Test Synthesis for Strongly Testable Datapaths using Datapath-Controller Functions*

Michiko Inoue Kazuhiro Suzuki† Hiroyuki Okamoto‡
Hideo Fujiwara
Graduate School of Information Science
Nara Institute of Science and Technology

Abstract

This paper proposes a test syntheses method for datapaths. The proposed method goes on design-for-testability while generating control sequences for justification and propagation at register-transfer level. Since the method fully utilizes functions of controllers as well as datapaths, it achieves small area overhead.

Keywords: *design-for-testability, RTL circuit, at-speed testing, hierarchical test generation, non-scan design.*

1. Introduction

Standard full scan design has the advantages of high fault coverage and short test generation time for stuck-at faults, while having many problems of large test data, long test application time, large hardware overhead and difficulty of at-speed testing. High level test synthesis is one approach to resolve these problems. This includes hierarchical test generation, register-transfer level (RTL) design-for-testability (DFT) and high level synthesis for testability. It has potentiality to reduce test cost drastically by utilization of high level information or abstraction.

This paper simultaneously considers DFT and test generation at RTL. Test generation is based on hierarchical test generation[1] which combines test generation for combinational modules at gate level and search for paths to control and observe the modules at RTL. The proposed method finds control and observation paths for each module while adding some DFT elements if it is difficult to establish such paths. There are some related works. Makris et al.[2] proposed an RTL analysis methodology that identifies the test justification and propagation bottlenecks, and Ghosh, Jha et al.,[3, 4, 5, 6] proposed a test synthesis method considering both test generation and DFT.

In the methods proposed by Ghosh and Jha et. al, control/data flows at normal mode are used for testing. They analyze the control/data flows and use them to justify test patterns to modules and to propagate the responses. If adequate paths cannot be found, they add extra MUXs and paths from the primary inputs to the MUXs to increase controllability. The methods enables all the operational modules to be tested hierarchically with small area overhead.

Our group proposed another test synthesis methods based on *strong testability* [7, 8, 9, 10, 11]. This testability guarantees *all the modules* including multiplexors(MUXs) in a datapath to be tested hierarchically. This robust property is necessary to become an alternative to the scan design. We derived *test plans*, which are sequences of control signals that form control and observation paths for modules under test. We have an *isolation* approach and an *integration* approach. In the isolation approach, a datapath and a controller are considered independently, and test plans are provided from a test plan generator embedded for *at-speed testing*[7, 10]. We proposed another DFT method[8] for controllers that achieves 100% fault efficiency. Combination of these methods[9] achieve 100% fault efficiency for RTL circuits. Some part of our methods were further improved after the technology transfer to industry[12, 13]. In the integration approach, we utilize function of a controller to provide the test plans. Nagai et. al[11] proposed a method that provides the test plans using the original function of the controller.

In this work, we propose another method for the integration approach. We augment functions of a controller so that it can provide test plans as well as control signals for normal operations. For a datapath, we also augment functions if necessary. This method can enhance testability with small cost, since the method fully utilizes functions of controllers as well as datapaths. We propose a branch-and-bound algorithm considering DFT overhead for both controller and datapath. The algorithm finds control and

*This work is supported in part by Japan Society for the Promotion of Science (JSPS) under the Grant-in-Aid for Science Research No.14658092.

†Currently, SANYO Electric

‡Currently, Matsushita Electric Industrial

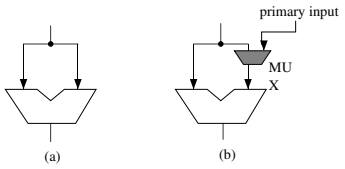

Figure 1. (a)violation,(b)preprocessing.

observation paths for all the modules with small overhead. In addition, our method has many advantages of test synthesis methods based on the strong testability. We can use a combinational ATPG and hence achieve high fault coverage and complete fault efficiency. Test application time is much shorter than scan design since we do not need scan shifting, and furthermore, we can apply at-speed testing. Though our method augments functions of controllers, we can apply the DFT method [9] for controllers after applying our method, and we can achieve 100% fault efficiency for RTL circuits.

2. Preliminary

We consider a register-transfer level(RTL) circuit composed of a controller and a datapath. They are connected by control signal lines from a controller to a datapath and status signal lines from a datapath to a controller. A controller is described by a finite state machine (FSM) and a datapath is composed by the following hardware elements and data signal lines connecting them. Hardware elements include primary inputs(PIs), primary outputs(POs), registers, modules. Modules are operational modules or multiplexors(MUXs).

In this paper, we assume the following restriction.

1. All the signal lines have the same bit width.
2. Each module has a data output port or status signal outputs. If a module has a data output, it is reachable to some primary output.
3. A controller has a reset state reachable from all the states.
4. For each module M, there are paths $P_1, P_2, \cdots, P_k$ from primary inputs to data input ports $i_1, i_2, \cdots, i_k$ such that any pair of P_i, P_j $(i \neq j)$ are disjoint or there is a register appearing in only one of these paths.

The last condition is required to guarantee justification of independent values at different (data) input ports of the same module without adding any path. This is violated, for example, if there is reconvergent paths without registers like Fig1(a). In such a case, we modify a given circuit in preprocessing, for example, like Fig1(b).

We propose a test synthesis method for datapaths. Our method is based on hierarchical test generation[1]. In the

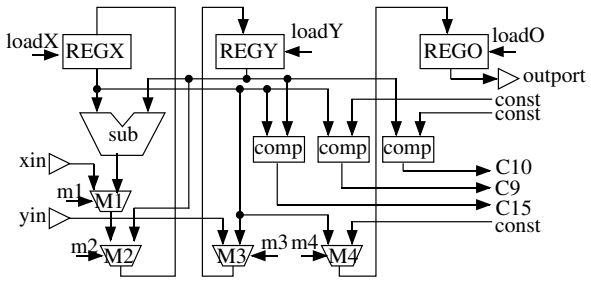

Figure 2. A datapath of GCD.

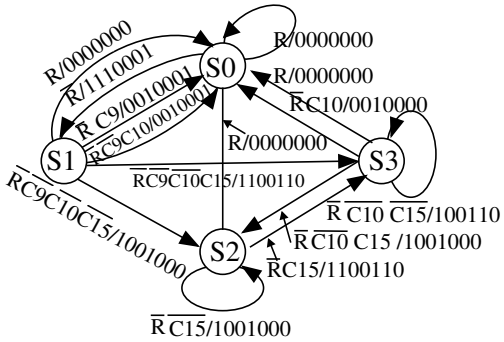

Figure 3. A controller of GCD.

hierarchical test generation, we first generate tests for each module using a combinational ATPG, then find *a test plan* at an RTL. A test plan is a sequence of control vectors that specifies *control paths* which justify the obtained tests to the module from primary inputs or control signal lines and *observation paths* which propagate the test responses to primary outputs or status signal lines.

We consider a circuit GCD as an example. Figures 2 and 3 show its datapath and controller, respectively. There are three status signal lines $C9, C10, C15$ and one primary input R of a controller. Figure4 shows control and observation paths where the circuits are expanded to some time frames, where bold lines represent the control and observation paths for a MUX $M1$ at the second time frame. In the control paths, the left input ports of $M1$ is controlled directly from a primary input xin and the right input ports is controlled from sub whose input ports are further controlled by paths $xin, M1, M2, REGX$ and $yin, M3, REGY$. The observation path is from $M1$ at the second time frame to the primary output at the 5th time frame where a dotted line means the register holds the value in that time frame. In the first time frame, the left inputs of $M1, M2, M3$ are selected and registers $REGX, REGY$ load values. These are controlled by a control vector. The test plan is a sequence of such control vectors for these 5 time frames.

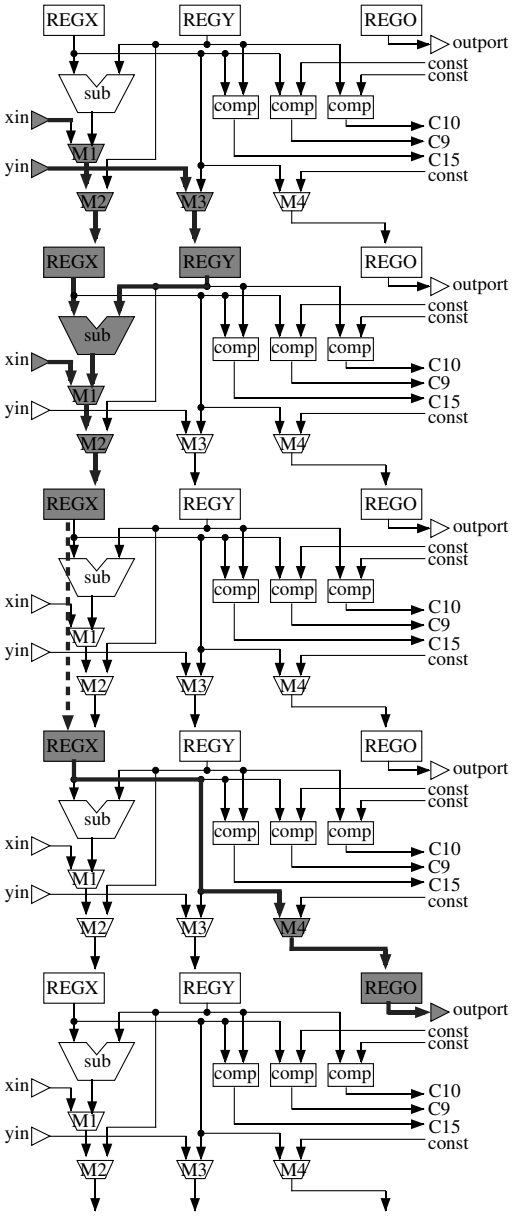

Figure 4. Control and observation paths.

In [7], we defined *strong testability* of a datapath as testability for hierarchical test for combinational modules.

Definition 1 ([7]) *A datapath is strongly testable iff there exists a test plan for each module M that makes it possible to apply any pattern to M and to observe any response of M.*

A strongly testable datapath has the following advantages. 1) Test pattern generation time is short since a combinational ATPG can be used, and 2) complete (100%) fault efficiency is achieved for the whole datapath since each module M is a small combinational circuit.

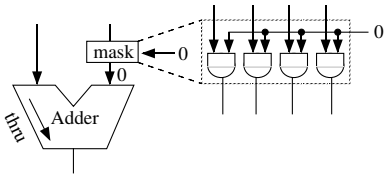

Figure 5. A mask element.

3. Test synthesis using datapath-controller functions

3.1 Problem

We consider test plan generation and DFT at the same time. We augment functions of an original controller so as to provide test plans while making a datapath strongly testable. We consider the following test synthesis problem.
Input: an RTL circuit (a datapath and a controller)
Output: an augmented RTL circuit (a strongly testable datapath and a controller capable of providing test plans for each module) and test plans
Objective: minimization of area overhead

We can augment functions of a controller by adding transitions, states, and primary inputs. For a datapath, we add hold function to registers or thru function to modules. Thru function of one input port of a module propagate any value of the input port to its output port without modification. Most of the popular operational modules (e.g. adder) can realize the thru function by using a mask element which generates a constant. Figure5 shows a mask element for an adder.

3.2 Test synthesis for datapaths

3.2.1 Idea

We consider how to augment functions of a controller and a datapath so that it can provide test plans like Fig4. Here we consider GCD as an example. Figure3 shows a Mealy FSM of its controller. An output value (i.e. a control vector) attached to each transition specifies values of $loadX$, $loadY$, $loadO$, $m4$, $m3$, $m2$, $m1$ in this order. Let (s, s', in, out) denote a transition from a state s to s' with an input in and an output out. In the first time frame in Fig4, the left inputs of $M1$, $M2$ and $M3$ are selected and $REGX$ and $REGY$ load values. This can be represented as a test vector "11x000x" where x means "don't care". If the original controller has a transition that outputs a control vector compatible with this, we may be able to use the transition. For example, a transition $(s0, s1, x, 1110001)$ has a compatible output with "11x000x". However, we have two more problems. First, we must generate a sequence

of control vector provided from a controller. Therefore, if we adopt $(s0, s1, x, 1110001)$, the following transition must start from $s1$. In our method, we check current and next states as well as output control signals, and if there is no compatible transition we add a new transition to the controller. Second, some transition depends on status signals, and it is difficult to control the status signals while generating test plans. To resolve this, we add transitions controlled from outside (additional primary inputs) and independent of status signals. We also augment functions of a datapath. For example, if it is difficult to control all the input ports of some operational module M, we add thru function to one input port i of M and try to control only i.

Any augmentation requires some area overhead, and our objective is to find the augmentation with the minimum area overhead. For a datapath, we can easily estimate area overhead required to augment thru function and hold function. For a controller, we introduce a cost function to estimate such area overhead. We propose a branch-and-bound algorithm that implicitly searches any possible sequences of transitions and finds the sequence that provides a test plan with the minimum augmentation.

3.2.2 Cost function for controllers

In our method, we do not consider any additional transition requires the same area overhead. For example, consider the case where we add a transition $(S1, S2, 000, 000000)$. If the original controller already has a similar transition $(S1, S2, 000, 100000)$, we can consider this augmentation requires small area overhead. We first define a distance between two transitions and define the augmentation cost for a transition t as the minimum distance between t and all the existing transitions.

First, we consider a distance of two transitions $(s1, s1', in1, out1)$ and $(s2, s2', in2, out2)$. Each transition (s, s', in, out) requires a logic that generates the output (s' and out) from the input (s and in). We estimate a difference between two logics by the difference between the input and the output of these logics. We treat x ("don't care ") in the different ways for the inputs and the outputs. If an input has x, it means that this transition occurs for both 0 and 1. On the other hand, x in an output means this transition generates 0 or 1. Figure6 shows distance functions of one bit d_{in} for inputs (Fig6(a)) and d_{out} for outputs (Fig6(b)).

We can naturally expands these distance functions for multi-bit. Let $b_1 b_2 \cdots b_n$ and $b'_1 b'_2 \cdots b'_n$ be vectors of n bits. We define $d_{in}(b_1 b_2 \cdots b_n, b'_1 b'_2 \cdots b'_n) = \sum_i d_{in}(b_i, b'_i)$ and $d_{out}(b_1 b_2 \cdots b_n, b'_1 b'_2 \cdots b'_n) = \sum_i d_{out}(b_i, b'_i)$.

For states, we estimate the difference at the worst case. Let $|S|$ be the number of states. In logic level, $\log |S|$ bit value is assigned to each state. Therefore, we estimate a difference between two difference states to be $\log |S|$.

	0	1	x
0	0	1	1
1	1	0	1
x	1	1	0

(a)

	0	1	x
0	0	1	0
1	1	0	0
x	0	0	0

(b)

Figure 6. Distance functions: (a)d_{in}, (b)d_{out}.

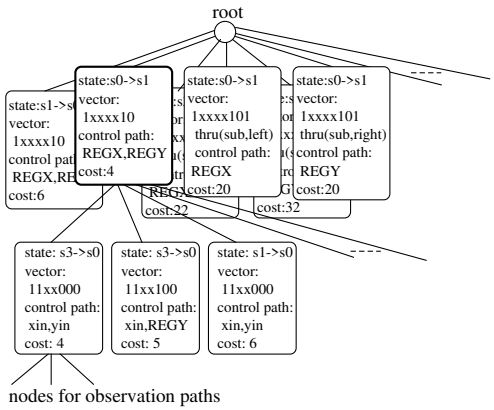

Figure 7. Branch-and-bound search

We define the cost function $cost(t, S, T)$ to add a transition t to a controller with a state set S and a transition set T as follows.

$$cost(t, S, T) = min_{t_e \in T} dist(t, t_e)$$
$$dist(t(= (s, s', in, out)), t_e(= (s_e, s'_e, in_e, out_e)))$$
$$= d_{in}(in, in_e) + d_{out}(out, our_e)$$
$$+ \log |S|(diff(s, s_e) + diff(s', s'_e))$$
$$diff(s, s_e) = \begin{cases} 1 & \text{if } s \neq s_e \\ 0 & \text{if } s = s_e \end{cases}$$

For example, consider a cost to add a transition $t = (s3, s0, TEST \cdot \overline{R}, 11x000x)$ to a controller in Fig3. A distance between an existing transition $t_e = (s3, s0, \overline{R} \cdot C10, 0010000)$ is as follows. To add t, we add a new primary input $TEST$, and therefore, the input of t_e becomes $\overline{TEST} \cdot \overline{R} \cdot C10$. Since values unspecified in the inputs are "don't care", $dist(t, t_e) = d_{in}(10xxx, 00x1x) + d_{out}(11x000x, 0010000) + \log 4(diff(s3, s3) + diff(s0, s0)) = 2 + 2 + 0 + 0 = 4$. Since this is the minimum distance, the cost for t is 4.

3.2.3 Branch-and-bound search

We propose a branch-and-bound search algorithm that generates test plans for modules while augmenting functions of both controller and datapath. We will explain how to generate a test plan for one module. The search starts with a

tree consisting of one root node. We start the search from the time frame where a test pattern is applied (e.g. the 2nd time frame in Fig4). We consider all possible control and observation paths for this time frame. In the case of $M1$ in Fig4, we can consider three cases for control paths: one uses both $REGX$ and $REGY$, the other two use $REGX$ or $REGY$ and thru function of sub. In addition, we consider every combination of the current and the next states. For each combination of these paths and these states for one time frame corresponds to a transition. Therefore, we can calculate the cost to add the transition. We assign one node for each of these transitions, and make it the child of the root. Then we expand nodes according to a branch-and-bound fashion. That is, in each stage of the search, we select the node such that the total cost from the root to the node is minimum, and generate the child nodes. We first generate nodes corresponding to the preceding transition of control paths as child nodes until all the control paths reach primary inputs, then generate the nodes corresponding to the succeeding transition of observation paths. Once we reach a node where observation path reaches a primary output, the path from the root to the node represents one candidate of a test plan. We then prune any node such that the total cost from the root to the node exceeds the cost of the obtained candidate. Figure7 shows a concept of the search tree, and we omit the detail of the algorithm.

We generate test plans for modules one by one. If the target module has control signals, we generate test plans for each value of the control signal. For example, we generate two test plans for one MUX that apply 0 and 1 for a test pattern to the MUX. In such a case, we must apply the search algorithm twice. However, the second search finishes very fast, since our branch-and-bound algorithm searches nodes with low cost first and the test plans obtained at the first search can almost be reused with no cost.

3.2.4 Reduction of the number of test pins

In general, there is a restriction of the number of I/O pins, and therefore, we may have a constraint on the number of additional I/O pins for test (test pins). Though our method requires new test pins to control additional transitions, we can avoid the increase of test pins. Figure8 illustrates the idea to keep the number of test pins. In Fig8(a), we consider to add new transition from $s0$ to $s3$ and need an additional test pin since two transitions are already added for test. We avoid the increase by adding a new state $s0'$ and transitions like Fig8(b), where a transition from $s0$ to $s0'$ outputs the control vectors to hold all the registers. The above technique guarantees that our method is available if at least one additional I/O pin can be used.

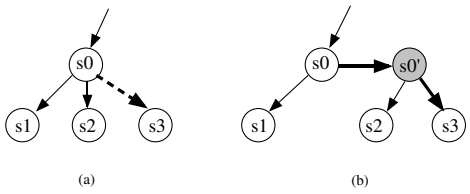

Figure 8. Reduction of the number of test pins.

Table 2. Hardware overhead

circuit	area overhead(%)						#test pin	
	our method + controller DFT[8]					full scan	ours	full scan
	C	DP	C'	MUX	total			
GCD	9.2	0	1.1	7.5	17.8	39.7	4	3
JWF	3.4	2.9	0.4	4.7	11.4	28.4	4	3
LWF	2.2	6.1	0.8	4.0	14.1	42.3	3	3
Paulin	0.6	2.2	0.2	0.6	3.7	6.3	4	3

4. Experiments

We applied the proposed method to 4 benchmark circuits GCD, 4th Jaumann Wave Filter(JWF), 3rd Lattice Wave Filter(LWF) and Paulin. We compared the results with full scan design reported in [8]. Table1 shows the characteristic of these circuits. The columns #state, #status, #control, #PI, #PO, $|$bit$|$, #Reg. and #Mod. denote the numbers of states, status signal lines, control signal lines, primary inputs and primary outputs, the bit-width of a datapath, the numbers of registers and modules, respectively. The experiments used a logic synthesis tool AutoLogicII(Mentor Graphics) and an ATPG tool TestGen(Synopsys) on SunBlade1000 (Sun Microsystems). The columns #gate denote the areas of circuits in gates after logic synthesis.

Table2 shows hardware overhead, where C and DP denote area overhead of augmentation for controllers and datapaths, respectively. After the test synthesis for datapaths, we apply the DFT method for controllers proposed in [9]. The column C' denote area overhead for this controller DFT, and MUX denote area overhead for MUXs to isolate controllers from datapath including MUXs to bring the values of status signals to the primary output of datapaths. The combination of two methods enables test generation using combinational ATPG for the whole RTL circuits.

For these circuits, our method achieves small area overhead, especially for Paulin. We analyzed the result for Paulin more, and found that our method added 7 thru functions to the datapath: one for control paths and 6 for observation paths. In our branch-and-bound algorithm, control paths are searched from the target module to primary inputs, and therefore, we can avoid the augmentation of thru

Table 1. Circuit characteristics

	controller				datapath						total
circuit	#state	#status	#control	#gate	#PI	#PO	bit	#Reg.	#Mod.	#gate	#gate
GCD	4	3	7	169	32	16	16	3	1	1350	1524
JWF	8	0	38	199	80	80	16	14	3	6671	6875
LWF	4	0	8	57	32	32	16	5	3	1924	1986
Paulin	6	0	16	123	64	64	32	7	4	24833	24965

Table 3. Test generation

	our method + controller DFT[8]				full scan	
circuit	C-ATPG (sec)	test plan gen. (sec)	fault eff. (%)	test appl. (cycle)	test gen. (sec)	test appl. (cycle)
GCD	0.70	3.71	100.0	774	171.51	6629
JWF	0.54	156.62	100.0	2779	2.88	20519
LWF	0.51	0.96	100.0	394	0.47	4066
Paulin	1.80	252.82	100.0	2703	4.68	16187

function to some module by controlling the all input ports of the module. On the other hand, we search observation paths from the target module to primary output. In this case, we add thru function to propagate the test response from an input port to an output port of a module. This implies that there is still room for improvement in the search algorithm.

Table3 shows the result of test generation and test plan generation by our method and full scan design. We generated tests for both datapaths and controllers. Since our method enables us to apply combinational ATPG for each module, time for combinational ATPG is short while achieving 100% fault efficiency. Test application time is much shorter than full scan design. However, for some circuits, our search algorithm consumes time. This is because our algorithm implicitly searches all possible paths.

5. Conclusions

We proposed a test synthesis method that simultaneously considers DFT and test generation for RTL circuits. The proposed method utilizes functions of controller and datapath as much as possible. Therefore, we can achieve 100% fault efficiency with small area overhead.

References

[1] B. T. Murray and J. P. Hayes, "Hierarchical test generation using precomputed tests for modules," *IEEE Trans. on Computer Aided Design*, Vol. 9, pp. 594–603, June 1990.

[2] Y. Makris and A. Orailoğlu, "RTL test justification and propagation analysis for modular designs," *Journal of Electronic Testing: Theory and Applications (JETTA)*, Vol. 13, pp. 105–120, Oct. 1999.

[3] I. Ghosh, A. Raghunathan, and N. K. Jha, "Design for hierarchical testability of RTL circuits obtained by behavioral synthesis," *IEEE Trans. on Computer Aided Design*, Vol. 16, pp. 1001–1014, April 1997.

[4] I. Ghosh, A. Raghunathan, and N. K. Jha, "A design for testability technique for register-transfer level circuits using control/data flow extraction," *IEEE Trans. on Computer Aided Design*, Vol. 17, pp. 706–723, Aug. 1998.

[5] I. Ghosh, A. Raghunathan, and N. K. Jha, "Hierarchical test generation and design for testability methods for ASPPs and ASIPs," *IEEE Trans. on Computer Aided Design*, Vol. 18, pp. 357–370, March 1999.

[6] S. Ravi, G. Lakshminarayana, and N. K. Jha, "TAO: Regular expression based register-transfer level testability analysis and optimization," *IEEE Trans. on VLSI Systems*, Vol. 9, pp. 824–832, Dec. 2001.

[7] H. Wada, T. Masuzawa, K. K. Saluja, and H. Fujiwara, "Design for strong testability of RTL data paths to provide complete fault efficiency," *Proc. of 13th International Conference on VLSI Design*, 2000, pp. 300–305.

[8] S. Ohtake, T. Masuzawa, and H. Fujiwara, "A non-scan approach to DFT for controllers achieving 100% fault efficiency," *Journal of Electronic Testing: Theory and Applications (JETTA)*, Vol. 16, pp. 553–566, Oct. 2000.

[9] S. Ohtake, H. Wada, T. Masuzawa, and H. Fujiwara, "A non-scan DFT method at register-transfer level to achieve complete fault efficiency," *Proc. of Asia and South Pacific Design Automation Conference 2000 (ASP-DAC2000)*, 2000, pp. 599–604.

[10] S. Ohtake, S. Nagai, H. Wada, and H. Fujiwara, "A DFT method for RTL circuits to achieve complete fault efficiency based on fixed-control testability," *Proc. of Asia and South Pacific Design Automation Conference 2001(ASP-DAC2001)*, 2001, pp. 331–334.

[11] S. Nagai, S. Ohtake, and H. Fujiwara, "A method of design for hierarchical testability for data flow intensive circuits at register-transfer level," *Journal of Information Processing Society of Japan*, Vol. 43, pp. 1278–1289, May. 2002.

[12] T. Hosokawa, H. Date, and M. Muraoka, "Test generation method using a compacted test plan table for RTL data path circuites," *Proc. of 20th VLSI test symposium*, 2002, pp. 328–335.

[13] H. Date, T. Hosokawa, and M. Muraoka, "A SoC test strategy based on a non-scan DFT method," *Proc. of 11th Asian Test Symposium*, 2002, pp. 305–310.