INFORMATION
SCIENCE
TECHNICAL
REPORT

NAIST-IS-TR98013
ISSN 0919-9527

New DFT Techniques of Non-Scan Sequential Circuits with Complete Fault Efficiency

Debesh Kumar Das, Satoshi Ohtake
and Hideo Fujiwara

November 1998

NAIST

〒 630-0101
奈良県生駒市高山町 8916-5
奈良先端科学技術大学院大学
情報科学研究科

Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan

New DFT Techniques of Non-Scan Sequential Circuits with Complete Fault Efficiency

Debesh Kumar Das[1]
Dept. of Comp. Sc. and Engg.
Jadavpur University
Calcutta-700 032, India
debeshd@hotmail.com

Satoshi Ohtake Hideo Fujiwara
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5, Takayama-Cho, Ikoma, Nara 630-0101, Japan
{satosi-o, fujiwara}@is.aist-nara.ac.jp

Abstract: As opposed to scan schemes, non-scan DFT allows at-speed testing. This paper suggests three techniques on non-scan DFT of sequential circuits. The novelty of the proposed techniques is that by using combinational ATPG tool 100% fault efficiency is guaranteed. Test sequences are generated from test patterns obtained by ATPG tool on the combinational part of the sequential machine. In all techniques, an additional circuit to reach invalid states (CRIS) is proposed to reach unreachable states on the state register of a machine. The second and third technique use an additional hardware called differentiating logic (DL), which uniquely identifies a state appearing in a state register. The design of this DL is universal, i.e., not dependent on the circuit structure. Hardware overhead of DL and CRIS is lower than that of full scan. Test generation and test application time are also found to compare favorably with those of earlier designs.

Keywords: sequential circuits, DFT, non-scan, complete fault efficiency, at-speed testing

[1] presently visiting in Nara Institute of Science and Technology, Japan

1. Introduction

Testing of VLSI circuits is very difficult in case of sequential circuits. To ease this problem, scientists often proposed design for testability (DFT) techniques. The main purpose of these DFT techniques is to append an additional hardware with the original circuit with the goal to reduce the test complexity. As the testing time for a circuit includes both test generation and test application time, designers attempt to decrease the both. While designing DFT, the designers should also be conscious of the quality of testing, which is often measured by *fault efficiency*. The ratio of number of faults detected or proved redundant by a test algorithm to the total number of faults in a circuit is known as the fault efficiency. Moreover, as ac testing is an important issue nowadays, at-speed testing is a basic requirement. So a good DFT design must have following goals – 1) to decrease test generation time, 2) to decrease test application time, 3) to have high fault efficiency, and 4) to achieve at-speed testing. Furthermore, hardware overhead should also be optimum in the designs.

The main reason for the difficulty in generating tests for sequential circuits is the poor controllability and observability of memory elements of these circuits. The oldest and most commonly used DFT technique to improve this controllability and observability is the full scan method [1,2] where all the flip-flops are arranged in a chain and their values are scanned in (out) before (after) each test. The technique reduces the test generation problem of a sequential circuit to that of a combinational one. Moreover, use of combinational ATPG guarantees complete fault efficiency in full scan technique. The problem of full scan technique is that it requires large hardware overhead and though it reduces the test generation time, test application time is high. Partial scan [3,4] technique suggests a cost effective alternative in comparison to full scan techniques, where instead of all the flip-flops, a subset of flip-flops are scanned. Though it decreases the hardware overhead, test application time is not improved. Moreover, as partial scan techniques require sequential test generation methods, high fault efficiency cannot be achieved. In both full and partial scan techniques at-speed testing is not possible.

To avoid the problems of scan techniques, non-scan approaches are proposed in [6-9]. In [6], a subset of all flip-flops are controlled from primary inputs using multiplexers. In [7], DFTs are designed using locally available lines. Both these techniques use sequential ATPG. In [8], non-scan design was targeted only to remove equivalent and isomorph redundancy. In all these approaches, though testing time may be improved, complete fault efficiency cannot not be achieved. Non-scan DFT approach with complete fault efficiency using combinational ATPG was first proposed in [9].

This paper suggests three new non-scan DFT techniques for sequential circuits. In all these methods, test sequences for different faults in a sequential machine are found by generating test patterns by a combinational ATPG tool used on combinational part of the machine. The novelty of the proposed techniques is that use of combinational ATPG tool guarantees complete fault efficiency. However, as each test pattern generated by this ATPG tool consists of values on primary inputs as well as state registers, some test patterns may consist some values which can never be reached by state transitions from reset states. To reach such values on state registers (invalid states), we propose a technique to append an extra logic called circuit to reach invalid states (CRIS) with the original machine. Among the proposed three techniques, the first one uses lowest hardware overhead, but it requires k additional observable points, where k is the number of flip-flops in the circuit. Use of one more additional circuit called as Differentiating Logic (DL) greatly reduces the number of additional observable points in second and third techniques. This logic DL is universal, i.e., not dependent on the circuit structure. First two techniques have low hardware overhead compared to that of full scan. First and third techniques have very low test application time. The second technique requires larger test application time in comparison to those of first and third, but this time is less than that of full scan technique. Three proposed

techniques, the method proposed in [9] and full scan technique are compared on benchmarks. Test length and hardware overhead are found to compare favorably with those of earlier designs.

2. Preliminaries

The general model of a synchronous sequential machine is shown in Fig. 1, consisting of a combinational circuit (CC) and a state register (SR) with k flip-flops. The machine has n primary input (PI) terminals fed by Binary variables x_1, x_2,…, x_n. The outputs [inputs] y_1, y_2,…, y_k [Y_1, Y_2,…, Y_k] of k memory elements of SR define the present [next] state of the machine. The behavior of the machine is described by the state transition diagram (STG). We assume that the machine has a reset state. Given an input, *transition from a state* S_i means the state transitions and change in output, if that input is applied in the machine with state S_i.

Consider a fault f in a sequential machine. Suppose this fault changes the transitions from a state S_i. Definitely to detect this fault, we have to first initialize the machine to state S_i.

Definition 1: Initialization state for a fault f in a machine is a state S_i, such that due to occurrence of f, transitions from S_i changes in STG.

Definitely, to initialize the machine to state S_i, we need a sequence of vectors to be applied in the machine from the reset state. This sequence is called as the justification sequence [5] for state S_i.

Definition 2: The *justification sequence* for a state S_i is a sequence of input vectors, application of which to the machine in the reset state, changes the state to S_i.

The problem is that there may not exist any such justification sequence for a state S_i. That is, with k memory elements, as the machine can have 2^k states, some of these states are unreachable from the reset state, though may be needed for initialization for testing.

Definition 3: A state that cannot be reached or which is hard to reach from reset state is known as an *invalid state,* else it is a *valid state*.

Note that our definitions for invalid and valid states in this paper are slightly changed from usual [5], as we include in the set of invalid states, also those states which are hard to reach from reset state. The list of invalid and valid states in a machine can be known from STG of the machine.

Even when initialization to a particular state is done, it may not be very easy to detect the fault. If the fault f is such that it changes the output lines of the circuit under some input combinations, then there is no problem to detect it, only one more test vector after the justification sequence is sufficient. But if the fault changes only the next state lines (suppose, it produces a state S_k instead of the expected state S_j under some input combination), then instead of one vector, we need a sequence of vectors to differentiate two states S_j and S_k.

Definition 4: A *differentiating sequence* [5] for a pair of states S_j and S_k, in a sequential circuit is a minimal length sequence of input vectors, such that the output response obtained by applying the sequence when the circuit is initially in S_j, is different from that obtained when the circuit is initially in S_k.

Let us extract the combinational circuit from a sequential machine, by replacing inputs [outputs] of SR by pseudo primary outputs (PPOs) [pseudo primary inputs (PPIs)]. Then this combinational circuit is known as the *combinational test generation model (CTGM)* of the sequential machine.

Example: CTGM of the machine of Fig. 1 is shown in Fig. 2.

Given the CTGM of a sequential machine, we try to generate the test vectors for this CTGM. Obviously each test vector is an ordered (n+k)-tuple, corresponding to n PIs and k PPIs.

Definition 5: A state in a machine is called a *test state,* if it appears in PPI lines of any test vector of CTGM of the machine.

Definition 6: A test pattern of the CTGM of a sequential machine is known to be a *valid test pattern,* if it contains a valid state in its PPIs. Otherwise, it is called as an *invalid test pattern.*

Definition 7: A state of a machine which is a test state and also a valid [an invalid] state is known as *valid test state* [*invalid test state*].

Note that in general, the number of test states is much smaller compared with the total number of states, 2^k.

3. New DFT designs

Given a sequential circuit with its STG, we try to design a non-scan DFT for it with a goal to attain complete fault efficiency.

From STG of the machine, first we find the set of valid and invalid states. Then we find the set of test vectors of CTGM of the machine using a combinational ATPG algorithm. If a test vector, obtained for CTGM is a valid test pattern, then obviously, this pattern can be reached from the reset state using original state transitions of the machine. But if it is an invalid test pattern, the value of PPIs cannot be set to the SR using state transitions of the machine. Actually, as some states are unreachable by state transitions from the reset state, this problem of state initialization to an unreachable state poses a major problem in the test generation of sequential circuits. In our designs, we adopt a new technique to reach these unreachable states. Notice that for testing of a circuit, we need not reach all invalid states, reaching only to invalid test states are sufficient. So our attention is to find a method to set invalid test states to SR.

3.1 The first technique

In our design to set the invalid test states to the SR, we append an extra logic called as CRIS (circuit to reach invalid states) to the original machine which generates all invalid test states of the machine. The DFT scheme is shown in Fig. 3. CRIS has the inputs as the next state lines of the original machine. In a similar approach recently [9], an additional circuit was also used to reach these invalid test states, where primary inputs are used as inputs to the extra logic. The method to design the CRIS is described in the following.

3.1.1 Designing CRIS

Suppose V is the set of valid states in the machine, and S_{ITS} denotes the set of invalid test states. Then obviously, any state $S_i \in V$ can appear in lines (Y_1, Y_2,…, Y_k) by proper transition from reset state. Similarly, any state $S_i \in S_{ITS}$ cannot appear in lines (Y_1, Y_2,…, Y_k) by any transition from reset state. The function of CRIS is to make also the appearance of these invalid test states at the inputs to SR. For this, to map V into ($V \cup S_{ITS}$), CRIS takes PPOs (Y_1, Y_2, …, Y_k) as the inputs, and produces new next state lines (Y'_1, Y'_2,…, Y'_k) as inputs to SR with the help of some control inputs. The control inputs are such that when each of them is at logic 0, output of CRIS is the same as input and when one or more of them is 1, it produces some invalid test state from a valid state. The question is how to optimize the circuit of CRIS with minimum number of control inputs to map V into ($V \cup S_{ITS}$). The problem of optimization of number of control inputs and hardware of CRIS is an open problem and yet to be solved. However, in this paper we give a heuristic approach.

To generate S_{ITS} from V, for each state $S_i \in S_{ITS}$, we first find how S_i can be produced from each state $S_j \in V$. For example, say a state (0101) is an invalid test state and that is to be produced from a valid state (0011). To do this, we need to complement 2^{nd} and 3^{rd} bits of (0011). This can be done by ORing Y_2 and ANDing Y_3 with a control input C such that $Y'_2 = Y_2 + C$ and $Y'_3 = Y_3 \bar{C}$; it means that with C = 0 output of CRIS is its input, whereas with C = 1 it can show 0101 at outputs (Y'_1, Y'_2, Y'_3, Y'_4) when input is (Y_1, Y_2, Y_3, Y_4): (0011). For each state $S_i \in S_{ITS}$, we find how S_i can be produced from each state in V. Among these different productions, we implement that one which requires minimum number of gates to produce S_i at output of CRIS from a valid state input at CRIS. If different invalid test states require same set of bits tso be complemented, we use the same control line. If any bit requires both ANDing and ORing, we replace the gate by XOR.

3.1.2 Number of control inputs and hardware overhead of CRIS

Theoretically, the number of such control lines requirement can be maximum k, and that happens when there is only one valid state in the machine and there are at least 2^{k-1} invalid test states. But practically, as number of invalid test states is much smaller (can be at most the number of test states) in comparison to total number ($=2^k$) of states, and that is not very high in comparison to number of valid states, control lines requirement and hardware overhead cannot be high. By our heuristic approach, benchmarks result depicts significantly low overhead in designs of CRIS. Only in two cases, it shows the use of two additional control lines with two two-input gates, in all other cases, one control input with one two-input gate is found to be sufficient.

3.1.3 Short test application time

As CRIS ensures generation of all invalid test states, then for each and every fault in the appended machine, we can always find the justification sequence to reach a test state. In Technique 1, as we make all next state lines as observable points, so whether or not a particular state (required for initialization for testing) is reached after justification sequence that is also observable. After this state is reached at present state lines, one more vector is sufficient to observe any change due to a fault. This vector is the test vector obtained by using combinational ATPG tool on CTGM of the machine. We use a hold mode in SR, while applying this test. The use of this hold mode can be described as follows.

Suppose an initialization state S_i for a fault is properly reached from reset state. When this initialization state appears in present state lines of SR, hold mode is activated. During hold mode, the state of the machine remains at S_i, whatever be the inputs at PIs. As several faults may require same initialization, for all such faults test vectors are applied consecutively holding the machine at state S_i. That is, we don't need to initialize the machine separately for each of the faults using same justification sequence repeatedly.

Also notice that as we observe the next state lines we don't need any differentiating sequence. Use of CRIS to reach unreachable states, use of no differentiating sequences and use of hold mode to avoid the repeated application of same justification sequence highly reduce test application time.

3.1.4 Short test generation time and complete fault efficiency

Use of combinational ATPG tool decreases test generation time. Use of this tool and use of CRIS to ensure the machine to reach any test state make fault efficiency to be 100%.

3.2 The second technique

In the first technique, observation of next state lines makes the length of a differentiating sequence always to be zero, but its drawback is the requirement of k additional observable points. To reduce this number in observable points, as well as to achieve differentiating sequences of short length we use one more additional circuit known as differentiating logic (DL), which takes PIs and PPIs as inputs. The complete scheme for DFT in technique 2 is shown in Fig. 4.

3.2.1 Design of Differentiating Logic (DL)

The number of outputs of DL is dependent on n and k. Two cases need to be considered: (i) $k \leq n$, (ii) $k > n$.

Case 1: $k \leq n$

In this case, DL has one output. The output function of the DL is given as $F = x_1y_1 + \bar{x}_1\,\bar{y}_1 + x_2y_2 + \bar{x}_2\,\bar{y}_2 + \ldots\ldots + x_ky_k + \bar{x}_k\,\bar{y}_k$ and the circuit to realize F is shown in Fig. 5. The function F has a unique property. For every combination of $(y_1, y_2,\ldots, y_k)$, $y_i \in (0,1)$, the sub-function contains a unique pattern in x_is. This pattern is such that for a pattern $(y_1, y_2,\ldots, y_k)$ at PPIs, if we apply a pattern X at PIs with $(x_1, x_2,\ldots, x_k) = (\bar{y}_1, \bar{y}_2,\ldots, \bar{y}_k)$, we get the output of DL as 0, and for any other pattern at PI the output of DL is always at logic 1. It implies that if the machine reaches a state $S_i(y_1, y_2,\ldots, y_k)$, then by applying a single input pattern, obtained by complementing each bit of $(y_1, y_2,\ldots, y_k)$, this state can be uniquely identified. That is, differentiating sequence of any two states is of unit length.

Example 1: The K-map for k=3 is shown in Fig. 6. Variables x_i's (y_i's) are used to label the map horizontally (vertically). A horizontal line in k-map corresponds to a state. Notice that in any horizontal line, there exists exactly one zero for a specific combination in x_i's, and for all other combination in x_i's, the minterms are true. Similarly, there exists exactly one zero in any vertical line. Say, for a state $(y_1,y_2,y_3) = (010)$, if we apply an input with $(x_1,x_2,x_3) = (101)$ we get F = 0, and for any other combination in (x_1,x_2,x_3) we obtain F = 1. Moreover, with $(x_1,x_2,x_3) = (101)$, for any state other than (010), we get F=1. It implies that differentiating sequence of any two states including (010) can be obtained by a single vector with $(x_1,x_2,x_3)=(101)$.

Case 2: $k > n$

In this case, as we have less number of inputs, same inputs are used repeatedly. DL has r outputs where $r = \lceil k/n \rceil$.

Each output line of DL realizes a function F_i $(1 \leq i \leq r)$ such that

$F_{j+1} = x_1y_{jn+1} + \bar{x}_1\,\bar{y}_{jn+1} + x_2y_{jn+2} + \bar{x}_2\,\bar{y}_{jn+2} + \ldots\ldots + x_ay_{jn+a} + \bar{x}_a\,\bar{y}_{jn+a}$ where a = n for $(0 \leq j < r-1)$, and $a=k-(r-1)n$ for $j=r-1$. If a is found to be 1, then we replace F_{j+1} by y_{jn+1}.

When n = 1, we get r = k, which means $F_i = y_i \ \forall\ i\ (1 \leq i \leq r = k)$. It means we don't need to have any special logic DL as shown in Fig. 5. This logic for DL is used to decrease the number of observable points. But for n=1, we cannot decrease this number (as r = k). So, for n=1, we apply only the first technique.

3.2.2 DL is universal

Notice that while in first technique the observable points are at the next state lines (inputs of SR), the second technique uses the present state lines (outputs of SR) to have the observable points. It is necessary, because in second technique to identify a state uniquely, we have to apply a specified pattern in PIs, so if we want to use the next state lines (inputs to SR) as inputs to DL, their values are changed with changing in PIs.

It can be noted that the design of this DL is universal, i.e., not dependent on the circuit structure.

3.2.2 Use of Hold mode

Hold mode is also used in the second technique, but its use is different to that of first technique. When we have to identify a state on present state lines, hold mode is activated during identification. If a state (y_1, y_2,…, y_k) is expected at present state lines for case 1 ($k \leq n$), we activate hold mode and apply an input at (x_1, x_2,…, x_k) lines of PIs such that $x_i = \overline{y_i}\ \forall\ i\ (1 \leq i \leq n)$. If the output of DL is 0, then the state of the machine is at the desired state. For case 2 ($k > n$), in stead of an input, an input sequence has to be applied at PIs keeping the hold signal active.

To detect a fault, we apply justification sequence from reset state to reach a state required for initialization for detecting a fault. Whether this initialization has properly reached that is checked by applying a proper input (or a sequence of inputs in case 2) at PIs with hold signal active. After that, with hold inactive, we apply the test vector obtained by ATPG tool used on CTGM. If this test vector does not change PO lines, then we need to identify any change on the state of the machine. That identification is done by again activating the hold signal with application of proper input (or a sequence of inputs) at PIs.

3.2.3 Techniques to achieve low test application time

By using DL in second technique, we decrease the number of additional observable points from k to r in comparison to first technique. Thus, differentiating sequence is of unit length in case of $k \leq n$, and for $k > n$, this length is $r = \lceil k/n \rceil$. Note that in case of full scan technique, this length is always n. Short differentiating sequence decreases the test application time.

To decrease the test application time further, we adopt another technique. Suppose, the detection of a fault f_1 requires the application of a test vector at the machine with state S_i. After proper initialization is done, application of the test vector may change the state of the machine to another state S_j, which again can be identified. But another fault f_2 may require same initialization state S_i, for which we need to reach the reset state and then to apply the same justification sequence, if we next want to test for f_2. In this case, instead of further initialization to S_i, we check, whether S_j is a test state. If S_j is a test state, we use it as an initialization state of another fault which is not yet detected. If S_j is not a test state or there is no other fault remaining to be detected with initialization state S_j, in that case we may attempt to reach again to state S_i from reset state to detect fault f_2. Following this technique, benchmark results shows less test application time in comparison to that of full scan, though this time is increased from that of first technique.

3.*2.4 Hardware overhead*

Amount of hardware overhead of DL is (2k+r) gates, where $r = \lceil k/n \rceil$. This overhead is less than that (=3k) of full scan technique for $r < n-1$.

3.3 The third technique

Though the second technique decreases the number of observable points, one drawback of it is that as observable points use the present state lines, initialization state for a fault is always lost. We cannot use the same justification sequence for different faults which need same initialization state for their detection.

To achieve the same test application time as that of first technique, using less observable points, the second technique is modified in the third technique, where a register is used to load the values of the next state lines and output lines of this register are fed into DL instead of PPIs. The complete scheme is shown in Fig. 7. Use of hold mode is similar to that of first technique, one justification sequence is used to initialize for detection of different faults. Whether, a specified

state can be reached in the machine that can be observed at output lines of DL. However this technique requires more hardware, as its hardware overhead needs one more register in comparison to that of second technique.

4. Experimental Results

General performance of the DFT Design can be described as in Table 1. The machine is considered to have n inputs and k flip-flops. O(ISG) and O(CRIS) indicate the overhead of invalid state generator (ISG) in the paper (ATS-98) of [9] and that of CRIS of this paper respectively. It is found experimentally that O(CRIS) < O(ISG). O(CRIS) was found to be maximum of two two-input gates in MCNC benchmarks. In techniques 1,2 & 3, c denotes the number of control inputs needed for CRIS. In most of the cases of benchmarks, the value of c is found to be 1, except in two cases, where it is found to be 2. In techniques 2 & 3, r denotes the value $\lceil k/n \rceil$.

Table 1: Overall comparison

Method	Pin overhead	Area overhead	Test generation time	Test application time	At-speed Testing
Full-scan	3	3k gates	low	high	Not possible
ATS-98 [9]	k+2	O(ISG) + 3k gates	low	low	possible
Technique-1	k+ 1+ c	O(CRIS)	low	low	possible
Technique-2	r+ 1+ c	O(CRIS) + (2k+r) gates	low	higher than tech. 1, less than full scan	possible
Technique-3	r+ 1+ c	O(CRIS) + (2k+r) gates + register (k flip-flops)	low	low	possible

Our experimental results on benchmarks are also shown. We compare our three methods with full scan and a recent work with complete fault efficiency [9]. Hardware overhead, test generation time and test application time are the parameters to be compared.

Benchmark specifications are shown in Table 2. AutoLogic II (Mentor Graphics) tool synthesizes the circuits from MCNC benchmarks [10]. Columns “name”, “#PIs”, “#POs”, “#states” denote the name, the number of primary inputs, the number of primary outputs and the number of states of the original sequential machines respectively. Columns “#FFs” and “area” denote the number of flip-flops and circuit areas after synthesis. In benchmark results, we show only those cases when number of inputs (n) > 1. For n=1, we apply only the first technique of our DFT designs.

Table 3 shows hardware and pin overhead. Columns “name” represent the different circuits. Columns “scan” and “ATS98 [9]” represent full scan and the method in [9] respectively. Columns “case 1”, “case 2” and “case 3” correspond to technique 1, technique 2 and technique 3 of our DFT design respectively. Hardware overhead of first technique is lowest and significantly small. Hardware overhead of both first and second technique is smaller than that of full scan. The third technique needs more hardware as an additional register of k flip-flops (k= # of flip-flops)

are used. We have considered 7 gates per flip-flop in third technique. In the techniques 2 & 3, number of gates are decreased by 3 from that given in the formula of Table 1, if the remainder in dividing k by n be 1. Pin overhead of proposed second and third techniques are same and in most cases it equals to that of full scan technique. In full scan technique pin overhead is always 3. The first technique, which has very low overhead in hardware requires more number of pins. This pin overhead of technique 1 is same as that in the method of [9].

Test generation and test application time for different methods are shown in Table 4. A combinational/sequential test generation tool TestGen (Sunrise) is used. Results show that test generation time is almost equal in five cases. This happens as all these techniques use combinational ATPG tool. Test application time is highest in case of full scan method. This time is almost equal in the method of [9], first technique and third technique. Second technique requires larger test application time in comparison to those of first and third techniques, but this time is short in comparison to that of full scan.

5. Conclusions

The paper suggests three new techniques on non-scan DFT. As test initialization is a major problem in testing of sequential circuits, it solves that problem by using an additional hardware called as CRIS (circuit to reach invalid states). It is found experimentally that hardware overhead of CRIS is also low. The techniques use combinational ATPG tool to find the test sequences of the machine. Among the three techniques, hardware overhead of the first technique is the lowest, but it requires k additional observable points. To decrease the number of observable points, a notion of differentiating logic (DL) is proposed in technique 2. Even with the use of this DL, hardware overhead is less than that of full scan. Use of this DL increases test application time in comparison to that of first technique, but this time is less than that of full scan. To achieve the test application time, same as that of first technique, an additional register is used in third technique. The novelty of these techniques is that they guarantee complete fault efficiency with at-speed testing. Hardware overhead, test generation time and test application time compare favorably with those of earlier designs.

Acknowledgement: Debesh Kumar Das is supported by JSPS-INSA fellowship. Satoshi Ohtake is under JSPS research fellowship. This work was supported in part by Semiconductor Technology Academic Research Center (STARC) under the Research Project and in part by the Ministry of Education, Science, Sports and Culture, Japan under Grant-in-Aid for Scientific Research B(2) (no.09480054). Authors would like to thank Toshimitsu Masuzawa, Tomoo Inoue, and Michiko Inoue of Nara Institute of Science and Technology for their helpful discussion.

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,* W. H. Freeman & Co., New York 1990.
3. S. T. Chakradhar, A. Balkrishnan and V. D. Agrawal, "An exact algorithm for selecting partial scan flip flops", *Proceedings of ACM/IEEE Design Automation Conference,* pp. 81-86, 1994.

4. P. S. Parikh and M. Abramovici, “A cost based approach to partial scan,” *Proceedings of ACM/IEEE Design Automation Conference*, 1993.
5. S. Devadas and K. Keutzer, “A unified approach to the synthesis of fully testable sequential machines,” *IEEE Transactions on Computer-Aided Design*, vol.10, pp. 39-50, 1991.
6. V. Chickermane, E. M. Rudnick, P. Banerjee and J. H. Patel, “Non-scan design-for-testability techniques for sequential circuits,” *Proceedings of 30th ACM/IEEE Design Automation Conference*, pp. 236-241, 1993.
7. I. Pomeranz and S. M. Reddy, “Design for testability for sequential circuits using locally available lines,” *Proceedings of Design, Automation and Test in Europe (DATE-98)*, p. 983-984, 1998.
8. D. K. Das and B. B. Bhattacharya, “Testable design of non-scan sequential circuits using extra logic,” *Proceedings of Asian Test Symposium*, pp. 176-182, 1995.
9. S. Ohtake, T. Masuzawa and H. Fujiwara, “A non-scan DFT method for controllers to achieve complete fault efficiency,” to appear in *Proceedings of Asian Test Symposium*, 1998.
10. S. Yang, “Logic synthesis and optimization benchmarks user guide,” *Technical Report 1991-IWLS-UG-Saeyang, Microelectronics Center of North Carolina.*

Table 2: STG characteristic

name	#PIs	#POs	#States	#FFs	Area(gates)
bbara	4	2	10	4	418.6
bbsse	7	7	16	4	781.2
bbtas	2	2	6	3	87.6
beecount	3	4	7	3	331.5
cse	7	7	16	4	1104.1
dk14	3	5	7	3	295.1
dk15	3	5	4	2	169.9
dk16	2	3	27	5	510.4
dk17	2	3	8	3	150.1
ex1	9	19	20	5	2740.5
ex2	2	2	19	5	416.9
ex3	2	2	10	4	192.8
ex4	6	9	14	4	479.2
ex5	2	2	9	4	183.7
ex6	5	8	8	3	667.0
ex7	2	2	10	4	189.6
keyb	7	2	19	5	1835.4
kirkman	12	6	16	4	2848.2
lion	2	1	4	2	104.0
lion9	2	1	9	4	340.9
mc	3	5	4	2	81.1
opus	5	6	10	4	567.6
planet	7	19	48	6	2791.1
planet1	7	19	48	6	2791.1
pma	8	8	24	5	1068.6
s1	8	6	20	5	2396.0
s1488	8	19	48	6	6190.2
s1494	8	19	48	6	6242.8
s208	11	2	18	5	2361.3
s27	4	1	6	3	416.3
s298	3	6	218	8	8720.8
s386	7	7	13	4	1241.1
s420	19	2	18	5	2217.5
s510	19	7	47	6	1184.2
s820	18	19	25	5	4411.0
s832	18	19	25	5	4543.7
sand	11	9	32	5	3860.9
sse	7	7	16	4	781.2
styr	9	10	30	5	2748.9
tav	4	4	4	2	228.9
tbk	6	3	32	5	8605.3
tma	7	6	20	5	802.7
train11	2	1	11	4	364.5
train4	2	1	4	2	83.3

Table 3: Hardware/pin overhead

name	Hardware Overhead (*gates*)					Pin Overhead				
	scan	ATS98[9]	case1	case2	case3	scan	ATS98[9]	case1	case2	case3
bbara	12	12	1	10	38	3	6	6	3	3
bbsse	12	12	1	10	38	3	6	6	3	3
bbtas	9	10	1	6	27	3	5	5	4	4
beecount	9	9	1	8	29	3	5	5	3	3
cse	12	0	0	9	37	3	5	5	2	2
dk14	9	9	1	8	29	3	5	5	3	3
dk15	6	0	0	5	19	3	3	3	2	2
dk16	15	31	1	11	46	3	7	7	5	5
dk17	9	0	0	5	26	3	4	4	3	3
ex1	15	15	1	12	47	3	7	7	3	3
ex2	15	34	2	12	47	3	7	8	6	6
ex3	12	20	1	11	39	3	6	6	4	4
ex4	12	12	1	10	38	3	6	6	3	3
ex5	12	20	1	11	39	3	6	6	4	4
ex6	9	0	0	7	28	3	4	4	2	2
ex7	12	20	2	12	40	3	6	7	5	5
keyb	15	15	1	12	47	3	7	7	3	3
kirkman	12	0	0	9	37	3	5	5	2	2
lion	6	0	0	5	19	3	3	3	2	2
lion9	12	20	1	11	39	3	6	6	4	4
mc	6	0	0	5	19	3	3	3	2	2
opus	12	12	1	10	38	3	6	6	3	3
planet	18	18	1	14	56	3	8	8	3	3
planet1	18	18	1	14	56	3	8	8	3	3
pma	15	15	1	12	47	3	7	7	3	3
s1	15	15	1	12	47	3	7	7	3	3
s1488	18	18	1	14	56	3	8	8	3	3
s1494	18	18	1	14	56	3	8	8	3	3
s208	15	15	1	12	47	3	7	7	3	3
s27	9	9	1	8	29	3	5	5	3	3
s298	24	165	1	20	76	3	10	10	5	5
s386	12	12	1	10	38	3	6	6	3	3
s420	15	15	1	12	47	3	7	7	3	3
s510	18	18	1	14	56	3	8	8	3	3
s820	15	15	1	12	47	3	7	7	3	3
s832	15	15	1	12	47	3	7	7	3	3
sand	15	0	0	11	46	3	6	6	2	2
sse	12	12	1	10	38	3	6	6	3	3
styr	15	15	1	12	47	3	7	7	3	3
tav	6	0	0	5	19	3	3	3	2	2
tbk	15	0	0	11	46	3	6	6	2	2
tma	15	15	1	12	47	3	7	7	3	3
train11	12	16	1	11	39	3	6	6	4	4
train4	6	0	0	5	19	3	3	3	2	2

Table 4: Test generation/application time

	Test Generation Time (*sec.*)					Test Application Time (*cycles*)				
name	scan	ATS98[9]	case1	case2	case3	scan	ATS98[9]	case1	case2	case3
bbara	0.32	0.32	0.36	0.39	0.37	339	88	86	171	89
bbsse	0.58	0.58	0.60	0.63	0.63	399	101	106	250	105
bbtas	0.06	0.06	0.05	0.06	0.06	71	27	29	43	54
beecount	0.19	0.19	0.21	0.19	0.26	199	60	63	69	79
cse	1.08	1.08	1.16	1.13	1.19	584	144	142	323	153
dk14	0.13	0.13	0.15	0.15	0.16	199	60	69	56	68
dk15	0.10	0.10	0.05	0.09	0.12	110	42	37	35	35
dk16	0.35	0.35	0.43	0.41	0.46	593	148	151	408	376
dk17	0.06	0.06	0.09	0.09	0.14	127	47	46	74	90
ex1	4.58	4.58	5.07	5.18	5.14	1679	323	335	838	340
ex2	0.27	0.27	0.29	0.24	0.35	461	119	107	196	307
ex3	0.12	0.12	0.12	0.14	0.17	249	72	61	84	133
ex4	0.31	0.31	0.32	0.33	0.29	294	76	77	212	83
ex5	0.11	0.11	0.10	0.15	0.15	244	71	64	90	116
ex6	0.46	0.46	0.45	0.50	0.47	243	70	71	69	69
ex7	0.08	0.08	0.13	0.18	0.12	194	60	61	33	123
keyb	4.90	4.90	5.18	5.20	5.47	1481	282	307	634	316
kirkman	13.88	13.88	12.79	12.89	12.76	2409	499	499	2796	508
lion	0.07	0.07	0.09	0.05	0.07	56	24	23	23	22
lion9	0.21	0.21	0.21	0.19	0.20	259	69	67	180	139
mc	0.06	0.06	0.06	0.06	0.10	44	20	19	22	23
opus	0.39	0.39	0.36	0.42	0.43	394	106	105	289	110
planet	4.14	4.14	4.13	4.25	4.38	1539	400	392	2002	407
planet1	4.28	4.28	4.02	4.18	4.12	1539	400	392	2002	407
pma	1.30	1.30	1.36	1.38	1.40	971	204	204	428	208
s1	5.19	5.19	5.14	5.23	5.05	1283	269	291	889	288
s1488	20.27	20.27	20.35	20.66	20.88	3051	615	660	5571	647
s1494	21.60	21.60	22.07	21.87	20.91	3065	645	677	4379	650
s208	9.35	9.35	7.85	8.11	7.80	1535	289	313	1645	319
s27	0.24	0.24	0.28	0.35	0.38	191	59	66	44	74
s298	82.40	82.40	79.58	83.91	86.98	9746	2429	2516	25924	5031
s386	1.23	1.23	1.31	1.27	1.29	554	131	144	404	134
s420	9.42	9.42	7.70	8.38	7.83	1439	273	305	1896	305
s510	1.06	1.06	1.13	1.22	1.21	930	195	194	1899	201
s820	16.09	16.09	16.39	16.24	16.26	2273	450	507	3010	484
s832	17.25	17.25	17.63	17.95	18.01	2225	438	516	2443	525
sand	9.26	9.26	8.87	8.72	9.12	1547	308	301	691	324
sse	0.61	0.61	0.58	0.60	0.58	399	101	106	250	105
styr	5.60	5.60	5.44	5.52	5.51	1385	296	319	793	309
tav	0.24	0.24	0.24	0.29	0.24	119	45	42	52	46
tbk	51.75	51.75	49.86	49.64	50.14	4469	793	780	1864	783
tma	0.85	0.85	0.88	0.90	0.99	623	151	157	252	162
train11	0.23	0.23	0.26	0.23	0.24	264	74	83	76	140
train4	0.05	0.05	0.05	0.05	0.07	41	19	18	12	21

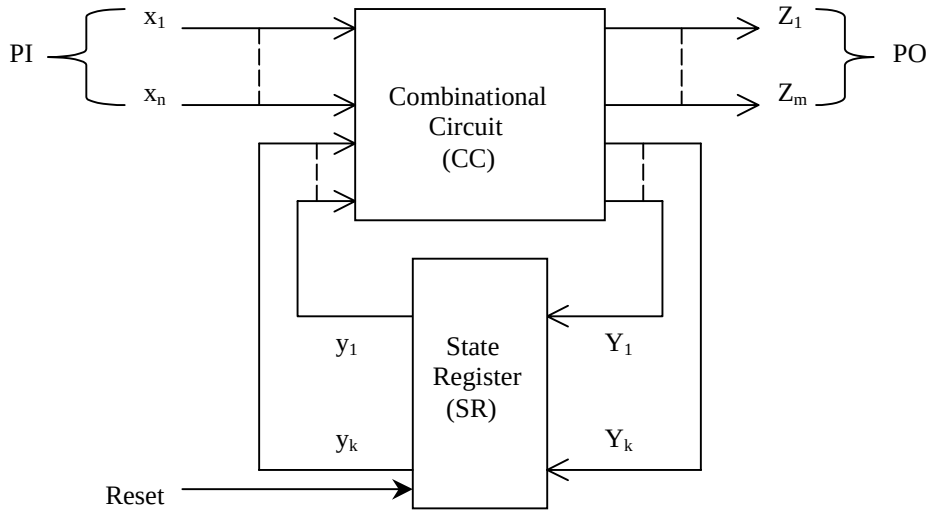

Fig. 1: The general model of a sequential machine

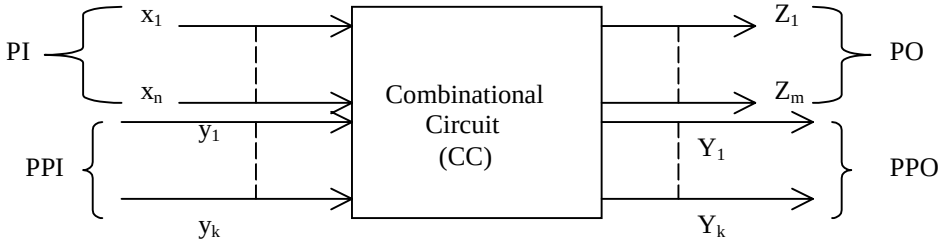

Fig. 2: CTGM of the machine of Fig. 1

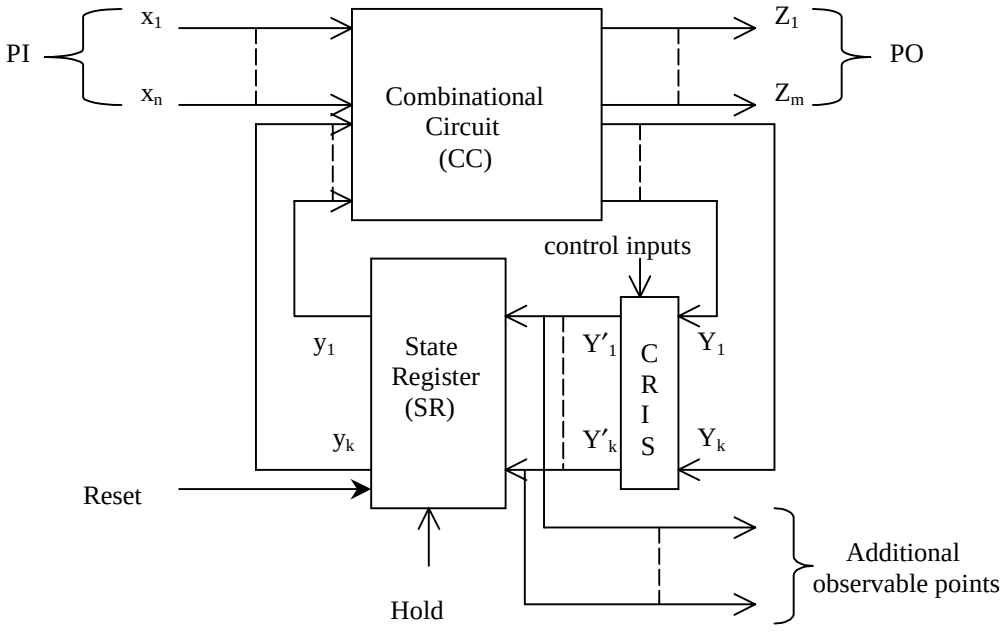

Fig. 3: DFT Design to achieve complete fault efficiency (Technique 1)

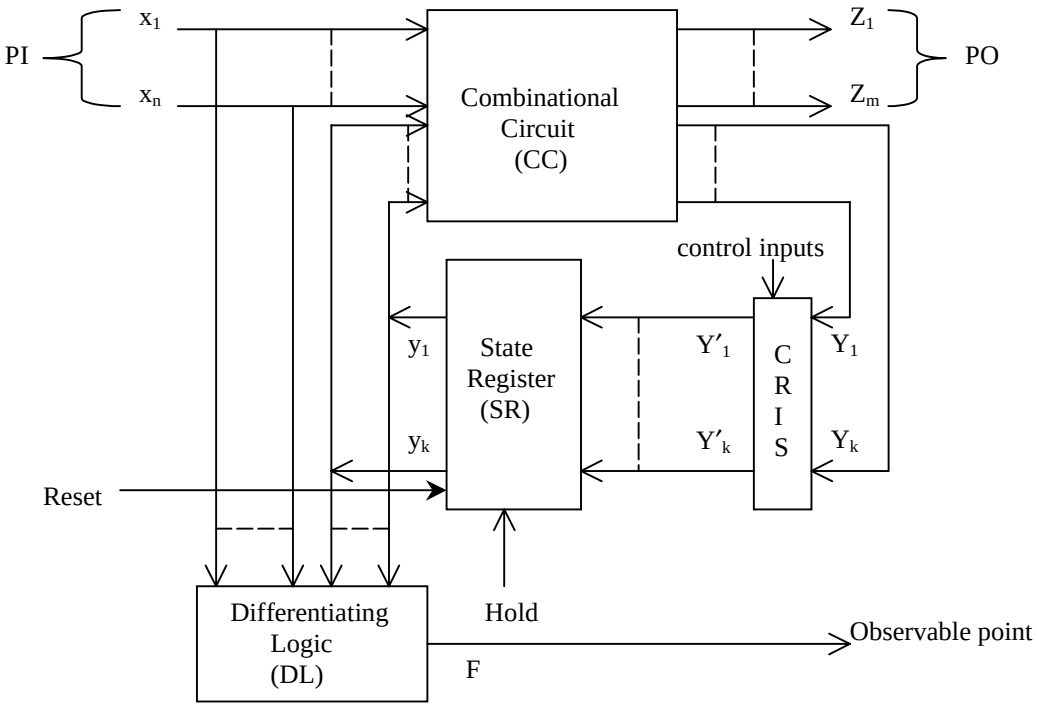

Fig. 4: DFT Design with complete fault efficiency and less observable points (Technique 2)

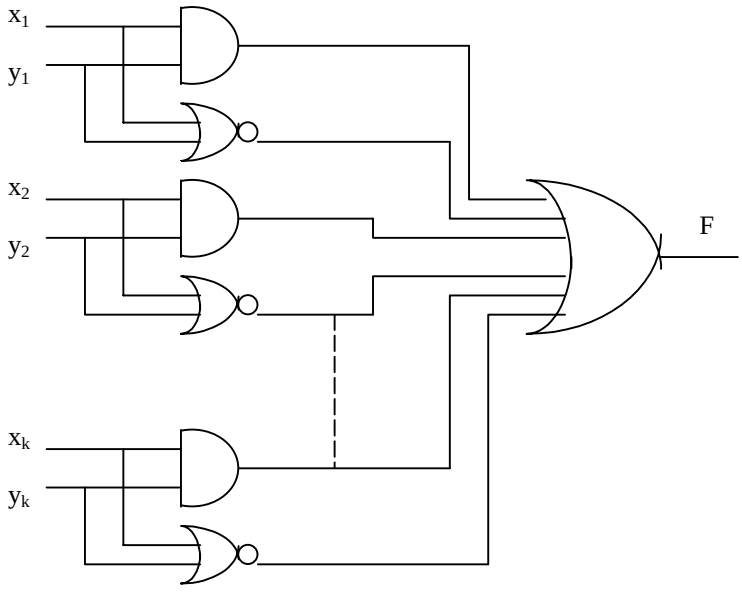

Fig. 5: Differentiating Logic (DL)

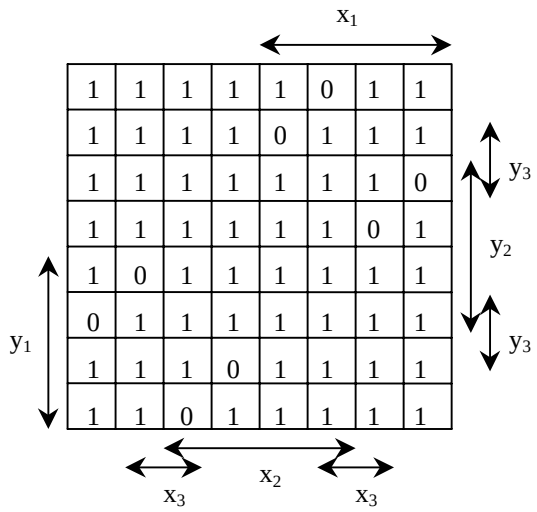

Fig. 6: K-map of DL for k=3 and n ≥ 3

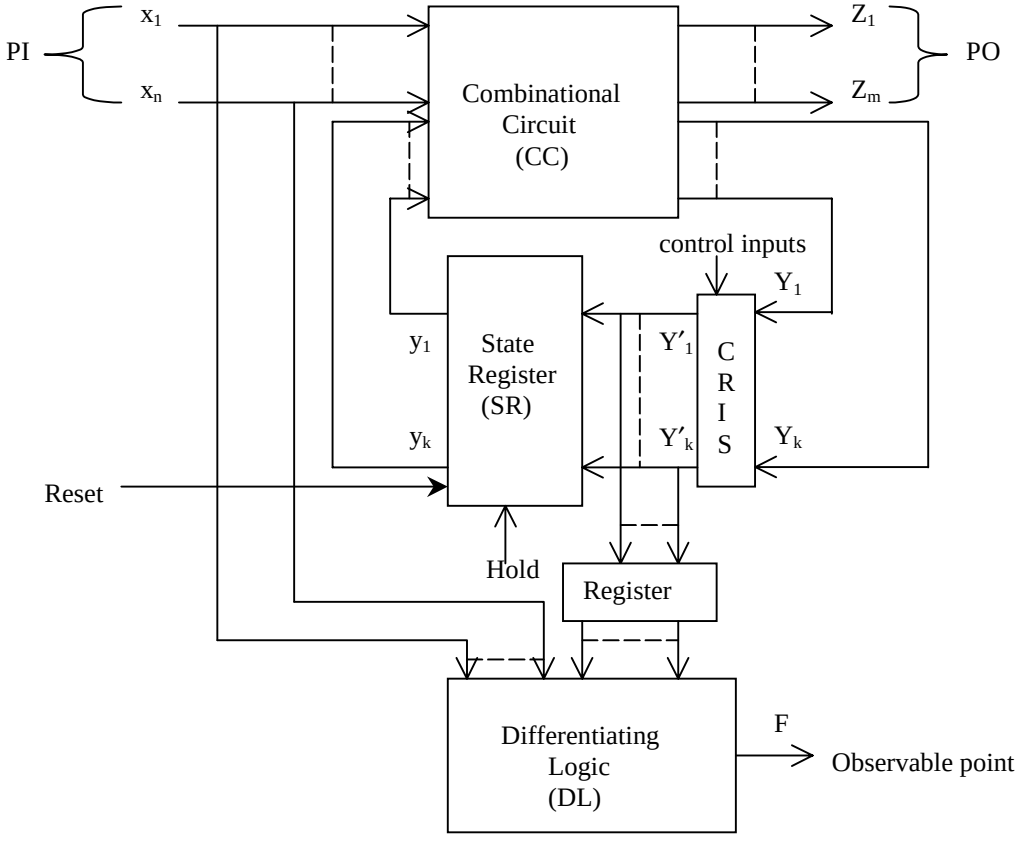

Fig. 7: DFT Design of Technique 3