Verification Pattern Generation for Core-Based Design Using Port Order Fault Model

Shing-Wu Tung
Computer and Communications Research Lab.
Industrial Technology Research Institute
Chutung, Hsinchu, Taiwan, R.O.C.
swtung@vlsi.ccl.itri.org.tw

Jing-Yang Jou
Dept. of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.
jyjou@bestmap.ee.nctu.edu.tw

Abstract

The lack of information about core’s internal structure is The designers must rely solely on the test set distributed by the core provider. Sometimes the stuck at fault (SAF) model and automatic test pattern generation (ATPG) are used to generate test vectors for those pre-defined blocks. However, a SAF test set could waste lots of time to verify the pre-verified internal structure of the cores. Therefore, in order to reduce the core-based design verification time, we should adopt the connectivity-based port order fault (POF) model instead of the stuck at fault model. In this paper, we compare the POF model with the SAF model and propose a method that the POF test set for functional verification can be generated by using the SAF-based ATPG tools with proper assignment of don’t care terms in inputs.

1. Introduction

The growth of the capacity of integrated circuits are likely to follow the prediction of Moore’s Law [1]. With advances in logic synthesis and integration of pre-designed cores, designing ASIC with more than 1 million gates is no longer impossible. However, to perform the design verification and testing on such VLSI circuits becomes a difficult task for engineers. The lack of information about a core’s internal structure makes the core-based design verification even harder. In order to generate proper verification patterns for core-based design, the stuck at fault (SAF) model and automatic test pattern generation (ATPG) tools are usually used. However, a SAF test set could waste lots of time to verify the pre-verified internal structure of the cores. So that the simulation and verification time will become the bottleneck of the entire design process. Therefore, in order to reduce the core-based design verification time, a connectivity-based port order fault (POF) model is proposed.

In this paper, we first define the three types of POFs and make a comparison between SAF and POF in Section 2. Iterative logic arrays (ILAs) such as RAM, ROM and datapath elements are typical pre-designed cores for core-based design. Section 3 uses an example to explain the constant testable (C-testable) patterns for ILA are not sufficient for detecting POF during core integration. Section 4 shows how POF test patterns can be generated using SAF based tools with proper assignment of don’t care terms in inputs. Experimental results are given in Section 5. Finally in Section 6 we summarize our results.

2. Preliminary

The stuck at fault (SAF) is the fault model that has been the most widely studied and used. Typically SAF assumes that a faulty signal line remains at a fixed logic level. But it does not accurately reflect the typical fault types of logic design. Based on our working experience, ten of the most likely happened design faults among the different views of all libraries are shown in figure 1. They are inconsistent port sequence, inconsistent pin name, inconsistent cell name, functional fault, inconsistent I/O type, syntax error, inconsistent timing data, primitive input without drive, floating nets and net with multiple active drivers.

On the other hand, the port order fault belongs to the group of pin-fault models. The POF assumes that a faulty cell has at least two I/O ports
misplaced. There are three types of port order faults [2].

**Definition 1.** The type I POF has at least one output $y_j$ misplaced with an input $x_k$, that is cell inputs $X' = \{x_1, \ldots, x_i, \ldots, y_j, \ldots, x_n, 1 \leq i \leq n, i \neq k\}$ and outputs $Y' = \{y_1, \ldots, y_i, \ldots, x_k, \ldots, y_m, 1 \leq i \leq m, i \neq j\}$.

**Definition 2.** The type II POF has at least two input ports misplaced, that is the inputs of the fault free cell is $X = \{x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n, i \neq j\}$ but the inputs of the faulty cell is $X' = \{x_1, \ldots, x_j, \ldots, x_k, \ldots, x_n, i \neq j\}$.

**Definition 3.** The type III POF has at least two output ports misplaced, that is the outputs of the fault free cell is $Y = \{y_1, \ldots, y_i, \ldots, y_j, \ldots, y_m, i \neq j\}$ but the outputs of the faulty cell is $Y' = \{y_1, \ldots, y_j, \ldots, y_k, \ldots, y_m, i \neq j\}$.

**Definition 4.** A design fault of a circuit $(K)$ is detectable if $F_{K'}(X) \neq F_{K'}(X)$ or $D_K(X) \neq D_K(X)$, where $X$ are inputs and $F_{K'}$ and $D_K$ represent the function and the delay of the corresponding faulty circuit.

**Definition 5.** A set of inputs which detect all possible POFs is called a complete POF test sequence.

Both SAF and POF are structural fault models. They both assume that components are fault-free and only the interconnections could be faulty. However the POF only considers the faults occur in I/O ports.

Depending on the number of faults can occur simultaneously in a circuit, the SAF is further classified into two classes: single stuck faults (SSF) and multiple stuck faults (MSF). For a circuit with $N_1$ I/O ports and $N_2$ internal nets, there are a total of $2(N_1 + N_2)$ SSFs and $3(N_1 + N_2) - 1$ MSFs.

The POF assumes the $N_2$ internal nets are fault free, therefore, there are a total of $N_1!$ possible POFs where $N_1! \div (N_1 - 1) \div 2$ are simple POF (only two ports are misplaced) and the others are complex POF (more than two ports are misplaced). According to the definition of POF, there are at least two ports misplaced for each type of port order faults. In other words, we don’t have to further classify the POFs into single or multiple faults.

### 3. Test Set for Adders

An ILA consists several identical cells with identical interconnections between cells. Adders, particularly ripple carry adders, are typical case of ILAs used in digital core-based designs. Due to their regularity, this type of circuit is easy of design and verification.

Based on the single combinational fault model, it has been proven that any ripple carry adder can be tested by a minimum test set of size 8 independent of the number of bits of the adder [3], namely, the ripple carry adder is C-testable. An example of a minimum test set is given in Table 1. However, the minimum test set can not detect the port order fault occurring at inputs of a 4-bit adder as shown in Figure 2. Bus signals being misplaced is a typical design fault in the integration stage of core-based designs. In order to detect those POFs, a test set of size $O(2 \times n)$ is needed where $n$ is the number of bits of the adder. An example of POF test set is given in Table 2.

The constant testable patterns are the best choice to verify the function of ILA. The verification time is significantly reduced as comparing to the exhaustive testing. However, it is not sufficient for detecting interconnection faults during core integration. Therefore, the linear testable patterns based on POF is needed to improve the quality of test set without significantly enlarge the verification time.
from the solution of \( f \) is the corresponding faulty functions of \( f \).

**Proof:**

The cofactor of \( f(x_1, x_2, \ldots, x_i, \ldots, x_n) \) with respect to variable \( x_i \) is \( f_{x_i} = f(x_1, x_2, \ldots, 1, \ldots, x_n) \). The cofactor of \( f(x_1, x_2, \ldots, x_i, \ldots, x_n) \) with respect to variable \( x_i \) is \( f_{x_i} = f(x_1, x_2, \ldots, 0, \ldots, x_n) \).

**Definition 6.** Boole’s expansion of a function \( f \), often called Shannon’s expansion, is defined as \( f(x_1, x_2, \ldots, x_i, \ldots, x_n) = x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}} \). [5]

**Lemma 1.** The test set to detect stuck-at 0 fault occurring at input port \( x_i \) is given by the solutions of the boolean equation \( x_i \cdot (f_{x_i} \oplus f_{\overline{x_i}}) = 1 \).

**Proof:** According to the well known definition of fault detection, a test \( t \) detects a fault \( f \) iff \( Z_f(t) \neq Z(t) \), that is \( Z_f(t) \oplus Z(t) = 1 \). Let the fault free function is expressed as \( Z = x_1 \cdot f_{x_1} + \overline{x_1} \cdot f_{\overline{x_1}} \), then the corresponding faulty functions of \( x_i \) stuck-at 0 is \( f_{x_i} \). The test to detect \( x_i \) stuck-at 0 is derived from the solution of

\[
Z \oplus Z_f = (x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}}) \oplus f_{x_i} = (x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}}) \cdot \overline{f_{x_i}} + (\overline{x_i} \cdot f_{\overline{x_i}}) \cdot (x_i + \overline{f_{x_i}}) = x_i \cdot f_{x_i} + x_i \cdot \overline{f_{x_i}} \cdot f_{x_i} = f_{x_i} \cdot (f_{x_i} \oplus f_{\overline{x_i}}) = 1
\]

The assignment of input values which satisfies the boolean equation \( x_i \cdot (f_{x_i} \oplus f_{\overline{x_i}}) = 1 \) is the test set. The first term \( x_i = 1 \) in the boolean equation activates the \( x_i \) stuck-at 0 fault and the solutions of the second term \( f_{x_i} \oplus f_{\overline{x_i}} = 1 \) setup the sensitized path to propagate the fault effect to output. Thus the lemma is proved.

**Lemma 2.** The test set to detect stuck-at 1 fault occurring at input port \( x_i \) is given by the solutions of the boolean equation \( x_i \cdot f_{x_i} \oplus f_{\overline{x_i}} = 1 \).

**Proof:** Similarly, as the proof of lemma 1, the test to detect \( x_i \) stuck-at 1 is derived from the solution of \( Z \oplus Z_f = 1 \), that is the assignment of input values which satisfies the boolean equation \( x_i \cdot f_{x_i} \oplus f_{\overline{x_i}} = 1 \) is the test set. Thus the lemma is proved.

**Lemma 3.** The test set to detect simple POF occurring at input ports \( x_i \) and \( x_j \) is given by the solutions of the boolean equation \( (x_i \oplus x_j) \cdot (f_{x_i} \oplus f_{\overline{x_j}}) = 1 \).

**Proof:** Let the fault free function is expressed as \( Z = x_i \cdot f_{x_i} + x_j \cdot f_{x_j} + x_i \cdot f_{\overline{x_j}} + x_j \cdot f_{\overline{x_i}} \), then the faulty function is \( Z_f = x_i \cdot f_{x_i} + x_j \cdot f_{x_j} + x_i \cdot f_{\overline{x_j}} + x_j \cdot f_{\overline{x_i}} \). The test to detect \( x_i \) and \( x_j \) port order fault is derived from the solution of

\[
Z \oplus Z_f = (x_i \cdot f_{x_i} + \overline{x_i} \cdot f_{\overline{x_i}}) \cdot (x_j \cdot f_{x_j} + \overline{x_j} \cdot f_{\overline{x_j}}) = x_i \cdot f_{x_i} \cdot f_{\overline{x_j}} + x_i \cdot f_{\overline{x_j}} \cdot f_{x_i} = (x_i \oplus x_j) \cdot (f_{x_i} \oplus f_{\overline{x_j}}) = 1
\]
Example: The complete SSF test set of a complex gate OA21(Z,A,B,C) with output function Z = (A+B) * C is given by all of the solutions that satisfy the boolean equations \( ACB = 1 \cup ABC = 1 \cup BC = 1 \cup C(A + B) = 1 \cup C(A + B) = 1 \). Therefore the complete SSF test set of OA21 is \{(A,B,C): 101, 001, 011, 111, 010, 100, 110\}.

Theorem: The simple POF can be detected by the complete SSF test set if the faulty ports are not functionally equivalent.

Proof: As the faulty ports \( x_i \) and \( x_j \) are not functionally equivalent, the cofactors \( f_{x_i \overline{x_j}} \) and \( f_{\overline{x_i} x_j} \) are different. Therefore there exist solutions for \( f_{x_i \overline{x_j}} \oplus f_{\overline{x_i} x_j} = 1 \). By applying the Shannon’s expansion of \( x_j \), the complete SSF test set for \( x_i \) is given by the solutions of boolean equation \( f_{x_i \overline{x_j}} = f_{x_i x_j} \cdot f_{\overline{x_i} x_j} + f_{x_i x_j} \cdot f_{\overline{x_i} \overline{x_j}} \cdot f_{\overline{x_i} \overline{x_j}} + f_{x_i x_j} \cdot f_{\overline{x_i} \overline{x_j}} \cdot f_{\overline{x_i} \overline{x_j}} = 1 \). The possible solutions come from the second and the third terms in the above boolean equation, \( f_{x_i x_j} \cdot f_{\overline{x_i} \overline{x_j}} + f_{x_i x_j} \cdot f_{\overline{x_i} \overline{x_j}} \cdot f_{\overline{x_i} \overline{x_j}} = 1 \), also satisfy the boolean equation in the lemma 3. Hence the simple POF between \( x_i \) and \( x_j \) are detectable by a subset of the complete SSF test set.

Therefore it is possible to generate the POF test with a SAF-based test generation tools by specifying the fault list of I/O SAFs. For example, the test set to detect POF between ports \( x_i \) and \( x_j \) can be derived from the solutions of \( \overline{x_i} \cdot (f_{x_i \overline{x_j}} \oplus f_{x_i x_j}) = 1 \cup x_i \cdot (f_{x_i \overline{x_j}} \oplus f_{x_i x_j}) = 1 \cup x_j \cdot (f_{x_i \overline{x_j}} \oplus f_{x_i x_j}) = 1 \).

However, a traditional SSF ATPG algorithm typically use a random pattern generator as the first step to generate low cost, fault independent tests then applies the fault-oriented algorithm to deal with those undetected faults. It does not guarantee that it can always find a solution which satisfy the equation in the lemma 3. By the way, the size of the complete SSF test set can be very large. In order to further reduce the solution space for POF test set generation, the traditional line justification process in SSF ATPG algorithm could be modified into the Algorithm 1 shown below. Thus the PIs with don’t care terms are assigned to the inverting value of the faulty input. Otherwise, an approximation method can be used to generate POF test sequence using SSF ATPG tools as shown in Algorithm 2. By specifying fault list of each input port to SSF ATPG tool one at a time, the test is guided to satisfy the term \( (x_i \oplus x_j) = 1 \) in the lemma 3.

Algorithm 1:

```
Justify(line, value)
{
    set line to value
    if line is a PI then return;
    /* else line is a gate output */
    c = controlling value of line;
    i = inversion of line;
    inval = value \oplus i;
    if (inval = c) then
        select one input j of line;
        Justify(j, c);
        for every input k of line other than j
            Justify(j, \overline{c});
    else
        for every input j of line
            Justify(j, \overline{c});
    end
}
```

Algorithm 2:

```
POF_test(I_list)
{
    input I_list; /* input port list */
    POF_test = NULL;
    foreach x_i in (I_list)
    {
        fault_list = \{x_i(s - a - 0, s - a - 1)\};
        t_{x_i} = SSF_ATPG(fault_list);
        /* The t_{x_i} is derived from setting the x_i bit in t_{x_i}, to zero */
        t_{\overline{x_i}} = t_{x_i} \oplus x_i;
        /* The POF_test is the concatenation of t_{x_i} */
        POF_test = POF_test + t_{x_i};
        + t_{\overline{x_i}};
    end
    end/* end of foreach x_i */
}
```

5. Experimental Results

The effectiveness of port order fault model for core-based design is demonstrated by generating verification test vectors for the ISCAS85 [4] benchmark circuits. Due to the huge number of all possible POFs, only the simple type II POFs are taken into consideration. It has been proven that the type II POF is dominated by the other two types of POFs [2]. The statistics of SSF and POF tests are shown in Table 3. The SSF tests are gen-
erated with a SSF-based ATPG tool by specifying the fault list of input SAFs. The POF tests are generated as shown in Algorithm 2 by the same SSF-based ATPG tool.

The results show as expected that the SSF tests do not detect all of the type-II POFs since those faults are not directly targeted by the SSF model. But the coverage of POF by SSF tests is quite high, ranging from 86% to over 98%. On the other side, due to the random assignment of unassigned inputs in the SSF ATPG tool we used, the coverage of POF test does not always achieve 100%; it is over 99% for all cases.

The complete POF test generation can be done by the following steps: (1) For each POF missed by the SSF tests, generate test patterns directly for those POFs. (2) The resulting test set is combined with the SSF tests to achieve 100% coverage of the detectable POFs.

The sizes of POF tests range from 166% to 963% of those of the SSF tests. On average, the POF tests are about 5 times as large as the SSF tests. The size of the SSF test is dependent on the internal structural of the circuit. For example, the circuits c499 and c1355 are functionally equivalent, but the SSF test size are 9 and 11 respectively. The only difference between c499 and c1355 is that all exclusive-or (XOR) gates of c499 have been expanded into their 4-NA2 gate equivalents (Figure 3) in c1355. However, the size of POF test is independent of the internal circuit structure, and it is proportional to the number of I/O ports. Thus the POF model is not only applicable in gate level circuits but also good for the higher level designs.

6. Conclusions

We have explored the relationship between SSF and POF, and shown how to generate the POF test set by SSF-based ATPG tools. Simulation and verification are becoming the bottleneck for the core-based system-on-a-chip designs. The core providers should guarantee the function of these pre-defined, pre-verified circuit blocks. The POF test set proposed in this paper can help system-on-a-chip designers focusing on verifying the core integration rather than verifying the core itself.

References


<table>
<thead>
<tr>
<th>vector</th>
<th>$c_0$</th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>11</td>
<td>00</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>$t_4$</td>
<td>1</td>
<td>00</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>$t_5$</td>
<td>1</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>$t_6$</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$t_7$</td>
<td>1</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 1. A minimum test set for ripple carry adders under the combinational fault model
Table 2. A port order fault test set for 4-bit adders

<table>
<thead>
<tr>
<th>Vector</th>
<th>$c_0$</th>
<th>$x_0/y_0$</th>
<th>$x_1/y_1$</th>
<th>$x_2/y_2$</th>
<th>$x_3/y_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>1</td>
<td>10</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>11</td>
<td>10</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>00</td>
<td>01</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>$t_4$</td>
<td>0</td>
<td>00</td>
<td>11</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>$t_5$</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>$t_6$</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>$t_7$</td>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

Table 3. The coverage of simple POF by SSF and POF tests for the ISCAS85 benchmarks.

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>No. of inputs</th>
<th>No. of simple POFs</th>
<th>No. of SSF tests</th>
<th>Coverage of POF by SSF tests</th>
<th>No. of POF tests</th>
<th>Coverage of POF by POF tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>c17</td>
<td>5</td>
<td>10</td>
<td>6</td>
<td>1.0000</td>
<td>10</td>
<td>1.0000</td>
</tr>
<tr>
<td>c432</td>
<td>36</td>
<td>630</td>
<td>26</td>
<td>0.9857</td>
<td>72</td>
<td>0.9968</td>
</tr>
<tr>
<td>c499</td>
<td>41</td>
<td>820</td>
<td>9</td>
<td>0.9878</td>
<td>82</td>
<td>1.0000</td>
</tr>
<tr>
<td>c880</td>
<td>60</td>
<td>1770</td>
<td>22</td>
<td>0.9808</td>
<td>120</td>
<td>0.9977</td>
</tr>
<tr>
<td>c1355</td>
<td>41</td>
<td>820</td>
<td>11</td>
<td>0.9768</td>
<td>82</td>
<td>0.9976</td>
</tr>
<tr>
<td>c9098</td>
<td>33</td>
<td>529</td>
<td>30</td>
<td>0.9886</td>
<td>45</td>
<td>0.9981</td>
</tr>
<tr>
<td>c2670</td>
<td>233</td>
<td>27028</td>
<td>48</td>
<td>0.8591</td>
<td>454</td>
<td>-</td>
</tr>
<tr>
<td>c3540</td>
<td>50</td>
<td>1226</td>
<td>21</td>
<td>0.9706</td>
<td>100</td>
<td>1.0000</td>
</tr>
<tr>
<td>c5315</td>
<td>178</td>
<td>15753</td>
<td>50</td>
<td>0.9751</td>
<td>356</td>
<td>0.9999</td>
</tr>
<tr>
<td>c6288</td>
<td>32</td>
<td>496</td>
<td>7</td>
<td>0.9254</td>
<td>62</td>
<td>1.0000</td>
</tr>
<tr>
<td>c7552</td>
<td>207</td>
<td>21321</td>
<td>43</td>
<td>0.9447</td>
<td>414</td>
<td>0.9952</td>
</tr>
</tbody>
</table>