SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test

A. Jutman, J. Raik, R. Ubar
Tallinn Technical University, Estonia,
email: artur@pld.ttu.ee

Abstract

In this paper we sum up the research that was done during the last decade on the topic of Structurally Synthesized Binary Decision Diagrams (SSBDDs). We describe general properties of SSBDDs that make this model very efficient for circuit structure dependent methods and algorithms. In addition, we describe a deterministic test generation algorithm based on SSBDDs and four efficient simulation methods of different classes: logic simulation, multi-valued simulation, timing simulation, and fault simulation. We investigate and show the origins of their common advantages and draw conclusions, which hold for all the described algorithms. The analysis is made on the basis of experimental data acquired when applying these algorithms to ISCAS'85 benchmark circuits. The experiments unveil some new properties of these benchmarks, which we also present in our paper.

1 Introduction

The constant increase of integration level of modern digital circuits imposes high and further growing requirements for the methods and algorithms used in CAD design, verification, and testing. The efficiency of any algorithm usually heavily depends on the underlying mathematical model. This made the search for better models a hot topic during the last several decades.

In 1986, Bryant proposed a new data structure called *Reduced Ordered Binary Decision Diagrams* (ROBDDs) [1,2]. He showed simplicity of manipulation and proved the model canonicity, what made it one of the most popular representations of Boolean functions. This model, however, suffers from the memory explosion problem, which limits its usability on large designs. Moreover, it cannot be used as a model for such methods that require a certain degree of structural information about the design.

During the last decade, many modifications to ROBDD model were proposed. These modifications were mostly aimed at fighting the memory explosion problem but very little was done to adopt BDDs for structural problems. For such problems, the general gate-level representation is traditionally used.

In this paper we show the advantageous properties of quite a mature but not widely known model of *Structurally Synthesized Binary Decision Diagrams* (SSBDDs). This compact model preserves the structural information about the modeled circuit and utilizes the partitioning of circuit into a set of subcircuits represented each by its own SSBDD.

We investigate common properties and advantages of four methods of logic-level simulation of digital systems and a deterministic test generation algorithm, which utilize SSBDDs as the underlying mathematical model. Logic-level simulation is still one of the most often used operations on digital designs during both design and test stages. This makes it a critical issue affecting the overall cost of a project. We show the origins of the SSBDD advantages in the logic-level simulation domain.

SSBDDs have already found application as an efficient mathematical model to represent digital circuits. They were introduced the first time in [3], [4] as *Structural Alternative Graphs* and generalized as *multiple-valued* decision diagrams in [5]. This model has several critical features that make it very attractive compared to other commonly used mathematical models, such as conventional BBDs (including state-of-the-art modifications) or a gate-level netlist.

First of all, the worst-case complexity (time) of generating SSBDD model from a circuit's net-list is linear in respect to the number of logic gates (due to circuit partitioning), while it is exponential for ROBDDs. Secondly, the size of the SSBDD model is linear in respect to the circuit size (again, ROBDD can be of exponential size). Thirdly, SSBDD model implicitly preserves structural information about the circuit while other BDD models do not. It also allows Boolean operations like AND, OR, or NOT with separate SSBDDs. Finally, it even reduces the model complexity compared to the gate-level representation, as algorithms running on the SSBDD model need no separate treatment of gates of different types (e.g. AND and OR gates are treated equally). Moreover, instead of considering each gate separately, it deals with *macros* – tree-like subcircuits (subcircuits with no reconvergent fanouts), which usually consist of several gates. Thus, each single node in an SSBDD represents a whole *signal path* from a macro input to the output of the macro. This is the most significant property, which allows development of efficient logic-level simulation algorithms. This property provides also fault collapsing for fault simulation and test generation.

Due to the above mentioned advantages the SSBDD model has been proposed as a representation model to solve various CAD problems like fast *deterministic* test pattern generation [12], efficient *design error* localization [13], *logic* and *multi-valued* simulation [6] for different purposes like hazards investigation, *delay fault* analysis, and *fault cover* analysis in dynamic testing. Efficient algorithms for *timing* and *fault* simulation were proposed in [7] and [8]. All of these methods use the advantage of the SSBDD model.

Current paper is organized as follows. In Section 2 we discuss the SSBDD model. The logic simulation is explained in Section 3, multi-valued and timing simulation methods are given in Section 4 and 5 correspondingly. Sections 6 and 7 give an overview of the SSBDD-based fault simulation and test generation. In Section 8 we discuss the experimental results and finally make concluding remarks in Section 9.

2 SSBDD Model: Important Properties

Def.1 A BDD that represents a Boolean function $y=f(X)$ over a set of Boolean variables $X=\{x_1, x_2, \dots, x_n\}$ is a directed acyclic graph $G_y=(M,\Gamma,X)$ with a set of nodes M and mapping Γ from M to M. Set M consists of two types of nodes: internal (non-terminal) M^N and terminal M^T, $M=M^N \cup M^T$. A terminal node m^T is labeled by a constant $e(m^T)\in\{0,1\}$ and is called *leaf*, while all non-terminal nodes $m\in M^N$ are labeled by variables $x\in X$, and have exactly two successor nodes. Let us denote the associated with internal node m variable as $x(m)$, then m^0 is the successor of m at the value $x(m)=0$ and m^1 is the successor of m at the value $x(m)=1$. An example of a BDD is given in Fig.1.

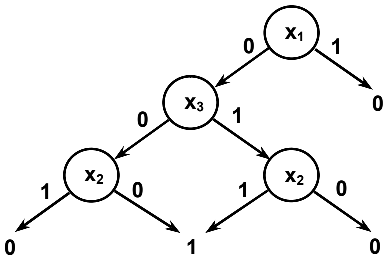

Fig.1. Binary Decision Diagram

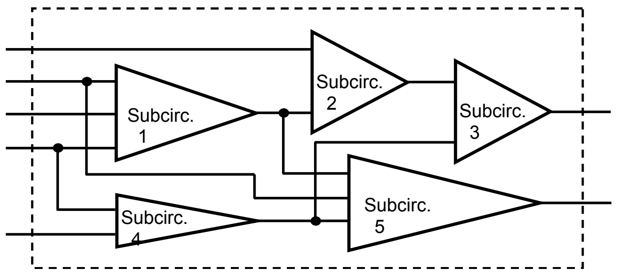

Fig.2. Combinational circuit partitioned into 5 fanout-free macros

There is a number of various kinds of BDDs have been proposed during last two decades [9]. The most of the modifications use the *Shannon decomposition* rule $y=\bar{x}f_{\bar{x}}+xf_x$. Here f_x is obtained from $y=f(x)$ by replacing variable x by value 1. The negative cofactor $f_{\bar{x}}$ is obtained by replacing variable x by value 0. This principle, along with the enforced total *variable ordering* on the graph and the *reduction rule*, produce a BDD with a canonical property. There are other kinds of BDDs that use different decomposition rules keeping, however, the property of canonicity [9].

SSBDD model is not a canonical model. In spite of this, as we will see later, it is a natural BDD representation of a digital circuit. It does not rely on the Shannon decomposition. As the basis, it uses the *equivalent parenthesis form* (EPF), that is, describes a digital circuit structurally.

Def.2 A BDD is called SSBDD, if there is a *one-to-one correspondence* between non-terminal nodes of the BDD and signal paths in the combinational circuit. Non-terminal nodes of an SSBDD are labeled by subscripted input variables, which can be inverted or not. In fact, SSBDD model is further defined by construction.

An SSBDD is constructed directly from a gate-level description of a combinational circuit by a graph superposition procedure. In this sense it is equivalent to EPF generation by superposition of Boolean functions. In prior to the superposition, the circuit must be partitioned into a set of tree-like fanout-free subcircuits (Fig. 2). Each such subcircuit will be represented by its own SSBDD. Therefore the whole circuit is represented not by a single SSBDD but by a set of separate SSBDDs connected by variables from extended set X.

Fig. 3 illustrates how an SSBDD is constructed from a combinational circuit. First, we set elementary BDDs: for AND and NOR gates. Then we use the superposition principle to combine

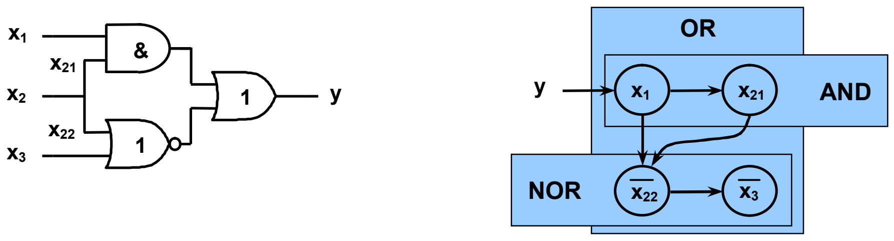

Fig.3. Illustration of the superposition principle

these into a whole single SSBDD just as OR gate combines the two gates into a single circuit. From the construction procedure it follows that it is possible to apply Boolean operations AND and OR to any two (or more) arbitrary SSBDDs. The result of this operation, a single SSBDD, will be then equivalent to the Boolean function resulting from the same Boolean operation applied to the same initial Boolean functions represented by the given (initial) SSBDDs.

Fig. 4 demonstrates two different SSBDD representations for the circuit from Fig. 3. For SSBDDs, it is agreed that the edge corresponding to the value $x(m)=0$ always goes down while the edge corresponding to $x(m)=1$ always goes right. Terminal nodes are not shown on this picture because it is also agreed that if $x(m)=0$ and no edge goes down from the node *m*, then one gets to the terminal node labeled by 0. Similarly, in the case of missing "right" edge, one reaches the terminal node "1".

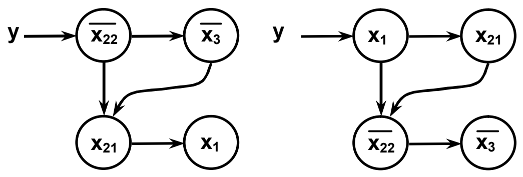

Fig. 4. Two different SSBDD representations for the combinational circuit from Fig. 3

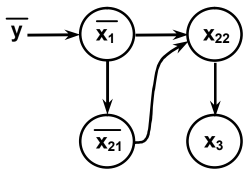

Fig. 5. An inverted SSBDD from Fig. 3

This unusual *alternative description style* used for SSBDD representation is possible due to introduction of the inversion of variables. In Fig. 4, variable x_{22} and x_3 are inverted. Otherwise it would be impossible to represent the NOR gate using this alternative description style.

It might seem useless to introduce the new style. However, it brings many advantages successfully utilized in the algorithms described in the following sections. It is not hard to notice that the SSBDD is a *planar graph* by construction. Together with the alternative description style this property allows to substantially reduce the search space of the algorithms working on the SSBDD model. In fact, this property is analogous to the property of monotony of Boolean EPFs. From Fig. 4 it is seen that no separate treatment for different logic gates needed in SSBDD model. This must be considered as an advantage compared to gate-level netlist representation.

Now we can mention that in addition to AND and OR operations, an operation of Boolean inversion can be introduced on SSBDDs. Fig. 5 illustrates an *inverted* SSBDD in respect to the SSBDD from Fig.3. In order to invert an SSBDD we have to perform two simple operations for each node of target SSBDD. First, swap the right and down edges. Second, invert all the variables.

Definition of the three Boolean operations (AND, OR, NOT) on SSBDDs allows to apply any arbitrary Boolean operation (as a combination of the defined operations) on an arbitrary number of SSBDDs. This property allows easy manipulations with the SSBDD model as well as easy construction of an SSBDD from a Boolean function.

Unlike the example in Fig 3, the automated SSBDD model construction procedure starts from a circuit output and moves along a signal path to the direction of inputs until a fan-out point is reached. Then the next signal path is chosen. The SSBDD construction stops when all the signal paths have been traversed up to their fanout points or primary inputs. The next SSBDD is constructed in the same way. The procedure stops when the whole circuit is covered. During such a procedure, the circuit partitioning is done in a natural way with no additional computational efforts.

SSBDD Construction Algorithm:

```
Construction (current gate)
{
  For each input of current gate
  {
    If the input is not primary or it is not a fanout
      Insert the next gate into the SSBDD
      Call Construction (next gate)
    End If
  }
Return
}
```

The latter recursive algorithm must be performed for each output and fanout point. It starts from the closest to the output (or the fanout point) logic gate and moves to the direction of inputs.

From the SSBDD model construction algorithm as well as from the definition we can derive the complexity of the model which is equal to the total number of signal paths in all the tree-like subcircuits. In the worst case this number is equal to the total number of all inputs of all the gates in the modeled circuit. In other words the size of this model has linear complexity *O(n)* in respect to the number of logic gates. Note that neither reordering nor decision making is done during the SSBDD model generation. This makes the worst case time of the model generation procedure linear in respect to the model size. Consequently, it is linear *O(n)* to the number of logic gates in the circuit as well.

Table 1. Comparison of sizes of different BDD models

Circuit	In	Out	Gates	ROBDD [2]	FBDD [11]	SSBDD
c432	36	7	232	30200	1063	308
c499	41	32	618	49786	25866	601
c880	60	26	357	7655	3575	497
c1355	41	32	514	39858	*N/A*	809
c1908	33	25	718	12463	5103	866
c2670	233	140	997	*N/A*	1815	1313
c3540	50	22	1446	208947	21000	1648
c5315	178	123	1994	32193	1594	2712
c6288	32	32	2416	*N/A*	*N/A*	3872
c7552	207	108	2978	*N/A*	2092	3552

Table 1 gives the model size comparison for several various BDD representations. As it is seen from the table, during the last decade there has been a substantial reduction in BDD model size from quite bulky ROBDDs to rather modest FBDDs (Free BDDs [11]). This became possible due to new sophisticated minimization algorithms and some modifications to the model itself. However, the SSBDD model still leads this race as it was twenty years ago. In average, it offers minimal model size independently of circuit function (for instance, there is no known efficient ROBDD representation for combinational multiplier). However, these two models are intended for different tasks as well. Since there is more than one SSBDD describing the same circuit, SSBDD model cannot be considered as a canonical one. This is the reason why the applications of SSBDDs and conventional BDDs are mostly different.

3 Logic Simulation

Two-valued *logic simulation* on SSBDDs is equivalent to path tracing procedure on graphs according to the values of variables at a given input pattern. An assignment to the variables *X* activates a path $l(m_0, m^T)$ from the *root* node m_0 to a terminal node m^T. The simulation procedure consists in tracing the path $l(m_0,m^T)$ and evaluating the $y=f(x)$ by finding the value *e* of the terminal node m^T.

Logic Simulation Algorithm:

```
For each SSBDD in the model
{
  Take the first node of current SSBDD
  While current node is not a terminal node
  {
    Evaluate current node and take the next node
  }
  Save the output value of current SSBDD
}
```

It is obvious that the worst case complexity of logic simulation is equal to the total number of nodes in SSBDDs. I.e. it is also linear $O(n)$.

4 Multi-Valued Simulation

For multi-valued simulation, we use a procedure based on calculation of Boolean derivatives on SSBDDs. Consi-der the set $S_5 = \{0, 1, \varepsilon, h, x\}$ for 5-valued simulation and a multi-valued vector $x^t = (x^t_1, x^t_2, \dots x^t_i, \dots x^t_n)$ for a transition period t. Denote a subset of literals with dynamic values at this vector by $X_D \subseteq X$, i.e.

$$X_D = \{x_i \,/ x^t_i \in S_5 \cap S_D\} = \{x_i \,/ x^t_i \in \{\varepsilon, h, x\}\}$$

Denote $l(m_i,m_j) = 1$, if there exists an activated path between the nodes m_i and m_j for a given vector x^t, otherwise, $l(m_i,m_j) = 0$.

Theorem 1. Given $y = f(x)$ and $x_i \in X$, the condition $dy/dx_i = 1$ for SSBDD $G_y = (M,\Gamma,X)$ where $x(m) \equiv x_i$ is equivalent to the following equation:

$$l(m_0,m) \wedge l(m^1, m^{T,1}) \wedge l(m^0, m^{T,0}) = 1$$

The proof of the theorem can be found in [6].

Note, Theorem 1 can be used for calculating Boolean derivatives only in the case where vector x^t is two-valued, because only in this case all the paths are activated uniquely. The general case, when x^t is a multi-valued vector, is considered in the following theorem.

Def.3 Let $\max\{l(m_i,m_j)\}=1$ over $x_k \in X_D$, if there exists at least one activated path between m_i and m_j for all two-valued assignments of $x_k \in X_D$, otherwise $\max\{l(m_i,m_j)\}=0$.

Theorem 2. Given $y=P(X)$ and $x_i \in X$, the condition

$$\max_{x_k \in X_D} \{dy/dx_i\}=1$$

for SSBDD $G_y = (M,\Gamma,X)$ where $x(m) \equiv x_i$ is equivalent to the following equation:

$$\max_{x_k \in X_D}\{l(m_0,m)\} \wedge \max_{x_k \in X_D} \{l(m^1,m^{T,1})\} \wedge \max_{x_k \in X_D} \{l(m^0,m^{T,0})\} = 1$$

The proof of the theorem is analogous to the proof of the Theorem 1.

For calculating the maximum of a Boolean derivative and proving that $\max\{dy/dx(m)\} = 1$, all dynamic values when tracing the path $l(m^1, m^{T,1})$ should be replaced by 1 and when tracing the path $l(m^0, m^{T,0})$ by 0. This follows from the property of monotony of Boolean EPFs. When tracing the path $l(m_0,m)$, all dynamic values should be replaced either by 1 or by 0 properly, so that the node m can be reached. In fact, instead of sequentially calculating the maximum of derivatives separately step by step for all the nodes m, where $x(m) \in X_D$, we can traverse all the paths from all the nodes m: $x(m) \in X_D$ in both directions by a single procedure based on nested calculation of all the derivatives. The successful utilization of the described idea allowed the creation of an efficient algorithm with linear worst-case complexity $2n$.

We do not give the multi-valued simulation algorithm here because it does not have a nice compact representation. It is thoroughly described in [6] instead.

5 Timing Simulation

The timing simulation approach is based on the same principle of calculation of Boolean derivatives on SSBDDs. The difference between these methods is that in multi-valued simulation we are tracing paths to search for the nodes with variables having dynamic values while in timing simulation we are searching for nodes that switch in current moment of time (i.e. a notion of time is introduced). Unlike in multi-valued simulation, the complexity of the problem of timing simula-tion itself is NP-complete and does not depend on the underlying model. It is due to the fact that a single input transition may result in exponentially long event sequences in certain circuits. On the other hand, the worst number of events at the output of a tree-like fanout free circuit is equal to the total number of events at its inputs. This means linear complexity. The number of events in a common circuit is somewhere in between. In fact, it is far from exponential case. Otherwise, the timing simulation of an ISCAS circuit for a modest number of input vectors would be impossible.

In the following we give a simplified timing simulation algorithm on SSBDD model. The detailed description of the algorithm, which exploits the property of monotony of Boolean EPFs and other SSBDD properties, is given in [7].

Timing Simulation Algorithm:

```
For each clock cycle
{
  Set current time to 0
  Perform Logic Simulation and evaluate the output
  Make list of all the nodes with transitions
  Sort list by time of transition in ascending order
  For all entries in the event list
  {
    Set current time to the value from the list entry
    Apply the specified transition
    Perform Logic Simulation and evaluate the output
    If current output value differs from previous
      Assign current time to the output transition
} }
```

We had to introduce a notion of time into SSBDD model in order to make the latter algorithm work. In this extended model a certain delay is assigned to each node. This delay is calculated as a sum of delays of all logic gates along the path represented by the corresponding node [6]. Fig. 6 illustrates this idea.

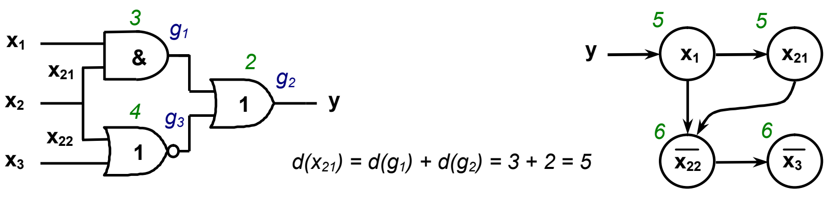

Fig. 6. Modeling delays on SSBDDs

6 Fault Simulation

In general, the fault simulation task is considerably complex, since several stuck-at faults can be simultaneously present in the circuit. A circuit with *n* lines can have $3^n - 1$ different stuck line combinations [10]. Needless to say that even a moderate value of n will result in an enormous amount of multiple faults. It is a common practice, therefore, to model only single stuck-at faults. An n-line circuit has $2n$ single stuck-at faults. In the following we will see that modeling stuck-at faults on SSBDD representations allows to reduce this number further.

It is possible to minimize the number of faults to be modeled by a technique called *fault collapsing*. Traditionally, this is done by implementing the relation of *fault dominance* on the set of faults. It is said that fault f_1 dominates f_2, if any test that detects f_2 will also detect f_1. Representing stuck-at faults by faults at SSBDD nodes can be viewed as a type of fault collapsing by applying fault dominance relations along the signal paths of the circuit.

While in the gate-level descriptions we model stuck-at faults at the interconnections between the gates, in SSBDD representations the faults are considered at nodes. For example, stuck-at-0 fault at a node is modeled with the 0-edge of the node being constantly activated, regardless of the value of the variable labeling this node. As it was stated in Def. 2, each SSBDD node represents a distinct path in the corresponding fanout-free circuit. By testing all the SSBDD node faults we will consequently test all the paths in the circuit and thus all the single stuck-at faults. This ability of SSBDDs to implicitly model logic level stuck-at faults is a very important property, which distinguishes it from other classes of BDDs.

Table 2 presents the number of uncollapsed faults, collapsed faults and SSBDD faults in eight ISCAS85 circuits. As we can see from the table, the traditional fault collapsing and SSBDD representations provide almost identical results. The difference in the number of faults is at most 8

Table 2. Comparison of collapsed and SSBDD faults

Circuit	Uncollapsed	Collapsed	SSBDD
c880	1550	**942**	994
c1355	2194	**1574**	1618
c1908	2788	1879	**1732**
c2670	4150	2747	**2626**
c3540	5568	3428	**3296**
c5315	8638	**5350**	5424
c6288	9728	**7744**	**7744**
c7552	11590	7550	**7104**

% (in the case of c1908). SSBDD achieves in average even 2 % better compaction of the fault list than the traditional approach, reducing the fault lists in average about 1.5 times. However, this advantage occurs partly because in ISCAS models, for every signal line, at least one fault is required to be included to the fault list. In a more advanced collapsing techniques the list can be further minimized. Nevertheless, Table 2 indicates that by generating SSBDDs, the fault list will simultaneously be reduced by a considerable amount of faults.

The advantage of the SSBDD based collapsing over the traditional one is that it allows us at the same time to rise to a higher abstraction level of circuit modeling. In the traditional case we would only minimize the number of faults but would still be working at the level of logic gates.

A simple fault simulation algorithm of complexity $O((n+1)^2)$ is given in the following.

Fault Simulation Algorithm:

```
Perform fault-free Logic Simulation
For each SSBDD
{
  Take the first node of current SSBDD
  While current node is not a terminal node
  {
    Evaluate current node
    Insert corresponding fault
    Perform Logic Simulation to evaluate output
    If current output differs from the fault-free
      Save current fault as detected
    Remove current fault and take the next node
} }
```

More sophisticated and efficient *parallel critical path tracing fault simulation* method is described in [5]. It is based on combining the parallel backward critical path tracing inside fanout free subcircuits with parallel forward critical path tracing between fanout free subcircuits for fanout stem analysis.

The basic idea of parallel fault simulation on SSBDD models is similar to this type of simulation at gate-level. Here, traditional style SSBDD descriptions are preferable since in these we do not have to keep track of whether a variable labeling a node is inverted or not. The simulation takes place as follows. Starting from the node with the highest index value, we repeat operation:

$$\left(x(m)\,\&\,x(m^1)\right)\vee\left(\overline{x(m)}\,\&\,x(m^0)\right)$$

for each node of the SSBDD. In this operation, m denotes the current node. m^0 and m^1 are its 0 and 1-successors, respectively. x(m) denotes the value of the variable labeling the node m. The result of the simulation will be the value calculated for the root node m_0.

7 Test Generation

The test generation approach presented in this paper would be an exact equivalent of PODEM [14] if SSBDD models consisted of a system, where for each logic gate an elementary BDD corresponded. In order to compare test generation on structurally synthesized BDD and gate level models, we have implemented two types of SSBDD synthesis: one, where BDDs are synthesized for each FFR and the other, where they are synthesized for each gate. In the following, different tasks of the SSBDD-based test generation process are explained.

Determining the D-Frontier. Similar to classical test generation approaches, D-frontier is determined by finding all the nodes labeled by variables with the value D (or D-bar) and performing X-path check for them.

X-Path Check. X-path check for a circuit line with the value D (or D-bar) checks for the existence of such a path from that line to a primary output, where all the lines have the value X. At the gate-level, the outputs of each gate along corresponding paths have to be checked. In the SSBDD representations, the primitives are reduced to SSBDDs, which significantly speeds up the procedure. In SSBDD models it is possible to perform X-path check by simply checking the values of variables corresponding to the respective SSBDDs.

Finding Boolean Derivatives. Calculating Boolean derivative for the nodes in SSBDD is similar to determining current backtrace objective in PODEM. In PODEM, the aim of selecting current objective was to propagate the fault effect (D or D-bar) to an output of a subsequent gate. In our approach, the primitives are SSBDDs and therefore the objective is to propagate the fault effect to the output of a subsequent fanout-free region.

In order to check that Boolean derivative for a node *n* in an SSBDD *G* is not zero, three paths have to be traversed. One from the root node of *G* to the node *n*. The two other paths have to be traversed downwards and rightwards from *n*, respectively. If a path from root to node *n* can not be traversed due to the values assigned to the variables in the SSBDD, or if the two last mentioned paths overlap, Boolean derivative for node *n* will be zero. The derivative will be zero also if the downward path exits the BDD rightwards or rightward path exits the BDD downwards. If the derivative is one, current objective will be to backtrace the variable whose value is X and that labels a node that was traversed along one of the three paths. There exists always at least one such variable. The backtrace value is determined by the traversal direction of the respective node.

The use of Boolean derivatives for choosing current objective is the reason why on SSBDDs higher fault coverages were achieved than in the case where BDD models were representing logic gates (see Table 2). Due to the fact that primitives are given on the level of FFRs rather than gates, many inconsistencies are detected earlier.

Backtrace. Backtrace on the SSBDD models is simple and very fast. Value *V* is backtraced on an SSBDD *G* as follows. Starting from the root node we traverse a path forced by variable values until the first node *n* labeled by variable *x(n)*, which has the value X, is reached. Subsequently, we will backtrace the value, which activates node *n* to the value *V*, on the SSBDD corresponding to the variable *x(n)*. Value *V* is recursively backtraced on the SSBDD model until primary inputs are reached.

D-Simulation. D-simulation is the most time-consuming procedure in current implementation. This can be expected since it is called repeatedly after each value assignment to the primary inputs. At present, we use three-valued simulation (0, 1, X) on two vectors, one for the fault-free and the other for the faulty circuit.

Fault Simulation. Subsequent to each generated test, fault simulation is performed in order to determine the faults covered by the generated vector.

In the following the general algorithm of SSBDD-based test generation is given. The following definitions are used.

Fault node is the SSBDD node under test.

Fault variable is the variable labeling the fault node.

Fault graph is the SSBDD that contains the fault node.

<u>*Test Generation Algorithm:*</u>

```
While D not propagated to a PO
{
  If fault variable = X
  { Backtrace value that activates the fault }
  Else
  {
    If fault is not activated
    { Backtrack }
    Else
    {
      If the value of fault graph not D
      {
        If Boolean derivative for fault node is 0
        { Backtrack }
      }
      Else
      {
        Determine the D-frontier
        Perform backtrace with current objective
} } } }
```

8 Experimental Results

The experiments with the four described methods of logic-level simulation were carried out on ISCAS'85 benchmarks. We run them for each circuit on two levels of abstraction: a gate-level and SSBDD model. This allows to measure directly the effect of the chosen model on simulation time.

In all cases the speed of simulation for SSBDD model was higher than that for gate-level representation [6,7]. For logic, multi-valued, and timing simulation the average speed-up varies from about 1,5 up to almost 4 times compared to algorithms working on the gate-level netlist model (Fig. 7). The fault simulation algorithm (fat dashed line in Fig. 7) shows the most noticeable acceleration. The runtime improves up to 7 times when simulation is performed on SSBDD model.

The rise of simulation performance (and, in fact, reduction of the capacity of required memory) becomes possible due to the model complexity reduction by shifting from lower gate level to a higher macro level of fanout free subcircuits (Fig. 2). On this macro level all the required structural information is implicitly preserved due to the use of SSBDDs.

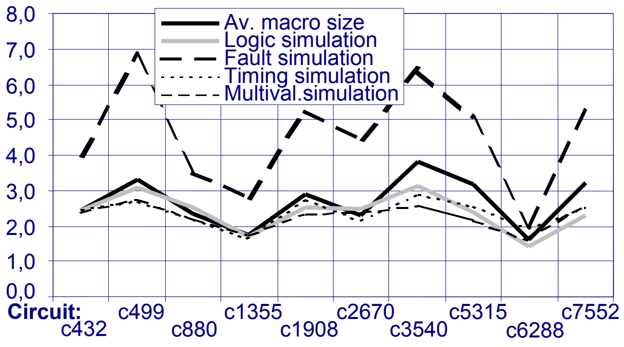

Fig. 7. Logic-level simulation speedup for different algorithms

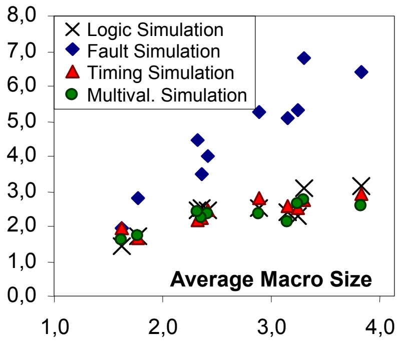

Fig. 8. Logic-level simulation speedup vs. average macro size

One can also notice a very interesting property of the methods researched, i.e. their results for all the circuits are correlated. The origin of this effect is the variance of the average size of macros (measured in the number of gates) for various circuits. This property is shown in Fig. 7 by the bold black line. The behavior of this line is also correlated to the behavior of the simulation curves. This unveils the fundamental property of the investigated methods, which is that the average simulation speed-up is directly proportional to the average size of a fanout-free subcircuit in the circuit (Fig 8).

The same property can be used also to describe ISCAS'85 benchmarks themselves. In fact, it is hard to detect any relationship that holds with no exception for all circuits from ISCAS 85 package. For instance, circuit size is not much informative if we need to estimate a BDD generation time. This is also useless for estimation of simulation speedup we earn from SSBDD model. Now we can arrange these benchmarks by the average size of fanout-free subcircuits (macros). This order is significant at least for an evaluation of methods, which use SSBDDs as the underlying model. The order is presented in Table 3.

Table 3. Benchmarks' order according to the average macro size

c6288	c1355	c2670	c880	c432	c1908	c5315	c7552	c499	c3540
1,62	1,77	2,32	2,36	2,42	2,90	3,15	3,24	3,30	3,83

Table 4 presents experimental results that were measured on a 233 MHz Pentium II PC under Windows 95 operating system. In the table, comparative test generation results on SSBDD and gate levels are given for two different timeout values. The achieved fault coverages were calculated by simulating faults of the test patterns on the noncollapsed gate-level model. Therefore, the coverages differ slightly from the ones of the ‘classical’, collapsed, fault model. As the Table shows, test generation times are 2.1 – 4.5 times faster at the SSBDD level than at the gate-level. Thus, SSBDD descriptions appear to be an efficient model for the test generation purposes.

Table 4. Comparison of SSBDD and gate-level test generation

Circuit	Fault coverage, %		Time, s	
	SSBDD	Gate	SSBDD	Gate
c432	97.33	87.06	0.10	0.32
c880	100.0	100.0	0.05	0.11
c1355	99.64	99.64	0.24	0.64
c1908	99.75	99.46	0.22	0.65
c2670	96.67	95.16	0.55	1.78
c3540	95.58	95.24	0.77	3.47
c5315	99.78	98.90	0.57	2.75
c6288	99.80	99.30	0.60	1.45
c7552	99.46	97.10	2.71	11.5

9 Conclusions

Common features and advantages of four logic-level simulation methods implemented on the SSBDD model are discussed in the paper. It is shown that the efficiency of the algorithms directly depends on the underlying model. The simulation speed-up in comparison with the gate-level simulation speed is linearly proportional to the average size of a macro measured in the number of gates. This statement holds for at least common realistic combinational circuits such as the ISCAS ’85 benchmarks.

The algorithm that most benefited from the usage of the SSBDD model is the fault simulation algorithm, as it utilizes at the same time fault collapsing together with the model simplification.

The experiments show also the advantages of SSBDD-based TG algorithm in respect to a similar algorithm working on the gate-level netlist.

For more details on each of the discussed approaches look in [6], [7], and [8].

References

[1] R. Bryant “Graph-based algorithms for Boolean function manipulation”, *IEEE Transaction on Computers*, 1986, vol. C-35, pp. 677-691.

[2] K.S. Brace, R.L. Rudell, and R.E. Bryant, "Efficient Implementation of a BDD Package," In *Proc. of the 27th DAC*, June 1990, pp. 40-45

[3] R. Ubar, “Test Generation for Digital Circuits Using Alternative Graphs (in Russian)”, in *Proc. Tallinn Technical University,* 1976, No.409, Tallinn Technical University, Tallinn, Estonia, pp.75-81.

[4] R. Ubar, “Beschreibung Digitaler Einrichtungen mit Alternativen Graphen für die Fehlerdiagnose,” *Nachrichtentechnik/Elektronik,* (30) 1980, H.3, pp.96-102.

[5] R. Ubar, “Test Synthesis with Alternative Graphs,” *IEEE D&T of Comp.* Spring 1996, pp. 48-59.

[6] R. Ubar, “Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams,” *OPA, Gordon and Breach Publishers, Multiple Valued Logic,* 1998, Vol.4, pp. 141-157.

[7] R. Ubar, A. Jutman, Z. Peng, “Timing Simulation of Digital Circuits with Binary Decision Diagrams”, in *Proc. of DATE 2001 Conference*, München, Germany, 2001, pp. 460-466.

[8] R. Ubar, “Parallel Critical Path Tracing Fault Simulation,” in *Proc. of the 39. Int. Wiss. Kolloquium*, Ilmenau, Germany, 1994, Band 1, pp. 399-404.

[9] A.Narayan, “Recent Advances in BDD Based Representations for Boolean Functions: A Survey”, in *Proc. 12th International Conference on VLSI Design*, Goa, India, 1999, pp. 408-413.

[10] M. Abramovici, et al., *Digital Systems Testing and Testable Design*, New York, IEEE Press, 1990, 652p.

[11] W. Günther, R. Drechsler, "Minimization of Free BDDs," In *Proc. of Asia and South Pacific Design Automation Conf.*, Hong Kong, Jan 1999, pp. 323-326

[12] J. Raik, R. Ubar, “Feasibility of Structurally Synthesized BDD Models for Test Generation,” *Compendium of Papers of the European Test Workshop*, Barcelona, May 27-29, 1998, pp. 145-146.

[13] A. Jutman, R. Ubar, “Design Error Diagnosis in Digital Circuits with Stuck-at Fault Model,” *Journal of Microelectronics Reliability. Pergamon Press, Vol. 40, No 2*, 2000, pp. 307-320.

[14] P.Goel, “An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits”, IEEE Trans. Comput., pp. 215-222, March 1981.