PAPER *Special Issue on Synthesis and Verification of Hardware Design*

Synthesis of Multilevel Logic Circuits from Binary Decision Diagrams

Nagisa ISHIURA†, *Member*

SUMMARY In this paper, a new method of synthesizing multi-level logic circuits directly from binary decision diagrams (BDDs) is proposed. In the simple multiplexer implementation, the depth of the synthesized circuit was always $O(n)$, where n is the number of input variables. The new synthesis method attempts to reduce the depth of circuits. The depth of the synthesized circuits is $O(\log n \log w)$ where w is the maximum width of given BDDs. The synthesized circuits are 2-rail-input 2-rail-output logic circuits. The circuits have good *testability*; it is proved that the circuits are *robustly path-delay fault testable* and also *totally self-checking* for single stuck-at faults.

key words: logic synthesis, binary decision diagrams, combinational circuits

1. Introduction

Logic synthesis is one of the most important techniques in computer-aided design of VLSIs and a lot of researches are undertaken[6], [9], [12]. We discuss in this paper a new technique for *multi-level logic synthesis*. The most successful and prevalent way of generating multi-level logic circuits from certain forms of specification is to go by way of two-level representation of Boolean functions. The synthesis techniques have become so efficient that it is possible to synthesize practical multi-level circuits, only if objective functions are once represented by two-level representation. However, it is well known that there are Boolean functions, among practically important ones, whose two-level representations become exponentially big with respect to the number of the input variables. Symmetric functions including parity functions and Boolean functions required for realizing arithmetic functions are such examples. Since it is infeasible to generate two-level representations for these functions, it is important to develop complementary synthesis techniques which does not go through two-level representation. As one solution to this subject, we propose a method of generating multi-level logic circuit by way of *binary decision diagram (BDD) representation*[1], [4] of given Boolean functions.

A BDD represents Boolean functions in the form of directed acyclic graph. It is known that many practical Boolean functions are represented by practical size of BDDs[7]. For example, the size of a BDD for n-bit parity function is $O(n)$. In general, symmetric functions of n input variables are represented by the BDD of size $O(n^2)$. There are also a lot of researches to minimize BDDs for given Boolean functions[5], [8], [11]. It is considered to be a promising approach to use BDDs as complementary intermediate representations to two-level representations.

There have been attempts to synthesize combinational circuits or MOS implementation of them from BDDs[3], [13]. They are based on an idea that a given BDD is easily converted into a circuit implementation by replacing each node of the BDD by a selector or a switch. The size of the circuit obtained by this straightforward method is $O(N)$ where N is the number of nodes of a given BDD. However, the depth of the circuit is $O(n)$, where n is the number of input variables, even for simple functions. Of course, it is possible to reduce the depth by applying multi-level minimization methods[12], [9], global optimization is not always guaranteed. In contrast, the method proposed in this paper aims at synthesizing multi-level circuits of small depth directly from BDD's. A tree-like circuit is synthesized based on the fact that the Boolean functions represented by a BDD is computed by repetition of *Boolean matrix multiplications*. The depth of the synthesized circuits is $O(\log n \log w)$, where w is the maximum width of the given BDD. If the w is bounded by some polynomial of n, then the depth of the circuits is $O(\log^2 n)$ and much smaller than $O(n)$.

Another important feature of circuits synthesized by this new method is *high testability*. The circuits synthesized are 2-rail input and 2-rail output logic circuits. It is of course possible to further reduce the size of the circuits by converting them into usual 1-rail input and 1-rail output logic circuits. On the other hand, if we use the synthesized circuits as they are, the circuits are 1) robustly path-delay fault testable and 2) totally self-checking for single stack-at faults.

In the following section, notation and definition of BDDs and Boolean functions are described. In Sect. 3, the relation between Boolean functions represented by a BDD and Boolean matrix multiplication. In Sect. 4, the new synthesis method is presented based on the computation method in Sect. 3, and the testability of

Manuscript received January 12, 1993.
Manuscript revised April 19, 1993.

† The author is with the Faculty of Engineering, Osaka University, Suita-shi, 565 Japan.

the synthesized circuit is discussed in Sect. 5.

2. Binary Decision Diagram and Its Quasi-Reduced Form

2.1 Binary Decision Diagram (BDD)

A *binary decision diagram* (*BDD*) over $\mathcal{B}=\{0, 1\}$ is defined as follows. This is an *ordered*[4]and *shared* (multi-output)[10] BDD.

Definition 1: A BDD over $\mathcal{B}$ is a 7-tuple $B=(X, N_V, N_C, I, e^0, e^1, l)$, where

$X=\{0, 1, \cdots, n-1\}$ is a set of *variable indexes*.

N_V is a set of *variable nodes*.

$N_C=\{c_0, c_1\}$ is the set of *constant nodes*.

$I=\{i_1, i_2, \cdots, i_m\}\subset(N_V\cup N_C)$ is the set of *initial nodes*.

$e^0, e^1: N_V\rightarrow(N_V\cup N_C)$ represents the *0-edge* and the *1-edge* of variable nodes.

$l: (N_V\cup N_C)\rightarrow(X\cup\{n\})$ represents *level* of nodes and satisfies the followings:

$l(v)=n$ iff $v\in N_C$,

$l(u)<l(e^0(u)),\quad l(u)<l(e^1(u)).$ □

As for the variable order, node of larger level is located at the position nearer to the constant nodes. c_0, c_1 are called the *0-node* and the *1-node*, respectively. Pairs of nodes $(v, e^0(v))$, $(v, e^1(v))$ are called the *0-edge* and the *1-edge* of node v, respectively. The size of BDD B is defined as the number of nodes of B and denoted by size (B). The above definition allows BDDs which have nodes unreachable from any nodes. In order to exclude such BDDs, we assume that node $v\in N_V\cup N_C$ satisfies either $v\in I$ or $\exists u[e^0(u)=v\vee e^1(u)=v]$.

Each node v of a BDD represents Boolean function $f_v: \mathcal{B}^n\rightarrow\mathcal{B}$ which is defined as follows.

$f_{c_0}=0$ (inconsistency), $f_{c_1}=1$ (tautology),

$f_v=\overline{x_{l(v)}}\cdot f_{e^0(v)}+x_{l(v)}\cdot f_{e^1(v)}\quad(v\in N_V)$.

The Boolean functions $\{f_{i_1}, f_{i_2}, \cdots, f_{i_m}\}$ which are represented by the initial nodes of BDD B are called the Boolean functions represented by B.

2.2 Reduced BDD and Quasi-Reduced BDD

Variable node v is *redundant* if v satisfies $e^0(v)=e^1(v)$. Two variable nodes v_1 and v_2 are *equivalent* if v_1 and v_2 satisfy $e^0(v_1)=e^0(v_2)$ and $e^1(v_1)=e^1(v_2)$. It is possible to delete redundant nodes and equivalent nodes in BDD B without changing the Boolean functions represented by B. This operation is called *reduction*. It is known that the unique BDD is obtained for each Boolean function by repeatedly applying reduction for BDDs[4]. This BDD is called a *reduced BDD*.

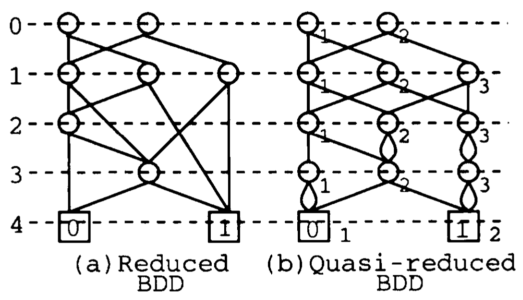


Fig. 1 Reduced BDD and quasi-reduced BDD.

BDD B is called *levelized* if every variable node v satisfies $l(e^0(v))=l(e^1(v))=l(v)+1$. Any BDD can be converted into a levelized BDD by adding redundant nodes. *Quasi-reduced BDD* is defined as a levelized BDD which has no equivalent nodes of the same level. Figure 1(a) is a reduced BDD and Fig. 1(b) is the quasi-reduced BDD that represents the same Boolean functions. Note that the 0-edge and 1-edge of a node is illustrated at the lower left and the lower right of the node respectively.

Proposition 1: Quasi-reduced BDDs have the following properties.

(1) There is a unique quasi-reduced BDD for each Boolean function.

(2) The size of a quasi-reduced BDD is at most n times as large as the size of the reduced BDD representing the same Boolean functions, where n is the number of input variables.

Input variable x_i is said to *support* function f_j if f_j is dependent on x_i (namely, $f_j|_{x_i=0}\neq f_j|_{x_i=1}$). For simplicity, we assume that there are no input variables that support none of the output functions. Namely, we assume that at least one of the nodes of each level is not a redundant node.

2.3 Reachability between Levels

A Boolean vector $a=(a_0, a_1, \cdots, a_{n-1})\in\mathcal{B}^n(n=|X|)$ is called an *assignment* to the variable vector $x=(x_0, x_1, \cdots, x_{n-1})$. We define *reachability* between nodes under an assignment.

Definition 2: $u\xrightarrow{a}v$ denotes that node v is *reachable* from u under assignment a, whose formal definition is as follows.

1) $u\xrightarrow{a}u$.

2) If $e^b(u)=v\wedge b=a_{l(u)}$ then $u\xrightarrow{a}v$.

3) If $u\xrightarrow{a}w$ and $w\xrightarrow{a}v$ then $u\xrightarrow{a}v$. □

For notational convenience, $u\xrightarrow{a}v\wedge v\xrightarrow{a}w$ is abbreviated as $u\xrightarrow{a}v\xrightarrow{a}w$. The Boolean functions represented by a BDD has close relation with reachability from the initial nodes to the constant nodes.

Proposition 2: The followings hold for each $v\in$

$(N_V \cup N_C)$.

$$f_v(a)=1 \quad \text{iff} \quad v \xrightarrow{a} c_1.$$

$$f_v(a)=0 \quad \text{iff} \quad v \xrightarrow{a} c_0.$$

3. Reachability Matrix

Definition 3: Let B be a quasi-reduced BDD. Let m_l and m_k be the numbers of nodes in levels l and k, respectively. Let us denote the nodes in level l by $\{v_l^1, v_l^2, \cdots, v_l^{m_l}\}$ and the nodes in level k by $\{v_k^1, v_k^2, \cdots, v_k^{m_k}\}$. Then *reachability matrix* $R_{l,k}$ is an $m_l \times m_k$ matrix whose (i, j)-element $r^{i,j}$: $\mathcal{B}^n \rightarrow \mathcal{B}$ is defined as follows:

$$r^{i,j}(a)=1 \text{ iff } v_l^i \xrightarrow{a} v_k^j. \qquad \square$$

Namely, (i, j)-element of $R_{l,k}$ is a Boolean function that becomes 1 if and only if j-th node of level k is reachable from i-th node of level l. For example, in the BDD in Fig. 1(b),

$$R_{0,1}=\begin{bmatrix} \overline{x_0} & x_0 & 0 \\ \overline{x_0} & 0 & x_0 \end{bmatrix}, \quad R_{1,2}=\begin{bmatrix} \overline{x_1} & x_1 & 0 \\ \overline{x_1} & 0 & x_1 \\ 0 & \overline{x_1} & x_1 \end{bmatrix}.$$

The following formula gives the general form of $R_{l,l+1}=[r^{i,j}]$. Elements of $R_{l,l+1}$ depends only on variable x_l.

$$r^{i,j}=\begin{cases} \overline{x_l} & (e^0(v_l^i)=v_{l+1}^j \wedge e^1(v_l^i)\neq v_{l+1}^j) \\ x_l & (e^0(v_l^i)\neq v_{l+1}^j \wedge e^1(v_l^i)=v_{l+1}^j) \\ 0 & (e^0(v_l^i)\neq v_{l+1}^j \wedge e^1(v_l^i)\neq v_{l+1}^j) \\ 1 & (e^0(v_l^i)=v_{l+1}^j \wedge e^1(v_l^i)=v_{l+1}^j) \end{cases} \qquad (1)$$

When $R_{l,h}$ and $R_{h,k}$ are given, $R_{l,k}$ can be computed by

$$R_{l,k}=R_{l,h}\cdot R_{h,k},$$

where$\cdot$means Boolean matrix multiplication (BMM). For example,

$$R_{0,2}=R_{0,1}\cdot R_{1,2}$$
$$=\begin{bmatrix} \overline{x_0}\,\overline{x_1}+\overline{x_0}x_1 & \overline{x_0}x_1 & x_0x_1 \\ \overline{x_0}\,\overline{x_1} & \overline{x_0}x_1+x_0\overline{x_1} & x_0x_1 \end{bmatrix}.$$

Note that $R_{0,n}$ represents reachability between the initial nodes and the constant nodes. For example, (1, 1)-element and (1, 2)-element of $R_{0,n}$ become 1 iff constant node c_0 and c_1, respectively, are reachable from the first initial node i_1. It means $r^{1,1}=f_{i_1}$ and $r^{1,2}=\overline{f_{i_1}}$. The following formula gives the general form of $R_{0,n}$.

$$R_{0,n}=\begin{bmatrix} \overline{f_{i_1}} & f_{i_1} \\ \overline{f_{i_2}} & f_{i_2} \\ \cdots & \cdots \\ \overline{f_{i_m}} & f_{i_m} \end{bmatrix}.$$

Namely, $R_{0,n}$ represents the Boolean functions represented by the BDD. $R_{0,n}$ can be computed by BMM from reachability matrixes for adjacent levels.

$$R_{0,n}=R_{0,1}\cdot R_{1,2}\cdot\cdots\cdot R_{n-1,n}. \qquad (2)$$

In the method presented in this paper, a combinational circuit that computes $R_{0,n}$ is synthesized according to the above formula. Circuits that computes $R_{l,l+1}$ and BMM are generated and combined. Reduction of depth of the circuit is achieved by constructing a *tree* of matrix multiplication circuits utilzing the property that matrix multiplication is *associative*.

4. Synthesis of Combinational Circuits

4. 1 Implementation of Boolean Matrix Multiplication

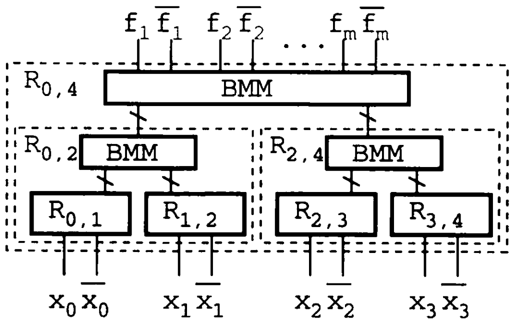
Let $P=[p^{i,j}]$, $Q=[q^{i,j}]$ be $p\times r$, $r\times q$ Boolean matrix. Then multiplication of P and Q results in $p\times q$ matrix $R=[r^{i,j}]$ whose (i, j) element is expressed as follows:

$$r^{i,j}=p^{i,1}\cdot q^{1,j}+p^{i,2}\cdot q^{2,j}+\cdots+p^{i,r}\cdot q^{r,j}$$
$$=\sum_{k=1}^{r} p^{i,k}\cdot q^{k,j}.$$

In the above formula, $\cdot$ represents *logical AND* and $+$ represents *logical OR*. Implementation of BMM by a combinational circuit is straightforward. It is implemented as a two-level *AND-OR* circuit. The circuit for each element of the resultant matrix has at most r product terms each of which consists of two literals. It follows that the total number of literals required for a whole BMM circuit is at most pqr. When we construct the circuit with *AND* and *OR* gates of two inputs, the depth and the number of gates are at most $1+\lceil \log r \rceil$ and $2pqr$, respectively.

4. 2 Synthesis of Combinational Circuit

Since the elements of $R_{l,l+1}$ are either 0, 1, $\overline{x_l}$ or x_l, as is shown in formula (1), $R_{l,l+1}$ is implemented only using connections. BMM is implemented as a two-level circuit as described in the previous subsection. Then a combinational circuit that implements the Boolean functions represented by a BDD is constructed according to formula (2). We can get different circuits depending on the order in which we apply BMM. In order to get the circuit of the smallest depth, we choose the order that leads to a tree of BMM circuits. For



$f_1 \overline{f_1}$ $f_2 \overline{f_2}$ ... $f_m \overline{f_m}$
$R_{0,4}$
BMM
$R_{0,2}$
BMM
$R_{2,4}$
BMM
$R_{0,1}$
$R_{1,2}$
$R_{2,3}$
$R_{3,4}$
$x_0 \overline{x_0}$ $x_1 \overline{x_1}$ $x_2 \overline{x_2}$ $x_3 \overline{x_3}$

Fig. 2 Block diagram of the synthesized circuit.

example, in the case of $n=8$, we construct a circuit according to the following association:

$$R_{0,8}=((R_{0,1}\cdot R_{1,2})\cdot(R_{2,3}\cdot R_{3,4}))\cdot((R_{4,5}\cdot R_{5,6})\cdot(R_{6,7}\cdot R_{7,8})).$$

Figure 2 shows a block diagram of the synthesized circuit for $n=4$. Outputs f_1, f_2, f_3 and f_4 represents f_{i_1}, f_{i_2}, f_{i_3} and f_{i_4}, respectively. The depth and the number of gates of the circuit are as in the following theorem.

Another constraint which should be considered in the synthesis method in this paper is optimization of the circuit. As is noticed in examples, reachability matrix contains a lot of 0's and 1's. Synthesized circuits can be much smaller by the transformation based on easiest simplification rules. In order to guarantee the testability discussed in the following section, we apply the transformation according to the following four rules.

1) $a\cdot 1 \rightarrow a$.
2) $a\cdot 0 \rightarrow 0$.
3) $a+1 \rightarrow 1$.
4) $a+0 \rightarrow a$.

We refer to the number of nodes in level l by the width of level l. The maximum width of a BDD is the maximum value of the width among all the levels. The depth and the size of synthesized BDDs are as follows.

Theorem 3: Let n and w be the number of the input variables and the maximum width of a given quasi-reduced BDD. Then the depth and the number of the gates of the synthesized circuit are $O(\log n \log w)$, $O(nw^3)$, respectively, under constant fan-in restriction.

Proof: As we shown above, the depth and the number of gates of BMM circuits, which computes multiplication of $p\times r$ and $r\times q$ Boolean matrixes, are at most $1+\lceil \log r \rceil$ and $2pqr$, respectively, when maximum fan-in is restricted to 2. Therefore the depth and the number of gates of each BMM circuit making up the synthesized circuits are $O(\log w)$ and $O(w^3)$ because p, q, and r are less than or equal to w. Since the whole circuit is made up of $n-1$ BMM circuits of $\log n$ levels, the depth and the number of gates of the circuit are $O(\log n \log w)$, $O(nw^3)$, respectively. □

5. Testability of the Synthesized Circuits

As is shown in Fig. 2, the synthesized circuit is a logic circuit of 2-rail inputs and 2-rail outputs (a 2-rail logic circuit). Although we can reduce the size of the circuit by transforming it into a single-rail logic circuit, the circuit has good testability if it is implemented as a 2-rail logic circuit.

(1) The synthesized circuit is *robustly path-delay fault testable*[6].
(2) The synthesized circuit is *totally self-checking*[2] for single stuck-at faults.

(1) means that all the path-delay faults, CMOS stuck-open faults and multiple stuck-at faults are testable. (2) means that the circuit is on-line testable. These testabilities are owing to the fact that the synthesized 2-rail logic circuit is *monotone*, namely, the circuit is constructed without negative gates. In the following subsections, the proofs for (1) and (2) are given. In order to establish the correspondence between assignments for given BDDs and input vectors for synthesized 2-rail logic circuits, we define mapping $d : \mathcal{B}^n \rightarrow \mathcal{B}^{2n}$ as follows.

$$d(a)=d_0d_2\cdots d_{n-1},\quad d_i=\begin{cases}01 & (\text{if } a_i=0)\\ 10 & (\text{if } a_i=1).\end{cases}$$

5.1 Robustly Path-Delay Fault Testability

We define the criticality of paths of a logic circuit. A sequence of signal lines in a logic circuit $s_1s_2\cdots s_m$ is called a path of the logic circuit if s_i is an input of gate g_i and s_{i+1} is the output of g_i(for $i=1, 2, \cdots, s_{m-1}$).

Definition 4: Let $d(a)$ be an assignment to a logic circuit. Path $p=s_1s_2\cdots s_m$ of the logic circuit is *critical* under $d(a)$ if a change of the signal value on s_1 propagates along p and observed at s_m.

Next, we define sensitization of edges and paths of BDDs.

Definition 5: Let $e=(u, v)$ be an edge of a BDD and a be an assignment for the BDD. Edge e is said to be *sensitized* by a if the following condition holds.

$$(e^0(u)=v\wedge a_{l(u)}=0)\vee(e^1(u)=v\wedge a_{l(u)}=1).\quad \square$$

A sequence of edges of a BDD $(v_1, v_2)(v_2, v_3)\cdots(v_{k-1}, v_k)$ is called a *path* of a BDD.

Definition 6: Path p is said to be sensitized by assignment a if all the edges constructing p are sensitized by a. □

Lemma 4: The number of paths from node v to node u which are sensitized at the same time by an assignment is at most one. In converse, if there is at least one path from v to u, there is an assignment that sensitizes the path.

Proof: We can describe a path from node u to node

v by the following expression using appropriate constants $b_0, b_1, b_2, \cdots, b_{k-1}$:

$$p=(v_0, v_1)(v_1, v_2)(v_2, v_3)\cdots(v_{k-1}, v_k),$$
$$v_0=u, v_k=v,$$
$$v_{i+1}=e^{b_i}(v_i) \quad (i=0, 1, 2, \cdots, k-1),$$

Let $l_u=l(u)$ and $l_v=l(v)$. The path from u to v that is sensitized by assignment a is described by the above expression by setting $b_0=a_{l_u}$, $b_1=a_{l_u+1}$, $b_2=a_{l_u+2}, \cdots$, $b_{k-1}=a_{l_v-1}$, which is unique. In converse, we can express any path from u to v by the above expression. By setting $a_{l_u}=b_0$, $a_{l_u+1}=b_1$, $=a_{l_u+2}=b_2, \cdots, a_{l_v-1}=b_{k-1}$, we can get assignment a that sensitizes the path. □

Let us denote the condition on which node v is reachable from node u by $r_{u,v}$. $r_{u,v}$ is a mapping $\mathcal{B}^n \rightarrow \mathcal{B}$ and $r_{u,v}(a)=1$ iff $u \xrightarrow{a} v$. The output of a BMM circuit represents $r_{u,v}$ for some nodes u and v. Let us denote the signal line that represents $r_{u,v}$ by $s_{u,v}$. Let us describe AND and OR gates whose output signal line is y and input signal lines are $a_1, a_2, \cdots, a_m$ by $y \leftarrow AND(a_1, a_2, \cdots, a_m)$ and $y \leftarrow OR(a_1, a_2, \cdots, a_m)$, respectively. Then $r_{u,v}$ is computed by the following subcircuit

$$s_{u,v} \leftarrow OR(s^1_{u,v}, s^2_{u,v}, \cdots, s^k_{u,v}),$$
$$s^i_{u,v} \leftarrow \begin{cases} s_{u,w_i} & (r_{w_i,v} \text{ is tautology}) \\ s_{w_i,v} & (r_{w_i,v} \text{ is tautology}), \\ AND(s_{u,w_i}, s_{w_i,v}) & (\text{otherwise}) \end{cases}$$

where w_i is a node of level $l_w(l(u)<l_w<l(v))$. $s^i_{u,v}$ represents the condition in which v is reachable from u by way of w_i. For this subcircuit, the following lemma holds.

Lemma 5: Let $d(a)$ be an assignment that sets the signal value of $s_{v,w}$, the output of an OR gate, to 1. Under this assignment, only one of the inputs $s^1_{u,v}$, $s^2_{u,v}, \cdots, s^k_{u,v}$ of the OR gate becomes 1 and the rest of them become 0. In converse, there is an assignment that sets the signal value of arbitrary input $s^i_{u,v}$ to 1 and the rest of the inputs to 0.

Proof: The fact that $s_{u,v}$ is 1 under assignment $d(a)$ is equivalent to $u \xrightarrow{a} v$. Similarly, the fact that $s^i_{u,v}$ is 1 under assignment $d(a)$ is equivalent to the fact that both r_{u,w_i} and $r_{w_i,v}$ are 1 and hence equivalent to $u \xrightarrow{a} w_i \xrightarrow{a} v$. Since only one path from u to v is sensitizable at the same time according to Lemma 4, only one of $s^1_{u,v}, s^2_{u,v}, \cdots, s^k_{u,v}$ becomes 1 and the others stay 0. The converse is also deduced from Lemma 4. Since there exists an assignment that sensitizes path $u \rightarrow w_i \rightarrow v$, $s^i_{u,v}$ can be set to 1. □

Lemma 6: If the value of primary output o is 1 under assignment $d(a)$, there are critical paths from primary inputs to o, which construct a tree.

Proof: Suppose output $s_{u,v}$ of a BMM circuit is taking value 1 and yet critical under assignment $d(a)$. Since only one input $s^i_{u,v}$ of the OR gate takes value 1 (Lemma 5), $s^i_{u,v}$ is critical. Then s_{u,w_i} and $s_{w_i,v}$ and signal lines connecting to them are also taking value 1 and critical.

Primary output o itself is taking value 1 and critical. By traversing signal lines from o according to the above discussion, we can get critical paths that reach primary inputs. We now prove that these critical paths construct a tree without reconvergences. Branches are caused only at AND gates. Let the inputs of an AND gate be s_{u,w_i} and $s_{w_i,v}$. The sets of the variables of the primary inputs that can reach s_{u,w_i} and $s_{w_i,v}$ are $\{x_{l(u)}, \cdots, x_{l(w_i)-1}\}$ and $\{x_{l(w_i)}, \cdots, x_{l(v)-1}\}$, respectively. Since the two sets are disjoint, there is no possibility of reconvergences. We can therefore conclude that the critical paths construct a tree. □

Let us denote the tree of critical paths for primary output o sensitized by assignment $d(a)$ by $C^o_{d(a)}$. Let p be a path from primary input i which is contained in $C^o_{d(a)}$. The responses for signal changes $0 \rightarrow 1$ and $1 \rightarrow 0$ can be observed at o because path p is critical. There are no hazards because there are no reconvergences (and also because the circuit is monotone). Therefore there is a robust test for the delay fault of path p.

If there is assignment $d(a)$ for an arbitrary path from a primary input to a primary output such that the path is contained in $C^o_{d(a)}$, the circuit is robustly path-delay fault testable. The premise is guaranteed by the following lemma.

Lemma 7: For an arbitrary path p from a primary input to a primary output, there is assignment $d(a)$ such that p is contained in $C^o_{d(a)}$.

Proof: Let $\{t_1, t_2, \cdots, t_k\}$ be set of all the output lines of BMM circuits and $R_{l,l+1}$ circuits that are contained in path p. Each t_j corresponds to the reachability between two nodes. Let us denote the two nodes by u_j and v_j. Let $T=\{u_1, v_1\} \cup \{u_2, v_2\} \cup \cdots \cup \{u_k, v_k\}$. Note that there are no two different nodes of the same level. Let q be a path in BDD that contain all the nodes in T, and let a be an assignment that sensitize q. The existence of a is guaranteed by Lemma 4. Since v_j is reachable from u_j for all j under assignment a, the signal values of $t_1, t_2, \cdots, t_k$ becomes 1. By the same discussion in the proof of Lemma 6, path p that contains those nodes is critical. □

Theorem 8: Synthesized circuits are robustly path-delay fault testable. □

5.2 Totally Self-Checking Property

A synthesized circuit has $2n$ inputs and $2m$ outputs. It accepts vectors in $D=\{01, 10\}^n$ and outputs vectors in $R=\{01, 10\}^m$. Vectors in D and R are called *code inputs* and *code outputs*, respectively. Vector in $\mathcal{B}^{2n}-D$ and $\mathcal{B}^{2m}-R$ are called *noncode inputs* and *noncode outputs*, respectively.

Definition 7: A circuit is *fault secure* if, for every fault from prescribed set F, the circuit never produces an incorrect code output for code inputs. A circuit is *self-testing* if, for every fault from F, the circuit produces a noncode output for at least one code inputs. A circuit is *totally self-checking* if it is both self-testing and fault secure. □

Lemma 9: Synthesized circuits are fault secure for single stuck-at faults.

Proof: Since the circuit is monotone, the effects of a single stuck-at fault on signal values are one way; the fault never causes both signal changes $0 \rightarrow 1$ and signal changes $1 \rightarrow 0$. It follows that signal values of a pair of outputs (0, 1) and (1, 0) never change into (1, 0) and (0, 1), respectively, and the circuit never outputs noncode outputs. □

Since a synthesized circuit is robustly path-delay fault testable, there is always a test vector for every single stuck-at fault. However, it does not necessarily mean that the circuit is self-testing, because the test vector may be a noncode input which assigns 0 to both of x_i and $\overline{x_i}$. Self-testing property of synthesized circuits is guaranteed by the following proof.

Lemma 10: Synthesized circuits are self-testing for single stuck-at faults.

Proof: We prove that there is, for each single stuck-at fault, a code input that detects the fault. First, we discuss faults in BMM circuits. Please remember the notation in Lemma 5:

$$s_{u,v} \leftarrow OR(s^1_{u,v}, s^2_{u,v}, \cdots, s^k_{u,v})$$

$$s^i_{u,v} \leftarrow \begin{cases} s_{u,w_i} & (r_{w_i,v} \text{ is tautology}) \\ s_{w_i,v} & (r_{w_i,v} \text{ is tautology}) \\ AND(s_{u,w_i}, s_{w_i,v}) & (\text{otherwise}) \end{cases}$$

According to the notation, we must consider eight cases for stack-at 0 (s-a-0) faults and stack-at 1 (s-a-1) faults of $s_{u,v}$, $s^i_{u,v}$, s_{u,w_i}, $s_{w_i,v}$. Since s_{u,w_i} and $s_{w_i,v}$ are symmetric, we only have to consider 6 cases.

1) s-a-0 fault of $s_{u,v}$: dominated by s-a-0 faults of $s^i_{u,v}$ in 3).

2) s-a-1 fault of $s_{u,v}$:

The existence of this fault makes $s_{u,v}$ always 1. If we map this effect to the original BDD, it corresponds to the fact that v is always reachable from u. We choose an initial node i from which there is a path to u and constant node c_b $(b \in \mathcal{B})$ to which there is a path from v. We also choose node v' to which there is a path from u and whose level is the same as v (Existence of node v' is guaranteed as follows: According to the simplification rules, the Boolean functions of $s_{u,v}$ is not tautology. It means that there is an assignment that makes v not reachable from u. Since the original BDD is a quasi-reduced one, there should be a node of the same level as v which is different from v and to which there is a path from u). Let a be an assignment which satisfies $i \xrightarrow{a} u \xrightarrow{a} v' \xrightarrow{a} c_{\bar{b}}$ and $v \xrightarrow{a} c_b$. This is possible because $f_v \neq f_{v'}$. Note that c_b is not reachable from i under assignment a. When the fault is not present, $i \xrightarrow{a} c_{\bar{b}}$ but not $i \xrightarrow{a} c_b$. However, when the fault is present, both $i \xrightarrow{a} c_{\bar{b}}$ and $i \xrightarrow{a} c_b$ hold because v is always reachable from u. This corresponds to the fact that both of the outputs for f_i and $\overline{f_i}$ take signal value 1 for input vector $d(a)$. Thus $d(a)$, which is a code input, is a test vector for the fault.

3) s-a-0 fault of $s^i_{u,v}$:

The effect of this fault corresponds to the fact that it becomes impossible to reach v from u by way of w_i under any assignments. A test vector for the fault is determined as follows. We choose an initial node i from which there is a path to u and constant node c_b $(b \in \mathcal{B})$ to which there is a path from v. Let a be an assignment that satisfies $i \xrightarrow{a} u \xrightarrow{a} w_i \xrightarrow{a} v \xrightarrow{a} c_b$. The existence of a is guaranteed by Lemma 4. When the fault is not present, c_b is reachable from i by way of u, w_i and v under assignment a and $c_{\bar{b}}$ is not reachable from i. However, when the fault is present, both c_b and $c_{\bar{b}}$ become unreachable from i. It corresponds to the fact that both of the outputs for f_i and $\overline{f_i}$ take signal value 0 for input vector $d(a)$. Thus $d(a)$, which is a code input, is a test vector for the fault.

4) s-a-1 fault of $s^i_{u,v}$: equivalent to s-a-1 fault of $s_{u,v}$ in 2).

5) s-a-0 fault of s_{u,w_i}: equivalent to s-a-0 fault of $s^i_{u,v}$ in 3).

6) s-a-1 fault of s_{u,w_i}:

The effect of this fault corresponds to the fact that v is reachable from u whenever v is reachable from w_i. By a similar discussion as in 2), there is a code input which detects the fault.

We next consider faults in $R_{i,i+1}$ circuits. This circuit is constructed according to formula (1). Let us denote the signal line that represents the reachability form u to v by $s_{u,v}$ and the signal line that represents x_i and $\overline{x_i}$ by s_{x_i} and $s_{\overline{x_i}}$, respectively. Then the circuit is described as

$$s_{u,v} \leftarrow \begin{cases} s_{\overline{x_i}} & \text{if } e^0(u) = v \wedge e^1(u) \neq v \\ s_{x_i} & \text{if } e^0(u) \neq v \wedge e^1(u) = v \end{cases}$$

The s-a-0 fault and the s-a-1 fault of $s_{u,v}$ is detected by the same discussion as in the BMM circuit. We must discuss s-a-0 faults and s-a-1 faults of s_{x_i} and $s_{\overline{x_i}}$. We only show the proofs for faults of s_{x_i} because the proofs for the $s_{\overline{x_i}}$ are the symmetric.

7) s-a-0 fault of s_{x_i}:

The effect of the fault corresponds to the fact that it is impossible to reach $e^1(u)$ from any node u of level i. Therefore, the fault is detectable if we choose test vector a that satisfies $i \xrightarrow{a} u \xrightarrow{a} e^1(u) \xrightarrow{a} c_b$, where i and c_b are a proper initial node and a proper constant node.

8) s-a-1 fault of s_{x_i}:

The effect of the fault corresponds to the fact that it is always possible to reach $e^1(u)$ from every node u of level i. Therefore, the fault is detectable if we choose test vector a that satisfies $i \xrightarrow{a} u \xrightarrow{a} e^0(u) \xrightarrow{a} c_b$, where i and c_b are a proper initial node and a proper constant node.

Now it is proved that all the single stuck-at faults are detectable by code inputs. □

Theorem 11: Synthesized circuits are totally self-checking for single stuck-at faults. □

6. Experimental Result

A synthesis program is implemented based on the method described so far. An experiment to synthesize multi-level circuits are conducted. The procedure for the experiment is as follows.

1) A BDD of the minimum size that represents Boolean functions of a given benchmark circuit is generated. SBDD package[10] was used for generating a multi-output BDD. The exact BDD minimization method in [8] was used to get the variable order except for '181' and 'adder16.' The orders for them were determined by hands (Note that the optimum variable orders for reduced BDDs are not always the optimum for quasi-reduced BDDs).
2) After removing attributed edges in SBDD package, the reduced BDD is converted into the quasi-reduced BDD.
3) A multi-level circuit is synthesized from the quasi-reduced BDD. In addition to the simplification stated in Sect. 4. 2, deletion of signal lines are achieved if there is a signal line that computes the same Boolean function as the signal lines (This simplification is considered to preserve the testability discussed in Sect. 5). No other optimization was applied.

Table 1 summarizes the result. The depth and the number of gates are ones obtained by using *and* and *or* gates of 4-inputs. CPU time shows the time required for 2) and 3) on the SPARCstation 1+. It is observed that the depth of the adders increases in proportion to the logarithm of the number of inputs. The result tells us that the depth is very small, but the circuit size is considerably large. One reason for this is that the circuits are 2-rail logic circuits with on-line testability. The other reason is that enough optimization has not been applied to the circuits. Since the circuits are considered to be globally optimal with respect to depth, they are expected to be a good initial solution for multi-level optimizer.

Table 1 Experimental Result.

circuit	depth	#gates	#literals	CPU time [sec]
dist	7	256	650	2.08
dk17	7	180	385	1.45
dk27	6	54	109	0.97
f51m	7	138	322	1.31
misex1	7	105	231	1.16
mlp4	7	363	890	1.97
rd53	6	47	98	0.86
rd73	6	76	162	0.96
rd84	7	107	231	1.14
root	7	131	313	1.24
sao2	9	190	430	1.56
sqr6	7	138	323	1.38
adder2	5	36	74	0.81
adder4	7	88	184	1.15
adder8	9	208	440	1.71
adder16	11	480	1024	3.74
181	9	1555	3873	122.69

7. Conclusion

A new method of synthesizing small depth and highly testable combinational circuits from BDDs is presented. The method is effective if the width of a BDD is small, but the circuit size becomes too large when the width of the BDD is large. Experimental results also show that the circuit size is very large. One reason for this is that enough optimization is not applied to the synthesized circuits. Detailed performance evaluation and investigation of applicability of other optimization methods is the future work to be carried out.

Acknowledgement

The author would like to thank to Mr. S. Minato of NTT for his SBDD package and also for his comments. I would also like to thank to Mr. H. Sawada of Kyoto University who offered me the optimum variable orders for benchmark circuits. I would like to thank to Mr. K. Abe for his help in the experiment for performance evaluation. The author would like to thank to Dr. M. Fujita of Fujitsu Laboratories for his discussions. I would also like to express sincere appreciation to all the members of Yajima Lab. of Kyoto University and Shirakawa Lab. of Osaka University for their comments and discussions.

References

[1] Akers, S. B., "Binary Decision Diagrams," *IEEE Trans. Comput.*, vol. C-27, no. 6, pp. 509-516, Jun. 1978.
[2] Anderson, D. A. and Metze, G., "Design of Totally Self-Checking Check Circuits for *m*-Out-of-*n* Codes," *IEEE Trans. Comput.*, vol. C-22, no. 3, pp. 263-269, Mar. 1973.
[3] Ashar, P., Devadas, S. and Keutzer, K., "Testability Properties of Multilevel Logic Networks Derived from

Binary Decision Diagrams," *Proc. Santa Cruz Conference on Advanced Research in VLSI*, Apr. 1991.

[4] Bryant, R. E., "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677-691, Aug. 1986.

[5] Butler, K. M., Ross, D. E., Kapur, R. and Mercer, M. R., "Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 417-420, Jun. 1991.

[6] Devadas, S. and Keutzer, K., "Synthesis and Optimization Procedure for Robustly Delay-Fault Testable Combinational Logic Circuits," *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 221-227, Jun. 1990.

[7] Ishiura, N. and Yajima, S., "A Class of Logic Functions Expressible by Polynomial-Size Binary Decision Diagrams," *Proc. Simulation and Synthesis Meeting and International Interchange*, pp. 48-54, Oct. 1990.

[8] Ishiura, N., Sawada, H. and Yajima, S., "Minimization of Binary Decision Diagrams Based on Exchanges of Variables," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 472-475, Nov. 1991.

[9] Matsunaga, Y. and Fujita, M., "Multi Level Logic Optimization Using Binary Decision Diagrams," *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 556-559, Nov. 1989.

[10] Minato, S., Ishiura, N. and Yajima, S., "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 52-57, Jun. 1990.

[11] Minato, S., "Minimum-Width Method of Variable Ordering for Binary Decision Diagrams," *IEICE Trans. Fundamentals*, vol. E75-A, no. 3, Mar. 1992.

[12] Muroga, S., Kambayashi, Y., Lai, H. C. and Culliney, J. N., "The Transduction Method — Design of Logic Networks based on Permissible Functions," *IEEE Trans. Comput.*, vol. C38, vol. 10, pp. 1404-1424, Oct. 1989.

[13] Sakurai, T., Lin, B. and Newton, A. R., "Multiple-Output Shared Transistor Logic (MOSTL) Family Synthesized Using Binary Decision Diagram," *UCB/ERL Report*, University of California at Berkeley, California, 1990.

Nagisa Ishiura was born in Kyoto, Japan, in 1961. He received B.E., M.E., and Ph.D. degrees in information science from Kyoto University, Kyoto, Japan, in 1984, 1986, and 1991, respectively. In 1987, he joined the Department of Information Science, Kyoto University, where he was an instructor until April 1991. Since May 1991, he has been a lecturer of the Department of Information Systems Engineering, Osaka University, Osaka, Japan. His current interests include design verification and test generation of digital circuits, logic synthesis, and hardware description languages. He is a member of IEEE and Information Processing Society of Japan.