ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
2015 Управление, вычислительная техника и информатика № 4 (33)

УДК 681.5.09
DOI: 10.17223/19988605/33/10

A.Yu. Matrosova, S.A. Ostanin, E.A. Nikolaeva, I.E. Kirienko

FULLY DELAY AND MULTIPLE STUCK-AT FAULT TESTABLE SEQUENTIAL CIRCUIT DESIGN

Supported by Russian Science Foundation (project № 14-19-00218).

In this paper we show that it is possible to derive a sequential circuit either from a transition table or from a State Transition Graph (STG), so that the sequential circuit has the short test detecting all multiple stuck-at faults at gate poles of the circuit, and delay of each circuit path is detectable

Keywords: finite state machine (FSM); state transition graph (STG); path delay faults (PDFs); multiple stuck-at faults; monotonous functions.

Under development of nanometer technologies it is not enough to test single stuck-at faults at gate poles of logical circuits. We need to test multiple stuck-at faults at gate poles along with delay faults of circuits. A path delay fault (PDF) model is often used at delay testing. In accordance with the conditions of fault manifestation, PDFs are divided into robust testable faults and non-robust testable faults [1, 2]. The PDF is robust testable on condition that the fault manifestation on a test pair does not depend on delays of other circuit paths. The PDF is non-robust testable if the fault manifestation is possible only when all other paths of a circuit are fault-free. Note that the robust testable PDF may be located and measured.

It is known that even for a simple circuit it is impossible to enumerate its multiple stuck-at faults, and only 20% of PDFs are robust testable. That is why it is very important to provide testability of logical circuits both for multiple stuck-at faults and delays during the circuit design.

Application of these synthesis methods to a combinational part of a sequential circuit may generate rather long paths in the obtained circuit. We suggest [4] a new method of the fully delay testable combinational part design that is oriented to cutting down the lengths of circuit paths in contrast to the lengths in the circuits obtained by the method suggested in [3]. In this method [4] the circuit behavior is represented with a composition of ROBDDs and monotonous products extracted from a state transition graph (STG). The experimental results have shown that the circuit paths in the resulted combinational parts [4] are cut from two to twenty times. Unfortunately, for these circuits we cannot find rather a short test detecting all multiple stuck-at faults at their gate poles.

For a high performance circuit it is desirable to detect multiple stuck-at faults together with PDFs. In paper [5] it is shown that if we apply the multilevel logic minimization method, based on algebraic division, to the system of irredundant sum of products (SoPs) describing the circuit behavior, we get a circuit testable for multiple stuck-at faults. We mean that a test detecting single faults of the literals of the system of irredundant SoPs detects all multiple stuck-at faults at gate poles of the resulted circuit.

In this paper we suggest the design method that allows deriving the sequential circuit that is both fully delay testable and multiple stuck-at fault testable.

The circuit behavior is represented either with a transition table or with the STG in which symbols of input and output alphabets have already encoded. For the STG we need to use additional inputs to provide above mentioned testability properties.

In Section 1 the approach to a sequential circuit design is suggested. In Section 2 the testability properties of the resulted circuit are investigated, and some means of cutting a test length are noted.

1. Sequential circuit design

Let the discrete device behavior be represented either with the transition table or the STG. We suggest the method of deriving the sequential circuit that implements the behavior and has the above mentioned testability properties. An example of the STG description is given in table 1.

Note that for encoding states we may use any unordered code. A code is unordered if any two code words are not comparable. Two code words α, β are not comparable if $\alpha \not\leq \beta$ and $\beta \not\leq \alpha$. For example, the code words 01010, 101111 are not comparable. We suggest encoding symbols of a state alphabet with unordered code words, in partly, code words of (m, n) code. Here n is a length of a code word and m is a number of its 1value components. To cut the length of code words we may choose value m being approximately equal to $\lceil 1/2 \rceil$. An example of the STG description after encoding states is given in table 2. Here we use (1, n) code).

Table 1

An example of the STG description

$x_1\, x_2\, x_3$	S	S	$y_1\, y_2\, y_3\, y_4\, y_5$
0 - -	1	1	0 0 0 1 0
- 0 -	1	1	0 0 0 1 0
1 1 -	1	2	1 0 0 1 0
- - 0	2	2	0 0 1 1 0
- - 1	2	3	1 0 1 1 0
1 0 -	3	3	0 1 0 0 0
0 - -	3	4	1 1 0 0 0
- 1 -	3	4	1 1 0 0 0
- - 0	4	4	0 1 0 0 1
- - 1	4	1	1 1 0 0 1

Table 2

The STG description after encoding states

$x_1\, x_2\, x_3$	$z_1\, z_2\, z_3\, z_4$	z_1', z_2', z_3', z_4'	$y_1\, y_2\, y_3\, y_4\, y_5$
0 - -	1 0 0 0	1 0 0 0	0 0 0 1 0
- 0 -	1 0 0 0	1 0 0 0	0 0 0 1 0
1 1 -	1 0 0 0	0 1 0 0	1 0 0 1 0
- - 0	0 1 0 0	0 1 0 0	0 0 1 1 0
- - 1	0 1 0 0	0 0 1 0	1 0 1 1 0
1 0 -	0 0 1 0	0 0 1 0	0 1 0 0 0
0 - -	0 0 1 0	0 0 0 1	1 1 0 0 0
- 1 -	0 0 1 0	0 0 0 1	1 1 0 0 0
- - 0	0 0 0 1	0 0 0 1	0 1 0 0 1
- - 1	0 0 0 1	1 0 0 0	1 1 0 0 1

We consider each line of table 2 as pair (u_i, h_i) of vectors. The first ternary vector (cube) u_i of the pair is represented with two first columns of Table 2, and has the length $n + p$ (in the considered example the length is 3 + 4 = 7). Here n is the number of circuit's inputs and p is the number of state variables (loops). The second Boolean vector h_i represented with two last columns of table 2 is a characteristic of the first vector. In this characteristic 1 values point to the functions taking 1 value on the first vector (cube), but 0 values point to the functions taking 0 value on the first vector (cube). Note that table 2 represents the system of incompletely specified Boolean functions that are divided into transition functions and output functions. Take into consideration that in Table 2 each incompletely specified function f_i of the system is represented with the cubes of on-set $M^1_{f_i}$ and the cubes of off-set $M^0_{f_i}$.

Give some definitions.

Call (w_j, h_j) as a system cube of an incompletely specified Boolean functions system, where characteristic h_j means that its 1 value component points to the function that take 1 value on the cube w_j; here we don't pay attention on 0 values of this characteristic.

System cube (w_j, h_j) is essential if $w_j \cap M^1_{f_i} \neq \emptyset$ and $w_j \cap M^0_{f_i} \neq \emptyset$ for each function f_i taking 1 value in characteristic h_j.

A system cube (w_j, h_j) covers a system cube (u_i, h_i) if $h_j = h_i$, and w_j covers u_i.

We differ cubes represented by ternary vectors without characteristics from system cubes supplied with characteristics.

Note that two cubes are orthogonal each other if there is at least one component that has 1 value in one cube and 0 value in another cube. For example, the cubes 010--01, 010--11 are orthogonal by the sixth component.

Two cubes are opposite two orthogonal each other if there are two components: in one of these components the cubes have 10 values, but in another component they have 01 values. For example, the cubes 0--101, 1-0001 are opposite two orthogonal by the first and the forth components.

In a set of unordered code words any two words are opposite two orthogonal.

Note that the cubes in table 2 corresponding to the different states are opposite two orthogonal; the cubes corresponding to the same state and having the different characteristics are surely orthogonal, but may be not opposite two orthogonal. To provide monotonous implementation of the system of incompletely specified Boolean functions represented by table 2, we need to support opposite two orthogonality of cubes correspond-

ing to the same state and having different characteristics. It demands adding inputs for the sequential circuit. It is desirable to cut the number of additional inputs as much as possible.

1.1. Injecting additional inputs

We suggest the algorithm of finding the minimal number of additional input variables to provide monotonous implementation of the system of incompletely specified Boolean functions, represented by the STG in which the states are encoded with unordered code words (here (m, n) code words).

For that we have to derive a direct graph G from a section of table 2. We include all system cubes corresponding to the same state in the section. The graph vertexes correspond to the cubes of the section of table 2, represented by the first column. Two vertexes are connected with edge if the corresponding cubes u_i, u_j are orthogonal but not opposite two orthogonal. The edge is directed from the vertex having 0-value in the cube u_i component to the vertex having 1 value in the same component of the cube u_j. If there is a cube, that is opposite two orthogonal any cube of table 2, then its corresponding vertex is absent in the graph G.

Two edges are compatible if the end of one edge is not the beginning of another one. A set of edges is compatible if any pair of its edges is compatible. A set of edges is maximal compatible if it is not contained in any compatible set of edges.

Algorithm of finding the minimal number of additional inputs.

1. Find for the graph G all compatible sets deriving sets of capacities 1, 2,…, q.

2. Form maximal compatible sets: S_1, S_2,…, S_t. Note that all sets of the capacity q are among maximal compatible ones.

3. Derive the Boolean matrix. Its lines correspond to the maximal compatible sets and its columns correspond to edges of the graph. Element ij of this matrix is equal to 1 if the maximal set corresponding to the line i contains the edge corresponding to the column j otherwise the element is equal to 0.

4. Find the shortest coverage of the columns by the lines.

The capacity of this *coverage is the minimal number of additional input variables for cubes of the section.*

Note that the intersections of some sets of the coverage may be not empty.

We execute injecting the input variables to the cubes of the section considered in the following way.

5. Let us consider set S_j from the shortest coverage S_1,…, S_t taking into account only the edges that are absent in the sets S_1,…, S_{j-1}. Inject the additional variable x_j and correlate 1 value of this variable to the cubes of vertexes, the edges of which belonging to S_j run from the vertexes and 0 values to the cubes of vertexes the edges of which belonging to S_j come to the vertexes. The rest cubes of the section get don't care value of the variable x_i.

6. Execute step 5 till all sets from the shortest coverage will be examined.

7. Fulfill steps 1-6 for each sections of table 2. Therefore all cubes of the same sections with different characteristics will be opposite two orthogonal.

The shortest coverage of maximal length among all sections is the minimal number of additional inputs of the sequential circuit that we derive.

Illustrate the above-mentioned algorithm for table 2. Here we have very simple graphs.

For the cubes corresponding to the state 1000 we have the graph G_{1000} (Fig. 1b) and the vectors (fig. 1, *a*). The edges (1, 3) and (2, 3) form one compatible set: it is needed one additional input variable x_4 (fig. 1, *c*). For the cubes corresponding to the state 0100 we have the graph G_{0100} (fig. 2, *b*) and the vectors (fig. 2, *a*). The edge (1, 2) forms one compatible set: it is needed one additional input variable x_4 (fig. 2, *c*).

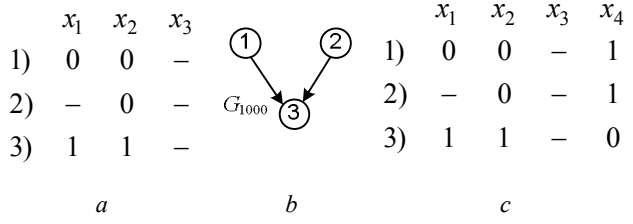

Fig. 1. Example for state 1000 in table 2

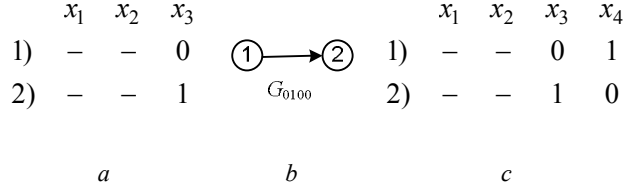

Fig. 2. Example for state 0100 in table 2

For the cubes corresponding to the state 0010 we have the graph G_{0010} (fig. 3, *b*) and the vectors (fig. 3, *a*). The edges form two compatible sets {(2, 1)} and {(1, 3)}: it is needed two additional input variables x_4 and x_5 (fig. 3, *c*). For the cubes corresponding to the state 0001 we have the graph G_{0001} (fig. 4, *b*) and the vectors (fig. 4, *a*). The edge (1, 2) forms one compatible set: it is needed one additional input variable x_4 (fig. 4, *c*).

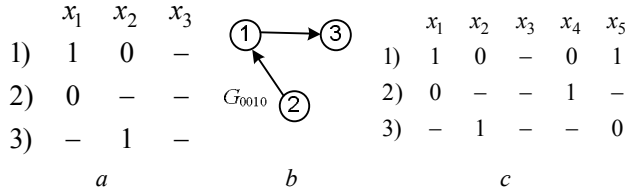

Fig. 3. Example for state 0010 in table 2

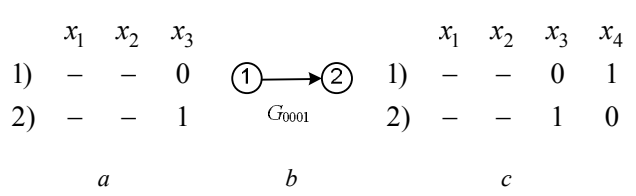

Fig. 4. Example for state 0001in table 2

The system of incompletely specified Boolean functions with additional input variables is represented in table 3. Change 0-values in the first and second columns of Table 3 for don't care executing merged system cubes and get table 4. It is known that table 4 is the implementation of the system of the incompletely specified Boolean functions represented with table 3.

Table 3

The STG description with additional input variables

$x_1\,x_2\,x_3\,x_4\,x_5$	$z_1\,z_2\,z_3\,z_4$	$z_1'\,z_2'\,z_3'\,z_4'$	$y_1\,y_2\,y_3\,y_4\,y_5$
0 - - 1 -	1 0 0 0	1 0 0 0	0 0 0 1 0
- 0 - 1 -	1 0 0 0	1 0 0 0	0 0 0 1 0
1 1 - 0 -	1 0 0 0	0 1 0 0	1 0 0 1 0
- - 0 1 -	0 1 0 0	0 1 0 0	0 0 1 1 0
- - 1 0 -	0 1 0 0	0 0 1 0	1 0 1 1 0
1 0 - 0 1	0 0 1 0	0 0 1 0	0 1 0 0 0
0 - - 1 -	0 0 1 0	0 0 0 1	1 1 0 0 0
- 1 - - 0	0 0 1 0	0 0 0 1	1 1 0 0 0
- - 0 1 -	0 0 0 1	0 0 0 1	0 1 0 0 1
- - 1 0 -	0 0 0 1	1 0 0 0	1 1 0 0 1

Table 4

The STG description with additional variables

$x_1\,x_2\,x_3\,x_4\,x_5$	$z_1\,z_2\,z_3\,z_4$	$z_1'\,z_2'\,z_3'\,z_4'$	$y_1\,y_2\,y_3\,y_4\,y_5$
- - - 1 -	1 - - -	1 0 0 0	0 0 0 1 0
1 1 - - -	1 - - -	0 1 0 0	1 0 0 1 0
- - - 1 -	- 1 - -	0 1 0 0	0 0 1 1 0
- - 1 - -	- 1 - -	0 0 1 0	1 0 1 1 0
1 - - - 1	- - 1 -	0 0 1 0	0 1 0 0 0
- - - 1 -	- - 1 -	0 0 0 1	1 1 0 0 0
- 1 - - -	- - 1 -	0 0 0 1	1 1 0 0 0
- - - 1 -	- - - 1	0 0 0 1	0 1 0 0 1
- - 1 - -	- - - 1	1 0 0 0	1 1 0 0 1

Thus we get the implementation of system $F(x_1,\ldots, x_{n+p})$ ($n(p)$ – the number of input (state) variables) of incompletely specified Boolean functions by a set of the essential cubes of this system. These cubes are represented by the lines of table 4. The completely specified Boolean functions of Table 4 are monotonous.

Table 5

The experimental results

Circuits	***n***	***m***	***s***	***p***	S_1	Σ_1	s_2	δ_1	δ_2	Σ_2
BBTAS	2	2	6	24	3	120	4	2	4	90
BEECOUNT	3	4	7	51	3	152	5	3	3	106
DK27	1	2	7	14	3	56	5	1	2	42
DK512	1	3	15	30	4	150	6	1	2	90
LION	2	1	4	15	2	40	4	2	3	28
LION9	2	1	9	25	4	150	5	2	3	89
MODULO12	1	1	12	24	4	120	6	1	2	72
S8	4	1	5	20	3	140	4	0	4	60
TRAIN11	2	1	11	25	4	150	6	2	3	83

The table 5 represents the experimental results oriented to finding the minimal number of additional inputs for benchmark circuits and simplification of circuit implementation at the expense of using monotonous functions. Here n is the number of primary inputs, m – primary outputs, s – number of states, p – number of products, s_1 and s_2 – number of bits for encoding internal states by the shortest Boolean vectors (s_1) and (m, n)-code words(s_2), δ_1 and δ_2 – number of additional inputs estimated by suggested algorithm (δ_1) and by adding

(m, n)-code words (depending on input variables) to provide opposite two orthogonality for the proper input cubes of the same section (δ_2), Σ_1 – sum of cube (product) ranks of implementation of STG description (system of completely specified Boolean function) after encoding states by the shortest Boolean vectors and without additional inputs, Σ_2 – sum of cube ranks of implementation of STG description suggested above in this paper.

We may use simplification method for this system of monotonous functions proposed by us in [6]. A simplification procedure is based on extension of the essential cubes of system $F(x_1,\ldots, x_{n+p})$ with maximal characteristics. Note that as a result of implementing system $F(x_1,\ldots, x_{n+p})$ of incompletely specified Boolean functions we get system of monotonous functions F^*.

1.2. Deriving circuit C

After getting the monotonous system F^* we suggest using the multilevel logic minimization method based on algebraic division followed covering obtained sum of products (SoPs) with gates [7]. In this case, we must previously transform each SoP of F^* into algebraic SoP in which a mergence of products is executed. Thus we get system F^* of monotonous SoPs. Applying the method [7] to this system we get circuit C that keeps system F^*. This means that when moving from the certain i-th circuit output to its inputs substituting the proper gate functions instead of internal circuit variables and eliminating brackets we obtain the corresponding monotonous SoP f_i^* from system F^*. Note that during substituting functions and eliminating brackets any simplifications are forbidden.

We may also use the special two-level synthesis method, suggested by us before [8]: it also keeps the system F^*.

Two-level synthesis methods are based on choosing a set of product factors and SoPs factors. A product factor of system F^* is derived from the certain product of system F^* by the elimination of some literals from the product. After getting a set of product factors, each system F^* product is covered by the certain factors from this set.

They say that the certain factors cover product K if multiplication of these factors gives rise to all literals in product K. If at least one literal appears two or more times under multiplication then such covering is called redundant otherwise the covering is called irredundant.

A SoP factor of system F^* is derived from the certain SoP of system F^* by elimination of some products from the system SoP. After getting a set of SoPs factors, each SoP of system F^* is covered by the certain SoP factors from this set.

They say that the certain factors cover the SoP f^* of system F^* if a disjunction of these factors gives rise to all products that are present in the SoP f^*. If the product appears two or more times under disjunction then such covering is called redundant otherwise the covering is called irredundant.

Note that a set of product factors and a set of SoP factors are chosen in such a way that each product and each SoP of system $\boldsymbol{F^*}$ has to be covered. As a result of covering, we get combinational circuit C.

We restrict our consideration of two-level synthesis methods in which only irredundant covering products and SoPs of system $\boldsymbol{F^*}$ are acceptable. Call such methods as two-level synthesis methods based on irredundant covering. These methods differ from each other by choosing factors and ways of covering,

Thus using either the multilevel logic minimization method or the two level synthesis method based on irredundant covering we get circuit C keeping system $\boldsymbol{F^*}$ of monotonous functions.

2. Testability properties of circuit *C*

2.1. Testability for multiple stuck-at faults at gate poles of circuit C

Consider single faults of literals of monotonous SoP f_i^* from F^*. Changing the certain literal in the certain product of a SoP for the constant 1 (0) call $b(a)$-fault. If several literals are fault, call it multiple fault of SoP. It is known [9] that all multiple faults of irredundant SoP consisting of prime implicants are detected with a test for single faults of literals of this SoP. This test is union of test patterns for each a-fault and test patterns

for each *b*-fault. Any f_i^* from *F** is irredundant SoP as it consists of monotonous products, and there are no product mergences.

It is also known that each *a*(*b*)-fault in irredundant SoP is detectable [10]. Test pattern α of the monotonous SoP for *a*-fault of product *K* is obtained by changing all don't care components of ternary vector corresponding to *K* for 0 values. Test pattern β for *b*-fault of product *K* and its literal x_i is obtained by changing the component corresponding to the fault literal x_i for 0 value in vector α (in test pattern for *a*-fault).

For example, we have monotonous SoP for state variable z_4' : $x_4z_3 \vee x_2z_3 \vee x_4z_4$.

Consider product $K = x_4z_3$ and literal x_4, then test pattern α for *a*-fault is Boolean vector 00010010.

Test pattern β for *b*-fault is Boolean vector 00000010.

Note that any multiple stuck-at fault at gate poles of the sub-circuit implementing monotonous SoP f_i^* is equivalent to the multiple fault of this SoP literals [5]. Consequently test for single faults of monotonous SoP f_i^* detects any multiple stuck-at fault at gate poles of the corresponding subcircuit from *C*. The length of this test is not more than sum of ranks of monotonous SoP f_i^* products and the number of this SoP products. Thus in order to derive test detecting any multiple stuck-at faults at gate poles of circuit *C* we may merge tests for single faults of literals of all monotonous SoPs of system *F**. Try to cut a length of a test detecting any multiple stuck-at fault at gate poles of circuit *C*.

For that we form SoP *f* including into it all products corresponding to the ternary vectors of Table 4 represented by the first and the second columns. Consider certain product *K* from SoP *f* and certain literal x_i. Find test pattern for *a*(*b*)-fault in above mentioned way. Let product *K* coincides with product *K** of SoP f_i^*, and literal x_i coincides with literal x_i^* from *K**.

Theorem 1. A test pattern for *a*(*b*)-fault of literal x_i from *K* is the test pattern for *a*(*b*)-fault for literal x_i^* of product *K**.

Proof. It follows from the condition: a set of the products of SoP f_i^* is subset of the products of SoP *f*. The theorem is proved.

Corollary 1. Test *T* for single *a*(*b*)-faults of SoP *f* is the test for the same faults of system *F** and, consequently, the test for multiple stuck-at faults at gate poles of circuit *C*.

Corollary 2. The length of test *T* is not more than the sum of ranks of SoP *f* products and the number of these products.

For the system of Table 4 the length of test *T* is not more than 29.

Note that the *T* length may be essentially cut as test patterns for *b*-faults of different products of SoP *f* may coincide [11].

2.2. Full path delay testability of circuit C

We say that circuit is fully delay testable if the delay of each path may be located and measured.

Consider monotonous SoP f_i^* of system *F** corresponding to the one output sub-circuit of *C*. The sub-circuit path α beginning on circuit input x_i and ending on its output originates literal x_{i_α} in the corresponding equivalent normal form (ENF) of the sub-circuit. If we exclude from ENF the sequences of numbers of gates representing the sub-circuit paths we get monotonous SoP f_i^* of system *F**. Consider one product *K** from SoP *f** and literal x_i corresponding to literal x_{i_α} of the ENF. In [12] it is suggested to consider delay fault of path α as a temporary fault of the corresponding literal x_{i_α} of the ENF, and the conditions of detecting robust PDF are formulated with using properties of ENF products. When finding test pair for falling (rising) transition of the path, it is suggested [12] to use test pattern for the corresponding $b_p(a_p)$-fault. This test pattern is vector v_2 of the test pair.

Note that a_p fault means that each appearance of literal x_{i_a} in ENF products is changed for constant 0 and b_p fault means that each appearance of literal x_{i_a} in ENF products is changed for constant 1.

Take into consideration that literal x_i^* corresponding to literal x_{i_a} in the ENF may appear in several products of monotonous SoP f_i^*.

As SoP f_i^* is monotonous and its products have no repeated variables, then for any literal of any product there exists the test pattern both for changing the literal by constant 0 and constant 1.

Consider product ***K**** from f_i^* and its literal x_i^*. Derive vector γ from the ternary vector representing product ***K**** by changing all don't care components for 0.

Theorem 2. Vector γ is the test pattern for the a_p-fault.

Proof. Vector γ turns all products of SoP f_i^* into 0 except product *K** that is it turns fault free SoP into 1. When a_p-fault appears, fault SoP takes 0 value on vector γ. The theorem is proved.

Derive vector δ from vector γ changing its 1 value component x_i^* for 0 value component.

Theorem 3. Vector δ is the test pattern for the b_p-fault.

Proof. Vector δ turns all products of fault free SoP f_i^* into 0. As literal x_i^* disappears in product *K** of fault SoP f_i^*, then the fault SoP takes 1 value on vector δ. The theorem is proved.

The test pattern for the a_p-fault is vector v_2 of the test pair for rising transition of path α. The test pattern for the b_p-fault is vector v_2 of the test pair for the falling transition of path α.

Let u be minimal cube covering vectors γ, δ.

Theorem 4. (δ γ) is the test pair for rising transition of the robust PDF.

Proof. According to Theorem 2, vector γ is vector v_2 for a_p fault. Vector v_1 must have opposite value [12] on variable x_i. This condition is executed. Cube u must be orthogonal [12] all products of SoP f_i^* except ones that contain literal x_i^* corresponding to path α. In this case, u is orthogonal all products of SoP f_i^* except product *K**. Note that the products of SoP f_i^* have no repeated variables. This means that all conditions for robust testable manifestation for rising transition of path α is executed. The theorem is proved.

Theorem 5. (γ δ) is the test pair for falling transition of the robust PDF.

The proof is similar to that of Theorem 4.

Corollary. The sequence (δ γ δ)((γ δ γ)) detects the robust testable PDFs for rising and falling transition of path α.

Thus we Have shown that each PDF of any sub-circuit of circuit *C* is robust testable. Note that test pairs consist of test patterns for $a_p(b_p)$-faults of SoP f_i^*.

Take into consideration that The fully delay and multiple stuck-at fault testability is provided, firstly, by using a description of the sequential circuit behavior with the system of monotonous functions. Secondly, we apply either the multilevel logic minimization method based on algebraic division or the two level synthesis method based on irredundant covering. Note that the conventional two-level synthesis method based on choosing product and SoP factors does not guarantee the fully delay testability property because of masking $a_p(b_p)$-faults.

Using a system of monotonous functions we increase the number of loops and the number of inputs of the sequential circuit but decrease ranks of SoP products.

Conclusion

We have shown that it is possible to derive the sequential circuit from a description of the FSM behavior either by the transition table or by the STG, so that the resulted circuit has short test detecting all multiple stuck-at faults at gate poles of the circuit, and a delay of each circuit path is detectable, i.e. robust testable. For that we need to use unordered code words for symbols of the state alphabet for the STG and symbols of input

and state alphabets for the transition table. It is necessary to add additional inputs when using the STG. As a result we get system of monotonous functions. Then applying either the multilevel logic minimization method based on algebraic division or the two-level synthesis method based on irredundant covering to the system, we derive combinational part *C,* fully testable for multiple stuck-at faults and path delay faults. The method of adding the minimal number of input variables for the STG description is suggested. This approach to the FSM design is connected with increasing of the numbers of inputs and loops of the resulted sequential circuit and decreasing its structural implementation.

REFERENCES

1. *Lin C.J., Reddy S.M.* On Delay fault testing in logic circuits // IEEE Trans. on Computer-Aided Design. 1987. V. 6, No. 5. P. 694–701.
2. *Bushnell M.L., Agrawal V.D.* Essentials of electronictesting for digital, memory and mixed-signal VLSI Circuits. Kluwer Academic Publishers. Boston, Mass, USA, 2000. P. 671.
3. *Matrosova A., Nikolaeva E., Kudin D., Singh V.* PDF testability of the circuits derived by special covering ROBDDs with gates // Proc. of the IEEE East-West Design&Test Symposium. Rostov-on-Don, 2013. P.1–5.
4. *Matrosova A.Yu., Mitrofanov E.V., Singh V.* Delay Testable Sequential Circuit Designs // Proc. of the IEEE East-West Design & Test Symposium (EWDTS 2013). Rostov-on-Don, 2013. P. 293–296.
5. *Matrosova A., Andreeva V., Ostanin S.* Easy testable combinational circuit design // Proc. Of the 6-th International workshop on Boolean problems. Germany. Freiberg, 2004. P. 237–244.
6. *Matrosova A.Yu., Andreeva V.V.* Minimizing the Boolean function system oriented toward self-checking finite-state machine design // J. of Optoelectronics, Instrumentation and Data Processing. 2008. V. 44, Issue 5. P. 459–467.
7. *Murgai R., Brayton R., Sangiovanni-Vincentelli A.* Logic Synthesis for Field Programmable Gate Arrays // Kluwer Academic Publisher. 1995. P. 425.
8. *Matrosova A., Kudin D., Nikolaeva E.* Combinational Circuits without False Paths // Proc. of the IEEE East-West Design&Test Symposium. Kiev. Ukraine, 2014. P. 1–6.
9. *Kohavi I., Kohavi Z.* Detection of multiple faults in combinational logic networks // IEEE Trans. Comput. 1975. VC-20, No. 6. P. 566–568.
10. *Matrosova A.Yu.* A fault-detection method for a synchronous device // Automation and Remote Control. 1978. V. 38, No. 12. P. 1849–1857.
11. *Andreeva V.V., Matrosova A.Yu.* Postroenie minimizirovannogo proveryaushego testa, obnaruzhivayushego neispravnost bezizbitochnoy DNA // Tomsk State University Journal. Supplement. 2006. No. 18. P. 34–39. (In russian)
12. *Matrosova A., Lipsky V., Melnikov A., Singh V.* Path delay faults and ENF // Proc. of the IEEE East-West Design&Test Symposium. St. Perersburg. Russia. 2010. P. 164–1677.

Matrosova Anjela Yurievna, doctor of science, professor. E-mail: mau11@yandex.ru
Ostanin Sergey Alexandrovich, PhD, associate professor. E-mail: sergeiostanin@yandex.ru
Nikolaeva Ekaterina Alexandrovna, PhD. E-mail: nikolaeve-ea@yandex.ru
Kirienko Irina Evgenievna. E-mail: irina.kirienko@sibmail.com
Tomsk State University, Russian Federation.

Поступила в редакцию 25 августа 2015 г.

Матросова А.Ю., Останин С.А., Николаева Е.А., Кириенко И.Е. (Томский государственный университет, Российская Федерация).

Проектирование полностью тестопригодных последовательностных схем для неисправностей задержек путей и кратных константных неисправностей.

Ключевые слова: конечный автомат; граф переходов; неисправности задержек путей; кратные константные неисправности; монотонные функции.

DOI 10.17223/19988605/33/10

При развитии нанотехнологий в электронике недостаточно тестировать одиночные константные неисправности на полюсах элементов логических схем. Необходимо тестировать кратные константные неисправности на полюсах элементов наряду с неисправностями задержек схемы. Одной из часто используемых моделей для тестирования времени задержки является модель неисправностей задержек путей схемы (НЗП). В соответствии со способом проявления неисправностей НЗП разделены на робастно тестируемые неисправности и неробастно тестируемые неисправности. НЗП робастно тестируемы, если проявление неисправности на тестовой паре не зависит от задержек на других путях схемы. НЗП неробастно тестируемы, если проявление неисправности возможно, только когда все другие пути схемы исправны.

Известно, что даже для простых схем сложно перечислить все кратные константные неисправности, и только 20% НЗП являются робастно тестируемыми. Именно поэтому очень важно обеспечить тестопригодность логических схем и для кратных константных неисправностей, и для неисправностей задержек путей на этапе проектирования схемы.

В данной работе мы предлагаем метод синтеза, который позволяет получать последовательностную схему, которая является полностью тестопригодной и для НЗП, и для кратных константных неисправностей.

Было показано, что для заданного описания поведения автомата возможно получить последовательностную схему так, что результирующая схема имеет достаточно короткий тест, обнаруживающий все кратные константные неисправности на полюсах элементов схемы, и НЗП для каждого пути схемы обнаружимы, т.е. являются робастно тестируемыми.

С помощью специального кодирования (неупорядоченными кодовыми словами) внутренних состояний автомата и введением дополнительных входов построим систему монотонных функций, являющуюся одной из реализаций, описывающих поведение автомата. Показано, что применение метода минимизации многоуровневых логических схем, основанного на алгебраическом делении, или метода двухуровневого синтеза, основанного на безызбыточном покрытии системы, позволяет синтезировать комбинационную часть схемы, которая является тестопригодной для заданного класса неисправностей. В работе также предложен метод поиска минимального числа дополнительных входных переменных.

REFERENCES

1. Lin, C.J., Reddy, S.M. (1987) On Delay fault testing in logic circuits. *IEEE Trans. on Computer-Aided Design*. 6(5). pp. 694-701. DOI: 10.1109/TCAD.1987.1270315
2. Bushnell, M.L. & Agrawal, V.D. (2000) *Essentials of electronictesting for digital, memory and mixed-signal VLSI Circuits*. Boston: Kluwer Academic Publishers.
3. Matrosova, A., Nikolaeva, E., Kudin, D., Singh, V. (2013) PDF testability of the circuits derived by special covering ROBDDs with gates. *Proc. of the IEEE East-West Design&Test Symposium*. Rostov-on-Don. pp.1-5.
4. Matrosova, A.Yu., Mitrofanov, E.V. & Singh, V. (2013) Delay Testable Sequential Circuit Designs. *EWD&T'2013. Proc. of the IEEE East-West Design&Test Symposium.* pp. 293-296. DOI: 10.1109/EWDTS.2013.6673138
5. Matrosova, A., Andreeva, V. & Ostanin, S. (2004) Easy testable combinational circuit design. *IWBP'2004. Germany, Freiberg, sept. 2004. Proc. Of the 6-th International workshop on Boolean problems*. pp. 237-244.
6. Matrosova, A.Yu. & Andreeva, V.V. (2008) Minimizing the Boolean function system oriented toward self-checking finite-state machine design. *Journal of Optoelectronics, Instrumentation and Data Processing.* 44(5). pp. 459-467. DOI: 10.3103/S8756699008050117
7. Murgai, R., Brayton, R. & Sangiovanni-Vincentelli, A. (1995) *Logic Synthesis for Field Programmable Gate Arrays*. Boston: Kluwer Academic Publisher.
8. Matrosova, A., Kudin, D. & Nikolaeva, E. (2014) Combinational Circuits without False Paths. *EWD&T'2014. Proc. of the IEEE East-West Design&Test Symposium.* September 2014. Kiev, Ukraine. pp. 1-6.
9. Kohavi, I. & Kohavi, Z. (1975) Detection of multiple faults in combinational logic networks. *IEEE Trans. Comput*. 20(6). pp. 566-568. DOI: 10.1109/TC.1972.5009008
10. Matrosova, A.Yu. (1978) A fault-detection method for a synchronous device. *Automation and Remote Control*. 38(12). pp.1849-1857.
11. Andreeva V.V. & Matrosova A.Yu. (2006) Postroenie minimizirovannogo proveryaushego testa, obnaruzhivayushego neispravnost bezizbitochnoy DNA. *Vestnik Tomskogo gosudarstvennogo universiteta – Tomsk State University Journal*. Supplement. 18. pp. 34-39. (In Russian).
12. Matrosova, A., Lipsky, V., Melnikov, A. & Singh, V. (2010) Path delay faults and ENF. *Proc. of the IEEE East-West Design&Test Symposium*. St. Perersburg. pp. 164-1677.