APPLICATION OF STRUCTURALLY SYNTHESIZED BINARY DECISION DIAGRAMS FOR TIMING SIMULATION OF DIGITAL CIRCUITS

Artur JUTMAN and Raimund UBAR

Computer Engineering Department, Tallinn Technical University, Raja 15, 12618 Tallinn, Estonia; {artur, raiub}@pld.ttu.ee

Abstract. Meeting the timing requirements is an important constraint imposed on highly integrated circuits, and the verification of timing of a circuit before manufacturing is one of the critical tasks to be solved by CAD tools. In this paper, we present a novel technique to speed up gate-level timing simulation that is based on Structurally Synthesized Binary Decision Diagrams (SSBDD), which have already found application as an efficient mathematical model to represent digital circuits. The new approach uses path delays instead of gate delays for tree-like subcircuits (macros). Therefore timing waveforms are calculated not for all internal nodes of the gate-level circuit, but only for outputs of macros. The macros are represented by SSBDDs, which enable a fast computation of delays for macros. We show that the speed-up of timing simulation is directly proportional to the average size of macros in the circuit. The new approach to speed up the timing simulation is supported by encouraging experimental results.

Key words: timing simulation, binary decision diagrams, delay modelling.

1. INTRODUCTION

The transition from the traditional ASIC (Application-Specific Integrated Circuits) to SoC (System-on-Chip) has led to new challenges in design methods, manufacturing, verification, and test. Timing simulation is a widely used method to verify the timing behaviour of a digital design. In a synchronous digital system, the timing property that is needed to be verified is that for each input vector transition the combinational logic settles to a stable state within a given clock period. One approach to ensure this is to use timing simulation.

There are different methods to model the delays in digital circuits, including the *zero-delay*, *unit-delay and multiple-delay* models [1]. While the zero-delay

models can be used to analyse combinational circuits without memories, and unit-delay models can be used to verify the logical behaviour of synchronous sequential circuits, they are inadequate for analysing the timing behaviour of digital circuits. For the timing behaviour, a multiple-delay model should be used. In such a model, each circuit element is assigned a delay which is an integer multiple of a time unit. Usually separate rise and fall delays are specified. If the gate delays are not functions of the direction of the output change, we can use a *transition-independent delay* model. In the following we use a *nominal-delay* model [2] with the assumption that the gate delays are known.

In the classical gate-level delay simulation [2], all the gates should be evaluated once per cycle which leads to a great amount of simulation with circuits of high complexity. The first approach to cope with the complexity was the event-driven method, when a gate is simulated only if a transition occurs at its input [3–9]. This approach has found broad application due to its flexibility and capability of handling different delay models and complex signal states. However, its major drawback is the useless processing of a great amount of events that do not change the output of any gate. It is also possible that exponentially long sequences of transitions can result from a single input change. Finally, most of the event-driven approaches are interpretive and cannot use the model compilation technique, which can substantially reduce the run time of the algorithm [10–12].

To suppress the useless events and evaluations, the demand-driven algorithm [13] and event-suppression approach [14] were proposed. Clock-suppression methods [15–19] were proposed for synchronous circuits in order to avoid evaluation of cyclic signals (synchronizers). For this class of circuits, cycle-based simulators [20–23] were also elaborated. They assume that only the final values at the output pins are reported. This means that intermediate simulations can be performed very fast using simple logic values (0 and 1) and the zero-delay model. Some of the event-driven approaches [24,25] use the extended Petri net model (timed Petri nets) to incorporate parallel and distributed simulation techniques. Multi-level timing simulation [26–28] as an addition to event-driven concept is one more attempt to reduce the complexity of the proposed model.

Another big cluster of timing simulation methods is symbolic approach [29–34]. These methods use *signal algebra* to represent different transitions and signal states in a circuit. Usually, this technique is not accurate enough unless a sophisticated algebra is used. For instance, in [32] a 9-valued algebra and in [34] a 13-valued signal algebra is proposed.

Our approach is based on SSBDDs which have already found application as an efficient mathematical model to represent digital circuits. They were introduced in [35,36] as structural alternative graphs, and generalized for the multiple-valued decision diagrams in [37]. SSBDD model has several critical features making it very attractive compared to other commonly used mathematical models, such as, for example, Ordered Binary Decision Diagrams

(OBDD) or a simple gate-level representation. First of all, the SSBDD model is generated from a circuit's netlist within linear time (for OBDD it can be exponential) compared to the circuit size. Secondly, the size of the SSBDD model is linear relative to the circuit size (again, OBDD can be of exponential size). Thirdly, SSBDD model preserves structural information about the circuit while other BDD models do not. And finally, it even reduces the model complexity compared to the gate-level representation, because instead of considering each gate separately, it deals with *macros* – tree-like subcircuits (i.e. subcircuits with no reconvergent fanouts), which usually consist of several gates.

In [38] SSBDDs were suggested for multivalued simulation of digital circuits for different purposes like hazards investigation [39], delay fault analysis [40], and fault cover analysis in dynamic testing [41]. Efficient algorithms for logic and fault simulation were described in [42] and [43], respectively. A fast deterministic test pattern generator based on SSBDDs was proposed in [44], and in [45] an efficient design error localization technique was introduced, which also utilizes the advantages brought by the SSBDD representation.

The use of SSBDDs in timing simulation allows modelling path delays in macros and calculation of timing waveforms at macros' outputs instead of providing such calculations at the output of each gate. To each path in a macro we assign a delay (or two delays in the case of transition-dependent delay model). For simplicity, in this paper, without loosing the generality, we consider the one-delay case for each path. The paths are considered only inside macros. For this reason, we avoid the exponential explosion of the number of paths processed. For example, assume that the subcircuit in Fig. 1 is represented by a macro. This macro is characterized by 6 paths and 6 delays, calculated on the basis of gate delays. When representing complex gates by macros, the number of macros is equal to the number of tree-like subcircuits in the complex gate. For example, a one-bit multiplexer is represented by a single macro.

In this paper we present a novel method for delay simulation based on Boolean derivatives and SSBDD representation. Section 2 describes *equivalent parenthesis forms* (EPF) for a given digital circuit. In Section 3 the main
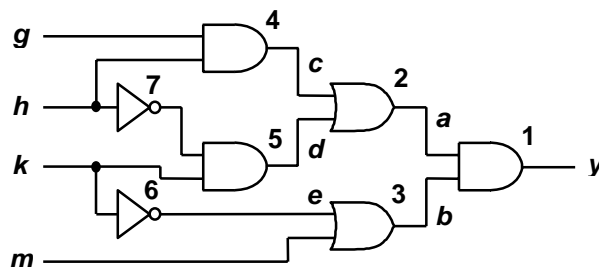
Fig. 1. Digital subcircuit.

considerations about timing simulation based on Boolean derivatives are given and in Section 4 an efficient implementation of this approach on SSBDDs is described. Our algorithms are explained in detail in Section 5. In Section 6 experimental results are given and, finally, Section 7 brings concluding remarks.

2. EQUIVALENT PARENTHESIS FORMS

Let us represent a digital circuit by an EPF synthesized by a superposition procedure directly from the gate-level description of a circuit. For synthesizing the EPF of a given circuit, numbers are assigned to the gates and letters to the nets. Then, starting at an output and working back toward the primary inputs, EPF replaces individual literals by products or sums of literals.

When an AND gate is encountered during backtracing, a product term is created in which the literals are the names of nets connected to the inputs of the AND gate. Encountering an OR gate causes a sum of literals to be formed, while encountering an inverter causes a literal to be complemented.

As an example, the procedure is illustrated by transforming the circuit in Fig. 1 to its EPF:

$$y = a_1 b_1 = (c_{12} + d_{12})(m_{13} + e_{13}) = (g_{124}h_{124} + f_{125}k_{125}) \wedge$$

$$\wedge (m_{13} + \neg k_{136}) = (g_{124}h_{124} + \neg h_{1257}k_{125})(m_{13} + \neg k_{136}).$$

When creating an equation by the superposition procedure described above, the identity of every signal path from the inputs to the outputs of the given circuit will be retained. Each literal in an EPF consists of a subscripted input variable or its complement, which identifies a path from the variable to the output. From the manner in which the EPF is constructed, it can be seen that there will be at least one subscripted literal for every path from each input variable to the output. It is also easy to see that the complemented literals correspond to paths, which contain an odd number of inversions.

3. EQUIVALENT PARENTHESIS FORMS AND TIMING SIMULATION

Let us have an EPF $y = P(x_1, x_2, ..., x_i, ..., x_n)$ where $x_i \in X$ are literals (inverted or not), which describe the behaviour of a digital circuit. Denote by $L(x_i) = (g_{i1}, g_{i2}, ..., g_{in})$ the signal path through the gates $g_{i1}, g_{i2}, ..., g_{in}$ from the output y up to the input x_i. Denote the delay of the gate g_{ij} by $d(g_{ij})$. For simplicity, here we use the same delay for all the gate inputs for both raising and falling transitions. However, this does not affect the generality of the approach.

Let us call $\partial y/\partial x_i$ *partial Boolean derivative*. The theory of Boolean differential calculus tells that if $\partial y/\partial x_i = 1$, then a transition of the signal at input x_i leads to a transition of the signal at output y. To take into consideration the timing aspect, we introduce a function $\partial y(t_y)/\partial x_1(t_x)$, where $\partial y(t_y)/\partial x_i(t_x) = 1$ means that the transition of x_i at the moment t_x causes the transition of y at the moment t_y.

Theorem 1. Given a single transition at the moment t_x on the input x_i with a single output of a circuit represented by EPF $y = P(x_1, x_2, ..., x_i, ..., x_n)$ with a single path $L(x_i) = (g_{i1}, g_{i2}, ..., g_{in})$ from x_i to y, the transition propagates up to y with the delay

$$d(x_i \to y) = d(g_{i1}) + d(g_{i2}) + ... + d(g_{in}), \qquad (1)$$

iff $\partial y(t_y)/\partial x_i(t_x) = 1$, where $t_y = t_x + d(x_i \to y)$.

Proof. Along the definition of partial Boolean derivatives, from $\partial y/\partial x_i = 1$ (here and afterwards t_y and t_x for y and x are dropped for better readability) it follows that the value of y depends on the value of x_i, hence the transition at x_i propagates up to y. Since the path $L(x_i) = (g_{i1}, g_{i2}, ..., g_{in})$, along which the transition propagates, is not a branch, and it also has no fanouts, no other reconverging paths can exist along which the same transition at x_i could influence the value of y. Hence, the delay of the transition at y may be produced only by the sum of the delays of the gates along the path $L(x_i)$, and the relationship (1) is valid.

In the general case, if transitions occur on several inputs, or a transition propagates along several reconverging paths, then the derivative $\partial y/\partial x_i$ may depend on the influence of other transitions which may result in a glitch at y. In other words, the value of the function $\partial y/\partial x_i = f(x_1, x_2, ..., x_{i-1}, ..., x_{i+1}, ..., x_n)$ depends in this case on the literals where values are undetermined (unknown), and the calculation of $\partial y/\partial x_i$ is impossible.

Now, let us introduce the set $S_5 = \{0, 1, \varepsilon, h, U\}$ for 5-valued simulation, where $\varepsilon(h)$ represents a waveform having a step-up transition from 0 to a final value of 1 (step-down transition from 1 to a final value of 0), and U represents undetermined or don't care waveform. These values $\varepsilon, h,$ and U are called *dynamic values.*

In Table 1 we give also the algebra introduced for the dynamic values $\{\varepsilon, h, U\}$ in [38]:

Let us have a network with EPF $y = f(x_1, x_2, ..., x_i, ..., x_n)$ and a multi-valued pattern $x^t = (x_1^t, x_2^t, ..., x_i^t, ..., x_n^t)$ at time t_x, where $x_i^t \in S_5$. Denote a subset of literals with dynamic values at t_x by $x_D = \{x_i | x_i^t \in \{\varepsilon, h, U\}\}$.

Table 1. Calculation of dynamic values

$\vee$	ε	h	U	$\wedge$	ε	h	U
ε	ε	U	U	ε	ε	U	U
h	U	h	U	h	U	h	U
U	U	U	U	U	U	U	U

Definition 1. We say $\max\{\partial y/\partial x_i\}=1$ iff there is at least one combination of values 0 or 1 for nonspecified x's which produce $\partial y/\partial x_i = 1$. Otherwise, $\max\{\partial y/\partial x_i\}=0$.

Lemma 1. The value of EPF $y = P(x_1, x_2, \ldots x_i, \ldots x_n)$ for a given network in the multi-valued alphabet S_5 is:

$$y = \wedge x_i = \vee x_i, \quad x_i \in x_D \cap \{x_i|\max\{\partial y/\partial x_i\}=1\}, \tag{2}$$

iff $x_D \cap \{x_i|\max\{\partial y/\partial x_i\}=1\} \neq \varnothing.$

Proof. If $\max\{\partial y/\partial x_i\}=1$ is valid for a single $x_i \in x_D$, then according to the definition of Boolean derivatives, $y = x_i$. In this case the same value of x_i occurs on the output (or inverted value if x_i is inverted). Suppose now that there are more than one literal $x_i \in x_D$ satisfying the condition $\max\{\partial y/\partial x_i\}=1$. In other words, there are more than one converging path in the network which propagate transitions towards the output. If two paths are converging, either AND or OR of multiple values from $\{\varepsilon, h, U\}$ is possible. From the equivalence of operations AND and OR on the set $\{\varepsilon, h, U\}$, it follows that the value of y can be calculated as a function of AND (or OR) of values $x_i \in x_D \cap \{x_i|\max\{\partial y/\partial x_i\}=1\}$.

Consider, for example, a transition pattern $g = k = m = 1$, $h = \varepsilon$ at the input of the circuit in Fig. 1. By calculating Boolean derivatives, we find: $\partial y/\partial h_{124} = h_{1257}$, and $\partial y/\partial h_{1257} = \neg h_{124}$. Since h_{124} and h_{1257} have dynamic values $h_{124} = h_{1257} = h = \varepsilon$, the calculation of the Boolean derivative is impossible. On the other hand, since $\max\{\partial y/\partial h_{124}\} = \max\{\partial y/\partial h_{1257}\} = 1$, and since $x_D \cap \{x_i|\max\{\partial y/\partial x_i\}=1\} = \{h_{124}, \neg h_{1257}\}$, we have $y = h_{124} \wedge \neg h_{1257} = \varepsilon \wedge \neg\varepsilon = U$. The value U on the output of the subcircuit in Fig. 1 means the possibility of a glitch at the given transition pattern.

Theorem 2. Given $|x_D| > 1$ at input pattern $x^t = (x_1^t, x_2^t, \ldots, x_i^t, \ldots, x_n^t)$ where $x_i^t \in S_5$, and a subset $x_D^* \in x_D$ where

$$\forall x_i \in x_D^* : (\max\{\partial y/\partial x_i\}=1) \,\&\, (d(x_i \to y) = \Delta_i), \tag{3}$$

274

there appears a transition on the output of a circuit $y = P(x_1, x_2, ..., x_i, ..., x_n)$ with the value

$$y = \wedge\, x_i, \tag{4}$$

at time $t_x + \Delta_i$ *where* Δ_i *is calculated by formula* (1).

Proof. Suppose there exist at least two inputs $x_i^t, x_j^t \in x_D^*$ with corresponding paths $L(x_i) = (g_{i1}, g_{i2}, ..., g_{in})$ and $L(x_j) = (g_{j1}, g_{j2}, ..., g_{jm})$ through the circuit. Suppose they have a joint path $L(g_{i,k}) = (g_{i1}, g_{i2}, ..., g_{i,k-1})$ starting from the output of a gate $g_{ik} \equiv g_{jk}$, $k > 0$, with the transition delay $\tau = d(g_{i1}) + d(g_{i2}) + ... + d(g_{i,k-1})$. From (3) it follows that the transitions evoked at the inputs x_i^t, x_j^t reach the inputs of the gate g_{ik} at the same moment $t_{k+1} = t_x + (\Delta_i - \tau - d(g_{ik}))$. On the other hand, from the condition $x_i^t, x_j^t \in x_D \cap \{x_i \,|\, \max\{\partial y / \partial x_i\} = 1\}$ and Lemma 1, it follows that the value of the signal at time $t_k = t_x + \Delta_i - \tau$ on the output of the gate g_{ik} belongs to the set $\{\varepsilon, h, U\}$, which means a transition (where U is a possible glitch). Since the path $L(g_{i,k})$ is also activated due to (3), the transition propagates to the output and shows itself at time $t_{k+1} + d(g_{ik}) + \tau = t_x + (\Delta_i - \tau - d(g_{ik})) + d(g_{ik}) + \tau = t_x + \Delta_i$.

Corollary. From Theorems 1 and 2 the following algorithm can be derived for timing simulation based on calculating Boolean derivatives of equivalent parenthesis forms.

Algorithm 1.

1. Calculate $\partial y / \partial x_i$ for $x_i \in x_D$ for the given transition x^t.
2. Take the lowest value of $\Delta_i = d(x_i \to y)$. If $\partial y / \partial x_i = 1$, fix the new value of y for time $t_x + \Delta_i$. Use formula (2) to check if a glitch is present. Remove x_i from x_D.
3. If $x_D = \varnothing$, stop, else repeat step 2.

4. TIMING SIMULATION ON SSBDDs

A structurally synthesized BDD $G_y = (M, \Gamma, X)$ with a set of nodes M and a mapping Γ from M to M is a BDD which represents an equivalent parenthesis form $y = P(x)$ of a gate-level network. The set of nodes consists of a subset of nonterminal nodes M^N and of a subset of terminal nodes M^T; $M = M^N \cup M^T$. There are one initial node $m_0 \in M^N$ and two terminal nodes $m^{T,e} \in M^T$, $e \in \{0,1\}$, in M. A one-to-one correspondence exists between nonterminal nodes $m \in M^N$ and the literals $x_i \in X$. The nodes $m \in M^N$ are

labelled by subscripted input variables (or the inverted variables) which identify a path from the input to the output of the network. The terminal nodes $m^{T,e} \in M^T$ are labelled by constants $e \in \{0,1\}$. The literal $x_i \in X$, which is associated with the node m, is denoted by $x(m)$. The mapping Γ defines the set of edges between the nodes of M whereas $\Gamma(m) \subset M$ is a set of successors of m, and $m^e \in \Gamma(m)$ is the successor of m for the value $x(m) = e$. A pattern x^t defines a set of activated edges in G_y. The edge between m and m^e is activated when $x(m) = e$ in the pattern x^t. Activated edges which connect nodes m_i and m_j make up an activated path $l(m_i, m_j)$. The path $l(m_i, m_j)$ consists of nodes $M(m_i, m_j) \subseteq M$. An activated path $l(m_0, m^{T,e})$ is called a full activated path.

Definition 2. A SSBDD $G_y = (M, \Gamma, X)$ represents an equivalent parenthesis form $y = P(X)$ of a gate-level network, iff for each pattern x^t a full path $l(m_0, m^{T,e})$ in G_y will be activated where $y = e$.

Two-valued test pattern simulation on SSBDDs is equivalent to path tracing procedure on graphs according to the values of variables at a given test pattern. At a given pattern x^t, in a SSBDD G_y, a full path $l(m_0, m^T)$ will be activated which determines the value of $y = x(m^T)$. The simulation procedure will consist of tracing the path $l(m_0, m^T)$ and finding the value of $x(m^T)$ at the terminal node m^T.

For multi-valued simulation, a procedure based on calculation of Boolean derivatives on SSBDDs will now be described. Denote $l(m_i, m_j) = 1$, if there exists an activated path between the nodes m_i and m_j at the given pattern x^t, otherwise $l(m_i, m_j) = 0$.

Theorem 3. Given $y = P(x)$ and $x_i \in X$, the condition $\partial y / \partial x_i = 1$ for SSBDD $G_y = (M, \Gamma, X)$, where $x(m) \equiv x_i$, is equivalent to the following equation:

$$l(m_0, m) \wedge l(m^1, m^{T,1}) \wedge l(m^0, m^{T,0}) = 1. \qquad (5)$$

The proof of the Theorem 3 can be found in [38].

Note that Theorem 3 can be used for calculating Boolean derivatives $\partial y / \partial x_i$ only in the case when pattern x^t is two-valued, because only in this case all the paths $l(m_i, m_j)$ are activated uniquely. In the general case, when x^t is a multi-valued pattern, to check the existence of a glitch, we have to generalize Eq. (5). The generalized case, based on maximums of Boolean derivatives, is considered in [38].

Using SSBDDs, it is possible to considerably speed up the calculations described in Algorithm 1, because it is not needed to trace all paths in Eq. (5) for each $x_i \in x_D$.
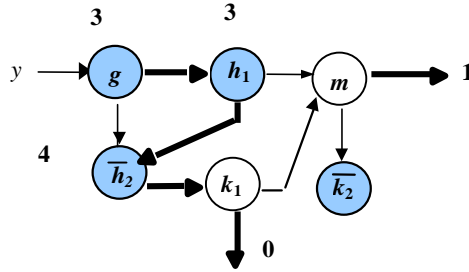
Fig. 2. SSBDD for the circuit in Fig. 1.

Table 2. Signal paths and delays of the example

Node	Path	Delay	Pattern
g	g, 4, 2, 1, y	3	h (10)
h_1	h, 4, 2, 1, y	3	ε (01)
$\neg h_2$	h, 7, 5, 2, 1, y	4	h (10)
k_1	k, 5, 2, 1, y	3	0
m	h, 3, 1, y	2	1
$\neg k_2$	k, 6, 3, 1, y	3	ε (01)

Example. An example of SSBDD for the circuit in Fig. 1 is presented in Fig. 2. The nodes of the graph, the corresponding paths in the circuit, and the path delays, calculated by Eq. (1), are depicted in Table 2 (here we assume that all the gates have a unit delay).

Consider a transition pattern given in Table 2. The bold arrows in Fig. 2 mark the activated path in the graph before the transition. The shaded nodes are those involved in the transition, i.e. where the direction of the activated path changes. For the nodes g and h_1 we have $\max\{\partial y/\partial g\} = \max\{\partial y/\partial h_1\} = 1$ [37]. Using the formula (2), we find that $g \wedge h_1 = h \wedge \varepsilon = U$ which means that at time $t = 3$ we may have a glitch on the output of the circuit.

5. THE TIMING SIMULATION ALGORITHMS

Using the SSBDD model gives us the possibility to minimize the number of macro inputs to be processed as well as the possibility to use some SSBDD features in order to increase the timing simulation efficiency.

In this section we describe several implementations of the Algorithm 1 on the SSBDD model. First, the general algorithm is given. Then we describe the single and double stack based approaches.

Given is a set of multivalued input patterns x^t at the input of a macro SSBDD $G_y = (M, \Gamma, X)$, and a set of delays $\Delta = \{d^e(m_i) | m_i \in M, e \in \{\varepsilon, h\}\}$. Certain values for both raise $d^\varepsilon(m_i)$ and fall $d^h(m_i)$ delays are specified for each node. We denote a variable in the node m_i as $x(m_i)$. The output of the algorithm is a single waveform for the output of each macro. The waveforms show all the transitions taking place there.

The general idea is as follows. Let the current time moment be t_x and the current pattern applied x^t. We are traversing the activated (before the transition) path $l(m_0, m^T)$ in the graph from the initial node m_0 to one of the terminal nodes m^T and checking if $x(m_i) \in x_D$ in order to find the node with transition that has minimum delay $d_{\min}$. The transition in this node is the first transition that may influence the macro's output. It will happen at the moment $t_y = t_x + d_{\min}$ iff $\max\{\partial y / \partial(x(m_i))\} = 1$. When the node is found, the current time t_x is changed to $t_x + d_{\min}$.

Our task now is to find the next $d_{\min}$. We go back to the initial node and traverse the path from the beginning taking into account that one of the values has already been changed. However, as we are probably traversing a new path, we can find a node with delay that is smaller or equal to the previous $d_{\min}$. This means that the transition in that node has also taken place and it is not interesting any more. In general, we are not interested in all delays $d^e(m_i) < t_x$. Suppose, we are in node m_i, $x(m_i) \in x_D$, somewhere in the middle of the path. The delay here is $d^e(m_i)$ and somewhere before (along the path $l(m_0, m_i)$) we have already found the next minimum delay $d_{\min}$. Then we will update the $d_{\min}$ with $d^e(m_i)$ iff $t_x < d^e(m_i) < d_{\min}$.

After we have reached a terminal node again, we check if it is different from the previously reached terminal node. If it is, we put the new transition to the output waveform labelling it with the current time moment. We continue the graph traversal procedure until no $d_{\min} > t_x$ is found. This means that all the transitions (which have influence on the macro output) in the macro have already taken place and the next vector should be taken. When all the vectors have been simulated for the given macro, a new macro is taken. The whole process stops when the whole circuit has been finished.

The above was the description of the general SSBDD-based timing simulation algorithm, which uses no stack. Note that in some cases we do not need to check all the nodes in the graph because those nodes will never lie on an activated path. To make the procedure even more efficient, we use a stack to store every encountered node along the path, with the delay which was taken as $d_{\min}$. Using the stack, we have no need to begin path traversal from the initial node every time. We can return to the last node m_s taken from the stack

and take the $d^e(m_s)$ as the next $d_{\min}$ and update it further as we start moving forward.

In the following, we give the description of a single-stack-based algorithm step by step.

Algorithm 2.

1. Initialization: $t = 0$, $d_{\min} = 0$, $i = 0$, $ptr = 0$, $stack(ptr).node = 0$, $stack(ptr).time = 0$, macro output is undefined.

2. If $t < d^e(m_i)$ go to 3. Otherwise take i as the index of m^0 if $x(m_i) = \{h, 0\}$ or as the index of m^1 if $x(m_i) = \{\varepsilon, 1\}$, go to 7.

3. If $x(m_i) \in x_D$ go to 4. Otherwise take i as the index of m^0 if $x(m_i) = \{\varepsilon, 0\}$ or as the index of m^1 if $x(m_i) = \{h, 1\}$, go to 7.

4. If $ptr = 0$ or $d^e(m_i) < stack(ptr).time$, go to 5. If not, go to 6.

5. $ptr = ptr + 1$, $stack(ptr).time = d^e(m_i)$, $stack(ptr).node = i$.

6. Take i as the index of m^0 if $x(m_i) = \varepsilon$ or as the index of m^1 if $x(m_i) = h$.

7. If m_i is not one of the terminal nodes, go to 2. If not, go to 8.

8. If macro output is different from the value of the terminal node we have come to, update macro output with the new transition and label it with time t.

9. If $ptr = 0$ stop, otherwise go to 10.

10. $t = stack(ptr).time$, $i = stack(ptr).node$, $ptr = ptr - 1$, $d_{\min} = stack(ptr).time$, go to 2.

Example. In Fig. 3 an example to illustrate the algorithm is given for the SSBDD in Fig. 2. The input pattern and the delays are the same as in Table 2. We start from the node g and go to the node h_1. As the stack was empty and g had a transition at the given moment of time, we put g and its delay into the stack. The node h_1 has a transition but the delay in it is not smaller than that in g. So we continue moving forward without updating the stack. The nodes h_2 and k_1 have no transitions this time. We just pass them by. Finally, we reach the terminal node $m^{T,0}$. Thus the initial value at the output y will be 0.
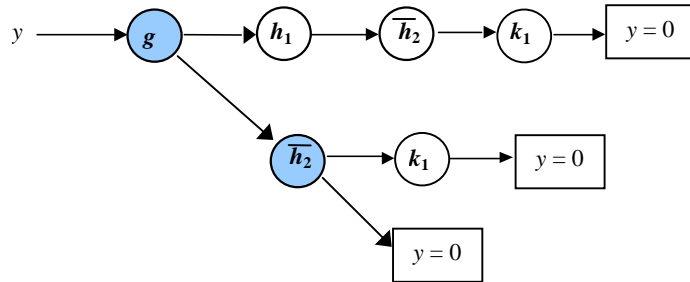

Fig. 3. Single-stack-based timing simulation for the SSBDD in Fig. 2.

We get back to the node taken from the stack (it is g) and go to another direction (the value in g has been changed). The current moment of time is 3 now. The node h_2 has a transition and the transition time is greater than the current moment of time. As the stack is empty again, we put h_2 and the delay into the stack and move forward. Finally, again we reach the same terminal node. Thus the output is stable. Again we get back to the node h_2 taken from the stack and reach the same terminal node, what means no change of the value on the output. Since the stack is empty now, the calculation terminates.

Note that despite the node k_2 has a transition, we did not examine this macro input at all. That is, we have to check all of the macro inputs and calculate derivatives for all $x(m_i) \in x_D$ only in the worst case.

In Algorithm 2 and the example above we use a stack to return each time not to the very beginning of the graph but exactly to the node with the next transition. However, not every transition along the activated path can influence the output of the macro. In the following we give an idea how to improve the Algorithm 2 by using this feature.

Given an input pattern that activates a full path $l(m_0, m^{T,e})$, which consists of the nodes $M(m_0, m^{T,e})$. We designate $M^e(m_0, m^{T,e}) = \{m | m \in M^N, m \in M(m_0, m^{T,e}), x(m) = e, e \in \{0,1\}\}$ the set of all nonterminal nodes along the path which hold the value e. Similarly, the set of all the nodes along the path which hold the value $\neg e$ are designated $M^{\neg e}(m_0, m^{T,e}) = \{m | m \in M^N, m \in M(m_0, m^{T,e}), x(m) = \neg e, e \in \{0,1\}\}$. In other words, we divide all the nodes along the activated path into two subsets. The first one $M^0(m_0, m^{T,e})$ contains all the nodes which hold the current value 0 and the other one $M^1(m_0, m^{T,e})$ contains all the nodes which hold the value 1. Terminal node $m^{T,e}$ does not belong to any of the two subsets. If the currently reached terminal node is $m^{T,0}$ then it is known that transitions in all the nodes $m \in M^1(m_0, m^{T,0})$ do not affect the output value (taking a new path, we will still reach the node $m^{T,0}$), and vise versa, for the node $m^{T,1}$ no transitions in nodes $m \in M^0(m_0, m^{T,1})$ can affect the output.

The above statement shows clearly that, standing in the terminal node $m^{T,e}$, we should consider only the nodes $m \in M^e(m_0, m^{T,e})$ as the potential sources of influence on the macro output. Therefore, we introduce a minor change to the Algorithm 2 using two different stacks for the nodes of $M^0(m_0, m^{T,e})$ and $M^1(m_0, m^{T,e})$. Standing each time at the terminal node, we check only the dedicated stack for the next transition to simulate it. If there are some transitions in another stack, they will be left not simulated because they cannot affect the macro output. That is, we have to simulate all the nodes with transitions on the current active path only in the worst case.

However, certain operations and comparison of data between two stacks should be added to make the algorithm work well. This generates some overhead and in the worst case the double-stack-based algorithm may work slower than the

single-stack-based one. This gave us an idea to try to use the two-stack approach only for finding the next moment of time but starting the traversal procedure from the initial node m_0. This helps us to avoid the time-consuming procedure of stack update. This means that we can win the time needed for stack update, but we lose the time needed for the path traversal from the beginning.

For different circuits all the three algorithms should give different results. It is logical to suppose that the simpler algorithms should work faster for smaller macros, but for bigger ones sophisticated stack-based algorithms can give better results. In the next section we will illustrate this statement by experimental data, but now let us give an example to illustrate the two-stacks-based algorithm.

Example. Consider the same SSBDD, the same input pattern, and the same delays as in the last example. Similarly, we begin with the node g and traverse the activated path until the end, but, differently from the single stack case, we store the node g in one stack and the node h_1 in another. We do not put nodes h_2 and k_1 into the stacks similarly to the previous example. Finally we reach the terminal node $m^{T,0}$. Thus the initial value at the output y is 0.

In this case, only nodes with transitions 0 to 1 can affect the macro output. Thus we have to check the corresponding stack. We find the node h_1 in this stack and go back to this node. However, at this point we cannot continue the graph traversal before we have checked another stack to see if it has a node which stands closer to the initial node and has a delay smaller than or equal to the delay in the node h_1. If there is such a node in another stack we have to go further to this node. This is the point where the overhead of the processing of stacks is added.

In another stack we find node g with the delay equal to the delay in h_1, so we move further to node g and start the traversal of newly activated path from that point. Both stacks are empty again. As the node h_2 has a transition and the delay is greater than the current moment of time, we put it into one of the stacks. Node k_1 does not have a transition, so we pass it by and come to the same terminal node (again $y = 0$).
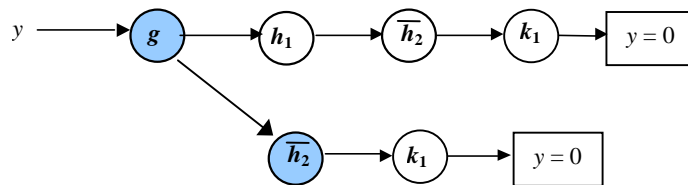

Fig. 4. Double-stack-based timing simulation for the SSBDD in Fig. 2.

We look at the stack, which corresponds to the situation where $y = 0$, and find it to be empty. This means that the simulation is over. In Fig. 4 an illustration of the algorithm is given. Compared to the single-stack-based algorithm (Fig. 3) it has one step less.

6. EXPERIMENTAL DATA

Experiments were carried out using two different types of benchmarks. The ISCAS'85 circuits were chosen since they are widely adopted benchmarks. However, the efficiency of simulation is highly dependent on the number of levels and on the number of gates in tree-like *subcircuits* (i.e., on the size of a macro) represented by graphs. Therefore we have also used 5 tree-like circuits with numbers of levels from 2 to 10 (numbers of gates from 3 to 1023). And we used two different input pattern generation modes: with single (S) or multiple (M) bit transitions allowed on inputs at the same time. In order to evaluate our approach, we used a gate-level event-driven timing simulation algorithm as a reference. Experimental results presented below clearly show a noticeable speed-up of this approach.

The results for tree-like circuits are illustrated for the case of single transitions in Fig. 5 and for the case of multiple transitions in Fig. 6. Simulation time is given in seconds for 30 000 random patterns. In our work we measured simulation time on both macro and gate levels. Note that the simulation time grows exponentially with the number of levels at the gate level (Fig. 5). The simulation time at the macro level (our approach) grows much slower (Fig. 5), but also not linearly. In Fig. 5 (right) all the four macro-level algorithms are shown. However, there is no noticeable difference in simulation time between them.
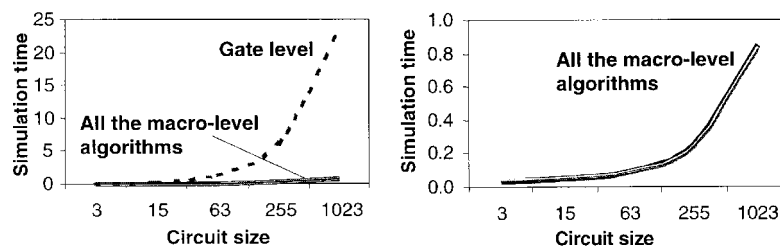

Fig. 5. Comparison of simulation times of different algorithms (single-bit transition mode).

For the multiple-bit change input pattern generation mode there is a noticeable difference in speed for the macro-level timing simulation algorithms (Fig. 6). The double-stack-based algorithm (lower thin line) is the fastest for this case and the algorithm with no stack (upper bold line) is the slowest. The timing simulation at the gate level needs time (dashed line) that grows again much faster than the time our approach requires.

Experimental results on ISCAS'85 benchmarks are given in Table 3. The best simulation times and speed-ups are shown in bold.

Note that the double-stack-based and no-stack-based algorithms give the best results. However, there is no big difference between the four algorithms but there is still a great speed-up compared to the gate-level algorithm. The speed of simulation based on the proposed method increases up to 3.54 times for patterns with single transition and up to 2.91 times for patterns with multiple transitions. For the case of ISCAS'85 benchmarks, no relationship between the circuit size and the simulation speed-up can be noticed.

However, Fig. 7 presents an interesting observation that the speed gain of timing simulation is directly proportional to the average size of a macro. The results are shown for the case of ISCAS'85 benchmarks and the double-stack
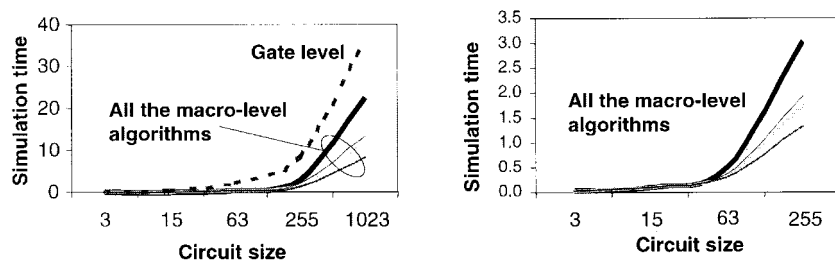

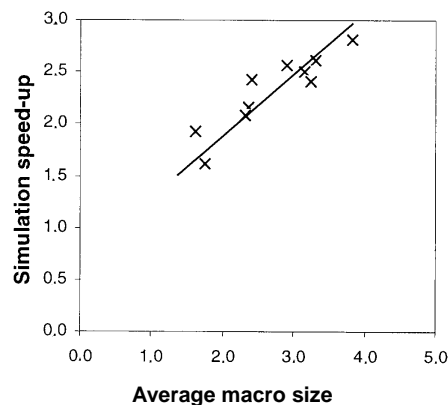**Fig. 6.** Comparison of simulation times of different algorithms (multiple-bit transition mode).

Fig. 7. Simulation speed-up compared to the average size (number of gates) of a macro.

283

Table 3. Experimental results for the ISCAS'85 circuits

Type of the algorithm	Name of the circuit and number of gates																			
	c432 232		c499 618		c880 357		c1355 514		c1908 718		c2670 997		c3540 1446		c5315 1994		c6288 2416		c7552 2978	
	Pattern generation mode																			
	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M	S	M
Simulation time for 10 000 patterns, s																				
No stack	**0.77**	1.49	**1.71**	**3.38**	1.34	2.52	**2.81**	**5.17**	**2.32**	**4.49**	**3.80**	7.85	**3.63**	8.69	**6.18**	15.2	**30.8**	139	**8.88**	24.4
Single-stack	0.86	1.47	1.93	3.73	1.53	2.56	3.41	5.76	2.69	4.71	4.64	8.21	4.26	8.71	7.36	15.4	32.3	**133**	10.7	24.5
Double-stack	0.80	**1.37**	1.79	3.77	**1.33**	**2.40**	2.94	5.20	2.36	4.50	4.02	**7.66**	3.78	**8.42**	6.55	**15.1**	31.6	141	9.22	**23.7**
Double-stack, no update	0.79	1.46	1.80	3.53	1.38	2.58	2.94	5.44	2.34	4.59	4.01	8.00	3.78	8.81	6.43	15.6	33.0	152	9.26	24.7
Gate level	**2.16**	**3.32**	**5.30**	**9.83**	**3.26**	**5.19**	**4.86**	**8.38**	**6.98**	**11.5**	**9.24**	**15.9**	**12.9**	**23.6**	**20.1**	**37.7**	**58.7**	**272**	**28.0**	**57.1**
G/M ratio																				
No stack	**2.81**	2.23	**3.10**	2.91	2.43	2.06	**1.73**	**1.62**	**3.01**	**2.57**	**2.43**	2.03	3.54	2.72	**3.26**	2.47	**1.90**	1.95	**3.15**	2.34
Single-stack	2.51	2.26	2.75	2.64	2.13	2.03	1.43	1.45	2.59	2.45	1.99	1.94	3.02	2.71	2.73	2.44	1.82	**2.04**	2.60	2.32
Double-stack	2.70	**2.42**	2.96	2.61	**2.45**	**2.16**	1.65	1.61	2.96	2.56	2.30	**2.08**	3.40	**2.81**	3.07	**2.50**	1.86	1.92	3.03	**2.41**
Double-stack, no update	2.73	2.27	2.94	2.78	2.36	2.01	1.65	1.54	2.98	2.51	2.30	1.99	3.40	2.68	3.13	2.42	1.78	1.78	3.02	2.31

simulation algorithm. The observed relationship gives us a new criterion for arranging hardly arranged before ISCAS'85 benchmarks.

For all the experiments we used a Sun Ultra 10 workstation with 440 MHz UltraSparc – IIi processor, 256 MB RAM, and SunOS 5.7.

7. CONCLUSIONS

A new approach to speed up gate-level timing simulation is proposed where, instead of gate delays, path delays for tree-like subcircuits (macros) represented by SSBDDs are used. SSBDDs capture the structure of a circuit whereas conventional BDDs do not allow that. At the same time, using SSBDDs for representing macros avoids exponential explosion of the model complexity. The number of paths in the circuit, processed by delay calculation, is a linear function of the number of gates.

Experiments were carried out on the ISCAS'85 benchmarks with the number of gates up to about 3000. The linear feature of the model complexity enables efficient simulation of complex realistic combinational circuits.

Four algorithms for this approach were implemented and their efficiencies compared. The timing simulation speed at the macro-level is up to 3.54 (2.43 on average) times faster compared to the gate-level simulation for the investigated set of ISCAS'85 benchmark circuits. The best among the macro-level algorithms is the double-stack-based one.

The high speed of simulation is achieved at the cost of some loss of simulation data. Instead of all waveforms for all nodes of the gate-level network, only the waveforms for the outputs of macros are calculated. This simplification is nevertheless acceptable for most industrial applications of timing simulation.

ACKNOWLEDGEMENTS

REFERENCES

1. Cheng, K. T. and Agrawal, V. D. *Unified Methods for VLSI Simulation and Test Generation*. Kluwer, Boston, 1989.
2. Abramovici, M., Breuer, M. A., and Friedman, A. D. *Digital Systems Testing and Testable Design*. IEEE Press, New York, 1999.

3. McWilliams, T. M. Verification of timing constraints on large digital systems. In *Proc. 17th ACM/IEEE Design Automation Conference*. Minneapolis, 1980, 139–147.

4. Bowden, K. Design goals and implementation techniques for time-based digital simulation and hazard detection. In *Proc. International Test Conference*. Philadelphia, 1982, 147–152.

5. Ishiura, N., Takahashi, M., and Yajima, S. Time symbolic simulation for accurate timing verification of asynchronous behavior of logic circuits. In *Proc. 26th ACM/IEEE Design Automation Conference*. Las Vegas, 1989, 497–502.

6. Doukas, D. and LaPaugh, A. S. Clover: A timing constraints verification system. In *Proc. 28th ACM/IEEE Design Automation Conference.* San Francisco, 1991, 662–667.

7. Martello, A., Levitan, S., and Chiarulli, D. Timing verification using hdtv. In *Proc. 27th ACM/IEEE Design Automation Conference*. Orlando, 1990, 118–173.

8. Linderman, M. H. *Simulation of Digital Circuits in the Presence of Uncertainty.* PhD thesis, Cornell University, Ithaca, 1994.

9. Devadas, S., Keutzer, K., Malik, S., and Wang, A. Verification of asynchronous interface circuits with bounded wire delays. *IEEE J. VLSI Signal Process.*, 1994, **7**, 161–182.

10. Maurer, P. and Lee, Y. S. Gateways: A technique for adding event-driven behavior to compiled simulations. *IEEE Trans. Comput.-Aided Des.*, 1994, **13**, 338–352.

11. Lewis, D. A hierarchical compiled code event-driven logic simulator. *IEEE Trans. Comput.-Aided Des.*, 1991, **10**, 726–737.

12. Wang, Z. and Maurer, P. LECSIM: A levelized event-driven compiled logic simulator. In *Proc. 27th ACM/IEEE Design Automation Conference*. Orlando, 1990, 491–496.

13. Smith, S., Mercer, M. & Brock, B. Demand driven simulation: BACKSIM. In *Proc. 24th ACM/IEEE Design Automation Conference.* Miami Beach, 1987, 181–187.

14. Devadas, S et al. Event suppression: Improving the efficiency of timing simulation for synchronous digital circuits. *IEEE Trans. Comput.-Aided Des.,* 1994, **13**, 814–822.

15. Ulrich, E. A design verification methodology based on concurrent simulation and clock suppression. *Proc. 20th ACM/IEEE Design Automation Conference.* Miami Beach, 1983, 709–712.

16. Takamine, Y. et al. Clock event suppression algorithm of VELVET and its application to S-820 development. In *Proc. 25th ACM/IEEE Design Automation Conference.* Anaheim, 1988, 716–719.

17. Weber, T. and Somenzi, F. Periodic signal suppression in a concurrent fault simulator. In *Proc. European Conference on Design Automation*. Amsterdam, 1991, 565–569.

18. Gai, S. and Montessoro, P. Creator: New advanced concepts in concurrent simulation. *IEEE Trans. Comput.-Aided Des.*, 1994, **13**, 786–795.

19. Razdan, R., Bischoff, G., and Ulrich, E. Clock suppression techniques for synchronous circuits. *IEEE Trans. Comput.-Aided Des.*, 1993, **12**, 1547–1556.

20. Barzilai, Z. et al. HSS – a high-speed simulator. *IEEE Trans. Comput.-Aided Des.*, 1987, **6**, 601–617.

21. Chiang, M. and Palkovic, R. LCC simulators speed development of synchronous hardware. *Comput. Des*., 1986, **25**, 87–92.

22. Wang, L.-T. et al. SSIM: A software levelized compiled-code simulator. In *Proc. 24th ACM/IEEE Design Automation Conference*. Miami Beach, 1987, 2–8.

23. Khoo, K.-Y. and Willson, A. N. Cycle-based timing simulations using event-streams. In *Proc. International Conference on Computer Design*. Austin, 1996, 460–465.

24. Kumar, D., Kohli, A., and Narayanswamy, V. Faster simulation of timed Petri nets via distributed simulation. In *Proc. 21st International Computer Software and Applications Conference*. Washington, D.C., 1997, 149–152.

25. Zuberek, W. M. Event-driven simulation of timed Petri net models. In *Proc. 33rd Annual Simulation Symposium.* Washington, D.C., 2000, 91–98.

26. Benkoski, J. *Statistical Timing Verification and Delay Fault Detection by Formal Signal Interaction Modeling in a Multi-level Timing Simulator.* PhD thesis, Carnegie Mellon University, Pittsburgh, 1989.

27. Ghosh, S. Dynamic multi-level simulation of digital hardware design. *Simulation*, 1987, **48**, 247–252.
28. Chew, M. P. and Strojwas, A. J. Utilizing logic information in multi-level timing simulation. In *Proc. 28th ACM/IEEE Design Automation Conference.* San Francisco, 1991, 215–218.
29. Eichelberger, E. B. Hazard detection in combinational and sequential switching circuits. *IBM J. Res. Dev.*, 1965, **9**, 90–99.
30. Breuer, M. A. and Harrison, L. Procedures for eliminating static and dynamic hazards in test generation. *IEEE Trans. Comput.*, 1974, **23**, 1069–1078.
31. Fantauzzi, G. An algebraic model for the analysis of logical circuits. *IEEE Trans. Comput.*, 1974, **23**, 576–581.
32. Kung, D. S. Hazard-non-increasing gate-level optimization algorithms. In *Proc. IEEE/ACM International Conference on Computer Aided Design.* Santa Clara, 1992, 631–634.
33. Chakraborty, T. J., Agrawal, V. D., and Bushnell, M. L. Delay fault models and test generation for random logic sequential circuits. In *Proc. 29th ACM/IEEE Design Automation Conference.* Anaheim, 1992, 165–172.
34. Chakraborty, S. and Dill, D. L. More accurate polynomial-time min-max timing simulation. In *Proc. 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems.* Eindhoven, 1997, 112–123.
35. Ubar, R. Test generation for digital circuits using alternative graphs. *Proc. Tallinn Polytechnical Institute.* 1976, No. 409, 75–81 (in Russian).
36. Ubar, R. Beschreibung digitaler Einrichtungen mit alternativen Graphen für die Fehlerdiagnose. *Nachr. tech. Elektron.*, 1980, **30**, 96–102.
37. Ubar, R. Test synthesis with alternative graphs. *IEEE Des. Test Comput.*, 1996, **13**, 48–59.
38. Ubar, R. Multi-valued simulation of digital circuits with structurally synthesized binary decision diagrams. In *Multiple Valued Logic.* Gordon and Breach Publishers, 1998, **4**, 141–157.
39. Andrew, R. An algorithm for 8-valued simulation and hazard detection in gate networks. In *16th International Symposium on Multiple Valued Logic.* Blacksburg, 1986, 273–280.
40. Mao, W. and Ciletti, M. D. A variable observation method for testing delay faults. In *Proc. 27th ACM/IEEE Design Automation Conference.* Orlando, 1990, 728–731.
41. Si, S. Dynamic testing of redundant logic networks. *IEEE Trans. Comput.*, 1978, **27**, 828–832.
42. Jervan, G., Markus, A., Raik, J., and Ubar, R. Automatic test generation system for VLSI. In *Proc. 1st Electronic Circuits and Systems Conference.* Bratislava, 1997, 255–258.
43. Ubar, R. Parallel critical path tracing fault simulation. In *Proc. 39. Int. Wiss. Kolloquium.* Ilmenau, 1994, **1**, 399–404.
44. Raik, J. and Ubar, R. Feasibility of structurally synthesized BDD models for test generation. In *Proc. European Test Workshop.* Barcelona, 1998, 145–146.
45. Jutman, A. and Ubar, R. Design error diagnosis in digital circuits with stuck-at fault model. *J. Microelectron. Reliability*, 2000, **40**, 307–320.

STRUKTUURSETE BINAARSETE OTSUSTUSDIAGRAMMIDE RAKENDAMINE DIGITAALSKEEMIDE ASÜNKROONSEKS SIMULEERIMISEKS

Artur JUTMAN ja Raimund UBAR

Nüüdisaegsete digitaalskeemide üha kasvav töösagedus sunnib tähelepanu pöörama viidete modelleerimisele (asünkroonsele simuleerimisele). Laialt tuntud meetodeid skeemi käitumise kontrollimiseks ajas on asünkroonne simuleerimine.

Töös on esitatud uus ventiilitaseme asünkroonse simuleerimise kiirendamise meetod, mis baseerub struktuursetel binaarsetel otsustusdiagrammidel (SSBDD). SSBDD-d on juba leidnud rakendust digitaalskeemide esitamise efektiivse matemaatilise mudelina. Uus lähenemisviis kasutab ventiilide viidete asemel puukujuliste alamskeemide (makrode) teede viiteid. Seetõttu ei arvutata ajadiagramme mitte iga ventiili väljundis, vaid ainult makrode väljundites. Iga makro on esitatud SSBDD-ga, mis võimaldab kiiret viidete arvutamist. On näidatud, et kiiruse kasv asünkroonsel simuleerimisel on proportsionaalne skeemi makrode keskmise suurusega. Uut asünkroonse simuleerimise kiirendamise meetodit on kontrollitud eksperimentaalselt.