On Generation of The Minimum Pattern Set for Data Path Elements in SoC Design Verification Based on Port Order Fault Model

Chun-Yao Wang, Shing-Wu Tung, and Jing-Yang Jou
[*]*Department of Electronics Engineering*
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.
{*wcyao,swtung,jyjou*}*@eda.ee.nctu.edu.tw*

Abstract

Embedded cores are being increasingly used in the design of large System-on-a-Chip (SoC). Because of the high complexity of SoC, the design verification is a challenge for system integrator. To reduce the verification complexity, the port order fault (POF) model proposed in [1] has been used for verifying core-based designs and the corresponding verification pattern generation have been developed [2] [3]. Adders and multipliers are the most often used data path elements in core-based designs. Due to their regularity, the development of the verification pattern sets can be achieved in a systematic method. In this paper, we present the algorithms of generating the minimum verification pattern sets for adders and multipliers and these pattern sets are much smaller than that obtained from the automatic verification pattern generation (AVPG) proposed in [3].

1. Introduction

Spurred by process technology leading to the availability of more than 1 million gates per chip, and more stringent requirements upon time-to-market and performance constraints, system level integration and platform-based design [4] are evolving as a new paradigm in system designs. A multitude of components that are needed to implement the required functionality make it hard for a company to design and manufacture an entire system in time and within reasonable cost. Hence, design reuse and reusable building blocks (cores) trading are becoming popular in the SoC era. However, present design methodologies are not enough to deal with cores which come from different design groups and are mixed and matched to create a new system design. In particular, verifying whether a design satisfies all requirements is one of the most difficult tasks.

[*]This work was supported in part by ROC National Science Council under Grant NSC90-2218-E-009-039

Verification is a process used to demonstrate the functional correctness of a design. **Testing** is a process that verifies whether the design was manufactured correctly. Fig. 1 shows the reconvergent paths model for both verification and testing [5]. The purpose of the verification is to ensure that a design meets its functional intent. But during testing, the finished silicon is reconciled with the netlist that was submitted for manufacturing. Therefore, when a design is claimed to be fully tested, i.e., 100% fault coverage, under a fault model, such as stuck at fault (SAF) model, that means it is manufactured correctly. However, designers still cannot guarantee that the chip satisfies the design specification if they do not verify it properly before manufacturing. The chip may be a manufactured correctly but designed incorrectly chip. Thus, designers spend about 70% of their efforts to the verification. But design verification is still on the critical path of the design flow [5].

Usage of cores divides the IC design community into two groups: **core providers** and **system integrators**. In traditional System-on-Board (SoB) design, the components that go from providers to system integrators are ICs, which are designed, verified, manufactured and tested. The system integrator verifies the design by using these components as **fault free building blocks**. SoB verification is limited to detecting faults in the interconnection among the components. Similarly, in System-on-a-Chip (SoC) design, the components are cores. The system integrator verifies the design by using the cores as **design error free building blocks**. **Based on this assumption, SoC verification could be focused on detecting the misplacements of the interconnection among the cores as the first step**. This higher level of abstraction decreases the complexity of design verification on a system chip and reduce the time on design verification of the entire system.

The focus of core-based design verification should be on

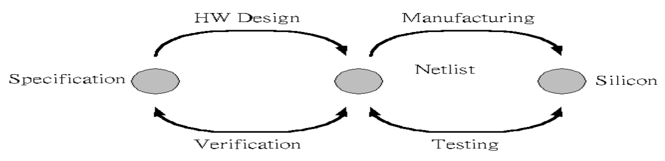

Figure 1. Reconvergent paths model for both verification and testing

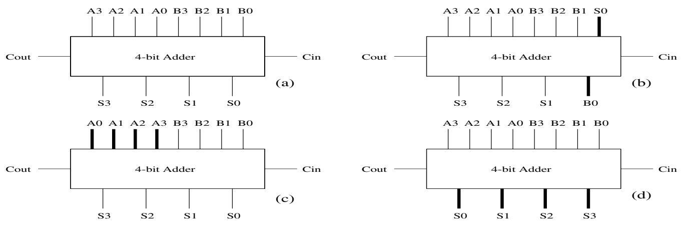

Figure 2. POF model

how the cores communicate with each other [6]. By creating the testbenches at a higher level, a connectivity-based design fault model, **p**ort **o**rder **f**ault (**POF**), proposed in [1] is used for reducing the time on core-based design verification [2] [3].

In [2], Tung et al. proposed a verification pattern generation algorithm based on the POF model. However, the algorithm generates the verification pattern set only for detecting the simple POF (SPOF) (two ports misplaced at a time). This simplified model is not enough to deal with all possible misplacements occurred in a real design during core integration phase.

In [3], Wang et al. proposed the AVPG algorithm to generate the verification pattern set for detecting all possible misplacements among the ports of the cores. However, this AVPG algorithm is developed for targeting general random logic functions, therefore, the generated verification pattern sets for adders and multipliers are not minimized.

In this paper, we present the algorithms of generating the minimum verification pattern sets for adders and multipliers and these pattern sets are much smaller than that obtained from the AVPG proposed in [3].

The remainder of this paper is organized as follows. The POF model is introduced in Section 2. Section 3 describes the mechanism of conducting POF verification. The verification pattern sets for adders and multipliers are presented in Section 4. Section 5 concludes the paper.

2. Preliminary

The POF model assumes that a faulty cell has at least two I/O ports misplaced. It also assumes that the components are fault free and only the interconnection among the components could be faulty. There are three types of POFs [1].

Definition 1: The type I POF is at least one output misplaced with an input. The type II POF is at least two inputs misplaced. The type III POF is at least two outputs misplaced.

Example 2.1: A fault free 4-bit adder is shown in Fig. 2(a). The function of the adder is $\{Cout, S(3:0)\} = A(3:0) + B(3:0) + Cin$. An example of the type I POF is shown in Fig. 2(b). Input $B0$ is misplaced with output $S0$. Fig. 2(c) shows an example of the type II POF. Input $A(3:0)$ are misplaced. Fig. 2(d) shows an example of the type III POF. Output $S(3:0)$ are misplaced.

It has been proven that the type II POF **dominates** the type I and III POFs [1]. Therefore, in this paper, we consider the **type II POF** only.

Definition 2: A **port sequence** is an input port numbers permutation which indicates the relative positions among these input ports. The **fault free port sequence** is a port sequence that none of the input ports is misplaced. An **equivalent POF** is a port sequence which evaluates to the same outputs as fault free port sequence after applying any patterns.

The input ports of an n-bit adder/multiplier are numbered from $1 \sim 2n$, where **1 $\sim$ n for augend/multiplicand**, and **n+1 $\sim$ 2n for addend/multiplicator**.

3. Integration Verification

In this section, we introduce IEEE P1500 [7], which is a standard under development and is used for embedded core testing, to reduce the complexity of design verification.

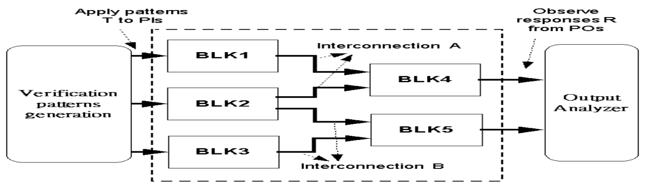

Figure 3. Generic verification scheme

Fig. 3 depicts a generic verification scheme for the core-based system chip. Since these cores, BLK1 $\sim$ BLK5, are pre-verified, the verification efforts during the integration phase should be focused on the interconnection among the cores. To verify the interconnection among the BLK1 $\sim$ BLK5, designers apply the patterns T to primary inputs (PIs) of the integrated design, then compare the responses R to the expected results in primary outputs (POs). If the responses R are inconsistent with the expected ones, some interconnection are misplaced. The generation of the patterns T depends on the functionalities of BLK1 $\sim$ BLK5.

As the complexity of cores increase or more cores are involved in the SoC integration, the patterns T become harder to generate.

To conquer this problem, we exploit the technique of design for testability (DFT) to conduct verification. The solution is **IEEE P1500 standard for embedded core test** (SECT). IEEE P1500 SECT is a standard under development that aims at improving ease of reuse and facilitating interoperability with respect to the test of core-based chips. The most important component in this standard is the **P1500 wrapper**. It is a thin shell around the core that provides the switching capability between the core and its various access mechanisms. A straight forward core integration methodology is proposed in the following paragraphs. For the cores to be integrated into a system, we first sort them in topological order from PIs to POs. The entire system is integrated block-wise and follows the topological order. As a block is added into the system, the verification patterns for the added block are generated and applied to the integrated system for verification.

The P1500 wrapper was proposed with a few pre-defined operations , such as core-internal test, core-external test, bypass, isolation and normal modes. To verify the interconnection among the circuit under verification (CUV) and its predecessors, the CUV is set in normal mode which allows the CUV to function in its normal system operation. The predecessors connected to the CUV **directly** are set in external test mode which allow verifying the interconnected wiring between cores via the ordinary input/output ports in the core wrappers. The other predecessors of the CUV are all set in bypass mode which allow the stimuli being bypassed through cores to the CUV.

For example, assume the topological order of the embedded cores are BLK1 $\sim$ BLK5 as shown in Fig. 3. In the beginning, the BLK1 $\sim$ BLK3 are added into the system. Since these blocks do not have any predecessors, it is not necessary to conduct the POF verification. As the BLK4 is added into the system, the BLK1 and BLK2 are the predecessors that directly connected to it. In order to verify the interconnection A among these blocks, the BLK4 is set in normal mode, and the BLK1 and BLK2 are set in external test mode to propagate the POF stimuli from PIs through the wrappers (of BLK1 and BLK2) to the inputs of the BLK4 as shown in Fig. 4. Hence, the verification patterns can easily go through the system from PIs to POs and verify the interconnection A. If there are any misplacements in the interconnection A, the inconsistent results will be observed in the output analyzer. The integration of the other blocks follows the same procedure to verify the interconnection. **This verification mechanism allows us solely focusing on the functionality of the added block when generating the verification pattern set and reduce the complexity of POF verification**.

By using the P1500 test structure for POF verification, we do not introduce any more hardware overhead in the chip implementation. In fact, we reuse the hardware overhead incurred in the testing phase.

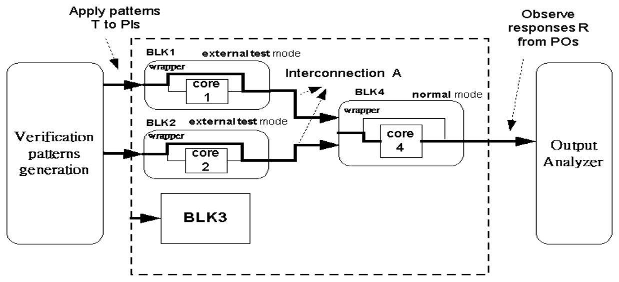

Figure 4. POF verification when integrating the BLK4

4. The Minimum Verification Pattern Sets for Data Path Elements

4.1. Undetected Port Sequence (UPS) Representation

The total number of POFs in an N-input core is N!-1. This number grows rapidly when N increases, for instance, as N=69, N!-1$\approx$1.7$\times$10^{98}. Therefore, an implicit representation is used to indicate the undetected port sequences (UPSs).

Example 4.1 is used to demonstrate the implicit representation for the UPSs.

Example 4.1: Given an 8-input core, the input ports are numbered from 1 to 8. The UPSs representation (12345678) represents the UPSs that caused by all possible misplacements among the port numbers in the same group, i.e., port 1 to port 8. The number of undetected POFs is 8!-1, and the 1 in the 8!-1 accounts for the fault free port sequence. The UPSs representation (125)(4)(3678) indicates the UPSs that caused by all possible misplacements among the port numbers 1, 2 and 5 and/or all possible misplacements among the port numbers 3, 6, 7 and 8. The number of the undetected POFs is 3!$\times$1!$\times$4!-1. The UPS representation (1)(2)(3)(4)(5)(6)(7)(8) represents 8!-1 POFs are all detected.

4.2. The Minimum Verification Pattern Set for Adders: TA_{2n}

We propose an algorithm to generate a pattern set which detects all nonequivalent POFs in an adder. The pseudocode of this algorithm, Generation_A, is shown in Fig. 5.

Before the explanation of the Generation_A algorithm, we state two definitions used in this algorithm first.
Definition 3: A^i_{2n} is the pattern with length $2n$ generated by the i^{th} iteration of the Generation_A algorithm.
Definition 4: TA_{2n} is the pattern set generated by the Generation_A algorithm. It consists of A^i_{2n}, where i=1 $\sim \lceil log_2 n \rceil$. The size of TA_{2n} is the number of patterns in TA_{2n} and is denoted as $|TA_{2n}|$. $|TA_{2n}|$= $\lceil log_2 n \rceil$.

```
Algorithm: Generation_A
Input: $n$($n$ + $n$ adder)
Output: $TA_{2n}$
{
    $TA_{2n} \leftarrow \emptyset$;
    t$\leftarrow \lceil log_2 n \rceil$;
    FOR (i=1 to t)
    {
        $a_1 \leftarrow \frac{2^t}{2^i}$ consecutive 1s;
        $a_2 \leftarrow \frac{2^t}{2^i}$ consecutive 0s;
        $a_{12} \leftarrow a_1$ concatenates $a_2$;
        $A^i_{2n} \leftarrow a_{12}$ replicates $2^i$ times;
        $if(n \neq 2^t)$ /* if $n$ is not equal to power of 2 */
            $A^i_{2n} \leftarrow (1 \sim n, 2^t + 1 \sim 2^t + n)^{th}$ bits of $A^i_{2n}$ ;
        $TA_{2n} \leftarrow TA_{2n} \cup A^i_{2n}$;
    }
    return($TA_{2n}$);
}
```

Figure 5. The pseudo-code of Generation_A algorithm.

In this algorithm, the input is n and the output is the verification pattern set TA_{2n}. For each iteration i, it generates a pattern A^i_{2n} and then puts it into the verification pattern set TA_{2n}. We illustrate this algorithm with Example 4.2.
Example 4.2: Given a 4-bit adder where n=4 and t=$\lceil log_2 4 \rceil$=2, there are two iterations. In the 1^{st} iteration, i=1, $\frac{2^t}{2^i} = \frac{2^2}{2^1} = 2$, a_1=11, a_2=00, a_{12}=1100, 2^i=2, replicates a_{12} two times to form A^1_8, A^1_8=**11001100**. In the 2^{nd} iteration, i=2, $\frac{2^t}{2^i} = \frac{2^2}{2^2} = 1$, a_1=1, a_2=0, a_{12}=10, 2^i=4, replicates a_{12} four times to form A^2_8=**10101010**. **TA$_8$** = {**11001100**, **10101010**}.

Lemma 1. *To activate a POF caused by the port misplacement of the port x and the port y denoted as POF(x, y), the assignments of x and y have to be different, either x=0 and y=1 or vice versa.*

Lemma 2. *The minimum number of patterns for activating all port sequences in an n-bit adder/multiplier is $\lceil log_2 2n \rceil$.*

Lemma 3. *For a 2n-input core, assume the POF(x, y) can be detected by a pattern with m bits 1s and (2n-m) bits 0s, then this pattern can actually detect m! $\times$ (2n-m)! POFs in total. This characteristic is called the* **domination property** *of a POF pattern.*

TA $_8$	Remaining UPSs	
A^1_8 11001100	(1256)(3478)	(15)(26)(37)(48)
A^2_8 10101010		

Figure 6. TA_8 and the remaining UPSs

Example 4.3: For a 5-input core, assume the verification pattern 11000 detects the POF(2,3) and the output of the pattern 11000 is A. Because the verification pattern 1**10**00 detects POF(2,3), the output of the verification pattern 1**01**00 must not be A (assume it is B). The additional misplacements among the 0 ports or between the 1 ports in 10100 make the pattern 10100 intact and the output is still B. Therefore, these additional misplacements combined with POF(2,3) are all detected by 11000, and the amount of them are 2! $\times$ 3!.

Lemma 4. *The equivalent POFs of an n-bit adder occurred only at the misplacements of the same weight bits.*

Definition 5: AP^m_{2n} is a pattern with length $2n$, these bits are numbered from $1 \sim 2n$, bit $1 \sim m$ and bit $n+1 \sim n+m$ are 1s, the others are 0s where $1 \leq m < n$.

Lemma 5. *When applying AP^m_{2n} into an adder, after any misplacements of 0 ports with 1 ports in AP^m_{2n}, the output of the adder will be different with the original one.*

Example 4.4: Given a 4-bit adder, AP^1_8=10001000, AP^2_8=11001100, and AP^3_8=11101110. When applying each AP^i_8 into the adder, if any misplacements of 0 ports with 1 ports are occurred in the AP^i_8, the output of the adder will be different with that of the fault free port sequence.
Theorem 1: *TA_{2n} is the minimum pattern set that can detect all nonequivalent POFs in an n-bit adder.*
Example 4.5: Given a 4-bit adder and the verification pattern set TA_8 shown in the 1^{st} column of Fig. 6. We want to illustrate that TA_8 can detect all nonequivalent POFs in a 4-bit adder and it is the minimum pattern set. First, we apply the pattern A^1_8, 11001100, into the adder. Assume the adder is fault free, the port sequence is 12345678 and the outputs is 11000 as shown in Fig. 7(a). This means "1100 + 1100 = 11000". However, if the port 2 and port 3 are misplaced with each other, the port sequence of the adder becomes 1**32**45678. When we apply the same pattern A^1_8 into it, the real pattern assigned into it is 1**01**01100, this makes the output become 10110 (1010 + 1100 = 10110)

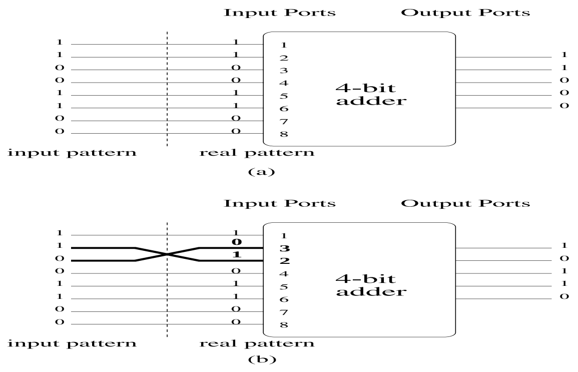

Figure 7. The fault free and faulty port sequences

as shown in Fig. 7(b). Because the fault free output **11000** and the faulty output **10110** are different, this pattern A_8^1 (11001100) can detect the port sequence 1**32**45678. According to Lemma 3, the additional port misplacements among the 0 ports and/or among the 1 ports in the 10101100 make the pattern 10101100 intact and all of them are detected by A_8^1. Similarly, if the port 2 and port 4 are misplaced with each other, the port sequence of the adder becomes 1**4**3**2**5678. When the A_8^1 is applied into the adder again, the real pattern assigned into it is 1**001**1100 and the output is different with the fault free output, too. Actually, according to Lemma 5, the port misplacements of 0 ports with 1 ports in the A_8^1 will cause the output different. Therefore, these port misplacements will be detected. As a result, the undetected port sequences (UPSs) are the port misplacements that cannot be distinguished from the outputs and they are the misplacements of the same assignment ports. They can be written as (1256)(3478) in the implicit UPSs representation, which is shown in the Fig. 6. For applying the pattern A_8^2, 10101010, since the updated remaining UPSs are (1256)(3478), we only focus on the possible misplacements in each group of the remaining UPSs. The port assignments of each group in the A_8^2 are AP_4^1. Therefore, according to Lemma 5, the misplacements of 0 ports with 1 ports in each group can cause the output different. As a result, the remaining UPSs after applying A_8^1 and A_8^2 are (15)(26)(37)(48) and are listed in the Fig. 6. Furthermore, according to Lemma 4, these remaining UPSs are exactly the equivalent POFs. Thus, TA_8 can detect all nonequivalent POFs in a 4-bit adder. TA_8 is the minimum pattern set as well. To detect all port sequences in an adder, these port sequences have to be activated first. According to Lemma 1, 2, the minimum number of patterns for activating all port sequences is $\lceil log_2 2n \rceil$=$\lceil log_2 8 \rceil$=3. However, according to Lemma 4, the misplacements of same weight bits are equivalent POFs, therefore, the pattern that only activates the equivalent POFs is removed and the minimum number of patterns is reduced to $\lceil log_2 2n \rceil - 1$ =$\lceil log_2 n \rceil$=2. This number is equal to $|TA_8|$.

4.3. The Minimum Verification Pattern Set for Multipliers: TM_{2n}

Definition 6: M_{2n}^i is the pattern with length $2n$ generated by the i^{th} iteration of the Generation_M algorithm.
Definition 7: TM_{2n} is the pattern set generated by the Generation_M algorithm. It consists of M_{2n}^i, where i=1 $\sim$ $\lceil log_2 2n \rceil$. The size of TM_{2n} is the number of patterns in TM_{2n} and is denoted as $|TM_{2n}|$. $|TM_{2n}|$= $\lceil log_2 2n \rceil$.

```
Algorithm: Generation_M
Input: $n$($n \times n$ multiplier)
Output: $TM_{2n}$
{
    $TM_{2n} \leftarrow \emptyset$;
    t$\leftarrow$ $\lceil log_2 2n \rceil$;
    FOR (i=1 to t)
    {
        $m_1 \leftarrow \frac{2^t}{2^i}$ consecutive 1s;
        $m_2 \leftarrow \frac{2^t}{2^i}$ consecutive 0s;
        $m_{12} \leftarrow m_1$ concatenates $m_2$;
        $M_{2n}^i \leftarrow m_{12}$ replicates $2^{i-1}$ times;
        $if(n \neq 2^{t-1})$ /* if $n$ is not equal to power of 2 */
            $M_{2n}^i \leftarrow (1 \sim n, 2^{t-1}+1 \sim 2^{t-1}+n)^{th}$ bits of $M_{2n}^i$ ;
        $TM_{2n} \leftarrow TM_{2n} \cup M_{2n}^i$;
    }
    return($TM_{2n}$);
}
```

Figure 8. The pseudo-code of Generation_M algorithm.

Example 4.6: Given a 4-bit multiplier where n=4 and t=$\lceil log_2 8 \rceil$=3, there are three iterations. In the 1^{st} iteration, i=1, $\frac{2^t}{2^i}$ =$\frac{2^3}{2^1}$ = 4, m_1=1111, m_2=0000, m_{12}=11110000, 2^{i-1}=1, M_8^1=**11110000**. In the 2^{nd} iteration, i=2, $\frac{2^t}{2^i}$ =$\frac{2^3}{2^2}$ = 2, m_1=11, m_2=00, m_{12}=1100, 2^{i-1}=2, replicates m_{12} two times to form M_8^2=**11001100**. In the 3^{nd} iteration, i=3, $\frac{2^t}{2^i}$ =$\frac{2^3}{2^3}$ = 1, m_1=1, m_2=0, m_{12}=10, 2^{i-1}=4, replicates m_{12} four times to form M_8^3=**10101010**. TM_8=$\{$**11110000**, **11001100**, **10101010**$\}$.

Lemma 6. *The equivalent POFs of an n-bit multiplier occurred only at the interchange of the multiplicand and multiplicator.*

Lemma 7. *Given four positive integers, A, B, C, and D. The output of A $\times$ B is different with that of C $\times$ D if A $>$ C and B $>$ D or if A = C and B $>$ D or if A $>$ C and B = D.*

Table 1. The comparisons between our approach and AVPG in verification pattern set size

			adder		multiplier	
n bits	# of POFs	# of exhaustive patterns	$\|TA_{2n}\|$	AVPG	$\|TM_{2n}\|$	AVPG
n	$(2n)! - 1$	$2^{(2n)}$	$\lceil log_2 n \rceil$	$2n-2$	$\lceil log_2 2n \rceil$	$2n-2$
2	23	16	1	2	2	2
4	40319	256	2	6	3	6
8	$\approx 2.1 \times 10^{13}$	65536	3	14	4	14
16	$\approx 2.6 \times 10^{35}$	$\approx 4.3 \times 10^{9}$	4	30	5	30
32	$\approx 1.3 \times 10^{89}$	$\approx 1.8 \times 10^{19}$	5	62	6	62

Theorem 2: *TM_{2n} is the minimum pattern set that can detect all nonequivalent POFs in an n-bit multiplier.*

Example 4.7: Given a 4-bit multiplier and the verification pattern set TM_8 shown in the 1^{st} column of Fig. 9. We explain Theorem 2 by using the same procedure as we did in Example 4.6. According to Lemma 6, the equivalent POF of this 4-bit multiplier is 56781234. After applying M_8^1, 11110000, except the equivalent POF, the misplacements of 0 ports with 1 ports in the 11110000 will cause the outputs different and will be detected. Therefore, the remaining UPSs are (1234)(5678) and 56781234. Since the updated remaining UPSs are (1234)(5678) and 56781234, we only focus on the possible misplacements in each group of the remaining UPSs. When we applying M_8^2, 11001100, into the multiplier, the port misplacements of 0 ports with 1 ports in the (1234) and (5678) will cause the multiplicand and multiplicator smaller, and the results of multiplication will be different according to Lemma 7. As a result, the updated remaining UPSs are (12)(34)(56)(78) and 56781234. Similarly, after applying M_8^3, 10101010, the remaining UPSs are (1)(2)(3)(4)(5)(6)(7)(8) and 56781234 as shown in the Fig. 9. The (1)(2)(3)(4)(5)(6)(7)(8) is the fault free port sequence and the 56781234 is the equivalent POF. Thus, TM_8 can detect all nonequivalent POFs in a 4-bit multiplier. TM_8 is also the minimum pattern set. The reason is as same as stated in Example 4.6. The minimum number of patterns for activating all port sequences is $\lceil log_2 2n \rceil$ and these activated port sequences in a multiplier are also detected. Thus, TM_8 is the minimum pattern set that can detect all nonequivalent POFs in an n-bit multiplier.

TM $_8$	Remaining UPSs		
M $^1_8$ 11110000	(1234)(5678) 56781234		
M $^2_8$ 11001100		(12)(34)(56)(78) 56781234	
M $^3_8$ 10101010			(1)(2)(3)(4)(5)(6)(7)(8) 56781234

Figure 9. TM_8 and the remaining UPSs

4.4. Summary

We have proposed two algorithms to generate the minimum verification pattern sets TA_{2n}, TM_{2n} to detect all nonequivalent POFs in adders and multipliers, respectively. Table 1 lists the number of POFs and the corresponding verification pattern set size in an n-bit adder/multiplier. We find that the growth of the number of POFs is much faster than that of required verification pattern set size. For example, as n=32, the number of POFs approximately reaches 1.3 $\times 10^{89}$, but $|TA_{64}|$ is only 5 and $|TM_{64}|$ is only 6. Furthermore, the $|TA_{2n}|$ and $|TM_{2n}|$ are much smaller than the results obtained by AVPG [3]. It minimizes the pattern set size from linearity to logarithm. Therefore, TA_{2n} and TM_{2n} are very efficient in detecting all possible misplacements occurred among adders/multipliers and other blocks.

5. Conclusions

Taking advantages of the regularity of adders and multipliers and using the domination property of a POF pattern, we present the minimum verification pattern sets of adders and multipliers for detecting all nonequivalent POFs, and these pattern sets are much smaller than that obtained by [3].

References

[1] S.-W. Tung, and J.-Y. Jou, "A logic fault model for library coherence checking," *Journal of Information Science and Engineering*, pp.567-586, Sep. 1998.

[2] S.-W. Tung, and J.-Y. Jou, "Verification pattern generation for core-based design using port order fault model," *in Proc. Asian Test Symposium*, pp.402-407, Dec. 1998.

[3] C.-Y. Wang, S.-W. Tung, and J.-Y. Jou, "An AVPG for SoC design verification with port order fault model," *in Proc. IEEE International Symposium on Circuits And System*, pp.V259-V262, May. 2001.

[4] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, "Surviving the SOC revolution - a guide to platform-based design," Norwell, Massachusetts, Kluwer Academic Publishers, 1999.

[5] J. Bergeron, "Writing testbenches-functional verification of HDL model," Norwell, Massachusetts, Kluwer Academic Publishers, 2000.

[6] J. A. Rowson, and A. Sangiovanni-Vincentelli, "Interface-based design," *in Proc. Design Automation Conference*, pp.178-183, Jun. 1997.

[7] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel, "Towards a standard for embedded core test:An example," *in Proc. IEEE International Test Conference*, pp.616-627, Sep. 1999.