PAPER Special Section on Test and Verification of VLSIs

Design for Testability Method to Avoid Error Masking of Software-Based Self-Test for Processors

Masato NAKAZATO[†*], **Michiko INOUE**[†a)], **Satoshi OHTAKE**[†], ***Members***, ***and*** **Hideo FUJIWARA**[†], ***Fellow***

SUMMARY In this paper, we propose a design for testability method for test programs of software-based self-test using test program templates. Software-based self-test using templates has a problem of error masking where some faults detected in a test generation for a module are not detected by the test program synthesized from the test. The proposed method achieves 100% template level fault efficiency, that is, it completely avoids the error masking. Moreover, the proposed method has no performance degradation (adds only observation points) and enables at-speed testing.

key words: *software-based self-test, processor, test program template, design for testability, error masking, at-speed testing*

1. Introduction

In recent years, it has been essential that processors with high performance and rich functionality have accurate and at-speed testing. Though the full-scan approach is commonly used due to its simplicity, it induces performance penalty, area overhead and excessive power consumption. The hardware built-in self-test (BIST), which is one of the other widely used techniques, applies pseudo-random test patterns to modules on the circuit at the normal operational speed. However, design modifications are required to make a circuit to be BIST-ready, and involve large amount of manual effort. The BIST also induces area overhead. Furthermore, an application of random patterns results in excessive power consumption.

A number of approaches [1]–[7] have been proposed for software-based self-test (SBST) as a promising approach to resolve the above problems. In SBST, we test a processor by executing a sequence of instructions called a test program. A processor can be tested by communicating with a memory, and thus it enables at-speed testing. We use communication between the memory and an outside ATE as pre- and post-processes of the execution of the test program.

Some methods among the SBST methods generate a test program based on test program templates targeting structural faults to achieve the high fault coverage [1], [2], [5]–[7]. In this approach, gate-level test generation is applied to generate test patterns for each module under test (MUT) of a processor (MUT test generation), and a test program is synthesized from the test patterns (test program synthesis), where a test program justifies the test pattern from the memory to the MUT and propagates the test response from the MUT to the memory. To guarantee the test program synthesis, test program templates are used. A test program template is an instruction sequence with unspecified operands that delivers a test pattern to an MUT and observes the test response. The approach extracts constraints on the input and output of the MUT from each template, and applies test generation for the MUT under the constrains. In this approach, we can easily synthesize a test program from a test pattern for the MUT. However, the justification and observation parts consider only behavior of a fault-free processor and do not consider behavior of a faulty processor, and such parts might not work as expected. In this case, some faults detected by a test pattern for an MUT may not be detected by the synthesized test program. We call such a phenomenon "error masking."

In this paper, we propose a design for testability (DFT) method that completely resolves the problem of error masking for any test program generated by the template-based SBST approach for the stuck-at fault. We show a sufficient condition to avoid error masking and propose the DFT method to make processors which satisfy the sufficient condition. The proposed method has advantages in that (1) it has no performance degradation in a sense that it does not add any gate in a critical path but only observation point, and (2) it enables at-speed testing. In the experimental results, we show the effectiveness of the proposed method on hardware overhead and test application time.

This paper is organized as follows. In Sects. 2 and 3, we show a processor model and test program generation using templates, respectively. In Sect. 4, we analyze error masking and define template level fault efficiency. In Sect. 5, we propose a sufficient condition to avoid error masking. In Sect. 6, we propose a DFT method of SBST for processors and the experimental results are shown in Sect. 7. Finally, the paper is concluded in Sect. 8.

2. Processor Model

A processor is specified by register transfer level (RTL) description. Figure 1 illustrates an example of a processor. A processor consists of combinational modules such as arithmetic logic unit (ALU) or multiplexer (MUX), sequential modules such as a controller, signals, and buses. A signal in an RTL description has a bit width, and is referred to as an RTL signal. A bus considered to be a tri-state bus [8]. For

Manuscript received February 13, 2007.
Manuscript revised August 6, 2007.

[†]The authors are with Nara Institute of Science and Technology (NAIST), Ikoma-shi, 630–0192 Japan.
[*]Presently, the author is with Toshiba Corporation (Semiconductor Company).
a) E-mail: kounoe@is.naist.jp

DOI: 10.1093/ietisy/e91–d.3.763

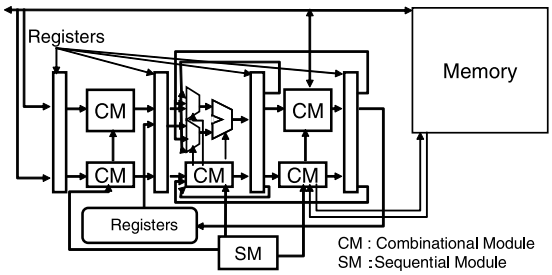


Fig. 1 An example of a processor.

a fault-free processor, we also assume the following about tri-state buses.

(1) Two or more inputs of a tri-state bus are not activated simultaneously.
(2) Each output of the tri-state bus has a masking circuit that generates a logic value ('0' or '1'), and an output of a tri-state bus is masked into some specific logic value if any input of the tri-state bus is not activated.

A processor is assumed to be synthesized while preserving the hierarchy of the modules, and therefore each module can be identified in a gate-level description.

3. Template-Based Test Program Generation

We first explain test program generation using test program templates. In the rest of this paper, we call a test program template a template. Figure 2 illustrates an example of a template, which consists of three sequences: a justification instruction sequence, a test instruction sequence and an observation instruction sequence. A justification instruction sequence is utilized for delivering test patterns to registers which are adjacent to inputs of an MUT. A test instruction sequence applies test patterns to the MUT and propagates the test response to registers that are adjacent to outputs of the MUT or the memory. An observation instruction sequence propagates the test response stored in registers to the memory. For each template, we extract constraints on the input space and the output space of the MUT and perform the MUT test generation under constraints.

Figure 3 illustrates a model of an MUT test generation under constraints extracted from a template. The input and output constraint are extracted from a template, where these constraints represent the relation between operands of a template and inputs of the MUT, and outputs of the MUT and outputs to the memory, respectively, in a fault-free processor. If a processor is fault-free, a test program synthesized from test patterns can justify test patterns of the inputs of the MUT and observe the test responses at some primary output. However, in the case of a faulty processor, the test program might not justify test patterns of the inputs of the MUT or observe the test responses. We call a test program obtained by the test program generation using templates *template-based test program* and, in the rest of the paper, we restrict fault model to a stuck-at fault.

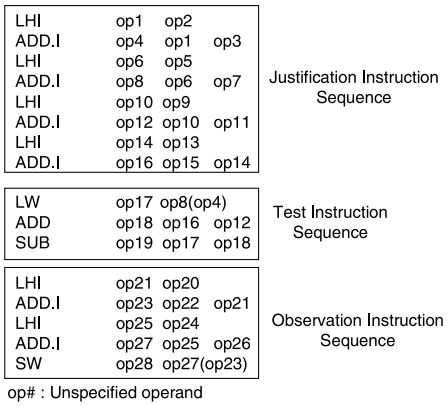


Fig. 2 An example of a template.

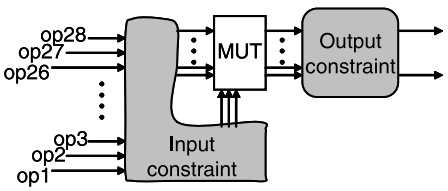


Fig. 3 A model of an MUT test generation.

4. Error Masking

4.1 Template Level Fault Efficiency

In the test program generation method using templates, justification instruction sequences, and observation instruction sequences are generated only in consideration of the behavior of the fault-free processor. When applying a test program to the faulty processor, errors may appear during the execution of these sequences. Therefore, we cannot guarantee that the test program justifies test patterns of the MUT, or observe the test response. This means that some faults detected in the MUT test generation may not be detected by the test program synthesized from the generated test patterns. We call this phenomenon "error masking." In this paper, we define template level fault efficiency (FE_T) as a measure to evaluate error masking as follows:

$$FE_T = \frac{F_{TP}}{F_{MT}},$$

where F_{MT} is the number of faults detected in the MUT test generation and F_{TP} is the number of faults detected by the test program among the F_{MT} faults. A template level fault efficiency of 100% means that there is no error masking.

4.2 Analyzing Error Masking

Figure 4 illustrates examples of error masking using a time frame expansion model of the execution of a test program, where each time frame corresponds to one clock. In the time frame expansion model, time frames that apply test patterns

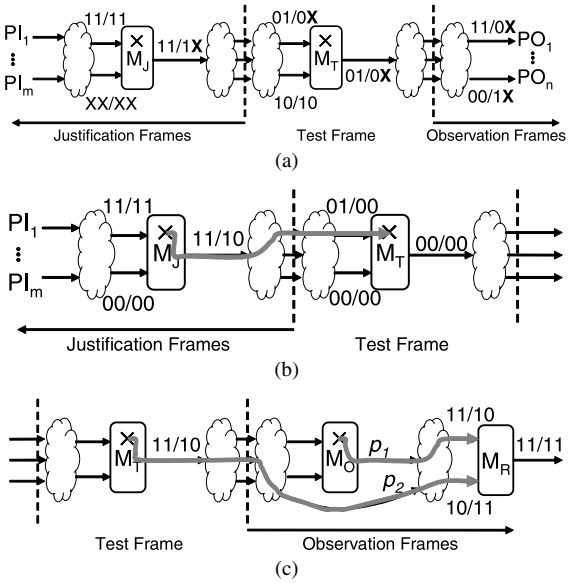


Fig. 4 Examples of error masking:(a) unknown values are propagated to RTL signals; (b) errors reach the MUT; (c) errors are propagated to two RTL signals and meet at some module in some frame.

to the MUT are called "test frame", while time frames before the test frame are called "justification frames" and time frames after the test frame are called "observation frames." Modules M_J, M_T and M_O denote the same MUT which appears repeatedly in different time frames. In this figure, we describe the value of an RTL signal as "the value in fault-free circuit/value in faulty circuit." We consider four values '0', '1', 'X' (uninitialized value) and 'Z' (high-impedance) for each bit of an RTL signal line. We consider 'X' and 'Z' to be unknown value, and an error of an RTL signal line means that at least one bit has different known values.

Figure 4 (a) illustrates an example of error masking induced by an unknown value. In Fig. 4 (a), unknown values reach an input of M_J and logic values reach another input. If M_J operates correctly, unknown values do not appear the output of M_J. However, if M_J operates incorrectly under the effect of the fault, an unknown value is propagated to its output. This causes that value '01/0X' is propagated to an input of M_T at the test frame and fails to excite the fault at the test frame.

Figure 4 (b) illustrates an example of error masking induced by a cycle of an RTL circuit. In this case, there is a cycle including the MUT. In Fig. 4 (b), the fault is excited at M_J in a justification frame and errors appear the output of M_J. The errors reach inputs of M_T in the test frame. A fault of M_T is not excited because incorrect test patterns reach to inputs of M_T.

Figure 4 (c) illustrates an example of error masking induced by a convergence of errors. In Fig. 4 (c), the same fault in M_T and M_O is excited, and the errors are propagated through two paths p_1 and p_2 meet at some module M_R. The multiple errors mask each other in M_R in the observation frame and no error is propagated to the output of M_R. In this case, there are reconvergent paths between the MUT and some module.

5. Sufficient Condition to Avoid Error Masking

In this section, we show a sufficient condition to avoid error masking. In the sufficient condition, we utilize a reconvergent path.

Definition 1 (Reconvergent Path): Let M and M' be modules. Paths p_i and p_j are reconvergent paths from M to M' if both p_i and p_j are paths from M to M', and p_i and p_j share no module except M and M'.

We give the following two assumptions for a test program.

- The value stored in the memory cell referred to during the execution of the test program is known. That is, in the fault-free processor, unknown values are not propagated from the memory to the processor.
- The memory cell written during the execution of the test program is initialized to a different value from the expected value, which will be stored in the memory cell during the execution of the test program.

In this paper, we consider a fault is detected in the following cases.

(A1) No value or an unexpected value is stored in some memory cell where the test program should write some expected value.
(A2) The test program fails to read a value from a memory cell designated to be read in the test program.

The second condition (A2) holds if the memory address lines are observable. However, even if they are not observable, we consider such incorrect behavior would cause some observable errors.

To guarantee that the proposed method can achieve 100% template level fault efficiency, we show a sufficient condition for a processor such that error masking does not occur during the execution of the template-based test program.

Theorem 1: For any template-based test program, error masking does not occur during the execution of the test program if a processor satisfies the following four conditions.

(1) Each register is initialized at the beginning of the execution of the test program.
(2) All the control signals of each tri-state bus and its masking circuits are observable.
(3) For each cycle, at least one RTL signal on the cycle is observable.
(4) For each pair of reconvergent paths, at least one RTL signal on the two paths is observable.

Proof:

Let f be a stuck-at fault detected by an MUT test generation in template-based test program generation. We consider the execution of a test program for f. Let M be a module with f.

First we show that f is detected or a value of any RTL signal of the processor is known. Since all the registers are initialized to known values at the beginning from condition (1), if some signal has an unknown value, it comes from the outside or is generated at the inside of the processor. If f is not detected, (A2) implies that values are read from the same memory addresses in both correct and faulty processors. Since the memory cells where the test program refers to have known values, unknown values are not propagated from the outside of the processor. Moreover, if f is not detected, condition (2) implies that there is no error on control signals or masking circuits of tri-state buses. Therefore, any output of any bus has a value of some activated input of the bus or a known value generated by its masking circuit. Since the value of any RTL signal can be determined by values of primary inputs, registers, and bus outputs, any RTL signal has a known value.

Then we show that f is detected or the test pattern reaches M. We assume that the test pattern of f does not reach the inputs of M. We consider the registers used in order to justify this test pattern in the correct operation of the processor. In this case, there is a bit b of a register among them such that b has a different value from the correct value. If f is not detected, any RTL signals have known values and the value of b is an error. Since an error is only caused by f of M, a path P through b from an output of M to an input of M exists and the error is propagated on P. Since at least one RTL signal on each cycle is observable from condition (3), at least one RTL signal on P is observable and the fault is detected. Therefore, f is detected or the test pattern for f reaches the inputs of M.

Finally, we show that the test response of M is propagated to an intended primary output or f is detected. The output of M can be observed at a primary output in the fault-free processor. Therefore, there exists a path P such that an error is propagated from M to an primary output. Suppose the fault is not detected. In this case, an error is not propagated to any primary output, and there exists a module M' such that the error is prevented from propagating on P. If M' is not faulty and errors are propagated to M' only through P, the errors are propagated to the outputs of M'. Therefore, (a) M' is faulty that is $M = M'$, or (b) errors are propagated to some inputs of M' which are not on P.

(a) If M' is M, errors are propagated on a cycle, and are observed from condition (3) and therefore f is detected.

(b) If errors are propagated to some inputs of M' which is not on P, errors are propagated on two reconvergent paths from M to M'. By condition (4), errors are observed and f is detected.

Therefore, a fault f detected by MUT test generation can be detected during the execution of a test program synthesized from the test pattern for f.

□

6. Design for Testability Avoiding Error Masking of Software-Based Self-Test

6.1 Formulation

We propose a DFT method to avoid error masking. First, we consider the following DFT elements since we add only initialization functions of registers and observable points to the original design in order to satisfy the sufficient condition in Theorem 1.

- Add an initialization function to a register
- Add an observation point to an RTL signal

Since an advantage of SBST is the possibility of at-speed testing, it is important that the processor after DFT also preserves the possibility of at-speed testing. Therefore, we capture the values of RTL signals at the normal operational speed. We use a multiple input signature register (MISR) for this purpose.

In order to satisfy the sufficient condition, it is necessary to add initialization functions to registers which do not have it. Therefore, it is no room to consider an optimization problem for initialization functions. We formulate the problem to minimize the number of observation points as follows.

***Error Masking Resolution Problem*:**

Input: An RTL description of a processor
Output: An RTL description of an augmented processor that can achieve 100% template level fault efficiency for any template-based test program
Objective: To minimize the sum of the bitwidths of RTL signals that are made observable

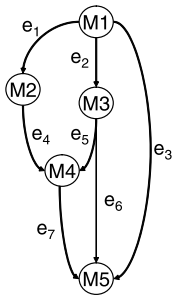
6.2 Algorithm

We propose a heuristic algorithm in order to solve the error masking resolution problem. In the proposed algorithm, we utilize a circuit graph, a reconvergent structure and a path dependency graph.

Definition 2 (Circuit Graph): The circuit graph is a directed graph of an RTL circuit $G_C = (V_{G_C}, E_{G_C})$, where $v \in V$ is a vertex corresponding to a combinational module, a sequential module, a register, a primary input and a primary output and $e \in E_{G_C}$ is an edge corresponding to an RTL signal and has the weight corresponding to the bitwidth of the RTL signal.

Definition 3 (Reconvergent Structure): Let M and M' be modules. A set of all the paths from M to M' is called a reconvergent structure S.

Definition 4 (Path Dependency Graph): Let $V_{ReconvP}$ be a set of paths in all the reconvergent structures. Let $E(p)$ be a set of edges in a path p. A reconvergent path dependency graph is a bipartite graph $G_{PD} = (V_{ReconvP} \cup V_e, E_{G_{PD}})$, where



Path:
p_1: e_1, e_4, e_7
p_2: e_2, e_5, e_7
p_3: e_2, e_6
p_4: e_3
p_5: e_1, e_4
p_6: e_2, e_5
p_7: e_5, e_7
p_8: e_6

Fig. 5 A circuit graph of the reconvergent structure.

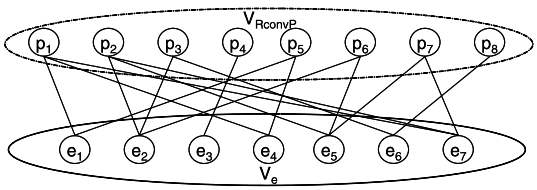


Fig. 6 A path dependency graph.

$V_e = \bigcup_{p \in V_{ReconvP}} E(p)$, and $E_{G_{PD}} = \{(p, e) \mid p \in V_{ReconvP}, e \in E(p)\}$.

Figure 5 illustrates an example of circuit graph of a reconvergent structure and names of paths in it. Figure 6 illustrates a path dependency graph corresponding to the reconvergent structure in Fig. 5. From the path dependency graph, we can identify which paths share an edge.

The proposed algorithm consists of the following four steps.

Step 1: For each register, an initialization function is added if the register does not have the function, and all the control signals of each tri-state bus and all the control signals of its masking circuits are made observable.
Step 2: The circuit graph G_C of the processor is generated.
Step 3: For each cycle in the circuit graph, at least one RTL signal on the cycle is made observable.
Step 4: For each reconvergent path, at least one RTL signal is made observable.

We describe the details the Step 1, Step 3 and Step 4 of the algorithm as follows.

Step 1:

For each register in the processor, an initialization function is added. This initialization function is controllable from a primary input. In general, the processor needs some controls from the outside, and if primary inputs required for this initialization function can be shared with the other controls then it is not necessary to add a new primary input.

Step 3:

In this step, we find a set of RTL signals such that every cycle has at least one RTL signal in the set. We perform the following four steps.

Step 3.1:

We find a cycle C such that the sum of the weight of edges is the minimum by using the minimum cost to profit ratio cycles algorithm in [9]. Then the edge e_i with the minimum weight in C is removed from G_C. This process is repeated until G_C becomes acyclic. For later steps, we store the set C_{min} of selected cycles. All the removed edges are restored to G_C.

Step 3.2:

Let E_{cut} denote a set of edges corresponding to the RTL signals to be observed. We initialize E_{cut} to be empty. From the set of edges in C_{min}, the edge e_i with the minimum weight among the edges that appear in the maximum number of cycles in C_{min} is selected. The edge e_i is removed from G_C and added to E_{cut}, and the cycles that include e_i are removed from C_{min}. This process is repeated until C_{min} becomes empty.

Step 3.3:

Step 3.1 and 3.2 are repeated until G_C becomes acyclic. The circuit graph obtained in this step is referred to as G_C^{α}.

Step 3.4:

If some of the edges in E_{cut} obtained by processing from Step 3.1 to Step 3.3 is restored to G_C^{α}, the circuit graph may not become cyclic. For each $e_c \in E_{cut}$, if the circuit graph becomes acyclic when e_c is added to it, e_c is removed from E_{cut} and e_c is restored to G_C^{α}. The circuit graph obtained in this step is referred to as G_C^{β}.

Step 4:

Let P be a set of paths in all the reconvergent structures in G_C^{β}. We find a set of RTL signals such that every path has at least one RTL signals in the set. We perform the following four steps.

Step 4.1:

We generate the path dependency graph $G_{PD} = (V_{ReconvP} \cup V_e, E_{G_{PD}})$ from all the reconvergent structures in G_C^{β}.

Step 4.2:

For each vertex $v_i \in V_e$ in G_{PD}, we calculate a bit rate R_{v_i}. The bit rate R_{v_i} is obtained by $R_{v_i} = \frac{W_{v_i}}{N_{v_i}}$, where W_{v_i} is the bitwidth of the RTL signal corresponding to v_i and N_{v_i} is the outdegree of v_i in G_{PD}. The bit rate means how may bits needed to make one path observable, and is used to observe more paths by less bitwidths.

Step 4.3:

The vertex $e_i \in V_e$ with the minimum bit rate in G_{PD} is selected, and e_i and its neighbors are removed from G_{PD}. The edge of G_C^{β} which corresponds to e_i is removed from G_C^{β}, and is added to E_{cut}.

Step 4.4:

Steps 4.2 and 4.3 are repeated until the number of paths in each reconvergent structure is less than or equal to one.

As the result of processing these steps, we observe RTL signals corresponding to edges in E_{cut}. These RTL signals are connected to an MISR.

We show an example of the execution of Step 3 and Step 4 by using the circuit graph G_C in Fig. 7. In Step 3.1, we find the set of cycles $C_{min} = \{C_1, C_2, C_3\}$ by removing edges e_6, e_3 and e_5, respectively, where C_1 includes edges (e_2, e_4, e_6), C_2 includes edges (e_2, e_3, e_7, e_1) and C_3 includes

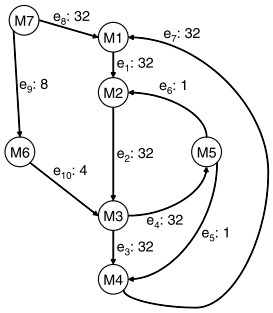


Fig. 7 An example of a circuit graph.

edges $(e_2, e_4, e_5, e_7, e_1)$. In Step 3.2, in the set of edges in C_{min}, the edge with the minimum weight among the edges that appear in the maximum number of cycles is e_2, and e_2 is added to E_{cut} and is removed from G_C. In this case, all the cycles are cut by e_2. Therefore, we complete Step 3. In Step 4.1, we generate the path dependency graph G_{PD} of G_C. In Step 4.2, the vertex with the minimum bit rate is e_5. In Step 4.3, the vertex e_5 is selected, and e_5 and its neighbors are removed from G_{PD}. The edge of G_C which corresponds to e_5 is removed from G_C, and is added to E_{cut}. However, the number of paths in each reconvergent structure is more than one. Therefore, we return to Step 4.2 and repeat similar processes. In this case, the edges e_6 and e_{10} are removed in each process and are added to E_{cut}. Finally, we obtain observation points e_2, e_5, e_6 and e_{10} and the total bitwidth of the observation points is 38.

7. Experimental Results

We evaluate the proposed method using a non-pipelined processor SAYEH [10] and a five-stage pipelined processor Dlx_N that is based on Dlx processor [11]. All the registers of SAYEH processor are resetable. All the registers except for registers of the register-file of Dlx_N processor are resetable.

Table 1 shows hardware overhead of the proposed method and the full-scan design for SAYEH and Dlx_N. In the column "DFT," "FS" and "PM" denote the full-scan design method and the proposed method, respectively. The columns "Area" and "HO" denote the area of the processor and the hardware overhead, respectively. In the columns "Area," "Original" and "Additional" denote the original area of the processor without DFT and the additional area that increases by applying the proposed DFT method to the processor, respectively. A unit of the area is an area for one NOT gate. In the columns "Additional" and "HO", the sub-columns "Init." and "OB" denote the additional area and the hardware overhead of the initialization function and MISR that increase by applying the proposed DFT method to the processor, respectively. In these sub-columns, "-" denotes no additional area or no hardware overhead. The column "#OB" denotes the number of observable bits. In the full-scan design method and the proposed method, an observable bit means the number of scan flip-flops and the number of inputs of MISR.

Table 1 Hardware overhead.

Processor	DFT	Area			#OB	HO(%)	
		Original	Additional				
			Init.	OB		Init.	OB
SAYEH	FS	12389	-	1485	165	-	11.99
	PM		-	2958	102	-	23.88
DLX_N	FS	55995	-	13635	1379	-	23.23
	PM		3968	4379	151	7.09	7.46

There are not any additional area or hardware overhead of the initialization function of SAYEH processor since all the registers are resetable. In the case of the full-scan design method, both processors do not induce the additional area and the hardware overhead of the initialization function. The number of observable bits of the proposed method becomes less than that of the full-scan design method for both processors. For Dlx_N processor, the hardware overhead of the proposed method is smaller than that of the full-scan design method. The Dlx_N processor has many registers including the architecture registers that appear in instruction set architecture and the pipeline registers to enhance the performance. Therefore, the full-scan design induces a large area overhead. For details of the area for Dlx_N processor, the area of the initialization function is almost the same as MISR for observable points. However, if Dlx_N processor has already had the initialization function for all the registers, the area of the initialization function is not required. The area of the initialization function depends on the design specification of the processor. On the other hand, for the SAYEH processor, the hardware overhead of the proposed method is larger than that of the full-scan design method. This is because that the SAYEH processor has a very area-optimized design with a lot of loops and a few registers; therefore, the proposed method needs many observation points whereas full-scan design requires little area overhead. Moreover, the hardware overhead per one observed bit of the proposed method is larger than for the full-scan design. However, this hardware overhead can be reduced if we compress the observed space before applying it to MISR.

In order to show the effectiveness of the proposed method, we apply the proposed DFT method to the arithmetic logic unit (ALU) in Dlx_N processor. Table 2 and Table 3 show the results of the MUT test generation for the ALU in Dlx_N processor and the execution of the template-based test program before/after the proposed DFT method is applied, respectively. We used the SBST method in [7] as an MUT test generation and a test program synthesis.

In Table 2, the columns "Total", "RF", "DF", "FC", "FE" and "TGT" denote the number of total faults of ALU, the number of the identified redundant faults, the number of the detected faults, the fault coverage, the fault efficiency, and the total test generation time for the MUT test generation, respectively. A unit of "TGT" is second. In order to identify redundant faults, we use the method in [7]. The total fault coverage and fault efficiency are 99.83% and 100.00%,

Table 2 MUT test generation for ALU.

Total	RF	DF	FC	FE	TGT (sec)
7030	12	7018	99.83	100.00	358.70

Table 3 Test program execution for ALU.

DFT	DF	EM	FC	FE	FE_T	TAT (clock)
Before	6948	54	98.83	99.00	99.26	7508
After	7022	0	99.89	100.00	100.00	7124

respectively. The total test generation time is 358.70 second. This test generation time is reasonable because the method in [7] generates combinational circuits as constraint circuits, and a combinational test generation is applied.

In Table 3, the columns "DF", "EM", "FC", "FE", "FE_T" and "TAT" denote the number of the detected faults, the number of faults undetected by error masking, the fault coverage, the fault efficiency, the template level fault efficiency and the test application time during the execution of the test program, respectively. A unit of "TAT" is clock. In Table 3, there exist 54 faults undetected by error masking before the proposed DFT method. However, after the DFT method is applied, the number of faults undetected by error masking is 0. The proposed DFT method can achieve 100% template level fault efficiency. The fault coverage after the proposed DFT is larger than that of before the DFT. Moreover, the proposed DFT method can also reduce about 5% of the total test application time.

8. Conclusions and Future Work

In this paper, we showed a sufficient condition to avoid error masking for template-based test programs, and proposed a design for testability method to satisfy the sufficient condition. The experimental results reveal that the proposed method achieves less hardware overhead than full-scan design if the processor features many registers and less loops or reconvergent paths. In general, modern processors oriented to high performance have many registers to accelerate their speed, while the structure tends to be simpler than the design oriented to area optimization. From this observation, we consider that the proposed method is suitable for such modern processors. The proposed method was no performance degradation in a sence that it adds only observation points to the original design and moreover, it enables at-speed testing. The reduction of the hardware overhead caused by the DFT method is the issue to be investigated in our future work.

Acknowledgements

Authors would like to thank Dr. Tomokazu Yoneda of Nara Institute Science and Technology for his useful comments. This research was supported in part by 21st Century Center of Excellence Program "Ubiquitous Networked Media Computing" and Japan Society for the Promotion of Science (JSPS) under Grants-in-Aid for Scientific Research B (No.15300018) and for Scientific Research C (No.18500038).

References

[1] L. Chen and S. Dey, "Software-based self-testing methodology for processor cores," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.20, no.3, pp.369–380, March 2001.
[2] L. Chen, S. Rabi, A. Raghunath, and S. Dey, "A scalable software-based self-test methodology for programmable processors," Proc. Design Automation Conference, pp.548–553, 2003.
[3] W.C. Lai, A. Krstic, and K.T. Cheng, "Test program synthesis for path delay faults in microprocessor cores," Proc. International Test Conference, pp.1080–1089, 2000.
[4] W.C. Lai, A. Krstic, and K.T. Cheng, "Instruction-level DFT for testing processor and IP cores in system-on-a chip," Proc. Design Automation Conference, pp.59–64, 2001.
[5] M. Inoue, K. Kambe, N. Hoashi, and H. Fujiwara, "Instruction-based self-test for sequential modules in processors," Proc. IEEE 5th Workshop on RTL and High Level Testing, pp.109–114, 2004.
[6] K. Kambe, M. Inoue, and H. Fujiwara, "Efficient template generation for instruction-based self-test," Proc. IEEE 13th Asian Test Symposium, pp.152–157, 2004.
[7] M. Inoue, M. Nakazato, S. Yokoyama, K. Kambe, and H. Fujiwara, "Efficient and effective test program generation for software-based self-test of pipelined processors," NAIST Technical Reports, no.2006005, Aug. 2006.
[8] Semiconductor Technology Academic Research Center (STARC), RTL design style guide, STARC, 2000. (In Japanese).
[9] K. Mehlhorn and S. Näher, LEDA, Cambridge university press, 1999.
[10] Z. Navabi, VHDL Analysis and Modeling of Digital Systems, McGraw-Hill, 1997.
[11] J.H. Hennesy and D.A. Patterson, Computer Architecture: A Quantative Approach, Morgan Kaufmann Publishers, 1996.

Masato Nakazato received the B.E. degree in Photonics from Ritsumeikan University, Shiga, Japan, in 2001, and the M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology, Nara, Japan, in 2005 and 2007, respectively. Presently he is an engineer with Toshiba Corporation (Semiconductor Company). His research interests are fault analysis, VLSI CAD, fault diagnosing technique, test generation for logic circuits and synthesis for testability.

Michiko Inoue received her B.E., M.E, and Ph.D. degrees in computer science from Osaka University in 1987, 1989, and 1995 respectively. She worked at Fujitsu Laboratories Ltd. from 1989 to 1991. She is an associate professor of Graduate School of Information Science, Nara institute of Science and Technology (NAIST). Her research interests include distributed algorithms, parallel algorithms, graph theory and design and test of digital systems. She is a member of IEEE, the Information Processing Society of Japan, and Japanese Society for Artificial Intelligence.

Satoshi Ohtake received the B.E. degree in computer science from the University of Electro-Communications, Tokyo, Japan, in 1995 and the M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology, Nara, Japan, in 1997 and 1999, respectively. He was a Research Fellow of the Japan Society for the Promotion of Science from 1998 to 1999. Presently he is an Assistant Professor at the Graduate School of Information Science, Nara Institute of Science and Technology. His research interests are VLSI CAD, design for testability, and test pattern generation. He is a member of IEEE Computer Society and IPSJ.

Hideo Fujiwara received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. He was with Osaka University from 1974 to 1985 and Meiji University from 1985 to 1993, and joined Nara Institute of Science and Technology in 1993. Presently he is a Professor at the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. His research interests are logic design, digital systems design and test, VLSI CAD and fault tolerant computing, including high-level/logic synthesis for testability, test synthesis, design for testability, built-in self-test, test pattern generation, parallel processing, and computational complexity. He is the author of Logic Testing and Design for Testability (MIT Press, 1985). He received many awards including Okawa Prize for Publication, IEEE CS (Computer Society) Meritorious Service Awards, IEEE CS Continuing Service Award, and IEEE CS Outstanding Contribution Award. He served as an Editor and Associate Editors of several journals, including the IEEE Trans. on Computers, and Journal of Electronic Testing: Theory and Application, and several guest editors of special issues of IEICE Transactions of Information and Systems. Dr. Fujiwara is a fellow of the IEEE, a Golden Core member of the IEEE Computer Society, and a fellow of the IPSJ.