Observation Point Oriented Deterministic Diagnosis Pattern Generation (DDPG) for Chain Diagnosis

Fei Wang\textsuperscript{1, 2}, Yu Hu\textsuperscript{1}, Yu Huang\textsuperscript{3}, Jing Ye\textsuperscript{1, 2}, Xiaowei Li\textsuperscript{1}\textsuperscript{*}

\textsuperscript{1} Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
\textsuperscript{2} Graduate University of Chinese Academy of Sciences, Beijing, China
\textsuperscript{3} Mentor Graphics Corporation, 300 Nickerson Rd., Marlborough, MA, 01752, USA
{wang_fei, huyu, yejing, lxw}@ict.ac.cn

Abstract

Scan is a widely used Design-for-Testability technique to improve test and diagnosis quality. Many defects may cause scan chains to fail. In this paper, an observation point oriented Deterministic Diagnostic Pattern Generation (DDPG) method was proposed for compound defects, which tolerates the system defects during scan chain diagnosis. Instead of sensitizing multiple paths proposed in our prior work, the proposed new DDPG method directly targets as many observation points as possible to observe the loading error occurred on the targeted scan cell. Experimental results on ISCAS’89 benchmark circuits show that the proposed DDPG method improves the effectiveness and efficiency of diagnosing compound defects, compared to our prior research.

1. Introduction

Scan design is a widely applied Design-for-Testability (DFT) industrial technique. It was reported in [1] that 10-30\% defects cause scan chain to fail, while in another paper [2], it was even reported that chain failures account for almost 50\% of chip failures. Therefore, scan chain failure diagnosis becomes an important research topic in recent years.

Past research in the diagnosis of scan chains can be classified into two major categories: software-based scan chain fault diagnosis techniques and hardware-based scan chain fault diagnosis techniques. Most of the previously proposed software-based chain diagnosis techniques [1] [3-5] [7] assume the system logic is defect-free. It may be an impractical assumption that leads to incorrect diagnostic results. When scan chain defects and system logic defects co-exist on one chip, it is called compound defects [9]. According to [9], the diagnosis hit rate for compound defects can be smaller than 40\% when assuming system logic is defect-free.

Although hardware based scan chain fault diagnosis methods [12-14] do not have to make any assumption on the system logic, they are rarely applied in the realistic industrial designs due to the hardware overhead of the special scan architecture and unconventional DFT flow.

Even though some previous work [2] [6] [8-10] proposed some techniques that can diagnose compound defects, these works are not Deterministic Diagnostic Pattern Generation (DDPG) methodology. The diagnostic quality substantially depends on the existing patterns, which may not guarantee to get the best diagnosis results all the time.

In this paper, we proposed a novel observation point oriented DDPG method that targets diagnosing scan chain faults while compound defects existing. The proposed DDPG improves the effectiveness and efficiency as our experimental results will show. The rest of the paper is organized as follows. Session 2 explains the fault models and terms used in this paper before reviewing some previous work. Session 3 describes the proposed DDPG algorithm in detail. Session 4 illustrates the proposed diagnosis technique based on the created DDPG patterns. Experimental results are shown in session 5. Session 6 concludes the paper.

2. Background

2.1 Scan Chain Fault Models

In this paper, we assume each scan cell in a scan chain is given an index in descending order from the scan input to the scan output and the faults occur on scan input side. The scan chain length (L) is the total number of scan cells in the scan chain. Six types of fault models in two categories [2] are employed in this work. One category is stuck-at faults including stuck-at-0 (SA0) and stuck-at-1 (SA1). The other category is timing faults including slow-to-rise (STR), slow-to-fall (STF), fast-to-rise (FTR) and fast-to-fall (FTF). Table 1 shows the six fault models and their effects on a scan chain composed of 8 scan cells. The affected data is indicated in underlined bold face.

### Table 1. The fault effects under various fault models

<table>
<thead>
<tr>
<th>Model</th>
<th>Expected Scan Out</th>
<th>Actual Scan Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA0</td>
<td>000001111</td>
<td>000000000</td>
</tr>
<tr>
<td>SA1</td>
<td>111100000</td>
<td>111111111</td>
</tr>
<tr>
<td>STR</td>
<td>111100000</td>
<td>111100000</td>
</tr>
<tr>
<td>STF</td>
<td>111100000</td>
<td>111111111</td>
</tr>
<tr>
<td>FTR</td>
<td>111100000</td>
<td>111110000</td>
</tr>
<tr>
<td>FTF</td>
<td>000001111</td>
<td>000001111</td>
</tr>
</tbody>
</table>

In this paper, observation points include all the POs and PPOs (Pseudo Primary Output). However some states of
observation points may be corrupted by scan chain fault. If the states of observation points can be correctly observed no matter whether scan chain is faulty or not and the observation points can capture the states of targeted scan cell, the observation point are called *Available observation points (AOPs)*. In this paper, we use AOPs to observe the loading error occurred on a targeted scan cell.

### 2.2 Related work

Prior software-based chain diagnosis techniques can be classified into two categories: (1) simulation based chain diagnosis and (2) using DDPG to generate dedicated diagnostic patterns.

The techniques proposed in [9] [10] belong to category (1). In [9], an algorithm was proposed to diagnose general compound defects. The failures are first partitioned into two groups by X-simulation. The failure bits in the first group are caused by system logic defects and the failure bits in the other group could be caused by either chain defects or system logic defects. System logic diagnosis techniques and chain diagnosis techniques are applied to those two groups of failures respectively. In [10], a special defect called DACS was explored. DACS stands for Defect that Affect Chain and System logic. However, the method cannot diagnose the compound defects if the scan chain failures are uncorrelated with the system logic failures.

![Fig. 1 Impaction of path sensitization](image)

The DDPG for compound defects firstly proposed in [11] belong to category (2). The basic idea of the DDPG is to propagate the state of the targeted scan cell to multiple reliable observation points. As system defects usually impact a small number of AOPs, we could still correctly determine the loading state of targeted scan cell even a dozen of system defects exist. To propagate the state of the targeted scan cell, the DDPG tries to establish the sensitized paths from the targeted scan cell to a set of reliable observation points as many as possible by constraining the off-path inputs of every gate along the propagation paths to non-controlling values. This is why we call the DDPG *path oriented DDPG*. The DDPG can guarantee that only system defects can result in unexpected responses on AOPs. However, it is difficult to determine which path should be selected. Moreover, in most of conditions, we cannot simply constrain the off-path inputs of some gates to non-controlling values because of the existing re-convergent fan-out regions. Consider an example shown in Figure 1, the state of the targeted scan cell $M$ can be propagated to $PO_1$ by setting $PI_1=1$ and cell $N=1$. It illustrates that although gate $G3$ is on the propagation path, when we randomly select one path from $M$ to $PO_1$, the off-path input of gate $G3$ could not be simply constrained to non-controlling value for fault propagation. As a result, we have to employ some time-consuming algorithms to analyze every fan-out branches on the pre-selected paths.

### 3. Observation Point Oriented Pattern Generation for Scan Chain Fault

Let us re-exam the example in Figure 1. In fact, it is not necessary to select fault propagation paths from the targeted scan cell to reliable observation point. Instead, we only need to guide an ATPG tool to find AOPs, therefore, this method is called *observation point* oriented DDPG in this paper. For example, to force ATPG using $PO_1$ to detect the fault at the targeted cell $M$, we change the original circuit to the circuit as shown in Figure 2. A copy ($A'$) of the sub-circuit of combinational logic ($A$) is created between $M$ and $PO_1$, and an XOR gate is inserted to check whether the responses of two sub-circuits ($A$ and $A'$) are different. Here $A$ represents the circuit where $M$ has no loading error while $A'$ represents the circuit that a loading error has occurred on the targeted cell $M$. Hence the state of $M$ is inverted before connecting to $A'$. To create a pattern, any ATPG tool can be employed to detect a SA0 fault on the output of the XOR gate. Therefore, the basic idea of DDPG is altered from establishing sensitized paths [11] to create as many failure bits as possible on reliable observation points when a loading error occurring on the targeted scan cell.

![Fig. 2 An example of observation point oriented pattern generation for scan cell $M$](image)

Three major contributions are made in this paper.

1. The circuit can be considered as a black box before running stuck-at ATPG. Therefore, we need not to analyze the structure of the circuit-under-test (CUT).

2. DDPG can automatically choose available observation points, as long as the ATPG can detect the SA0 fault on the output of the XOR gate. Therefore, we need not to pre-select paths from a targeted cell to a set of reliable observation points as in our previous work [11].

3. Constraints are not imposed on the fault propagation path directly. Therefore, the searching space explosion problem can be avoided even the number of paths is tremendously large in a circuit with many re-convergent fan-out regions.

### 3.1 Pattern Generation Algorithm Overview

The basic idea of the DDPG algorithm is to create as many failure bits as possible on reliable observation points when a loading error occurs on the targeted scan cell. The purpose of creating failure bits on multiple reliable observation points is to reduce the impact from system logic defects.

We extend the specific example shown in Figure 2 to a more general model shown in Figure 3. Here Cb (Combinational Circuit) is the combinational logic of a circuit-under-test between PPIs and PPOs. CCb (Copy
Combinational Circuit) is a copy of Cb. Cb represents the circuit where the targeted scan cell has no loading error while CCb represents the circuit that a loading error has occurred on targeted scan cell, therefore the targeted PPI is inverted before connecting to CCB. Inputs and outputs of Cb and CCB are all connected to a Validity Checker, surrounded by the dash line as shown in Figure 3. As explained later in detail, the Validity Checker, which includes an Input Checker and an Output Checker, is used to check whether a pattern satisfies the constraints for activating and propagating the loading error at the targeted scan cell.

An Input Checker in Validity Checker is employed to check the constraint (1.A) while an Output Checker is employed to check the constraint (1.B). We will illustrate meanings of various constraints in detail in the next two subsections. If all constraints are satisfied with a pattern, signal “Valid” is set to “1”. Therefore, to set signal “Valid” to “1”, an ATPG tool can be employed to detect a SA0 fault on signal “Valid”. Thus, this technique converts the diagnosis problem into a single stuck-at fault ATPG problem, which can be easily solved by existing ATPG tools. Once the SA0 fault is detected, the created pattern is used as the scan chain diagnostic pattern. If the SA0 fault cannot be detected, instead of claiming this targeted scan cell is un-diagnosable we could set \( k = k-1 \) and repeat the above procedure until \( k = 1 \). That is to say, if we could not find \( k \) available observation points, we try to find \( k-1 \) available observation points.

### 3.2 Input Checker

Input Checker imposes two kinds of constraints on the loading values of a created pattern:

(2.A) The targeted scan cell should be sensitive to the fault.

(2.B) Fault sensitive transitions should not occur on scan cells used for fault propagation. That is to say, the value of the scan cells used for fault propagation on the faulty scan chain should be correctly loaded. Constraint (2.B) is necessary because if the states of scan cells used for fault propagation are corrupted, we do not know the unexpected responses on AOPs are either caused by faults in system logic or caused by loading errors. To illustrate the structure of Input Checker, we use an example as shown in Figure 4. Assume a STR fault needs to be detected.

To satisfy (2.A), the targeted scan cell should be sensitized to excite the STR fault, therefore, the targeted scan cell \( F_{PPI} \) is constrained to “1” while its immediate downstream neighbor (cell \( F_{PPI} \)) is constrained to “0”. Similarly, if the fault type is FTR or FTF, we constrain the immediate upstream neighbor (cell \( F_{PPI} \)) of cell \( F_{PPI} \).

Once the sensitive transition is created, the output of AND gate G1 should be 1.

To satisfy (2.B), we use CU (Checking Unit) to constrain all scan cells except the targeted scan cell in the faulty scan chain. For example, if a 0→1 transition occurs at the cell pair \( F_{PPI}, F_{PPI} \) on the faulty scan chain, the output of the corresponding CU gate Gi will be “0”, which leads signal “IC” and signal “Valid” to “0”. It means such a pattern is impossible to detect a SA0 at signal “Valid”. It hence implies, when a pattern is successfully created, it must satisfy the constraint (2.B). Actually, (2.B) is not a strong constraint. Firstly, these constraints are only on faulty scan chains. Secondly, most of the cells in faulty scan chain can be considered as don’t-care-bits based on our experience and applying values to don’t-care-bits will not impact fault resolution. Thirdly, as for the scan cells with certain states, only the scan cells whose states are “0” may cause constraint conflicts. Finally, as for the scan cell with “0” state, conflicts will not happen if the immediate downstream of the scan cell can be set to “1”.

### 3.3 Output Checker

The basic idea of Output Checker is to check whether the observation points are AOPs. If an observation point is an AOP, the corresponding bit of G1, G3 or G4 shown in Figure 5 should output “1”. The gate with a black dot calculates the number of “1” output by G1, G3 and G4. That
is to say, the gate with a black dot calculates the number of AOPs. If the number of AOPs is greater than or equal to $k$, signal "OC" is set to ‘1’. The inputs of Output Checker coming from Cb and CCb consist of three kinds of constraints as shown in Figure 6: (3.A) POs and (3.B) PPOs in good scan chains denoted as $G_{PPO}$, and (3.C) PPOs in the faulty scan chain denoted as $F_{PPO}$.

(3.A) $PO_{k,n}$ and $PO'_{0,n}$ denote the responses on POs from Cb and CCb respectively. We use XOR gate G1 to compare the responses. If responses on one of POs are different, the corresponding bit of G1 outputs “1”, which shows the corresponding bit is an AOP.

(3.B) The PPOs in good scan chains are connected to XOR gate G3 that has similar function as described in (3.A).

(3.C) As captured states in the faulty scan chain may be contaminated during unloading responses, CUs are employed to check whether fault sensitive transitions will occur on PPOs. Meanwhile, the PPO is also connected to corresponding XOR gate G2 to compare the response. Only when the corresponding outputs on CU and on XOR gate are both “1”, the corresponding output of G4 is “1”. For stuck-at faults, the reliable observation points include the downstream cells of Lower Bound (LB) in the faulty scan chain, as well as good scan chains and POs. For SA1 (SA0) fault, we set LB to the cell that is closest to scan chain input and has been observed a 0 (1) on the Automatic Test Equipment (ATE). For example, if the observed values of a scan chain with SA0 fault are $11110100$, the LB is scan cell 3.

**Fig. 6 Output Checker for timing faults**

In Figure 6, the gate with a black dot constrains the minimum number of AOPs that should be found by DDPG. If the number is greater than or equal to a predefined value, such as $k$, signal "OC" is set to ‘1’ otherwise it is set to ‘0’.

Figure 7 Pick two available observation points from three reliable observation points.

In Figure 6, the gate with a black dot constrains the minimum number of AOPs that should be found by DDPG. If the number is greater than or equal to a predefined value, such as $k$, signal "OC" is set to ‘1’ otherwise it is set to ‘0’.

**4. Scan Chain Fault Diagnosis**

After the patterns are generated, we apply them on the ATE and collect the failure log file to run scan chain diagnosis. The whole diagnosis process has two phases described as follows.

1. Calculate load error probability for each scan cell.
2. Calculate the suspect score.

The load error probability can be calculated by equation (1). We use an example to explain equation (1). When we target scan cell $C_i$, $Pat_C$ can create three AOPs ($a, b, c$) on the faulty chain. Assume the loaded state of $C_i$ is fault-free, the expected machine responses of cells ($a, b, c$) is (1, 1, 1). If the loaded state of $C_i$ is faulty, the responses at ($a, b, c$) should lead us to observe (0, 0, 0) on ATE. Above calculation is based on system logic is fault-free. However, some system logic defects may contaminate the observed responses. For example, if there is one defect making cell $c$ captures a “1”, we will observe cells ($a, b, c$) = (0, 0, 1), then $\text{LEP}(C_i) = 2/3 = 0.67$. It means, using $Pat_C$, the probability of cell $C_i$ having an incorrect loading value is 0.67.

$$\text{LEP}(C_i) = \frac{\# \text{ of failure AOPs}}{\# \text{ of total AOPs}}$$

(1)

After we have a load error probability for each scan cell, we use the same technique proposed in [11] to plot a $p$-graph and to calculate the suspect score. The x-axis of the $p$-graph indicates the scan cell indexes and the y-axis represents LEP for every cell. The edge of the $p$-graph is where the load error probability is dramatically changed and the location of the edge is most likely the defect location. Equation (2) is employed to calculate a suspect score ($\text{suspect}_{i}$) for each cell $C_i$. A sliding window with size of $2r+1$ is defined for each cell $C_i$ at the center of the sliding window. $C_i$ is calculated with the LEP of scan cells from $i-r$ to $i+r$ using Equation (2). The maximum score of $\text{suspect}_{i}$ will be reported as the suspect scan cell. The fundamental principle of Equation (2) is based on a median filter. So any glitches on the $p$-graph that may lead to misdiagnosis are likely to be removed. According to the suspect score, a graph called suspect profile can be plotted whose peak point indicates the faulty location. Examples of $p$-graph and suspect profile will be given in the next section.

$$\text{Suspect}_{i} = 1 - \text{abs} \left( \frac{1}{2} - \frac{\sum_{c} \text{LEP}(C_c)}{2r+1} \right), \quad i \in \{0, 1, 2, \ldots, L - 1\}$$

(2)

**5. Experimental Results**

Experiments were performed on five real chip based ISCAS’89 benchmark circuits [15] and each benchmark is considered having two scan chains. We developed an ATPG tool based on a SAT engine [16]. Each targeted scan cell should have at least two AOPs, which means the custom-defined parameter $k$ is initially set to 2. Based on our experiences, all of the benchmark circuits can meet the $k=2$ experimental setup.

To evaluate the proposed diagnosis algorithm, we initially create simulated failure logs that are caused by scan chain faults alone. Next we randomly pick a gate from the system.
logic and inject a SA0 or SA1 fault to create failure log caused by compound defects.

Flt_type[1] = SA1;  Flt_type[2] = SA0;

For (each circuit) {
  For (chain = 1; chain <= 2; chain++) {
    For (cell = 0; cell < L; cell++) {
      For (type = 1; type <= 6; type++) {
        Inject one fault with Flt_type[type] at cell on chain;
        Run the proposed pattern generation algorithm;
        Simulate generated patterns to create failure log_A;
        Run the proposed diagnosis flow on fail log A;
        Randomly inject a SA1 or a SA0 system logic fault;
        Simulate generated patterns to create failure log B;
        Run the proposed diagnosis analysis flow on fail log B;
      }
    }
  }
}

Fig. 8 Experimental setup to evaluate Hit_Rate

For each circuit, we inject six types of chain faults shown in Table 1 into every scan cell, one at a time. The flow of experiment setup is shown in Figure 8.

We use the same definition of Hit_Rate [9]. If the reported suspects include the injected fault site, it is called a hit. We calculated the Hit_Rate under two conditions that system logic is fault-free and a SA0/SA1 fault is injected into the system logic. The Hit_Rate in both conditions are “1”.

Table 2. The number of faults injected into CUT when misdiagnosis happens

<table>
<thead>
<tr>
<th>CUT</th>
<th># of FTF faults</th>
<th># of FTR faults</th>
<th># of SA0 faults</th>
<th># of SA1 faults</th>
<th># of STF faults</th>
<th># of STR faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>s9234</td>
<td>18</td>
<td>14</td>
<td>7</td>
<td>38</td>
<td>38</td>
<td>17</td>
</tr>
<tr>
<td>s13207</td>
<td>16</td>
<td>18</td>
<td>39</td>
<td>42</td>
<td>31</td>
<td>14</td>
</tr>
<tr>
<td>s15850</td>
<td>19</td>
<td>22</td>
<td>31</td>
<td>9</td>
<td>39</td>
<td>15</td>
</tr>
<tr>
<td>s38417</td>
<td>45</td>
<td>46</td>
<td>30</td>
<td>41</td>
<td>35</td>
<td>43</td>
</tr>
<tr>
<td>s38584</td>
<td>29</td>
<td>47</td>
<td>34</td>
<td>59</td>
<td>42</td>
<td>44</td>
</tr>
</tbody>
</table>

Next, to evaluate the reliability of the proposed algorithms, we incrementally inject more SA0 or SA1 faults into the system logic of every circuit until a misdiagnosis happens. In Table 2, it shows the number of injected system logic faults that starts to make misdiagnosis of the injected chain fault. As shown in Table 2, for most of the circuits, the proposed diagnosis algorithm can tolerate more than a dozen of faults in the system logic. As an example illustrated in Figure 9, we show the p-graph and the suspect profile of s38584 when a FTR fault occurs on the scan cell C_{356}. The X-axis indicates the indexes of the scan cells and the Y-axis composes p-graph and the suspect score. The blue lines represent for p-graph while the red lines represent for suspect profile.

Table 3. Diagnostic resolutions (Average/Worst)

<table>
<thead>
<tr>
<th>CUT</th>
<th>SA0</th>
<th>SA1</th>
<th>FTF</th>
<th>FTR</th>
<th>STF</th>
<th>STR</th>
</tr>
</thead>
<tbody>
<tr>
<td>s9234</td>
<td>2.27/9</td>
<td>1.55/9</td>
<td>1.00/2</td>
<td>1.01/2</td>
<td>1.01/2</td>
<td>1.01/2</td>
</tr>
<tr>
<td>[5]</td>
<td>15.8/33</td>
<td>5.7/14</td>
<td>5.5/13</td>
<td>4.8/14</td>
<td>3.3/8</td>
<td>6.9/19</td>
</tr>
<tr>
<td>[11]</td>
<td>2.21/6</td>
<td>1.75/5</td>
<td>1.19/2</td>
<td>1.16/3</td>
<td>1.22/3</td>
<td>1.21/4</td>
</tr>
<tr>
<td>s13207</td>
<td>*</td>
<td>1.97/7</td>
<td>2.00/7</td>
<td>1.01/2</td>
<td>1.01/2</td>
<td>1.01/2</td>
</tr>
<tr>
<td>[11]</td>
<td>2.26/7</td>
<td>2.00/7</td>
<td>1.02/2</td>
<td>1.03/3</td>
<td>1.03/3</td>
<td>1.01/2</td>
</tr>
<tr>
<td>s15850</td>
<td>*</td>
<td>2.12/8</td>
<td>1.89/6</td>
<td>1.09/5</td>
<td>1.01/2</td>
<td>1.05/5</td>
</tr>
<tr>
<td>[5]</td>
<td>2.7/7</td>
<td>2.1/7</td>
<td>2.7/7</td>
<td>2.1/7</td>
<td>2.0/7</td>
<td>2.1/7</td>
</tr>
<tr>
<td>[11]</td>
<td>2.23/6</td>
<td>2.30/8</td>
<td>1.04/2</td>
<td>1.03/2</td>
<td>1.02/2</td>
<td>1.07/2</td>
</tr>
<tr>
<td>s38417</td>
<td>*</td>
<td>2.80/9</td>
<td>2.28/8</td>
<td>1.01/2</td>
<td>1.01/2</td>
<td>1.01/2</td>
</tr>
<tr>
<td>[11]</td>
<td>2.03/9</td>
<td>2.13/8</td>
<td>1.27/4</td>
<td>1.18/4</td>
<td>1.16/3</td>
<td>1.20/5</td>
</tr>
<tr>
<td>s38584</td>
<td>*</td>
<td>1.95/7</td>
<td>1.78/7</td>
<td>1.02/5</td>
<td>1.02/5</td>
<td>1.02/5</td>
</tr>
<tr>
<td>[5]</td>
<td>4.1/12</td>
<td>4.6/15</td>
<td>3.6/12</td>
<td>5.6/13</td>
<td>3.8/12</td>
<td>3.4/10</td>
</tr>
<tr>
<td>[11]</td>
<td>2.93/10</td>
<td>1.73/8</td>
<td>1.02/3</td>
<td>1.03/2</td>
<td>1.03/2</td>
<td>1.03/2</td>
</tr>
</tbody>
</table>

* This work
the system logic respectively. As shown in the p-graph, the variance of the amplitudes among all spikes is increased with the number of system logic defects, which illustrates more observation points are contaminated by the faults in the system logic. However, the peak still can be identified and pinpoint the fault location. We continuously inject more faults into the system logic. A misdiagnosis happened when the number of faults injected into the system logic reaches 47. As shown in Figure 9 (d), the impact of the faults in the system logic is so large that no peak value can be identified from suspect profile. In this condition, the suspect profile looks like white noise spectrum. Therefore, the diagnosis of faulty scan cell is failed.

Table 3 lists the average and worst diagnosis resolutions of the benchmark circuits, in the form of (Average/Worst). The average diagnosis resolution is the average number of suspect list identified among all scan cells in a circuit and the worst diagnosis resolution is the largest number of suspect list identified among all scan cells in a circuit [5]. No matter whether or not a fault is injected into the system logic, the diagnostic resolution is not degrade in our experiments and all faulty scan cells are successfully diagnosed. On average scale the proposed method outperforms the methods in [11] because the searching space is effectively limited.

The average pattern generation time of a scan cell is listed in Table 4. The pattern generation algorithms proposed in this work and in [11] are both run on a Xeon Linux workstation with 2GB memory. A SAT engine [16] is employed as the core of ATPG. We can see that the pattern generation time for a scan cell of our work ranges from 0.15s to 1.36s and most of average pattern generation time is less than 1 seconds, while the pattern generation time of the path oriented method proposed in [11] ranges from 0.15s to 14.47s.

### Table 4. Average pattern generation time (second) for a cell

<table>
<thead>
<tr>
<th>CUT</th>
<th>SA0</th>
<th>SA1</th>
<th>FTF</th>
<th>FTR</th>
<th>STF</th>
<th>STR</th>
</tr>
</thead>
<tbody>
<tr>
<td>s9323</td>
<td>4.48</td>
<td>1.09</td>
<td>0.44</td>
<td>0.44</td>
<td>0.49</td>
<td>0.15</td>
</tr>
<tr>
<td>s13207</td>
<td>1.85</td>
<td>1.01</td>
<td>0.07</td>
<td>0.41</td>
<td>0.44</td>
<td>0.43</td>
</tr>
<tr>
<td>s15850</td>
<td>3.15</td>
<td>1.04</td>
<td>1.19</td>
<td>0.66</td>
<td>1.15</td>
<td>0.39</td>
</tr>
<tr>
<td>s38417</td>
<td>3.83</td>
<td>4.93</td>
<td>0.83</td>
<td>0.86</td>
<td>0.69</td>
<td>1.60</td>
</tr>
<tr>
<td>s38584</td>
<td>1.47</td>
<td>2.43</td>
<td>0.32</td>
<td>0.35</td>
<td>0.67</td>
<td>0.46</td>
</tr>
</tbody>
</table>

* This work

### 6. Conclusions

In this paper, we proposed an observation point oriented scan chain Deterministic Diagnostic Pattern Generation (DDPG) technique that is primarily targeting compound defect diagnosis. The proposed algorithm tries to create as many failure bits as possible on reliable observation points when a loading error occurs on the targeted scan cell. Experimental results performed on ISCAS’89 benchmark circuits show that pattern diagnostic quality and pattern generation efficiency are both enhanced.

### References