Test Generation for Designs with Multiple Clocks

Xijiang Lin and Rob Thompson

Mentor Graphics Corp.
8005 SW Boeckman Rd.
Wilsonville, OR 97070

Abstract

To improve the system performance, designs with multiple clocks have become more and more popular. In this paper, several novel test generation procedures are proposed to utilize multiple clocks in the design effectively and efficiently in order to dramatically reduce test pattern count without sacrificing fault coverage or causing clock skew problem. This is achieved by pulsing multiple non-interactive clocks simultaneously and applying a clock concatenation technique. Experimental results on several industrial circuits show significant test pattern count reduction by using the proposed test generation procedures.

Categories and Subject Descriptors

B.8.1 Reliability, Testing, and Fault-Tolerance

General Terms

Algorithms, Design

Keywords

ATPG, Clock Domain, Scan Design

1 Introduction

To test full scan designs, traditional Automatic Test Pattern Generation (ATPG) tools will generate basic test patterns which only allow to pulse one clock between scan loading and unloading in order to avoid clock skew problem caused by pulsing several clocks simultaneously. In today's VLSI designs, multiple clocks have been extensively used to improve the system's performance. Therefore, if the basic test patterns generated by the traditional ATPG tool are still used to test these designs, the number of test patterns required to achieve the desired fault coverage will increase significantly. As the test pattern count has great impact on both the test application time and the memory requirement of the tester, it is desired that the ATPG tool can take the multiple clocks in the design into account and use them in an effective way to reduce the test pattern count dramatically as well as to avoid any clock skew problem.

DAC 2003, June 2-6, 2003, Anaheim, California, USA.

In order to reduce the test pattern count in designs with multiple clocks, there exist two approaches[1]:

- *Clock domain analysis*: This approach utilizes the non-interactive relations among different clocks by analyzing clock domains in the design. A group of clocks belongs to the same clock domain if there is no functional path among the state elements captured by the different clocks in the group. Since the clocks in the same clock domain can be pulsed simultaneously without causing clock skew problem, the traditional ATPG tool can apply all of those clocks together during the test generation after defining them as equivalent clocks. Because faults needed to be captured by the different clocks are possibly tested by the same test pattern, the test pattern count can be reduced. Since the clocks in the same clock domain are defined as equivalent, this approach does not utilize the non-interactive relations among the clocks effectively. Consider a design with three clocks, {*clk1*, *clk2*, *clk3*}. If any pair of clocks can be pulsed simultaneously without causing clock skew problem, defining *clk1* and *clk2* as equivalent will forbid pulsing clocks *clk1* and *clk3* together as well as *clk2* and *clk3* together during test generation. As a result, it may increase the test pattern count.
- *Clock concatenatio*n: In this approach, different clocks are pulsed sequentially between scan loading and unloading such that the generated test pattern will include more than one cycle and only one clock is pulsed in each cycle. Since no overlap exists between the applied clocks, there is no risk of clock skew during data captures. Because the test data volume is dominated by the scan data, and the test application time is dominated by the scan loading and unloading operation, the cost of storing and applying one multiple cycle test pattern is very close to that spent applying a test pattern with single cycle. Compared with pulsing one clock per test pattern, the observation points in the different clock domains are used more efficiently, such that the generated test pattern count can be reduced. The drawbacks of this approach are that both sequential test generator and sequential fault simulator are required and the test generation time will be much longer than that taken by the

first approach. However, if all the clocks in the design interact with each other, this approach is the only way to reduce the test pattern count without requiring to mask out control points and observation points.

To reduce the test generation time in the second approach, the combinational test generator and the fault simulator are enhanced in [3] to generate test patterns that pulse clocks sequentially. To guarantee a safe capture behavior, the state elements that have potential clock skew problem are masked out as neither control point nor observation point. Due to losing controllability and observability of some scan cells, this approach can achieve better performance when the interaction among different clocks in the design is minimum.

In this paper, we focus on a test generation processes that efficiently generate highly compact test pattern sets for designs with multiple clocks based on the two approaches described above. When utilizing clock domain analysis, the proposed test generation procedure does not require defining clocks in the same domain as equivalent. Instead, we create a clock interaction table to guide the test generator to avoid clock skew problem among the different clocks dynamically. As a result, we could implicitly enumerate all the possible combinations of non-interactive clocks and maximize the number of clocks to be pulsed together, as well as eliminate unnecessary clock pulsing. When generating a compact test pattern set by applying clock concatenation, the fault effects captured by previous clocks must not be disturbed by the clocks applied subsequently. Besides enhancing test generator to achieve this goal, we also use design rule checking to determine which group of clocks can be applied sequentially and identify the scan cells that could be disturbed by more than one clock in advance, in order to reduce test generation complexity.

The paper is organized as follows. In Section 2, we describe the criteria to identify the interaction among the different clocks in the design and give the test generation procedure by applying the clock interaction table derived from the clock domain analysis. In Section 3, we show an efficient test generation procedure based on the clock concatenation technique. To reduce test pattern count further, the clock concatenation based approach is combined with the clock domain analysis based approach. The enhanced procedure is described in Section 4. Experimental results for several industrial circuits are given in Section 5. Section 6 concludes the paper.

2 Test Generation Based on Clock Domain Analysis

2.1 Clock Classification Criteria

In order to avoid clock skew problems, the basic rule to allow to pulse two clocks simultaneously is given below.

Basic rule: Two clocks, *clk1* and *clk2*, cannot be pulsed simultaneously if there exists any combinational data path between the state elements controlled by this two clocks, i.e., if the state elements updated by *clk1* can reach the data port of any state element updated by *clk2* through combinational logic or *vice versa*, they cannot be applied together.

As an example, no combinational data path exists between the flip-flops *DFF1* and *DFF2* in Figure 1(a). Applying *clk1* and *clk2* simultaneously will not cause any clock skew problem. In Figure 1(b), the output of *DFF3* can reach the data input of *DFF4* through an AND gate. If the rising edge of *clk4* reaches *DFF4* later than the rising edge of *clk3* due to clock skew, the new data captured into *DFF3* may be captured by *clk4* to *DFF4*. As a result, we should not allow *clk3* and *clk4* to be pulsed simultaneously during testing.

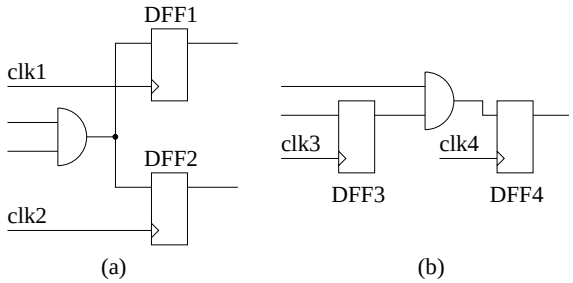

Fig. 1: Examples for basic rule

Based on the basic rule, we could build a data transfer graph[3] to describe the data transfers across different clocks and this graph is used to identify the group of clocks that can be pulsed together without causing clock skew problem. However, in modern designs, the relationship among the different clocks becomes more and more complex. The data transfer graph itself is not sufficient to describe the interaction among the clocks. Considering the circuit in Figure 2(a), both *clk1* and *clk2* can reach the clock port of *DFF1*. The data captured into *DFF1* will be determined by the earlier rising edge between *clk1* and *clk2* when both *sel1* and *sel2* have logic value 1. As a result, race condition exists between these two clocks. During test generation, we must avoid to pulse them simultaneously even if they satisfy the basic rule.

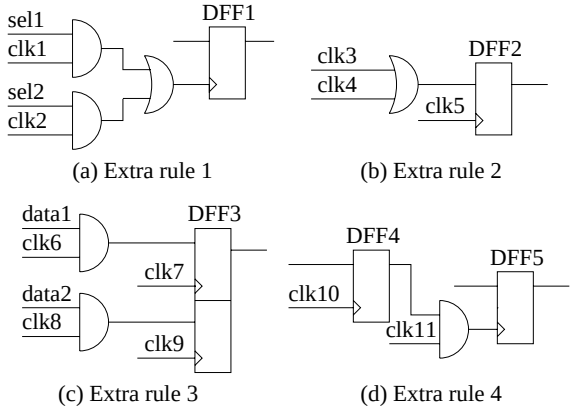

Fig. 2: Examples for extra rules

We propose several extra rules to check the interaction among clocks. During the test generation, two or more clocks can be applied together only when they pass both the basic rule and the extra rules. The proposed extra rules are described below.

Extra rule 1: Two clocks *clk1* and *clk2* cannot be pulsed simultaneously if both clocks can reach to the same clock port of any state element through any combinational logic.

An example circuit that violates the extra rule 1 is shown in Figure 2(a).

Extra rule 2: Two clocks *clk1* and *clk2* cannot be pulsed simultaneously if *clk1* can reach the data port of a state element controlled by *clk2* through any combinational logic.

In Figure 2(b), both *clk3* and *clk4* reach the flip-flop *DFF2*'s data port through an OR gate. If *clk3* is at its off state, logic 0, and *clk4* and *clk5* are pulsed simultaneously, the data captured into *DFF2* is undetermined due to data holding time may be violated. However, *clk3* and *clk4* in this example can be pulsed together if they satisfy all checking rules.

Extra rule 3: Two clocks *clk1* and *clk2* cannot be pulsed simultaneously if they can reach to the different clock ports of the same state element through any combinational logic.

A flip-flop with two ports is shown in Figure 2(c). When *clk7* and *clk9* are pulsed together, the data captured into the flip-flop is undermined if different logic values exist at the inputs of the different data ports. Based on the extra rule 2, we should not pulse the clocks {*clk6*, *clk7*} and {*clk8*, *clk9*} simultaneously. However, pulsing the clocks {*clk6*, *clk9*} and {*clk7*, *clk8*} will not create any problem.

Extra rule 4: Two clocks *clk1* and *clk2* cannot be pulsed simultaneously if both *clk2* and the data captured by *clk1* can reach to the clock port of the same state element through any combinational logic.

In Figure 2(d), the data captured by *clk10* is used to gate the clock *clk11*. If *clk11* is later due to the clock skew problem, the new capture value rather than the old value in *DFF4* is incorrectly used to gate *clk11*.

2.2 Clock Interaction Table

By applying the clock classification criteria described in the above section, we can classify the clocks into several non-interaction groups, i.e., all clocks in the same group are potential candidates to be pulsed together during test generation in order to reduce the test pattern count. Since a clock may be included in more than one group, to enumerate all possible clock groups explicitly is not an efficient way to guide the test generation. In our implementation, we use a *n*-by-*n* symmetric matrix *I* to record the analysis results for the interaction among the clocks, where *n* is number of clocks in the design. A data item *I*(*i*, *j*) in the matrix *I* is set to be 1 if the *i*-th clock cannot be applied together with the *j*-th clock because they do not pass the basic rule and the extra rules. Otherwise, the value 0 will be assign to the data item *I*(*i*, *j*).

Table 1: Clock interaction table for the circuit shown in Figure 2(c)

I	*clk6*	*clk7*	*clk8*	*clk9*
clk6	0	1	0	0
clk7	1	0	0	1
clk8	0	0	0	1
clk9	0	1	1	0

As an example, the clock interaction table for the circuit shown in Figure 2(c) is given in Table 1. Both *I*(*clk6*, *clk7*) and *I*(*clk8*, *clk9*) are set to be 1 due to the extra rule 2 is violated, and *I*(*clk7*, *clk9*) is set to be 1 due to the extra rule 3 is violated.

When applying the clock interaction table during test generation, the procedure *ckeck_interaction_clocks*() described below is used to avoid generating test patterns with any clock skew problem. This procedure is called when the test generator requires to turn a clock clk_{on} on in order to detect the fault currently targeted by ATPG.

Procedure: check_interaction_clocks(clk_{on})

/* clk_{on} is the clock required to be turned on now */

1. Check potential clock skew:
   - **(a)** For each clock clk_i in the design: If clk_i is already turned on and $I(clk_{on}, clk_i)$ is equal to 1, return FAIL.
2. Avoid clock skew:
   - **(a)** For each clock clk_i in the design: If clk_i has not been turned off and $I(clk_{on}, clk_i)$ is equal to 1, turn it off.
3. Turn clk_{on} on and return PASS.

In above procedure, we first check if clk_{on} can be applied together with any already on-clocks. If the check fails, the procedure returns FAIL to ask the ATPG to backtrack. The second step in the procedure will turn off all clocks which cannot be applied together with clk_{on} in order to avoid clock skew problem.

2.3 Test Generation Procedure

In order to apply clock domain analysis to generate more compact test set, dynamic compaction needs to be used during the test generation. In this section, we describe how to integrate the results from clock domain analysis into dynamic compaction. The proposed test generation procedure is given below. For simplicity, test patterns with single cycle are generated in this procedure. We will consider test generation for multiple cycle test patterns in Section 4.

Procedure: ATPG_for_clock_domain()

1. Apply clock classification criteria to identify the interaction among all the clocks in the design and create clock interaction table.
2. Create fault list and set test pattern set *T* equal to NULL.
3. While there exist undetected faults in the fault list, do
   - **(a)** Randomly pick an undetected fault *f* from the fault list.
   - **(b)** Set all the bits in the test cube *C* with one cycle to be X.
   - **(c)** Set *num_fails*=0 and *primary_fault_flag*=1.
   - **(d)** Apply deterministic test generator to specify unspecified bits in *C* in order to detect the fault *f*. During the test generation, if any clock is required to be turned on, call the procedure *check_interaction_clocks*() to turn off the clocks that will cause skew problem, but keep all other clocks' value as X if they are not specified yet.
   - **(e)** If test generation successes, update *C* by the new expanded test cube, set *primary_fault_flag*=0, and go to Step (i).
   - **(f)** If *primary_fault_flag* is 1, go to Step (a).
   - **(g)** Set *primary_fault_flag*=0 and increase *num_fails* by 1.

(h) If *num_fails* is greater than maximum number of allowed failures, go to Step (j).

(i) Randomly pick next undetected fault *f* from the fault list such that the fault effect of *f* will not be captured by the clocks with off-state in *C*. Then, go to Step (d).

(j) Set all clocks in *C* with unspecified value to have their off-state value and then randomly fill all unspecified bits in *C*.

(k) Fault simulate *C* for all undetected faults and drop the detected faults from the fault list.

(l) Add *C* to the test set *T*.

4. Return the generated test pattern set *T*.

The above procedure is similar to traditional dynamic compaction except for the differences listed below:

- After test generation for the primary target fault succeeds, the traditional dynamic compaction will force all unspecified clocks to at their off-state in order to avoid clock skew problem. In our procedure, we use the clock interaction table to determine the clocks needing to be turned off. All other clocks are left as unspecified in order to increase the chance to detect the faults in those clock domains by expanding the test cube.
- The proposed procedure is more flexible in selecting the clocks to be pulsed together. It implicitly enumerates all possible combinations of the non-interactive clocks by using the clock interaction table and only applies as many clocks together as necessary. For example, if *clk1*, *clk2*, and *clk3* can be pulsed simultaneously, the procedure may only turn both *clk1* and *clk2* on, but turn *clk3* off if turning *clk3* on cannot detect extra faults. Thus, it avoids the shortcoming of the strategy of grouping the clocks first and defining the clocks in each group as equivalent, as mentioned in Section 1. As a result, it improves the performance of the test generator to generate more compact test set.
- Traditional test generation typically targets faults according to a predetermined order. In the proposed procedure, we randomly pick the fault to be targeted next since we found that this strategy generates more compact test set than using a fixed fault order. Furthermore, when we select the secondary target faults in Step (i), we will use clock values to filter the number of secondary target faults to be considered next. This is based on the observation that a fault in the design may require to turn a specific clock on in order to detect it. The analysis to determine the relations between faults and clocks can be done before the test generation. We will not discuss it in this paper.

3 Test Generation Based on Clock Concatenation

When all clocks in the design are interact with each other, clock domain analysis is unable to group the clocks to be pulsed simultaneously. Thus, only one clock can be pulsed in each cycle. If we still generate single cycle test pattern, many state elements controlled by non-pulsed clocks do not have the opportunity to observe fault effects in anything other than the logic already tested by a scan chain test. As a result, test pattern count will be increased significantly because observation points are not used effectively in each test pattern.

To reduce the test pattern count while not risking clock skew, we can generate test patterns with multiple cycles where different clocks are applied sequentially in different cycles, i.e., turn one clock on in each cycle. If the fault effects captured into scan cells during any cycle in a test pattern are not disturbed by the clocks applied subsequently in the same test pattern, the faults in the different clock domains can be detected by one test pattern through pulsing those clocks sequentially.

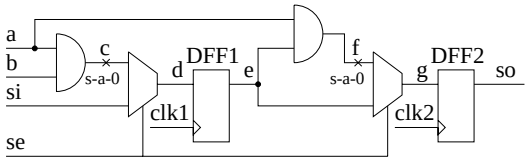

Fig. 3: Example for applying clocks sequentially

Considering the circuit shown in Figure 3, two clocks, *clk1* and *clk2*, are included in this circuit. Due to violating the basic rule, they cannot be pulsed simultaneously. It will require two test patterns to detect the faults *c* stuck-at-0 and *f* stuck-at-0 if one cycle per test pattern. However, if we apply the test vector {*a*,*b*,*si*,*se*,*clk1*,*clk2*}={1,1,X,0,pulse,0} first, the fault effect of *c* stuck-at-0 will be captured into scan cell *DFF1* and the fault-free machine value of *DFF1* will be set to 1. Before doing scan unloading, we apply another test vector {*a*,*b*,*si*,*se*,*clk1*,*clk2*}={1,X,X,0,0,pulse} and it will capture the fault effect of *f* stuck-at-0 into *DFF2*. Since *clk1* is at its off-state in the second vector, the fault effect of *c* stuck-at-0 captured into *DFF1* is not disturbed by pulsing *clk2*. Thus, we can detect both faults in a test pattern including two cycles after doing scan unloading.

To generate multiple cycle test patterns as described above, we propose the procedure shown below. In the proposed procedure, we assume only one clock is pulsed in each cycle and the design is a full scan design.

Procedure: ATPG_with_clock_concatenation()

1. Classify clocks into two groups, i.e., master clock group M_{clk} and slave clock group S_{clk}.

2. Mark scan cells which can be disturbed by more than one clock.

3. Create fault list and set test pattern set *T* to be NULL.

4. While there exist undetected faults in the fault list, do

(a) Set both disturbed observation scan cell set D_{cell} and off-clock set O_{clk} to be NULL.

(b) Set initial test cube *C* to be NULL and number of cycles N_{cycle} in *C* to be 0.

(c) While N_{cycle} is less than maximal number of cycles allowed:

(i) Append one additional cycle at the end of *C*.

(ii) In the added cycle, force all clocks in O_{clk} to

their off-state and all other primary inputs to have unspecified value initially.

(iii) Imply all the clock inputs for each scan cells in D_{cell} to be off in the added cycle in order to avoid disturbance of these scan cells.

(iv) Randomly pick an undetected fault f from the fault list such that its fault effect can be captured by a clock not in O_{clk}. If no such kind fault left, go to Step (d).

(v) Activate f in the last cycle and apply the deterministic test generation to specify unspecified bits in C to detect f.

(vi) If the test generation for f fails, go to Step (iv).

(vii) Increase N_{cycle} by 1.

(viii) Let clk_f in the last cycle be the capture clock to observe f in the scan cell $cell_f$. Set all clocks except clk_f to their off-state in the last cycle of C in order to avoid clock skew. Moreover, if $cell_f$ is marked in Step 2, add it to D_{cell}.

(ix) Apply dynamic compaction to specify as many unspecified bits in C in order to detect additional undetected faults. During the dynamic compaction, all faults are activated in the last cycle only. For each new observation scan cell, add it to D_{cell} if it is marked in Step 2.

(x) If $clk_f \in M_{clk}$($clk_f \in S_{clk}$), add all clocks in S_{clk}(M_{clk}) to O_{clk}.

(xi) If all clocks are included in O_{clk}, go to Step (d).

(d) Randomly fill all unspecified bits in C.

(e) Fault simulate C for all undetected faults and drop the detected faults from fault list.

(f) Add C to the test pattern set T.

5. Return the generated test pattern set T.

The first step in the above procedure is typically used for LSSD designs. LSSD designs includes two types of clocks, master clock and slave clock. Since it is impossible to unload both the master and slave elements of a single scan cell in a test pattern, there is no benefit in pulsing the master clock and slave clock sequentially. In the proposed procedure, we will give freedom to chose any type of clock in the first cycle of the test pattern. However, only the clocks with the same type as the clock in the first cycle are allowed to be pulsed in the subsequent cycles. This is achieved in Step (x) by adding all the clocks in the different group to O_{clk}. For non-LSSD designs, all clocks belong to master clock and S_{clk} is NULL.

The analysis carried out in Step 2 is used to mark the scan cells which can be disturbed by more than one clock. Considering the circuit in Figure 2(a), if *clk1* is used to capture the fault effect in the first cycle of the test pattern, we cannot apply *clk2* in the subsequent cycles since it will disturb the value captured earlier. In the proposed procedure, we will add the scan cell to the disturbed observation scan cell set D_{cell} if it is marked as a disturbed cell and is used to observe certain fault during the test generation. As shown in Step (iii), all chosen observation scan cells are prevented from being disturbed by implying all the clock inputs of the scan cells in D_{cell} to their off-value. For the scan cells not marked in Step 2, it is unnecessary to explicitly imply all the clock inputs to their off-value since our procedure uses each capture clock only once in each test pattern. This is achieved by adding all used capture clocks to the off-clock set O_{clk}. Step (ii) will force all the clocks in O_{clk} to their off-state when appending a new cycle. In this way, we can minimize the test generation effort to avoid disturbing the observation scan cells.

All other steps in the proposed procedure are straightforward. During the test generation, one additional cycle is appended to the end of the previous test cube and dynamic compaction is applied to maximize the number of faults detected in the additional cycle by activating the faults in the new added cycle only. The test generation is finished when all possible clocks are used in the generated test pattern or the predefined cycle limit for the test pattern is reached.

4 Mixed Test Generation Procedure

In Section 3, we assume one clock pulse each cycle in order to avoid the risk of clock skew. To reduce the number of cycles in each test pattern further, we can mix the clock domain analysis based approach with the clock concatenation based approach. In the mixed approach, more than one clock is allowed to be pulsed in each cycle of the test pattern. To achieve this goal, the procedure *ATPG_with_clock_concatenation*() proposed in Section 3 is modified in the following ways:

- During test generation for the fault f in Step (v), the procedure *check_interaction_clocks*() needs to be called to turn off all the clocks that will cause clock skew with current on-clocks in the same cycle. All non-interactive clocks will keep their values as unknown if possible.
- In Step (viii), it is unnecessary to force all unspecified clocks to their off state in the last cycle first. This operation is carried out after the dynamic compaction is finished in the last cycle in order to maximize the number of clocks to be pulsed in the same cycle.
- All clocks with on-value in the last cycle will be added to O_{clk} after the dynamic compaction in Step (ix) in order to use a clock at most once.

5 Experimental Results

All proposed test generation procedures were integrated into a commercial test generation tool. Several industrial circuits were used to evaluate the number of generated test patterns by applying three proposed test generation procedures. The test generation times reported in this section are obtained on a LINUX workstation with 2.4GHz Pentium 4 processor.

In Table 2, we show the test generation results for 4 different industrial circuits by applying four different test generation procedures: the procedure for generating test patterns with single cycle and single clock pulse under column *Single Clock*, the procedure for generating test patterns with single cycle and clock domain analysis under column *Clock Domain*, the procedure for generating test patterns with two cycles and clock concatenation under column *Clock Concatenation*, and the procedure for generating test patterns with

Table 2: Test generation results by applying proposed test generation procedures for stuck-at fault model

Ckt	#Flts	#Clks	#Doms	Maximum Number of Allowed Test Cycles in Each Test Pattern										
				One Cycle					Two Cycles					
				Single Clock		Clock Domain			Clock Concatenation			Mixed		
				#Pats	CPU	#Pats	CPU	%Red	#Pats	CPU	%Red	#Pats	CPU	%Red
ckt1	1209815	36	5	5476	329	2691	251	50.9	2765	861	49.5	1621	589	70.4
ckt2	2131449	11	5	6661	3349	5461	3134	18	3959	8446	40.6	3219	7682	51.7
ckt3	2558092	8	5	2480	1584	2264	1562	8.7	1389	2575	44	1289	2493	48
ckt4	4394833	24	13	10186	6060	7271	5196	28.6	5573	18646	45.3	5008	13286	50.8

two cycles and the mixing of clock domain analysis and clock concatenation under column *Mixed*. Dynamic compaction is applied in all four procedures. The number of generated test patterns and the test generation time in seconds for each test generation procedure is given under the columns *#Pat* and *CPU*, respectively. For the last three test generation procedures, we also show the percentage of test pattern reduction compared with the first test generation procedure under the column *%Red*. The total number of collapsed faults and number of clocks in each circuit are shown under columns *#Flts* and *#Clks*. After the clock interaction table is created by doing clock domain analysis, we use greedy algorithm to classify clocks that can be pulsed simultaneously into groups. However, each clock can only be included in one group. The total number of clock groups created in this way are reported under the column *#Doms*.

From Table 2, it can be seen that the test generation procedure based on clock domain analysis not only effectively reduces the number of test patterns with single cycle from 8.7% to 50.9%, but also reduces the test generation time when comparing with the traditional test generation procedure that generates test patterns with single cycle and single clock pulse. If we consider the relation between the number of clocks and the number of clock domains in the design, the test generation results show that the more non-interactive clocks in the design, the more effective the clock domain analysis based approach is to reduce the test pattern count.

When we apply clock concatenation technique alone to generate compact test set, it can be seen from Table 2 that significant reduction of the number of test patterns is achieved even if the maximum number of allowed test cycles is only set to be 2. On average, 44.9% test pattern count reduction is obtained when compared with the test pattern count under the column *Single Clock*. Moreover, the clock concatenation technique is more effective in terms of test pattern count than the clock domain analysis in general. For example, we can reduce the test pattern count by 44% when using the clock concatenation technique for *ckt3*. But only 8% test pattern count reduction is achieved for the same circuit when using the clock domain analysis. When considering the test generation time, the clock concatenation based test generation requires 2 to 3 times longer than the traditional test generation procedure.

When mixing the clock domain analysis with the clock concatenation, we can reduce the test pattern count further as shown under the column *Mixed*. On average, the mixed method generates 10.8% fewer test patterns when compared with using the clock concatenation technique alone. Moreover, the CPU time in the mixed method is reduced as well.

6 Conclusions

Utilizing the clocks in designs with multiple clocks effectively and efficiently will dramatically reduce the test pattern count without losing fault coverage. The test generation based on clock domain analysis is more effective in reducing the test pattern count for the designs with large number of non-interactive clocks and it avoids generating test patterns with the risk of clock skew. The clock concatenation based approach can be used for the design even if all clocks are interactive with each other. This approach typically generate more compact test set than the clock domain analysis based approach. However, when we combining these two approach, significantly extra test pattern reduction can be achieved for all the industrial circuits under the experimentation.

References

[1] "Designs with Multiple Clock Domains: Avoiding Clock Skew and Reducing Pattern Count Using DFTAdvisor™ and FastScan™," Technical White Paper, in http://www.mentor.com/dft.

[2] R. Press and R. Illman, "ATPG Pattern Compaction: The Next Wave," Technical White Paper, in http://www.mentor.com/dft.

[3] V. Jain and J. Waicukauski, "Scan Test Data Volume Reduction in Multi-Clocked Designs with Safe Capture Technique", in Proc. of ITC, pp. 148-153, 2002.

[4] "ATPG Tools Reference Manual - FastScan, FlexTest, and TestKompress," Mentor Graphics Corp., 2002.