Testing of Clock-Domain Crossing Faults in Multi-Core System-on-Chip*

Naghmeh Karimi†, Zhiqiu Kong†, Krishnendu Chakrabarty†, Pallav Gupta‡, and Srinivas Patil§

†Electrical & Computer Engineering Department
Duke University
Durham, NC 27708
{*naghmeh.karimi, zhiqiu.kong, krish*}*@duke.edu*

‡Test CAD Technology Group
Intel Corporation
Folsom, CA 95630
pallav.gupta@intel.com

§SoC Enabling Group
Intel Corporation
Austin, TX 78704
srinivas.patil@intel.com

***Abstract*—Manufacturing test for clock-domain crossing (CDC) defects is a major challenge for multi-core system-on-chip (SoC) designs in the nanometer regime. Setup- and hold-time violations in flip-flops situated on clock boundaries may lead to catastrophic failures, even when circuits are equipped with synchronizers at clock boundaries. In this work, we comprehensively study the effect of CDC faults, and propose a number of fault models to target such defects. In addition, we develop an automatic test-pattern selection method for CDC fault detection. This work is motivated by the fact that CDC faults cannot always be detected by conventional ATPG methods. The results of applying the proposed method to a number of IWLS'05 benchmarks demonstrate the effectiveness of our approach.**

***Keywords*-Clock-domain crossing; multi-clock domain circuit; Test-pattern selection**

I. Introduction

Advances in very large scale integrated (VLSI) technology have enabled designers to integrate multi-core SoC on a single die. As the number of intellectual property (IP) cores in an SoC increases, high-speed communication between the cores becomes a major challenge.

This problem is exacerbated when cores operate with different clock signals fed either by different PLL sources, or by a common PLL source but with different phases and frequencies. Consequently, intra-core communication must be handled carefully.

In multi-clock designs, a CDC occurs whenever data is transferred from one clock domain to another. Depending upon the relationship between the sender and receiver clocks, various types of problems may arise during data transfer. Some of these problems are metastability, data loss, and data incoherency [1].

The effect of metastability must be neutralized when data are transferred between different clock domains. This can by accomplished by employing synchronizers at clock boundaries. In their simplest form, synchronizers are flip-flops that reside in the sender and receiver domains. The receiver synchronizer samples the data coming from the sender domain. The sender synchronizer removes unwanted transitions (glitches) that occur before the combinational output of the sending logic settles. Even if metastability is eliminated, incorrect operation resulting from convergence of synchronized data or improper synchronization of data-transfer protocols (handshaking) can result in functional errors [2]. Although computer-aided design (CAD) tools are used to formally verify a CDC design for correct functional behavior before the fabrication process, post-silicon multi-clock circuits may still exhibit incorrect behavior due to process variation-induced violation of setup- and hold-time at the boundary flip-flops, even when they are equipped with synchronizers at clock boundaries. Current manufacturing test content is not developed to screen defective units resulting from CDC errors. Thus, we comprehensively study CDC faults with the goal of increasing test quality to reduce the number of shipped defective chips [3].

Transition delay fault (TDF) testing is well-known and widely used in industry to target timing defects. Despite their benefits, current transition ATPG tools are not adequate for detecting CDC faults because these tools do not model the interaction between the logic residing on a clock boundary while generating a test pattern to target a specific TDF. Therefore, fault models and ATPG methodologies need to be developed to target these faults. In this work, we have developed the first manufacturing test method that targets CDC faults in multi-core SoCs. The main contributions of this work are as follows:

- We analyze the various functional errors associated with CDCs and develop novel fault models for CDC testing. These fault models are based on classical fault models, but with additional constraints;
- We propose the first test-pattern selection method for CDC testing;
- We demonstrate the efficacy of our proposed CDC testing method using commercial CAD tools, and present results on a number of IWLS'05 benchmark designs.

The remainder of this paper is organized as follows. Section II presents related work on CDC verification and testing. In Section III, we develop CDC fault models representing the faulty behavior of CDC designs in the presence of physical defects. We propose our test-pattern selection method to

*This research was supported in part by NSF under grant No. CCF-0903392, and by SRC under Contract No. 1992.

1081-7735/11 $26.00 © 2011 IEEE
DOI 10.1109/ATS.2011.68

target CDC faults in Section IV. Implementation details about the proposed method are presented in Section V, while experimental results to demonstrate the effectiveness of this technique are discussed in Section VI. We conclude this paper in Section VII.

II. Related Work

To the best of the authors' knowledge, prior work in CDC testing has been focused primarily on pre-silicon validation. There is no study involving post-silicon CDC test in the open literature.

Various types of errors are discussed in [4], [5], and error recovery is described based on CDC synchronization signals between different clock domains. In [6], enable-based synchronization and data coherency involving CDC is presented. The authors proposed using a static CDC verification approach to target metastability, convergence, and other CDC problems that traditional CAD tools, such as logic simulation and static timing analysis, do not address.

The authors in [7] used an assertion-based verification approach to verify proper functionality for CDC signals. Two methods based on model checking for verifying synchronizers in CDC designs were proposed in [8]. In [9], a formal verification technique to detect CDC design errors was discussed.

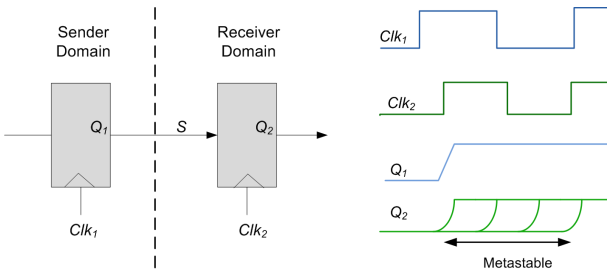
In [10], an assertion-based method was presented to reduce the risk of clock-related errors. A built-in CDC test for globally asynchronous locally synchronous (GALS) systems was proposed in [11]. CDC jitters were analyzed, and a design-for-test (DFT) method to emulate the delay caused by the jitters was presented in [12]. In [13], the delay of synchronizers was increased to eliminate the non-determinism caused by metastability. A variety of CDC verification tools have been developed by several CAD companies.

As mentioned earlier, all of the current methods address only pre-silicon validation of CDC designs. However, in the presence of process variations, post-silicon testing of CDC is also required to ensure correct operation, even if thorough pre-silicon validation was performed before tapeout.

III. CDC Fault Model

To be able to screen for CDC defects, the faulty behavior of these defects must be logically represented using a fault model. In this section, we propose the first CDC fault model to capture the erroneous behavior.

In a synchronous circuit, the proper operation of a flip-flop depends on the stability of its input signal for a certain period of time before (setup-time) and after (hold-time) its clock edge. If setup- and hold-times are violated, the flip-flop output may oscillate for an indefinite amount of time, and may or may not settle to a stable value before the next active clock edge. This unstable behavior is known as *metastability*. Fig. 1(a) shows an example of a multi-clock circuit in which



(a) CDC circuit. (b) Metastability on Q_2.

Figure 1. An example of a CDC circuit and metastability.

signal S is launched by Clk_1, and needs to be captured properly by Clk_2. As shown in Fig. 1(b), if a transition on S happens very close to the active edge of Clk_2, a setup-time violation occurs, which may lead to metastability on Q_2.

CDC faults mainly occur due to setup- and hold-time violations on flip-flops residing at clock boundaries. Therefore, we can categorize them as follows:

- Faults that occur due to setup-time violation;
- Faults that occur due to hold-time violation.

If a flip-flop experiences a setup-time violation, it does not sample a change in value at its data input. In a hold-time violation, however, it may incorrectly capture a data change at its input. We next describe the fault model for each case.

A. Setup-Time Violation

Fig. 2(a) illustrates a sample waveform for the CDC circuit shown in Fig. 1(a). If signal S experiences an unexpected delay, and its value toggles during the setup-time window of the receiver flip-flop, the receiver flip-flop may capture the value "0" even though the expected value is "1". As per the example, because the output value of the sender flip-flop does not change in the subsequent clock cycle, $Q2$ gets its expected value of "1" in the next clock cycle. In this case, the setup-time violation of the receiver flip-flop can be modeled as a slow-to-rise fault with a delay of one clock cycle.

Fig. 2(b) shows another possibility of a setup-time violation for the same CDC circuit. First, the receiver flip-flop experiences a setup-time violation. Then, the output of the sender flip-flop changes before the next active clock edge arrives for the receiver flip-flop. Thus, the receiver flip-flop cannot capture the transition of signal S, and remains unchanged. In this case, the setup-time violation of the receiver flip-flop can be modeled by a slow-to-rise fault with an infinite delay.

In general, if a value change of a CDC signal S violates the setup-time of the receiver flip-flop, then the faulty behavior can be modeled as a transition (slow-to-rise or slow-to-fall) fault with a delay of k clock cycles, where k = 1 if the pulse observed in signal S is at least 1.5 times

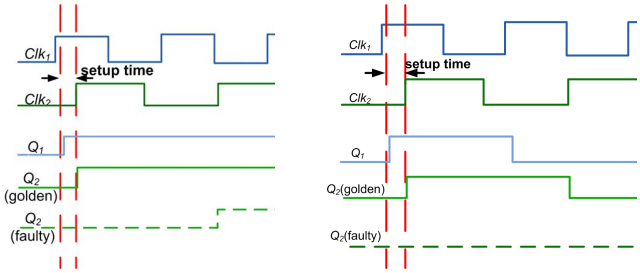


Figure 2. Timing waveforms showing setup-time violations for the circuit in Fig. 1(a).

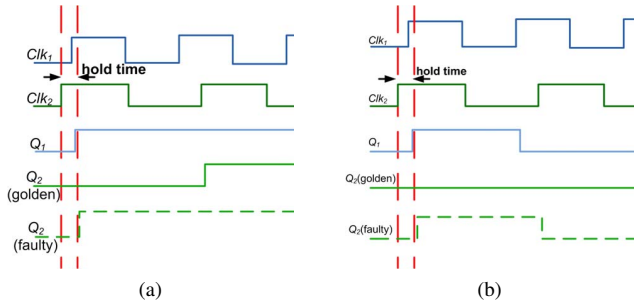


Figure 3. Timing waveforms showing hold-time violations for the circuit in Fig. 1(a).

wider than the receiver clock period. Otherwise, $k = \infty$. In the rest of this paper, a CDC fault arising due to setup-time violations will be referred to as a *S-CDC* fault.

B. Hold-Time Violation

If a flip-flop experiences a hold-time violation, data changes on its input may be incorrectly sampled. Fig. Fig. 3(a) shows a sample waveform for the CDC circuit shown in Fig. 1(a). If signal S changes during the hold-time interval of the receiver flip-flop, an incorrect change on the output may be observed. As per the example, the receiver flip-flop gets an output value of "1" one clock cycle earlier than expected. In this case, the hold-time violation at the receiver flip-flop can be modeled as a transient fault with a duration of one clock cycle.

Fig. 3(b) shows another example of a hold-time violation for the CDC circuit of Fig. 1(a). First, the receiver flip-flop experiences a hold-time violation. Then, the output of the sender flip-flop changes before the next active edge of the receiver flip-flop. Thus, the receiver flip-flop captures the transition of signal S. Similar to the previous example, the hold-time violation of the receiver flip-flop can be modeled as a transient fault with a duration of one clock cycle.

In general, if any value change of a CDC signal S violates the hold-time of the receiver flip-flop, then the faulty behavior can be modeled as a transient fault with a duration of one clock cycle. In the rest of this paper, a CDC fault arising due to hold-time violations will be referred to as a *H-CDC* fault.

IV. Proposed CDC-Oriented Double-Capture (CoDC) Testing Scheme

Based on the fault models described in Section III, we now describe a test strategy to detect S-CDC faults. We do not consider H-CDC faults in this paper; a thorough treatment of them is left for future work.

TDF testing is widely used in industry to target timing defects. However, traditional TDF ATPG tools are not adequate for detecting S-CDC faults because they do not consider the interaction between the logic residing on a clock boundary while generating a test pattern to target a TDF fault. In order to detect S-CDC faults, we need to generate a transition in the sender domain, and observe its effect in the receiver domain.

For TDF detection, an ATPG tool generates a transition at the fault site, and propagates the transition to an observable point. While this condition is also required for detecting S-CDC faults, it is not sufficient. That is, the transition at the fault site must be provided by the sender domain. Thus, a major shortcoming of traditional ATPG tools is that they do not consider this constraint. Consequently, they cannot guarantee coverage of CDC faults. To overcome this limitation, this paper addresses a testing scheme to directly target S-CDC faults.

Launch-on-Shift (LoS) and Launch-on-Capture (LoC) are two widely used TDF testing methods. Since LoC is easier to implement in practice, we only consider this method to detect S-CDC faults.

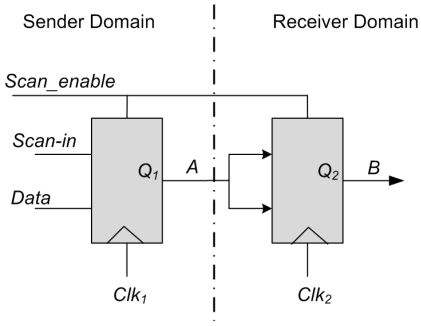


Figure 4. A CDC example for illustrating our proposed test-pattern selection method.

Fig. 4 shows a CDC example that is used to illustrate the proposed test-pattern selection method. For the sake of clarity, only the synchronizer flip-flops are shown. In this case, the functional and scan paths on the clock boundary are identical. However, our method is readily applicable (without any loss of generality) to circuits with different functional and scan paths on the clock boundaries. Assume that we want to target the S-CDC fault modeled by a slow-to-rise fault at the output of the receiver flip-flop. To detect this fault, we need to first generate a rising transition on signal A. Then, we must transfer this transition to signal B in

the next active edge of Clk_2. To be able to detect this CDC fault on signal B, we need to apply three test vectors instead of the two that are applied by the traditional LoC method. Fig. 5-6 show the active paths highlighted in bold for the three steps needed to detect the fault. These steps are as follows:

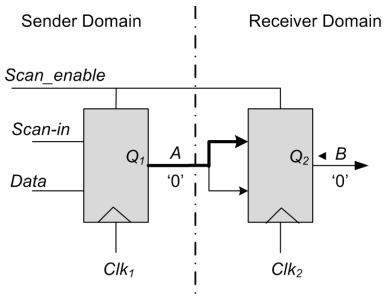
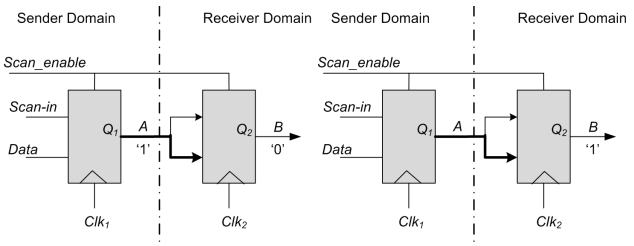


Figure 5. Illustration of Step 1 to target slow-to-rise CDC fault on signal B (scan path highlighted in bold).



(a) Step 2. (b) Step 3.

Figure 6. Illustration of Steps 2-3 to target slow-to-rise S-CDC fault on signal B (functional path highlighted in bold).

- **Step 1:** Shift a test vector (V_1) in scan mode such that A and B get a value of "0" (Fig. 5).
- **Step 2:** Operate in functional mode (apply functional clocks Clk_1 and Clk_2, respectively) and generate vector V_2 such that A and B get a value of "1" and "0", respectively (Fig. 6(a)).
- **Step 3:** Operate in functional mode, and generate vector V_3 such that B gets a value of "1" (Fig. 6(b)).

If $k > 0$ synchronizers are placed between the sender and receiver flip-flops, then Step 2 involves one cycle of functional clock Clk_1 but $k + 1$ cycles of functional clock Clk_2. Steps 3 remains unchanged.

This CDC testing scheme, which we refer to as CDC-oriented Double-Capture testing (CoDC), comprises of one launch and two capture cycles. To target the S-CDC fault modeled by a slow-to-rise fault on signal B, we need to generate vectors V_1-V_3 in consecutive clock cycles such that the output of the sender and receiver clock boundary flip-flops (A and B) get the values shown in Table I. Note that the transition on B is at-speed with respect to Clk_2. Thus, a S-CDC fault on B can be easily detected.

Table I
SYNCHRONIZER VALUES FOR DETECTING SLOW-TO-RISE S-CDC FAULT ON SIGNAL B.

Cycle #	Mode	Value at A	Value at B
1	Scan	0	0
2	Functional	1	0
3	Functional	–	1

– is a don't care.

No assumptions are made or restrictions are placed on the clocking scheme. The clock signals are fed either by different PLL sources, or by a common PLL source but with different phases and frequencies.

The CoDC approach can also be used to target CDC faults when data transfer between two clock domains is accomplished through asynchronous handshaking. In such scenarios, the data arrives at the receiver domain within an upper limit of n clock cycles. We can test for CoDC faults by applying n functional clock cycles using Clk_2 and using a transition detector to record a transition on B in the window of n clock cycles.

V. CoDC Implementation Details

To implement the CoDC method, we used commercial EDA tools for ATPG and logic simulation. Since two or more synchronizers are often added in practice to the receiver domain to handle asynchronous data transfers, we assume (without any loss of generality for the proposed method) in our experiments that the receiver domain for each CDC includes two synchronizers. Fig. 7 shows a flowchart of the steps taken for applying the proposed method. For each circuit under test, full-scan insertion was first performed. Next, we extracted all connected pairs of flip-flops residing at clock boundaries. For each sender/receiver flip-flop pair, we used a commercial ATPG tool to generate test vectors targeting the slow-to-rise fault on the output of the sender flip-flop under the constraint that the output of the receiver flip-flop holds the value "0". In this way, the first and second steps of the CoDC scheme are performed simultaneously. Due to the limitations of the ATPG tool, we could only extract up to 255 test vectors for each fault. We were not able to extract all possible test vectors that satisfy the first two requirements of the CoDC method.

In the next step, we used a commercial logic simulator to simulate the generated sets of test vectors and extract the subsets satisfying the third requirement of the CoDC method, i.e., generating a value of "1" on the output of the receiver flip-flop in the second clock cycle of the functional mode. We built a scoreboard for each S-CDC fault and the test vectors that detected that fault. Finally, a minimum set covering algorithm was applied to the test vectors to select a minimal set that detected all slow-to-rise S-CDC faults.

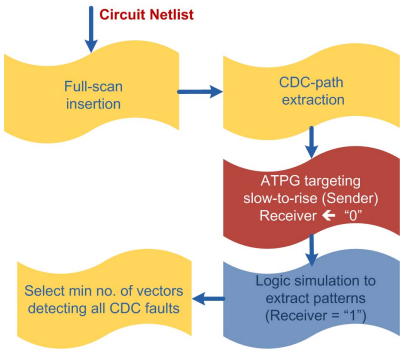


Figure 7. Flowchart for the implementation of the CoDC method.

VI. Experimental Results And Analysis

In this section, we provide details of the simulation setup used to evaluate the effectiveness of the CoDC scheme. Then, we present results on a number of IWLS'05 benchmarks, and discuss our observations.

A. Experimental Setup

We applied the CoDC method to four IWLS'05 benchmarks that contain multiple clock domains. They are the WISHBONE AC 97 Controller (`ac97_ctrl`), the WISHBONE Memory Controller (`mem_ctrl`), the USB function core (`usb_funct`), and the Ethernet IP core (`ethernet`) [14]. We used commercial EDA tools for test pattern generation, and logic and fault simulation. We implemented our own tools and scripts for performing pattern selection and evaluation. They were implemented in C++ and Python.

To generate a test pattern set that detects TDFs as well as S-CDC faults, top-off ATPG was performed after pattern selection to meet the fault coverage requirement for TDFs. The final pattern set for our procedure therefore consists of the selected pattern set plus the top-off ATPG pattern set.

All experiments were performed on a dual-processor Pentium Intel server running at 3.2 GHz with 2 GB of memory. CPU time for the flow described in Fig. 7 was estimated by aggregating the times needed for the different steps. While the runtime for the ATPG tool was small (a few seconds per S-CDC fault), invoking the commercial logic simulator multiple times leads to the runtimes ranged from 9 to 25 minutes per fault. Therefore, to reduce the cost of detecting S-CDC faults, we are developing a test generation scheme targeting S-CDC faults under the constraints discussed in Section IV.

B. Experimental Results

In this subsection, we present the results in four parts. The first part deals with the gate-level specification of each benchmark used in this study. The second part discusses the distribution of faults in each benchmark. The third part compares traditional LoC/TDF testing with the our CoDC method. Finally, the fourth part demonstrates the effect of pattern sampling on CDC fault detection.

Note that in our experiments, we only considered slow-to-rise S-CDC faults. We expect to get similar results for slow-to-fall S-CDC faults without any change in methodology. In each benchmark with two clock domains, we only considered the CDC paths from clock domain 1 to 2. The CDC paths from clock domain 2 to 1 were not considered. Again, we expect similar results if we consider the additional CDC paths. In addition, we treat `ethernet` benchmark as three different testcases based on the different CDC paths: `ethernet(1-2)`; `ethernet(2-3)`; `ethernet(3-1)`. In the `ethernet(i-j)` testcase, $i, j \in \{1, 2, 3\}$, we consider the CDC paths from Domain i to Domain j.

1) Benchmark Details: Table II lists various details about the benchmarks used in our experiments. The `ethernet` benchmark has three clock domains. All other benchmarks have two clock domains.

Table II
Benchmarks Statistics

Benchmark	# Clock domains	# Flip-Flops	# Gates
`ac97_ctrl`	2	2,199	28,083
`mem_ctrl`	2	1,083	22,015
`usb_funct`	2	1,746	25,531
`ethernet`	3	10,544	153,948

2) Number of Faults: Table III shows the number of transition and S-CDC faults in each benchmark. The second column shows the number of slow-to-rise (slow-to-fall) faults for each circuit, and the third column shows the number of CDC faults that can be attributed to violations on flip-flop setup times at clock domain boundaries (*i.e.*, S-CDC faults).

As mentioned earlier, for benchmarks with two clock domains, we only considered CDC paths from clock domain 1 to 2. For the `ethernet` benchmark, we considered CDC paths from clock domains 1 to 2, 2 to 3, and 3 to 1.

Table III
Number of Faults for the Benchmarks

Benchmark	Slow-to-rise(fall)	Slow-to-rise(fall) S-CDC
`ac97_ctrl`	36,510	734
`mem_ctrl`	38,082	342
`usb_funct`	40,076	1,342
`ethernet(1-2)`	160,394	560
`ethernet(2-3)`	160,394	206
`ethernet(3-1)`	160,394	1,918

3) Pattern Set Comparison: In this section, we highlight the effectiveness of the CoDC method in detecting CDC faults.

Detected CDC faults: To compare the traditional LoC method against the CoDC scheme, we first investigated the

number of S-CDC faults detected by each method for the four benchmarks. We applied our method to each benchmark and extracted the set of test patterns satisfying the constraints discussed in Section IV. Next, top-off ATPG was performed to meet the fault coverage requirement for TDFs. As a result, the final pattern set was the union of the CoDC and the top-off ATPG pattern sets.

To evaluate the number of S-CDC faults detected by the traditional LoC method for each benchmark, we used a commercial ATPG tool to generate test patterns detecting all slow-to-rise TDFs for that benchmark. Then, we used the generated patterns to perform logic simulation. Finally, the subset of the generated patterns that satisfied the constraints discussed in Section IV was extracted, and the S-CDC faults detected by these vectors were reported.

A limitation of the commercial ATPG tool used in this work is that it can only generate up to 255 test vectors for each fault, i.e., it cannot generate all possible test patterns satisfying the above constraints. This shortcoming limits the coverage of the CoDC method, i.e., the S-CDC faults that are reported as being undetected using CoDC may have been detected if the ATPG tool did not limit the number of generated patterns.

Table IV compares the LoC testing method with the CoDC method in terms of detecting S-CDC faults. The second and third columns in this table show the total number of the S-CDC faults and the size of its testable subset, respectively.

When we used the commercial ATPG tool to generate test patterns satisfying the first two requirements of the CoDC method, no pattern was generated for some faults. We considered these faults to be untestable S-CDC faults, and reported the remaining faults (for which the ATPG tool could generate at least one test pattern) as testable S-CDC faults. The numbers reported in the third column can be viewed as an upper bound on the number of testable S-CDC faults for each benchmark. For some faults, the ATPG tool can generate a number of test patterns satisfying the first two conditions of the CoDC method, but when we simulate the circuit using these patterns, the third condition may not be satisfied. However, these faults may actually be untestable. In such cases, the corresponding faults are reported as being undetected but testable. Again, if the ATPG tool did not limit the number of generated patterns for each fault, and was able to generate all test patterns for a fault, we would have been able to simulate all the generated patterns for each fault, and report the exact number of untestable faults.

The fourth column in the table lists the number of S-CDC faults detected by the traditional (baseline) LoC method. The fifth column lists the total number of S-CDC faults detected by the CoDC method. As discussed above, the numbers reported in this column are a lower bound in each case on the number of detected faults for the proposed method.

The sixth column reports the number of S-CDC faults detected by the top-off ATPG step. Note that if the ATPG tool would allow us to generate all patterns that satisfy the first two requirements of the CoDC method for each fault (instead of limiting to 255 patterns), no additional S-CDC faults would be detected by top-off ATPG. However, since we can target only 255 patterns per fault in the previous ATPG step, top-off ATPG can sometimes provide additional patterns that satisfy all three conditions of the CoDC method. Finally, the last column lists the total number of S-CDC faults detected by a combination of the CoDC method and top-off ATPG. For the benchmark circuits considered here, the traditional LoC method only detects 37% of the testable S-CDC faults on average. On the other hand, the CoDC method detects over 62% of these faults; in combination with top-off ATPG, 66% of S-CDC faults are detected.

Detected slow-to-rise transition faults: Another set of results that we present in this section investigates the number of slow-to-rise transition faults detected by each method. Table V shows the total number of slow-to-rise transition faults in each benchmark as well as the number of detected faults using either the LoC method or the CoDC method with top-off ATPG. As shown in this table, the number of detected faults in both cases are nearly equal. These results and the results in Table IV show that the proposed CoDC method with top-off ATPG enables us to detect a significant number of S-CDC faults (66%) without adverse impact on TDF coverage.

Pattern set: The next set of results compares the number of test patterns generated by the LoC, CoDC, and top-off ATPG methods. As shown in Table VI, with only 18% more test patterns on average, a significantly higher percentage of S-CDC faults can be detected by applying the CoDC method.

4) Pattern Sampling Comparison: To assess the effectiveness of the double-capture method, we used a commercial ATPG tool to generate up to 255 patterns for each fault satisfying the first two conditions of the CoDC method. We then simulated the generated patterns to identify those that also meet the third requirement of the CoDC method. However, extracting results for all faults takes a considerable amount of CPU time if all 255 test patterns are considered for each fault. Therefore, we performed sampling with different sample sizes to speed up logic simulation by over an order of magnitude.

Tables VII-IX show the results of pattern sampling for three benchmarks. Table VII shows the number of detected slow-to-rise faults for different pattern sets (n = 5, 10, 15, and 20). As shown in this table, for each benchmark, the number of slow-to-rise faults detected by CoDC + top-off ATPG for different sample size is nearly equal to the number of slow-to-rise faults detected with sample size of 255 (Table V). Therefore, sampling has no significant impact on the number of slow-to-rise faults detected by CoDC + top-off ATPG.

Table IV
DETECTED S-CDC FAULTS

Benchmark	# S-CDC faults	# Testable S-CDC faults	# Detected by LoC method	# Detected by CoDC method	# Additional detected by top-off ATPG	# Detected by CoDC method + top-off ATPG
ac97_ctrl	734	136	28	113	0	113
mem_ctrl	342	340	57	183	1	184
usb_funct	1,382	1,218	425	553	8	561
ethernet(1-2)	560	40	26	32	0	32
ethernet(2-3)	206	196	136	104	40	144
ethernet(3-1)	1,918	1,755	228	985	3	988

Table V
DETECTED SLOW-TO-RISE FAULTS

Benchmark	# Slow-to-rise faults	# Detected by LoC method	# Detected by CoDC method	# Additional detected by top-off ATPG	# Detected by CoDC method + top-off ATPG
ac97_ctrl	36,510	33,653	14,730	18,929	33,659
mem_ctrl	38,082	17,940	8,330	9,604	17,934
usb_funct	40,076	35,429	25,595	9,837	35,432
ethernet(1-2)	160,394	155,205	20,135	135,068	155,203
ethernet(2-3)	160,394	155,199	29,162	126,041	155,203
ethernet(3-1)	160,394	155,228	68,693	86,536	155,229

Table VI
COMPARISON OF NUMBER OF TEST PATTERNS

Benchmark	LoC	CoDC	Top-off ATPG	CoDC + top-off ATPG	% increase
ac97_ctrl	389	82	371	453	16
mem_ctrl	790	160	767	927	17
usb_funct	812	542	718	1,260	55
ethernet(1-2)	4,695	30	4,762	4,792	2
ethernet(2-3)	4,710	104	4,839	4,943	5
ethernet(3-1)	4,704	961	4,406	5,367	14

As expected, the number of S-CDC faults detected by the CoDC method with sampling is less compared to the case when 255 patterns are considered per fault. As shown in Table VIII, the number of detected S-CDC faults using CoDC + top-off ATPG for $n = 5$ is, on average, 51% of the number of the S-CDC faults detected using CoDC and top-off ATPG without sampling. For a sample size of $n = 20$, on average 71% of the S-CDC faults that were detected without sampling are still detected, and the number of faults detected is considerably larger than that detected by traditional LoC/TDF ATPG.

VII. CONCLUSIONS

We have analyzed the effect of CDC faults in multi clock-domain circuits and developed fault models to represent the incorrect behavior of these circuits in the presence of CDC faults. We have also developed a test-pattern selection method for detecting CDC faults. Experimental results for the IWLS'05 benchmark circuits show that the patterns extracted by the proposed method detects a significant number of the CDC faults compared with the patterns generated by commercial ATPG tools. In addition, the results show that by using a combination of the proposed method and top-off ATPG, almost all testable TDFs can be detected with a negligible overhead in the number of test patterns.

In this study, our goal was to leverage a commercial ATPG tool to extract a set of test patterns that detect S-CDC faults and TDFs. As a continuation of this work, we are developing a more efficient, custom ATPG tool to target S-CDC faults.

REFERENCES

[1] Y. Feng, Z. Zhou, D. Tong, and X. Cheng, "Clock domain crossing fault model and coverage metric for validation of SoC design," in *Proc. Design Automation & Test in Europe Conf.*, Mar. 2007, pp. 1–6.

[2] R. Ginosar, "Fourteen ways to fool your synchronizer," in *Proc. Intl. Symp. Asynchronous Circuits and Systems*, May 2003, pp. 89–96.

[3] M. Cole and D. Cohen, "Staying in sync," *Electronics*, vol. 5, no. 3, pp. 42–45, June-July 2007.

[4] "Clock domain crossing - Closing the loop on clock domain function implementation problems," Cadence Design Systems, Tech. Rep., 2004, "http://w2.cadence.com/whitepapers/cdc_wp.pdf" (last accessed 9 June, 2011).

[5] N. Hand, "The need for an automated clock-domain crossing verification solution," Mentor Graphics, Tech. Rep., May 2006, "http://www.mentor.com/products/fv/techpubs/emulation-systems-f1fc6a19-9e95-4fd0-8d84-d5e7cf0fc12a-dt?selid=28966" (last accessed 9 June, 2011).

[6] S. Sarwary and S. Verma, "Critical clock-domain-crossing bugs," *Electronics Design, Strategy, News*, pp. 55–60, April 2008.

[7] C. Kwok, V. Gupta, and T. Ly, "Using assertion-based verification to verify clock domain crossing signals," in *Proc. Design and Verification Conf.*, Feb. 2003, pp. 654–659.

Table VII
SAMPLING RESULTS: NO. OF DETECTED SLOW-TO-RISE FAULTS FOR DIFFERENT PATTERN SETS

Benchmark	Method	$n = 5$	$n = 10$	$n = 15$	$n = 20$
ac97_ctrl	CoDC	10,990	12,743	13,786	14,448
	Top-off	22,671	20,917	19,873	19,215
	CoDC + top-off	33,661	33,660	33,659	33,663
mem_ctrl	CoDC	5,145	6,307	6,521	6,729
	Top-off	12,787	11,631	11,412	11,205
	CoDC + top-off	17,932	17,938	17,933	17,934
usb_funct	CoDC	21,865	23,618	24,052	24,465
	Top-off	13,566	11,812	11,376	10,963
	CoDC + top-off	35,431	35,430	35,428	35,428

Table VIII
SAMPLING RESULTS: NO. OF DETECTED S-CDC FAULTS FOR DIFFERENT PATTERN SETS

Benchmark	Method	$n = 5$	$n = 10$	$n = 15$	$n = 20$
ac97_ctrl	CoDC	62	73	79	83
	Top-off	21	12	6	8
	CoDC + top-off	83	85	85	91
mem_ctrl	CoDC	47	68	78	84
	Top-off	41	13	13	4
	CoDC + top-off	88	81	91	88
usb_funct	CoDC	233	328	388	424
	Top-off	283	197	149	112
	CoDC + top-off	516	525	537	536

Table IX
SAMPLING RESULTS: COMPARISON BETWEEN NO. OF TEST PATTERNS FOR DIFFERENT PATTERN SETS

Benchmark	Method	$n = 5$	$n = 10$	$n = 15$	$n = 20$
ac97_ctrl	CoDC	61	69	71	75
	Top-off	365	369	373	370
	CoDC + Top-off	426	438	444	445
mem_ctrl	CoDC	46	64	71	75
	Top-off	767	800	811	781
	CoDC + Top-off	813	864	882	856
usb_funct	CoDC	228	323	383	418
	Top-off	768	744	722	749
	CoDC + Top-off	996	1,067	1,105	1,167

[8] T. Kapschitz and R. Ginosar, "Formal verification of synchronizers," in *Correct Hardware Design and Verification Methods*, ser. Lecture Notes in Computer Science, D. Borrione and W. Paul, Eds. Springer, 2005, vol. 3725, pp. 359–362.

[9] B. Li and C. Kwok, "Automatic formal verification of clock domain crossing signals," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan. 2009, pp. 654–659.

[10] M. Litterick, "Pragmatic simulation-based verification of clock domain crossing signals and jitter using SystemVerilog assertions," in *Proc. Design and Verification Conf.*, Feb. 2006, pp. 1–6.

[11] C. Leong *et al.*, "Built-in clock domain crossing (CDC) test and diagnosis in GALS systems," in *Proc. Intl. Symp. Design and Diagnostics of Electronic Circuits and Systems*, Apr. 2010, pp. 72–77.

[12] T. Ly, N. Hand, and C. Kwok, "Formally verifying clock domain crossing jitter using assertion-based verification," in *Proc. Design and Verification Conf.*, 2004.

[13] M. Su, Y. Chen, and X. Gao, "A general method to make multi-clock system deterministic," in *Proc. Design Automation & Test in Europe Conf.*, Mar. 2010, pp. 1480–1485.

[14] C. Albrecht, "IWLS 2005 benchmarks," in *Proc. Intl. Wksp. Logic Synthesis*, 2005.