ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2008, Vol.37, No.4

THE USE OF A SOFTWARE PROTOTYPE FOR VERIFICATION TEST GENERATION

Eduardas Bareiša, Vacius Jusas, Kęstutis Motiejūnas, Rimantas Šeinauskas

Software Engineering Department, Kaunas University of Technology
Studentų St. 50-404, LT–51368 Kaunas, Lithuania

Abstract. The software prototype model can be used for the generation of the verification test. The input stimuli, which form essential activity vectors, are selected from randomly generated ones on the base of software prototype. The essential activity vectors correspond to the terms of logical functions of output the existence of which is tested during the verification. The verification test is formed on the base of the essential activity vectors according to the defined rules. The quality of the verification test is measured by the following parameters: the length of test, the fault coverage of the stuck-at faults, the fault coverage of the pin pair faults, and the number of the essential activity vectors. The paper presents the experimental results for the benchmark suite ISCAS'85. The value of this approach is highlighted by the fact that the selected input stimuli detect the same stuck-at faults as the initially generated test set.

1. Introduction

In the last few years the major challenge, which the semiconductor industry is confronted with, has been to design devices in significantly less time with far more complex logical functionality. At the very top of the list of challenges to be solved is verification. The goal of the verification is to ensure that the design meets the logical functional requirements as defined in the logical functional specifications. Verification of the devices takes 40 to 70 percent of the total development effort for the design. The increasing complexity of hardware designs raises the need for the development of new techniques and methodologies that can provide the verification team with the means to achieve its goals quickly and with limited resources.

The verification of the design based on simulation is very similar to the testing process of the device. In the testing case, the manufactured device is checked against the model of the design. The test patterns are used for this purpose. The test patterns are generated on the base of the model of the design in order to uncover physical defects, which may emerge in the device during the manufacturing phase. The test patterns are supplied to the manufactured device. If the response of the test pattern differs from the expected one according to the model, it is concluded that the device has a defect.

In the verification case, the design is checked, whether it meets the requirements and the specification. During the verification based on the simulation, the verification test stimuli are generated on the base of the model at a high level of the design in order to uncover the possible mistakes, which emerged during the design process. The main problem is how to choose the verification test stimuli and to determine the expected responses that it would be possible to state that the design does not have inaccuracies and meets the specification.

Usually in the initial stages of the design, the software prototype of the device is created in order to simulate the logical functionality of the device. We will use the software prototype model for one clock cycle, which will determine the responses on the outputs and the values of the next state according to the stimuli on the inputs and the values of the previous state. The input stimuli and the state elements can be chosen on the base of such a software prototype model. But an unambiguous relationship has to be defined between the state bits of software prototype and the flip-flops of the synthesized circuit that it would be possible to compare the responses on the outputs and the state values of these two different models. Generally, let's assume that verification test consists of input stimuli and output responses neglecting the fact that some inputs and outputs correspond to the state elements.

In the paper, the input stimuli, which form essential activity vectors, are selected from randomly generated ones on the base of software prototype. The output activity vector shows which input values have the influence on the output values. The essential activity vectors have the largest amount of active inputs. The essential activity vectors correspond to the terms of logical functions of output the existence of which is tested during the verification. The number of activity vectors depends on the termination condition of the

generation. The different values of the termination condition allow obtaining the different completeness of the verification test.

The paper is organized as follows. We review the problems of the verification in Section 2. We introduce the activity vectors in Section 3. We explore the process of the verification in Section 4. We present the results of the experiment in Section 5. We finish with conclusions in Section 6.

2. Related work

Logical functional verification of the devices is crucial and takes a substantial part of the entire design cycle time. Logical functional verification is widely acknowledged as the bottleneck in the hardware design cycle.

A wide variety of verification technology options are available, broadly classified as simulation-based technologies, static technologies, and formal technologies [1]. Simulation-based technologies allow the verification of the devices early in the design cycle, enabling fast time-to-market. Although design time can be shortened in a modern design environment, the verification effort grows exponentially as devices become more complex. Under time-to-market pressure, we must have a proper verification methodology for complex device development flow. The simulation is still the most widely used form of device verification: millions of cycles are spent during simulation using a combination of random and directed test cases in traditional design flow. Certain heuristics and design abstractions are used to generate directed random test cases. However, it is very time-consuming to write all the test programs manually. This brings about the necessity of developing an automatic program generator to speed up the verification work.

Built-in Self Test (BIST) methodology can be used for the verification of the devices [1]. But the test for this methodology can be obtained only when the structural level of the device is available and the completeness of this test is always problematic. Lichtenstein et al. [2] proposed an approach to verification test generation called as Model Based Test-Generation. This approach allows the incorporation of complex testing knowledge. The architecture model, which is comprised of logical functional blocks, is used. Fournier et al. [3] proposed a pseudo-random test program generator, Genesys, a follow-on of the model based test generation. Genesys enables the combination of randomness and control, thus generating high quality tests. The architecture model is used, as well. Fine and Ziv [4] addressed one of the main challenges of simulation based verification, by providing a new approach for Coverage Directed Test Generation. This approach is based on Bayesian networks and computer learning techniques. The specification driven and constraints solving based method to automatically generate test programs from simple to complex ones for advanced microprocessors is presented in [5]. Microprocessor architectural automatic test program generator can produce not only random test programs but also a sequence of instructions for a specific constraint by specifying a user constraints file. It is well studied and reported in the literature that for a tool to be scalable with larger designs, it is important to handle the design at higher levels of abstraction. An Automatic Assembly Program Generator that handles the design at the behavioral RTL level is presented in [6]. The Generator is based on logical function-oriented test generation schemes, hence making it scalable and usable for some specific tasks. In recent years special purpose verification languages have been developed to support automatic stimulus generation. Behm et al [7] reported on experience with a new test generation language for processor verification. Al-Asaad and Hayes [8] presented a simulation-based method for combinational design verification that aims at complete coverage of specified design errors using conventional ATPG tools. All common design errors can readily be mapped into stuck-at faults and a systematic method to perform this mapping is presented. The experimental results show that complete test sets for stuck-at faults detect almost all detectable errors. The experiments demonstrate that high coverage of the modeled design errors can be achieved with small test sets. Ugarte and Sanchez [9] presented an assertion checking technique for behavioral models that combines a non-linear solver and state exploration techniques and avoids expanding behavior into logic equations. In order to generate proper verification patterns for core-based design, the stuck-at fault model and automatic test pattern generation (ATPG) tools are usually used. In order to reduce the core-based design verification time, a connectivity-based port order fault (POF) model was proposed [10]. The POF assumes that a faulty cell has at least two I/O ports misplaced. In [11], a set of metrics, the Event Sequence Coverage Metrics are introduced. The approach is based on an automatic method to extract the control flow of a circuit which can be explored for coverage analysis and ATPG.

All the mentioned above verification methods rely on the behavior model written in the special language, on the architecture models consisting of the logical functional blocks or manipulate the ATPG used models. The software prototype model can be used for the verification purposes, as well. The software prototype model is written in the programming language at the early stages of the design process. Therefore, it is purposeful to use this model for the generation of the verification test. The software prototype model allows expressing the logical functionality of the device on the base of input stimuli and state variable values. The state variables are considered as inputs. They are considered as outputs when response is captured. The logical functionality of the device is defined by one clock cycle model. Next, we will introduce the activity vectors, which are obtained on the base of software prototype and form the verification test.

3. Activity vectors

Let's assume that the software prototype model has n inputs and m outputs. We denote the input stimulus by $P=<p_1, p_2, \ldots, p_i, \ldots, p_n>$, where $p_i= \{0, 1\}$, i=1, 2, …, n. The activity vector $P^j=<p^j_1, p^j_2, \ldots, p^j_i, \ldots, p^j_n>$ is associated with output j. A component of the activity vector can take on one of the following values: 0, 1, N, V. The value V shows that the complement of the value 1 on the input i changes the value to the opposite on the output j. The value N shows that the complement of the value 0 on the input i changes the value to the opposite on the output j. The activity vectors $P1^j$ set the value 1 on the output j, meanwhile the activity vectors $P0^j$ set the value 0 on the output j. The values V and N are the active values. The activity vector summarizes n + 1 input stimuli that differ only by single value. Let's say, we assign the following input stimulus $<X_1, X_2, X_3, X_4, X_5>$ = <01011> for the benchmark circuit C17 (0). This input stimulus sets the value 1 on the output $y1$. We complement every value of this stimulus one by one and we derive the following activity vector: <0VN11>. The activity vector summarizes the following input stimuli: <01011>, <11011>, <00011>, <01111>, <01001>, <01010>. These input stimuli set the value 1 on the output, except the third one and the fourth one.

M activity vectors $P1^j$ or $P0^j$ can be derived for every input stimulus P (M denotes the number of outputs). The activity vector P_a can dominate the activity vector P_b, and we will represent this feature like $P_a>P_b$. The activity vector P_b has the active values only on the same inputs as the activity vector P_a, and the active values of the vector P_b are equal to the active values of the vector P_a on the same inputs. If the active values of the vectors P_a and P_b are the same, then the activity vectors P_a and P_b are equal. The prerequisites of dominating the vector P_b by the vector P_a are presented in Table 1.

Table 1. The prerequisites of covering

P_a	V	N	V	V	N	N
P_b	V	N	1	0	0	1

The activity vector P_a dominates the activity vector P_b, if at least one of the conditions, which are in the last four columns of Table 1, is satisfied. The vector, which is not dominated by the other vectors, is essential. After analysis of input stimuli, the sets of essential vectors $A1^j$ and $A0^j$ are formed for every output j. The vectors in set $A1^j$ set the value 1 on the output j, while the vectors of set $A0^j$ set the value 0 on the output j.

Let's consider the benchmark C17 presented in 0. The input stimulus $P = <X_1, X_2, X_3, X_4, X_5>$ = <01110> sets the following output values: <$y1$, $y2$> = <00>. The results of complementing every input value one by one are presented in 0. We obtain the following activity vectors according to this table: $P0^{y1}$ = <$N1VV$0>, $P0^{y2}$ = <01VV0>. In the same way, for the input stimulus P = <00110> we obtain the following activity vectors according to 0: $P0^{y1}$ = <N0110>, $P0^{y2}$ = <00110>. These vectors are not essential, because they are covered by the previous vectors.

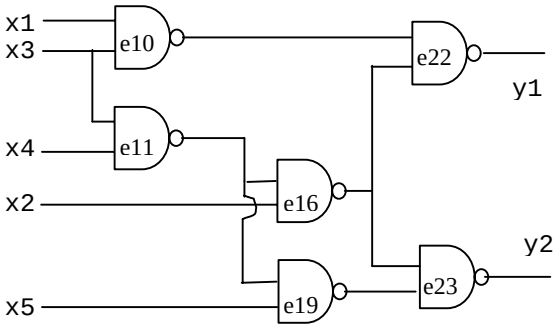

Figure 1. Benchmark circuit $C17$

Table 2. The complement of input values of stimulus <01110>

p_1	p_2	p_3	p_4	p_5	y_1	y_2
0	1	1	1	0	0	0
1	1	1	1	0	1	0
0	0	1	1	0	0	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	0

Table 3. The complement of input values of stimulus <00110>

p_1	p_2	p_3	p_4	p_5	y_1	y_2
0	0	1	1	0	0	0
1	0	1	1	0	1	0
0	1	1	1	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	1	0	0

After analysis of all the possible input stimuli, we obtain the following essential activity vectors:

$A0^{y1}$ = {< $N1VV$0>, <1NN10>, <NN100>}
$A1^{y1}$ = {< $V0V$10>, <0$V1N$1>, <0VN10>}
$A0^{y2}$ = {< 1$N01N$>, <11VV1>}
$A1^{y2}$ = {< 101NV>, <10$N1V$>, <1$V1N$0>, <0VN10>}

We notice that the active values of the essential activity vectors correspond to the variables of the terms of the direct and inverse logical functions. Then, we obtain the following logical functions:

$$y1 = X_1X_3 + X_2\overline{X}_4 + X_2\overline{X}_3$$
$$\overline{y1} = \overline{X}_1X_3X_4 + \overline{X}_2\overline{X}_3 + \overline{X}_1\overline{X}_2$$
$$y2 = \overline{X}_4X_5 + \overline{X}_3X_5 + X_2\overline{X}_4 + X_2\overline{X}_3$$
$$\overline{y2} = \overline{X}_2\overline{X}_5 + X_3X_4$$

But the complete correspondence not always exists between the values of the essential activity vectors and

the variables of the terms of the logical function. It is possible to think of the example where the active values of the activity vectors are a subset of the variables of the terms of the logical function. Let's consider the logical function TTF of three logical variables:

$$Y = X_1X_2X_3 + \overline{X}_1\overline{X}_2\overline{X}_3$$

$$\overline{Y} = X_1\overline{X}_2 + X_1\overline{X}_3 + \overline{X}_1X_2 + X_2\overline{X}_3 + \overline{X}_1X_3 + \overline{X}_2X_3$$

After analysis of all the possible input stimuli, we obtain the following essential activity vectors:

$A1$ = {< *VVV*>, <*NNN*>}

$A0$ = {< *V*00>, <0*V*0>, <00*V*>, <*N*11>, <1*N*1>, <11*N*>}.

That corresponds to the following terms of the logical function:

$$Y = X_1X_2X_3 + \overline{X}_1\overline{X}_2\overline{X}_3$$

$$\overline{Y} = X_1 + X_2 + X_3 + \overline{X}_1 + \overline{X}_2 + \overline{X}_3$$

We notice that the active values of the activity vectors of the set $A0$ form the incomplete terms of the inverse logical function. The active values of the activity vectors can produce the incomplete terms if not all possible input stimuli are considered. Such a situation arises for the large circuits. But that it is not the case for the example logical function TTF.

Conjecture. The active values of the essential activity vectors of the sets $A1^j$ and $A0^j$ of the output j correspond to the complete or incomplete terms of the direct and inverse logical function.

We cannot prove this conjecture, but, on the other hand, we were unable to find the example that would contradict to our conjecture. The investigation is difficult because the logical function of the output can be expressed in many different ways. The logical function is not obligatory minimal in all the cases.

Because the essential activity vectors correspond to the complete or incomplete terms of the logical function of the outputs, there is a possibility to check whether the output responses of the synthesized circuit not contradict to the existence of the term of the logical function. Let's consider how it is possible to determine the membership of the term in the logical function of the output. The term consists of the input logical variables. The variable of the term can be in complemented or uncomplemented form. The term is completely defined if the values of the uncomplemented variables are equal to 1, whereas the values of the complemented variables are equal to 0. The term $X_1\overline{X}_2X_3$ will be completely defined, if the value 1 will be assigned to the variables X_1 and X_3, and the value 0 – to the variable X_2. The input stimulus, which completely defines the term of the direct (inverse) logical function, sets the value 1 (0) on the output. If the term $X_1\overline{X}_2X_3$ belongs to the direct logical function, then any input stimulus, which completely defines the considered term, sets the value 1 on the output. If the term $X_1\overline{X}_2X_3$ belongs to the inverse logical function, then any input stimulus, which completely defines the considered term, sets the value 0 on the output. Generally, the term of the logical function determines the input stimuli that set the same value on the output.

Condition 1. All the input stimuli, which completely define the term, always set the same value on the output.

This condition is necessary, but not sufficient. Any two terms of the logical function, the variables of which do not contradict each other, will satisfy Condition 1. The Condition 2 defines the prerequisites for the single term.

Condition 2. Every variable of the term has the corresponding input stimulus that the assignment of the opposite value to the variable of the term invokes the value change on the output.

Let's assume that the accordance of the input stimuli to the Condition 1 and Condition 2 confirms the existence of the term of the logical function. The activity vector does not support completely both the conditions of the existence of the term. Firstly, the only n-k input stimuli, which completely define the term, are considered, where n – the number of inputs, k – the number of the active inputs. Additionally, the accordance to Condition 2 is satisfied only for a single input stimulus. For example, Condition 2 for the term $X_1\overline{X}_2$ of the logical function TTF of three variables is satisfied by two input stimuli 100 and 101. Therefore, when we consider the single input stimulus, we cannot obtain a minterm, but only the term $X_1\overline{X}_2$. Based on this observation, we will introduce a Rule 1 indicating how to construct the minterms from the terms.

Rule 1. The two terms defined by the activity vectors can be combined into a single term, if the combined terms have the different variables and the values of the inactive inputs coincide with the values of the active inputs.

The constructed term has to satisfy Condition 1 and Condition 2. For the logical function TTF on the base of Rule 1, we obtain the following essential activity vectors: $A0$ = {<*V*00>, <0*V*0>, <00*V*>, <*N*11>, <1*N*1>, <11*N*>}. The combination of <*V*00> and <1*N*1> allows obtaining the term $X_1\overline{X}_2$, and the combination of <*V*00> and <11*N*> allows obtaining the term $X_1\overline{X}_3$. In such a way, we can obtain all the terms of inverse function:

$$\overline{Y} = X_1\overline{X}_2 + X_1\overline{X}_3 + \overline{X}_1X_2 + X_2\overline{X}_3 + \overline{X}_1X_3 + \overline{X}_2X_3.$$

This logical function is not minimal If we combine the activity vector with the single other activity vector, we could obtain the minimal logical function: $\overline{Y} = X_1\overline{X}_2 + X_2\overline{X}_3 + \overline{X}_1X_3$.

The size of the sets of the activity vectors $A1^j$ and $A0^j$ directly depends on the size of the set of the input stimuli considered. The number of the activity vectors is directly proportional to the number of the terms of the logical function of the output. Therefore, after

finding the appropriate number of the activity vectors, the number of the activity vectors does not increase more. We could use this feature for the generation of the verification test. The input stimuli are generated randomly and selected only those ones that increase the set $A1^j$ or the set $A0^j$ of the essential activity vectors. The input stimuli selected according to this rule can be used as the verification test. The generation of the random input stimuli becomes ineffective when the generation process does not lead to the selection of new input stimuli. If the generation does not lead to the increase of the sets $A1^j$ or $A0^j$ during the predefined time limit, the generation is stopped. Using this simple algorithm, the verification tests were generated for the ISCAS'85 benchmark circuits. The results are presented in Section 5.

The random generation is not the most effective way to find the activity vectors. We noticed that the process of finding the activity vectors is more effective, if we use an adjacent generation of the stimuli for the selected ones [12, 13]. The adjacent activity vector differs from the selected one by a single value only. The change of the active value allows obtaining the activity vector, which sets the opposite value on the output, whereas the change of the inactive value allows obtaining the new activity vector in some cases. The probability that the randomly generated stimuli will differ by a single value is small. Therefore, the generation of the stimuli that are adjacent to the selected ones allows to enrich the random search and to speed up the process of finding the new activity vectors [12, 13].

Additionally, the active values of the activity vectors can be considered as the terms of the logical function of the output and they could be used for the generation of the new input stimuli. In order to obtain the activity vector of the set $A1^j$ ($A0^j$) having the most possible number of the active values, it needs to define as many as possible of the values of the activity vector of the opposite set $A0^j$ ($A1^j$) having the single complemented active value only. This feature allows creating various deterministic input stimuli generation methods that enrich ineffective random search.

4. Generation of the verification test

Verification is used to determine the correspondence of the design to the specification. The software prototype can be used instead of the specification during the verification. In such a case, a large amount of the input stimuli can be generated randomly, and the comparison of the responses of the software prototype model and the model of synthesized circuit can be carried out (Figure 2). The verification process takes long hours if the long sequences of the input stimuli are generated. The simulation of the input stimuli on the model of the synthesized circuit requires much more time than the simulation of the input stimuli on the software prototype model. Therefore, it is purposeful on the base of the software prototype model to select the input stimuli, which are essential for the verification of the model of the synthesized circuit, and to use them for the verification. The selection of the input stimuli on the base of the software prototype can be carried out in parallel with other design activities. Such a parallelism can shorten the verification time. The main problem of the selection is how to determine which input stimuli are essential. We will base the selection on the conjecture that the essential activity vectors correspond to the terms of the logical functions of the output. Therefore, we will assume that the essential activity vectors, which will be selected from the randomly generated stimuli. can form the verification test.

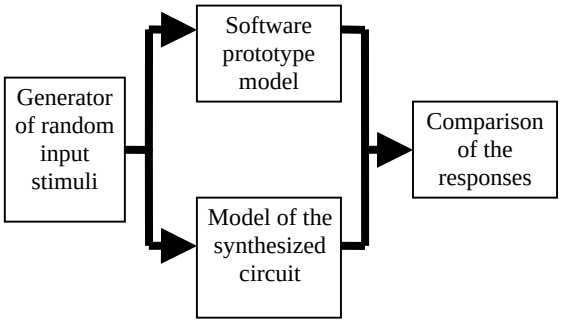

Figure 2. Scheme of the verification

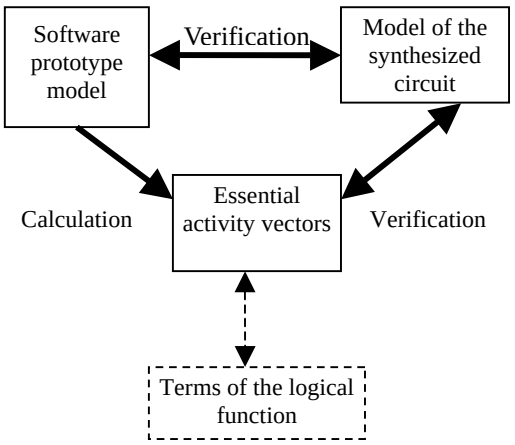

Figure 3. Verification structure

During the verification, it is determined the correspondence of the model of the synthesized circuit to the software prototype model. The essential activity vectors selected from the randomly generated stimuli can not reveal completely the functionality of the software prototype model. But the obtained activity vectors have to correspond to the functionality of the model of the synthesized circuit. The verification structure is shown in Figure 3.

The essential activity vectors can be regarded as the simplified functional description of the device obtained from the software prototype model. Therefore, it is possible to verify whether the designed circuit corresponds to the set of the essential activity vectors. During the verification between the set of the essential activity vectors and the model of the designed circuit, the responses of the every essential activity vector are

checked against the responses of the model of the synthesized circuit. We will call this process the verification of the essential activity vector. The verification of the essential activity vectors does not require the use of the responses of the software prototype model, because the response of the activity vector is determined by its dependence to the sets $A0^j$ or $A1^j$. If the verification of all essential activity vectors is complete, we will consider that the model of the synthesized circuit corresponds to the software prototype model.

The essential activity vectors correspond to the terms of the logical functions of the output. Therefore, we can study the verification of the terms of the logical functions and compare the verification of the essential activity vectors and the verification of the terms. During the term verification, the compliance of the responses of the model of the design to the existence of the term is determined. Such an action is equivalent to the fulfillment of Condition 1 and Condition 2.

The term of the logical function defines the input stimuli, which set the same value on the output. Let's assume, a term of the 10-input logical function consists of the following 4 variables: $X_2\overline{X}_5X_8\overline{X}_{10}$. This term implies that the values of the other 6 variables $X_1X_3X_4X_6X_7X_9$ do not have any influence on the output value, which will be set to 1, if X_2 is assigned to 1, X_5 – 0, X_8 – 1, X_{10} – 0. 2^{n-k} input stimuli have to be considered, where k – the number of the variables of the term. In such a way, the fulfillment of Condition 1 is checked. If not all the combinations will be evaluated, the assessment of Condition 1 will not be complete.

The three following cases have to be considered for the verification of the essential activity vector:

- $V1$ – the single input stimulus, which corresponds to the activity vector, is used for the verification. In such a way, the fulfillment of Condition 1 is verified at least once.
- $V2$ – the k input stimuli, which differ from the activity vector by one active input value, are used for the verification, where k – the number of the active variables of the term. The captured k output responses have to be opposite to the response of the activity vector. In such a way, the fulfillment of Condition 2 is verified for the term of the logical function that corresponds to the activity vector.
- $V3$ – the n-k input stimuli, which differ from the activity vector by one inactive input value, are used for the verification. In such a way, the fulfillment of Condition 1 is verified additionally n-k times for the term of the logical function that corresponds to the activity vector.

All three cases can be considered during the verification of any design. The use of all three cases enables the generation of n+1 input stimuli, which differ by single value.

Let's assume that for the input stimuli <0110001100> the activity vector $0V10N01V0N$ is obtained, which corresponds to the term $X_2\overline{X}_5X_8\overline{X}_{10}$. For the case $V1$, we will form the input stimulus <0110001100>, and we will measure the response of the model of the synthesized circuit to this stimulus. The value of the response has to be 1. The existence of the term is acknowledged by the input stimuli, which differ by single input value V or N, because these input stimuli change the output value to the opposite one. The following input stimulus: <0010001100>, <0110101100>, <0110001000>, <0110001101> formed for the case $V2$ will set the value 0 at the output. If this activity vector is essential, the considered input stimuli check the fulfillment of Condition 2 by the corresponding term of the logical function. For the case $V3$, the input stimuli that differ by the single inactive input value acknowledge the fulfillment of Condition 1 by the term. The following input stimuli fall into this category: <1110001100>, <0100001100>, <0111001100>, <0110011100>, <0110000100>, <0110001110>. They set value 1 at the output. Generally, the use of all three cases for the verification of the activity vector does not guarantee that all the input stimuli required for the verification of the term will be considered. Therefore, the terms obtained from the activity vectors cannot be precisely determined. They cannot be used for the synthesis of the device, but they can be used for the verification and for the test generation. The indetermination of the terms can reduce the reliability of the verification, but this obstacle does not deny the possibility of the use of the terms obtained from the activity vectors.

During the generation, only those input stimuli are selected, which form the new activity vectors for the sets $A1^j$ or $A0^j$. The essential activity vectors are included into the verification test. The selected stimuli can be included into the verification test as well. The activity vectors for several outputs can be obtained from the single selected input stimuli.

The increase of the size of the random search space decreases the amount of the selected input stimuli, which augment the sets $A1^j$ and $A0^j$. The same selected stimuli can appear in several activity vectors of the different outputs. Such selected stimuli AIP link together several activity vectors, and they can be used for the verification of the terms of several outputs at once. In such a case, the responses at the additional outputs can be added to the selected input stimuli. The selected input stimuli can be verified according to all three cases $V1$, $V2$, $V3$ taking into account the responses of all the outputs. For the cases $V1$ and $V3$, it is enough to mark the active inputs only, meanwhile for the case $V2$, the outputs, on which the value will be changed according to changed value on the input, have to be known. The number of the selected input stimuli can be several times less than the number of the activity vectors. The appropriate example is presented in Table 4.

Table 4. Combining of several activity vectors

	Output $y1$		Output $y2$		Outputs $y1$ and $y2$	
	Input stimuli	response $y1$	Input stimuli	response $y2$	Input stimuli	response $y1,2$
$V1$	1NN11	0	10N1V	1	1NN1V	0,1
$V2$	11011	1	10111	0	11011	1,0
	10111	1	10010	0	10111	1,0
					10010	0,0
$V3$	00011	0	00011	1	00011	0,1
	10001	0	11011	1	10001	0,1
	10010	0	10001	1		

Let's assume that the selected input stimulus P = <10011> sets the following values on the outputs: <01>. We obtain the activity vector $A0^{y1}$ = <1NN11> for the first output and the activity vector $A1^{y2}$ = <10N1V> for the second output. Then the linked activity vector for both outputs is <1NN1V>. These vectors are presented in the row under name $V1$ of Table 4, and they correspond to the case $V1$. The verification of the activity vector $A0^{y1}$ according to the case $V2$ uses the following input stimuli: <11011>, <10111>, meanwhile the verification of the activity vector $A1^{y2}$ according to the case $V2$ uses the following input stimuli: <10111>, <10010>. The similar situation is for the case $V3$. The input stimuli are presented in the rows under name $V3$ of Table 4. As we have mentioned, the linked activity vector for both outputs is <1NN1V>. Therefore, the verification of the selected input stimuli according to the case $V2$ uses the following three input stimuli: <11011>, <10111>, <10010>. Meanwhile, the verification of the selected input stimuli according to the case $V3$ uses the following two input stimuli: <00011>, <10001>. As we can notice, some input stimuli are repeated, when the activity vectors are used for the verification. When the verification is based on the selected input stimuli, we obtain the same input stimuli as in the case of the activity vectors. The input stimulus is selected if at least one essential activity vector is constructed on its base. Therefore, the selected input stimulus can form inessential activity vectors for the other outputs. When the verification is based on the selected input stimuli, the inessential activity vectors will be used, and the verification will be directed to the larger number of the outputs. Hence, the selected input stimuli are more useful for the verification than the activity vectors. Additionally, the number of the selected stimuli is less than the number of the activity vectors.

5. Experiments

We will assess the quality of the verification test by the following parameters: the length of test L, the fault coverage of the stuck-at faults FC, the fault coverage of the pin pair faults PPC,, and the number of the essential activity vectors AV. The very important parameter is the length L of the test, because the duration of the verification depends directly on it. As we have mentioned previously, the fault coverage FC of the stuck-at faults is the good indicator of the possibility to assess the presence of the errors in the design. But this is possible when the synthesized circuit is available.

During the whole design process from the software prototype to the synthesized circuit, the relationships between the inputs and outputs must be kept the same. Any pair (i, j) of the input and output can be related through the odd and/or even number of the inversions. The input i and the output j are connected through the even number of the inversions, if there exists an input stimulus that the value on the input i and the value on the output j are the same, and the change of the value on the input i invokes the change of the value on the output j. Analogously, the input i and the output j are connected through the odd number of the inversions, if there exists an input stimulus that the value on the input i and the value on the output j are opposite, and the change of the value on the input i invokes the change of the value on the output j. The change of the parity of the relationship between the input and output would indicate the fault during the design process.

The relationship between the inputs and outputs corresponds to the functional fault model, which is used for the functional test generation according to the software prototype model [12]. The test patterns are selected from the random generated ones on the base of the pin pair (PP) fault model. In the same way, the activity vectors are selected that possess at least one active input related to some output. The relationship between the inputs and outputs (the number of the functional faults) is the feature of the design that does not change during the design process. The estimation of all the possible relationships is complex and not always it could be established at the functional level. But generally, the number of the detected functional PP faults and the coverage PPC of the PP faults indicate the quality of the verification test, when the total number of the detectable functional PP faults is known.

The fourth parameter that indicates the quality of the functional test is the number AV of the essential activity vectors. Every essential activity vector is related to some term of the logical function. The larger

number of the activity vectors ensures the better recognition of the existence of the terms of the logical function. If the verification test reveals the existence of all the terms of the logical function, then it would be possible to state that the behavior of the software prototype and the behavior of the synthesized circuit are identical. Therefore, the fourth parameter is important, as well.

The circuits of the benchmark suite ISCAS'85 have been selected for the experiments because they are more complicated for testing purposes than combinational parts of the sequential circuits of the benchmark suite ISCAS'89. Let's assume that the input stimuli are used for the verification according to case *V*1.

Table 5. The parameters of the verification test

Circuit	L	FC%	PPC%	#AV	#AIP
C432	500000	100.00	100.00	14803	12423
C499	500000	100.00	100.00	69336	68870
C880	500000	100.00	100.00	23086	17001
C1355	500000	100.00	100.00	69336	68870
C1908	500000	100.00	100.00	28561	20632
C2670	500000	96.03	81.69	63838	39349
C3540	500000	100.00	99.96	67520	45441
C5315	500000	100.00	100.00	426629	180931
C6280	500000	100.00	100.00	53197	26440
C7552	500000	98.80	69.08	551981	273212
Average	500000	99.48	95.07	136829	75317

Table 5 presents the results of the verification test that was obtained from the 500 000 input stimuli generated randomly. The initial randomly generated test detects all the stuck-at faults, except for the circuits C2670 and C7552. The randomly generated input stimuli especially were unsuitable for the detecting of the stuck-at faults of the circuit C2670. The randomly generated test does not fully cover the PP faults for the mentioned circuits and the circuit C3540. The number of the PP faults for the circuits of the benchmark suite ISCAS'85 is known, therefore, the PP fault coverage can be calculated. The number of the essential activity vectors is shown in the column under name #AV. The last column holds the number of the selected input stimuli from set of the initial 500 000 random input stimuli. These stimuli form the essential activity vectors. The selected input stimuli allow shortening the verification test on the average 500 000 / 75317 = 6.64 times. We have to notice that the selected input stimuli detect the same stuck-at faults and PP faults, and they form the same essential activity vectors as the initial random stimuli. Therefore, the quality of the selected stimuli according to parameters FC, PPC and AV does not decrease but the number of them is 6.64 times less. The verification test based on the selected stimuli allows shortening the verification time in the same quantity except for the circuit C7552, for which the verification time can be less only about 2 times.

Let's consider the suitability for the verification of the functional test for the pin pair triplets PT faults [12]. This test differs from the PP fault test that every input pattern detects PP faults in pairs. The results are presented in Table 6.

Table 6. The PT fault test

Circuit	L	FC%	PPC%	#AV
C432	873	100.00	100.00	2304
C499	3458	100.00	100.00	7015
C880	7011	100.00	100.00	12917
C1355	3481	100.00	100.00	7043
C1908	2901	99.62	100.00	16879
C2670	4575	100.00	100.00	13388
C3540	7175	100.00	100.00	35296
C5315	8578	100.00	100.00	66599
C6280	2016	100.00	100.00	41888
C7552	15409	99.91	100.00	127876
Average	5512	99.95	100.00	33120

As we can see, the use of PT fault test for the verification allows shortening the length of test 75315 / 5512 = 13.66 times in comparison with the length of the verification test that is based on the selected stimuli. The PT fault test detects more stuck-at faults, detects all the PP faults, but it allows to form 136829 / 33120 = 4.13 times less essential activity vectors.

We also present the results of the PP fault test for the comparison purposes (Table 7). The PP fault test is 5512 / 772 = 7.14 times shorter than the PT fault test. The PP fault test detects fewer stuck-at faults, and it allows forming 33120 / 7916 = 4.18 times less essential activity vectors.

Table 7. The PP fault test

Circuit	L	FC%	PPC%	#AV
C432	117	97.8	100.00	395
C499	1077	100.0	100.00	4462
C880	381	100.0	100.00	1810
C1355	1011	100.0	100.00	4384
C1908	620	96.6	100.00	5272
C2670	448	99.6	100.00	3001
C3540	515	98.9	100.00	4978
C5315	1169	100.0	100.00	20436
C6280	268	100.0	100.00	6478
C7552	2115	99.6	100.00	27942
Average	772	99.25	100.00	7916

As we can notice, the significantly shorter verification test can detect almost all the stuck-at faults and PP faults, but its ability to check the existence of the terms of the logical function is much lesser. Therefore, the dominant factor of the assessment of the quality of

the verification test is the number of the essential activity vectors.

The procedure that selects the input stimuli from the randomly generated ones for the formation of the essential activity vectors is the verification test generation procedure. The quality of the verification test is assessed by the number of the selected stimuli, which depends on the termination condition of the generation. In such a way, the proportion of the number GP of the randomly generated stimuli to the number AIP of the selected stimuli is the indicator of the quality of the verification test. This proportion shows how many times the time of the verification is shortened and it can be used as the termination condition of the generation.

The ratio GP/AIP is shown in 0 (column 2) when 500 000 random input stimuli were generated. We see that this ratio does not exceed 10 for four circuits, and 500 000 random input stimuli are not enough in order to obtain the high quality of the verification test for these circuits. The smallest ratio for the circuit C7552 makes clear why the selected input stimuli do not detect all the stuck-at faults. The ratio for the circuit C2670 is larger than 10, but nevertheless some stuck-at faults remain undetected. Of course, the ratio GP / AIP can be set, for example, to 100, but the experimental research revealed that the use of the value of the proportion larger than 10 makes the generation ineffective, because the generation selects only few new activity vectors. In such a way, the termination condition of the generation is tightly related to the quality of the verification test.

Table 8. Ratio GP / AIP

Circuit	GP/AIP	Length	TetraMAX
C432	40.25	73	57
C499	7.27	99	54
C880	29.41	78	62
C1355	7.27	99	86
C1908	24.23	142	118
C2670	12.70	122	105
C3540	11.00	242	167
C5315	2.76	193	130
C6280	18.91	115	43
C7552	1.83	309	211
Average	15.56	147	103

The obtained verification test can be the initial basis for the construction of the manufacturing test in order to detect all the stuck-at faults. The simple way is to use the fault simulation of the synthesized circuit and on this base to select the test patterns that detect stuck-at faults. If the obtained verification test is too large for the fault simulation, then this test can be minimized on the base of the functional faults at the functional level of the circuit. The input stimuli that do not detect the new faults also should be removed, and then the number of the remaining input stimuli is quite similar to the number of the input stimuli obtained at the gate level of the circuit. These numbers are shown in the third and fourth columns of 0, respectively. We see that the manufacturing test obtained on the base of the verification test is on the average 147 / 103 = 1.43 longer than the test obtained at the gate level by the test generation tool TetraMAX.

6. Conclusion

The paper presents an approach to the generation of the verification test, which is based on the software prototype written according to the specification. The formation of the essential activity vectors from the randomly generated input stimuli decreases the number of the input stimuli used for the verification. The selected input stimuli that form the essential activity vectors also can be used for the verification without loss of the verification coverage. The number of the input stimuli used for the verification and the quality of the verification test depend on the termination condition of the generation. The value of this approach is confirmed by the fact that the selected input stimuli detect the same stuck-at faults as the initially generated test set. The essential activity vectors and selected input stimuli can be augmented by the adjacent input stimuli that differ by the single value. The number of the essential activity vectors that corresponds to the number of the verified terms of the logical function is the measure of the quality of the verification test. The larger number of the essential activity vectors means the higher quality of the verification. An ideal case is when the essential activity vectors are obtained for all the terms of the logical function. The verification test can be used in order to obtain the manufacturing test.

References

[1] **K. Batcher, C. Papachristou**. Instruction Randomization Self Test For Processor Cores. *Proceedings of the* 17*th IEEE VLSI Test Symposium* (VTS'99), *San Diego, CA, USA, IEEE Computer Society, April* 25-30,1999, 34-40.

[2] **Y. Lichtenstein, Y. Malka, and A. Aharon**. Model-Based Test Generation for Processor Design Verification. *Innovative Applications of Artificial inteligence (IAAI), AAAI Press*, 1994.

[3] **L. Fournier, Y. Arbetman, and M. Levinger**. Logical Functional Verification Methodology for Microprocessors Using the Genesys Test-Program Generator – Application to the x86 Microprocessors Family. *Proceedings of the Design, Automation and Test in Europe* (DATE'99), *Munich, Germany, March* 9-12, 1999**,** 434-441.

[4] **S. Fine, A. Ziv**. Coverage Directed Test Generation for Functional Verification using Bayesian Networks. *Proceedings of the* 40*th Design Automation Conference* (DAC'03), 2003, 286-291.

[5] **T. Li, D. Zhu, Y. Guo, G. Liu, S. Li**. MA^{2}TG: A Functional Test Program Generator for Microprocessor Verification. *Proceedings of the* 2005 8*th Euromicro conference on Digital System Design* (DSD'05), 2005, 76-183.

[6] **K.U. Bhaskar, M. Prasanth, V. Kamakoti, K. Maneparambi**l. A Framework for Automatic Assembly Program Generator (A^{2}PG) for Verification and Testing of Processor Cores. *Proceedings of the* 14*th Asian Test Symposium* (ATS '05), 2005, 40-45.

[7] **M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, M. Vinov**. Industrial Experience with Test Generation Languages for Processor Verification. *Proceedings of the* 41*st Design Automation Conference* (DAC'04), 2004, 36-40.

[8] **H. Al-Asaad, J.P. Hayes**. Design Verification via Simulation and Automatic Test Pattern Generation. *Proceedings of the* 1995 *International Conference on Computer-Aided Design (*ICCAD '95), 1995, 174-180.

[9] **I. Ugarte, P. Sanchez**. Functional Vector Generation for Assertion-Based Verification at Behavioral Level Using Interval Analysis. *Proceedings of the Eighth IEEE International Workshop on High-Level Design Validation and Test Workshop*, 2003, 102-107.

[10] **S.-W. Tung, J.-Y. Jou**. Verification Pattern Generation for Core-Based Design Using Port Order Fault Model. *Proceedings of the* 7*th Asian Test Symposium,* 1998, 402-407.

[11] **D. Moundanos, J.A. Abraham**. Using Verification Technology for Validation Coverage Analysis and Test Generation. *Proceedings of VLSI Test Symposium*, 1998, 254-259.

[12] **E. Bareiša, V. Jusas, K. Motiejūnas, R. Šeinauskas**. The Realization-Independent Testing Based on the Black Box Fault Models. *Informatica*, 2005, *Vol*.16, *No*.1, 19-36.

[13] **E. Bareisa, V. Jusas, K. Motiejunas, R. Seinauskas**. Logical Functional Digital Systems Testing. *Kaunas, Technologija*, 2006.

Received August 2008.