IEEE the 14th Asian Test Symposium (ATS'05), pp. 306-311, Dec. 2005.

A DFT Method for RTL Data Paths Based on Partially Strong Testability to Guarantee Complete Fault Efficiency

Hiroyuki Iwata , Tomokazu Yoneda , Satoshi Ohtake , Hideo Fujiwara
Graduate School of Information Science Nara Institute of Science and Technology
Kansai Science City 630-0192, Japan
Email:{*hiroyu-i,yoneda,ohtake,fujiwara*}*@is.naist.jp*

Abstract

This paper presents a non-scan design-for-testability (DFT) method that guarantees complete fault efficiency (FE) for register transfer level (RTL) data paths. We first define the ***partially strong testability*** *as a characteristic of data paths. Then we propose a DFT method to make a data path partially strongly testable and a test generation method for partially strong testable data paths based on the time expansion model (TEM). The proposed DFT method can reduce hardware overhead drastically compared with the previous method based on strong testability. Moreover, the proposed DFT method can generate test patterns with complete FE in practical time and allow at-speed test.*

key words: design-for-testability, data paths, strong testability, partially strong testability, complete fault efficiency

1. Introduction

With the progress of semiconductor technology, testing of VLSI becomes more difficult and the cost is increasing. Therefore, it is important to achieve high FE with low cost.

For combinational circuits, test patterns with 100% FE can be obtained by an automatic test pattern generator (ATPG) [1]. For sequential circuits, the test generation can be modeled by an iterative combinational arrays [1] so that a combinational test generation method can be used. However, test generation for sequential circuits is more complex than that for combinational circuits because of the number of time frames needed for the justification and the error propagation. In the worst case, the number of time frames is the exponential function of the number of FFs. To ease the complexity of the test generation, DFT techniques have been proposed.

The most widely used DFT techniques for sequential circuits is the full scan approach [1, 2]. In the full scan approach, test generation algorithm for combinational circuits can be applied. Therefore, this approach can achieve 100% FE. However, it requires large hardware overhead and can not allow at-speed test.

To avoid these disadvantages, non-scan DFT methods have been proposed [3, 4, 5]. A DFT method for RTL data paths based on strong testability [4] can achieve 100% FE and allows at-speed test. Moreover, compared with the full scan design, it achieves shorter test application time and lower hardware overhead. This method is based on hierarchical test generation [6] that consists of the following two steps. First, test patterns for each hardware element are generated. Then, a test plan (control vector sequences) for each hardware element is generated so that any test pattern can be propagated from primary inputs to its inputs and test responses can be propagated from its outputs to a primary output. Strong testability can be satisfied by adding thru function to all operational modules and hold function to some registers. Thru function propagates the value of the input of the hardware element to the output of the hardware element without changing the value. Hold function holds the value of the register. However, we can propagate any test pattern to each hardware element even if thru function is not added to all operational modules. Moreover, it is not necessary to propagate any test pattern to each hardware element. We can test each hardware element by applying all the values that appear in normal operation to the inputs of the hardware element.

In this paper, we introduce partially strong testability of a data path and a DFT method for making a data path partially strongly testable. Partially strong testability guarantees that for every signal line l, any value that appears at l in normal operation can be set to l while strong testability guarantees that every value whether it appears in normal operation or not can be set to l. In the proposed DFT method, only part of the hardware elements are augmented with DFT elements in order to make a data path partially strongly testable. Therefore, the hardware overhead is drastically lower than that required strong testability. Moreover, we propose a test generation method for partially strongly testable data paths. In the proposed method, a time expansion model (TEM) [7] is extended for partially strongly testable data paths. The number of time frames

needed for justification and error propagation is the sum of a sequential depth from a primary input to a primary output along a simple path and a sequential depth of a loop. Therefore, the method can achieve 100% FE in practical test generation time by using a combinational ATPG. Furthermore, the proposed method can achieve shorter test application time than the method based on strong testability and allow at-speed test.

2. Preliminary

2.1. Data Paths

In RTL description, a VLSI circuit generally consists of a controller and a data path. In this paper, a data path and a controller are separated and only a data path is the object of testing. A data path consists of hardware elements and signal lines. Hardware elements are primary input (PI), primary output (PO), hold register (HR), load register (LR), multiplexer (MUX), operational module (CM) and observational module (OM). Inputs of a hardware element are classified into data inputs and control inputs. Similarly, outputs of a hardware element are classified into data outputs and status outputs. A data input is connected with a data output of the other hardware element by a signal line. A control input is fed the signal from a controller. A status outputs feed the signal to a controller. We assume that control inputs are controllable and status outputs are observable. Moreover, we assume that all data inputs and outputs of hardware elements have the same bit width as a restriction for a data path.

Each hardware element has at most two data inputs, at most one control input, at most one status output, at most one data outputs. We classify CM into two types : CMA and CMB. CMA is a operational module that can be set any value to the data output by controlling the data inputs. CMB is a operational module except for CMA. MUXs, CMs and OMs are called combinational hardware elements.

2.2. Strong Testability

Definition 1 (Strong Testability [4]) *A data path is strongly testable iff there exists a test plan for each hardware element m that makes it possible to apply any pattern to m and to observe any response of m.*

If a data path *DP* is strongly testable, test patterns with 100% FE for m in *DP* generated by using a test generation algorithm for combinational circuits can be applied the test patterns from PIs to m and the responses can be observed from m to a PO.

Let m be a hardware element in a data path, let d_{mi} be the number of time frames required for propagation of the test patterns along a simple path from a PI to m, let d_{mo} be the number of time frames required for propagation of the responses along a simple path from m to a PO. In the DFT method based on strong testability, the length of a test plan for m is $d_{mi}+d_{mo}+1$.

3. Partially Strong Testability

In this paper, we define partially strong testability to apply combinational ATPG to a TEM of a data path and guarantee that the number of time frames of the TEM is about the same as the length of a test plan of strong testability. An example of partially strongly testable data path and a TEM is shown in Figure 1.

Definition 2 (The Range of a Signal Line) *Let R_l be a set of values that can appear at a signal line l by controlling PIs. Then, R_l is called the range of line l. The range of the inputs/outputs of hardware elements connected with line l is defined as the range of line l.*

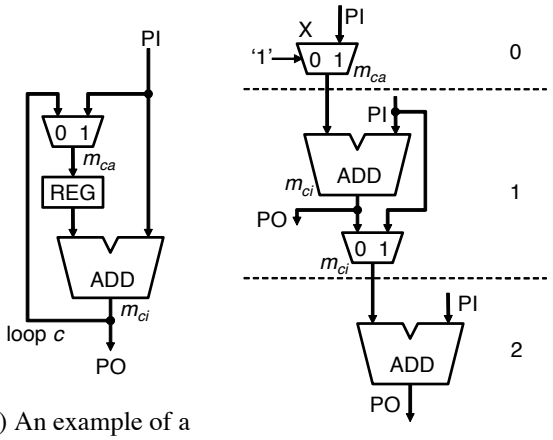
Definition 3 (Dependency) *Let R_{li} and R_{lj} be the range of a signal line l_i and l_j, respectively. There exists a dependency between l_i and l_j if l_i and l_j can not be simultaneously set to any value in R_{li} and R_{lj}, respectively. A dependency of the inputs/outputs of hardware elements connected with l_i and l_j is defined as the dependency between l_i and l_j.*

Definition 4 (Partially Strong Testability) *Let L be a set of loops in a data path and let c be a loop in L. Let M_c be a set of CMs with two data inputs and MUXs on c. A data path DP is partially strongly testable iff L satisfies the following conditions.*

1. *The data output of each hardware element $m_{ca} \in M_c$ can be set to any value along a simple path.*
2. *Let m_{ci} be a hardware element in M_c Let d_c be the number of time frames required for propagation of the values from m_{ca} to m_{ci}. There is no dependency between a data output of m_a at time t and a data input on c of m_{ci} at time $t+d$.*

Theorem 1 *If DP is partially strongly testable, any detectable fault in DP are detectable in a TEM for partially strong testable data path and the number of time frames of the TEM is the maximum $d_{mi}+d_{mo}+1+n_{mCM}$. Let n_{mCM} be the number of CMs with two data inputs exist on a simple path from a PI to a PO via m.*

Proof: If *DP* is an acyclic structure, it is proved that any detectable fault in *DP* are detectable in a TEM by [8]. By condition 1 and 2 of partially strongly testability, any value in normal operation can be propagated to any hardware element on any loop by performing time expansion once along the loop. Moreover, any value in normal operation can be applied from PIs to any hardware element m by d_{mi} frames and any value in normal operation can be propagated from m to a PO by d_{mo} frames. Then, at most n_{mCM} frames are needed so that there cannot exist a dependency between data inputs of CMs on a simple path via m. Therefore, if there are loops in *DP* and *DP* is partially strongly testable, any detectable fault in *DP* are detectable because a TEM of the *DP* can be treated as acyclic structure. The number of time frames of a TEM is the maximum $d_{mi}+d_{mo}+1+n_{mCM}$.



(a) An example of a partially strongly testable data path

(b) An example of a TEM for partially strongly testable data path

Figure 1. Partially strong testability

4. DFT for Partially Strong Testability

4.1. Problem Formulation

A given data path can be made partially strongly testable by adding thru function and hold function. We first formulate a DFT problem for making a data path partially strong testable as the following optimization problem.

Definition 5 (DFT for Partially Strong Testability)
input: *a data path*
output: *a partially strongly testable data path and a TEM of a partially strongly testable data path*
Optimization: *minimizing hardware overhead*

For CMs such as adders and multiplexers, we realize thru function by using a mask element instead of MUX because the former requires lower area.

4.2. Overview of DFT Algorithm

This subsection describes the overview of the proposed heuristic DFT method. The DFT algorithm consists of the following three phases.

1. Construct a control forest: To satisfy the condition 1 of partially strong testability, we construct a set of simple paths which guarantee to propagate any value in the range of each signal line. The simple path is called a control path and a set of the paths is called a control forest. Thru function is added to CMs to propagate any value.
2. Resolve dependencies: To satisfy the condition 2 of partially strong testability, we find whether there exists a dependency between data inputs of every hardware element. Hold function is added to LRs to resolve a dependency.
3. Generate a time expansion model: For test generation, we generate a TEM. Hold function is added to LRs so that any detectable fault in a data path can be detected in only one TEM.

4.2.1. Control Forest Construction In the proposed method, in order to reduce the CMs to be added thru function, if there are paths such that any value can be propagated along the paths without adding thru function, the paths are included in a control forest as much as possible. A control forest is constructed in order to decrease the dependencies of the control paths as much as possible and the ratio for the paths whose range is different from the range of a PI used as a control forest is the minimal.

We select the hardware elements and the paths from PI as the control forest. Let m be a hardware element, let m_t be a hardware element which is connected with a data output of m and not yet selected from m, and let n_m be the number of m_t. Let C_C be a set of hardware elements such that the paths connected with all the data inputs of the hardware element are selected and there exists the path that is connected with a data output of the hardware element and is not yet selected. Let C_M be a set of hardware elements with the minimum n_m in C_C. Let C_{REG} be a set of registers such that the path connected with a data inputs of the register is selected and all the path connected with a data output of the register are not yet selected. Let C_{CM} be a set of CMs such that at least one path connected with the data inputs of the CM is selected and all the path connected with a data output of the CM are not yet selected. The steps for constructing a control forest are shown as follows.

1. C_C is initialized to a set of PIs. C_{REG},C_{CM} are set to empty. All the hardware elements and the paths are not selected.
2. This step is repeatedly performed until C_M is empty. C_M is initialized to a set of hardware elements with the minimum n_m in C_C. We select $m \in C_M$ and delete m from C_M. Then, if there is no m_t, m is deleted from C_C. We select m_t and the path between m and m_t and operate as follows.
   - If m_t is a MUX and m_t is selected in the first time, m_t is added to C_C.
   - If m_t is a HR or a LR, m_t is added to C_{REG}.
   - If m_t is a CMA, all the paths connected with data inputs of m_t are selected and m_t is never added to C_C, m_t is added to C_C and deleted from C_{CM} iff m_t exists in C_{CM}.
   - If m_t is a CMA which does not satisfy above condition or m_t is a CMB, m_t is added to C_{CM}.

   The selection from m is continuously performed until m_t which is never selected is selected.
3. If C_C is not empty, the process returns to step 2.
4. If C_{REG} is not empty, all the registers in C_{REG} is added to C_C and deleted from C_{REG} and the process returns to step 2.
5. If there exists a CMA $m \in C_{CM}$, m is added to C_C and deleted from C_{CM} and the process returns to step 2. Then, if there exists no CMA in C_{CM} and there exists

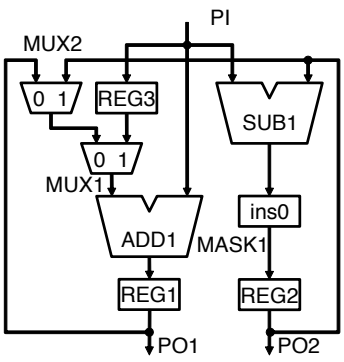


Figure 2. An example of a data path

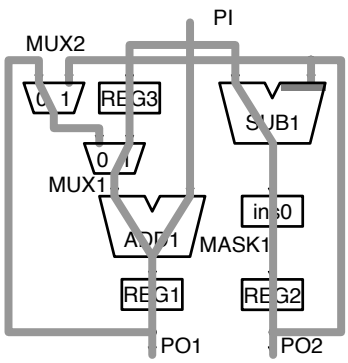


Figure 3. An example of a control forest

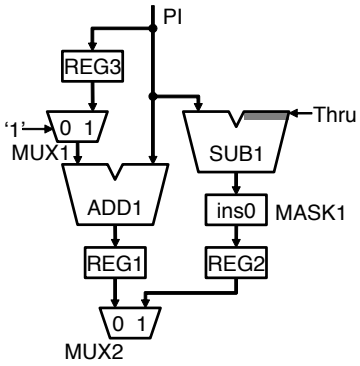


Figure 4. Control paths for MUX2

a CMB $m \in C_{CM}$, m is added to C_C and deleted from C_{CM} and it returns to step 2.

6. Let m be a hardware element, let $in1_m$ be the data input of m connected with the path selected in the first time and let out_m be a data output of m. If m satisfies at least one of following conditions, thru function that propagates from $in1_m$ to out_m is added to m.
   - There exists no register on the re-convergent path in a control forest which re-converge with m. And m exists on a loop or a loop exists on the path connected with the output of m.
   - On the step 5, m is a CMA included in C_{CM}.
   - m is a CMB. And m exists on a loop or a loop exists on the path connected with the output of m.

An example of a control forest for Figure 2 is shown in Figure 3. In the example of Figure 2, after step 1-4, the hardware elements and paths except SUB1 or subsequent ones are selected. In step 6, thru function is added to SUB1.

4.2.2. Dependency Resolution It is assumed that there exists a dependency between two data inputs of a hardware elements if the control paths for the hardware element are a re-convergent path in a control forest which re-converge with the hardware element and the sequential depths of the re-convergence path are same.

Let m_c be a hardware element such that there is a dependency between two data inputs of m_c. If m_c exists on a loop or there exist HRs on the control paths from m_c to PIs, the dependency needs to be resolved. To resolve the dependency, the hold time of the registers adjacent to one data input of m_c is figured out. If there exists the LRs in the registers, hold function is added to the LRs.

An example of control paths is shown in Figure 4. There exist control paths with the sequential depth 1 or 2 from the PI to the left data input of MUX2. There exists a control path with the sequential depth 1 from the PI to the right data input of MUX2. Therefore, there exists the dependency between the two data inputs of MUX2. The dependency must be resolved because MUX2 exists on a loop. REG1 is holded one cycle so that the dependency can be resolved.

4.2.3. Time Expansion Model Generation A TEM is used for test generation of partially strongly testable data path. In the proposed method, a hold function is added of the process of a TEM generation so that only one TEM can be generated.

Let t be the current time frame number of a TEM. Let C_O,C_{REG1}.C_{REG2} be a set of hardware elements, respectively. The steps for a TEM generation are shown as follows.

1. C_O is initialized to a PO. C_{REG1},C_{REG2} are set to empty. t is set to zero.
2. This step is repeatedly performed until C_O is empty. A hardware element $m_d \in C_O$ is deleted from C_O and m_d is added to the TEM at t. Then, we operate as follows. If m_d is added to the TEM in the first time, hardware elements connected with all the data inputs of m_d are added to C_O. If m_d is added to the TEM in the second time or more, hardware elements connected to m_d on the control forest are added to C_O. If m_d is a HR or a LR, m_d is added to C_{REG1}. If there exists the dependency between the two data inputs of m_d, the registers are holded according to the already figured out in 4.2.2 so that the dependency can be resolved. The signal lines between m_d and the hardware elements added to C_O are added to the TEM. Moreover, if the signal lines connected with POs, the POs are added to the TEM.
3. If C_{REG1} is not empty, the registers in C_{REG1} which are loaded at t are added to C_{REG2}. Let M_D be a set of CMs on the paths from the registers in C_{REG2} to POs in the TEM. If there is the dependency between two data inputs of $m \in M_D$ and m exists on a loop, the dependency is resolved in order to guarantee observation of the values. The registers which are reached from a data input of m in the TEM are holded so that the dependency can been resolved and deleted from C_{REG2}. If

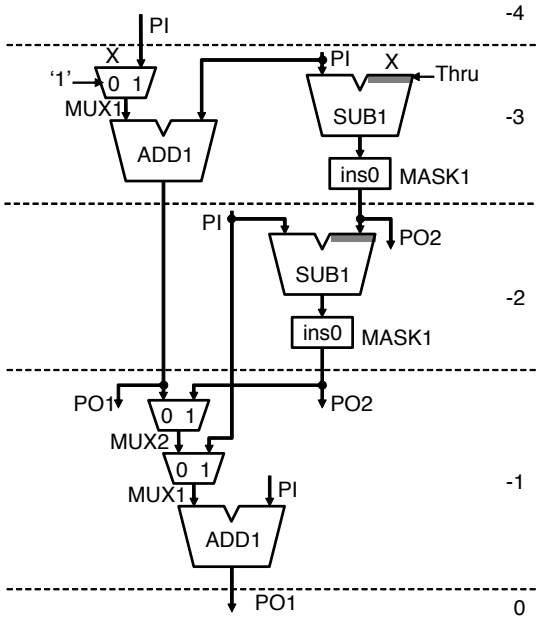


Figure 5. An example of a TEM

there exists LRs in the register, hold function is added to the LRs. The hardware elements connected with the data inputs of the registers in C_{REG2} are added to C_O and the signal lines between the registers and the hardware elements are added to the TEM. The registers in C_{REG2} are deleted from C_{REG1} and C_{REG2}. Then, the t is decremented one and the process returns to step 2.

4. One of the POs and OMs which are not added to the TEM is added to C_O. Then, the process returns to step 2 as t is the adequate time not to appear combinational hardware elements redundantly as possible.

An example of a TEM for Figure 2 is shown in Figure 5. In Step 1, C_O is initialized to PO1. In Step 2, there exists the dependency between the two data inputs of MUX1 and MUX2. Then, REG1 is holded one cycle at -1. In Step 5, the operation from PO2 is not performed because PO2 appears at $-1, -2$.

The hardware elements with the data input connected with no hardware element in the TEM have thru function. The control inputs of the hardware elements are controlled according to the control forest and the data inputs are set to unspecified values. A combinational ATPG is applied to a generated TEM using the above-mentioned restrictions. The generated test patterns are transformed so that the test patterns can be applied to the original data path.

5. Experimental Results

We evaluate the effectiveness of the proposed method by experiments. RTL benchmark circuits used for the experiments are GCD, LWF, JWF and PAULIN, which are popularly used circuits [4, 5], and RISC and MPEG, which are more practical and larger circuits designed by a semiconductor company [4, 5]. Circuit characteristics of these circuits are shown in Table 1. Column "bit" denotes the bit width of the circuits. Column "#PI", "#PO", "#Reg", "#MUX", "#CM" and "#OM" denote the numbers of primary inputs, primary outputs, registers, multiplexeres, operational modules and observational modules, respectively. In our experiments, we used AutoLogicII (MentorGraphics) as a logic synthesis tool with its sample libraries to synthesize those circuits. In Table 1, column "Area" denotes the total circuit size after synthesis.

We used TestGen (Synopsis) as a sequential/combinational ATPG tool on Sun Blade 2000 (Sun Microsystems). Test generation for sequential circuits using a TEM requires a combinational ATPG which can deal with multiple stuck-at faults. In this experiments, since TestGen can not deal with multiple stuck-at fault, we use the circuit model which can express multiple stuck-at faults in a time expansion model as single stuck-at fault [9].

The results of the hardware overhead are shown in Table 2 and the results of test generation are shown in Table 3. Column "org", "FS", "ST" and "PST" denote results of the original circuits (without DFT), of the circuits modified by the full-scan design, of the circuits modified by the method of [4] and of the circuits modified by the proposed method, respectively. The value in parentheses in column "Fault efficiency" shows the fault coverage. Column "Test generation time" denotes the time spent on the ATPG and does not include the time spent on the DFT since the time spent on the DFT is negligible as compared with the time spent on the ATPG. The test application time of the full scan designs are calculated by assuming the case where these circuits have single scan chain.

The proposed method can always achieve drastically lower hardware overhead than both "FS" and "ST" can. Especially for GCD, LWF, JWF and RISC, there is no need for DFT since the original circuits satisfies the partially strong testability as they are.

We observe that the test generation time of FS, ST and PST which can achieve 100% FE are much shorter than that of org. Moreover, PST can achieve almost same test generation time as ST which is based on hierarchical test generation. We consider that faults were efficiently detected by the fault simulation in PST since the whole circuit is the target for test generation in PST. However, PST requires a little bit long test generation time compared to FS. This is because PST uses a TEM for test generation.

In the result of test application time, PST can achieve shorter test application time than ST because faults were efficiently detected by the fault simulation in PST. Moreover, PST can achieve much shorter test application time than FS. This is because PST does not require scan-shift oper-

Table 1. Data paths characteristics

Circuits	bit	#PI	#PO	#Reg	#MUX	#CM	#OM	area
GCD	16	2	1	3	3	2	3	1010.7
LWF	16	2	2	5	5	3	0	1364.7
JWF	16	4	5	14	25	3	0	4208.3
Paulin	16	2	2	7	8	7	0	4329.0
Tseng	16	3	2	6	5	9	0	3163.9
Risc	32	1	3	40	84	20	3	39834.8
MPEG	8	56	148	241	207	161	0	48709.7

Table 2. Hardware Overheads.

Circuits	Hardware overhead[%]		
	FS	ST	PST
GCD	16.5	3.8	0.0
LWF	20.4	9.6	0.0
JWF	18.5	3.1	0.0
Paulin	9.0	3.5	1.8
Tseng	10.5	8.4	2.4
Risc	16.9	7.1	0.0
Mpeg	14.0	8.0	0.8

Table 3. Test generation results.

Circuits	Fault efficiency[%] (Fault coverage)				Test generation time[sec]				Test application time[clock]			
	org	FS	ST	PST	org	FS	ST	PST	org	FS	ST	PST
GCD	100.00 (100.00)	100.00 (100.00)	100.00 (100.00)	100.00 (100.00)	3.18	0.20	0.83	0.39	133	2351	387	197
LWF	100.00 (99.90)	100.00 (99.90)	100.00 (100.00)	100.00 (99.90)	0.33	0.21	0.62	0.24	56	2674	250	70
JWF	100.00 (99.95)	100.00 (99.95)	100.00 (100.00)	100.00 (99.95)	3.63	0.60	0.62	1.74	244	14849	769	608
Paulin	99.23 (99.20)	100.00 (100.00)	100.00 (100.00)	100.00 (100.00)	297.50	0.83	0.86	4.69	147	2824	875	526
Tseng	99.01 (98.96)	100.00 (100.00)	100.00 (100.00)	100.00 (100.00)	703.90	0.34	1.08	0.87	590	4752	633	523
Risc	99.37 (99.34)	100.00 (99.97)	100.00 (100.00)	100.00 (99.69)	12210.81	55.17	76.56	60.02	2271	621284	5520	3420
Mpeg	88.30 (87.16)	100.00 (99.45)	100.00 (99.51)	100.00 (99.45)	68947.90	1.84	1.18	13.34	1216	185183	107359	8448

ations. Furthermore, FS cannot allow at-speed test while other approaches can.

6. Conclusion

In this paper, we defined partially strong testability as a characteristic of a data path. We also proposed a DFT method for RTL data paths based on partially strong testability. The proposed method achieves 100% FE within practical test generation time by using combinational ATPG. It also allows at-speed testing. Furthermore, the hardware overhead required by the proposed method is drastically lower and the test application time is shorter compared with the method for strong testability [4].

Acknowledgments

The authors would like to thank Prof. Michiko Inoue of Nara Institute of Science and Technology for her valuable discussions. This work was supported in part by 21st Century COE Program and in part by Japan Society for the Promotion of Science (JSPS) under Grants-in-Aid for Scientific Research B(2)(No. 15300018).

References

[1] H. Fujiwara, *Logic Testing and Design for Testability*, The MIT press, 1985.

[2] M. Abramovici, M. A. Breuer and A.D.Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[3] R. B. Norwood and E. J. McCluskey, "Orthogonal scan:Low overhead scan for data Paths," *Proc.1996 Int.Test Conf.*,pp.659-668(1996)

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

[5] S. Ohtake, S. Nagai, H. Wada and H. Fujiwara,"A DFT method for RTL circuits to achieve complete fault efficiency based on fixed-control testabillity," *Proc. of ASP-DAC*, pp.331-334, 2001.

[6] J. Lee and J.H. Patel, "Hierarchical test generation under architectural level functional constraints," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1144-1151, Sept. 1996.

[7] T. Inoue, T. Hosokawa, T. Mihara, and H. Fujiwara, "An optimal time expansion model based on combinational ATPG for RT level ciruits," *Proc. IEEE the 7th Asian Test Symp.*, pp. 190-197, Dec. 1998.

[8] T. Inoue, D. K. Das, T. Mihara, C. Sano and H. Fujiwara, "Test generation for acyclic sequential circuits with hold registers," *International Conference on Computer Aided Design*, pp.550-556, Nov. 2000.

[9] H. Ichihara, and T. Inoue, "A method of test generation for acyclic sequential circuits using single stuck-at fault combinational ATPG," *IEICE Trans. Fundamentals*, Vol. E86-A, No. 12, pp. 3072-3078, Dec. 2003.