SOC Test Architecture Optimization for Signal Integrity Faults on Core-External Interconnects

Qiang Xu[†], Yubin Zhang[†] and Krishnendu Chakrabarty[‡]
[†] Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
[‡] Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA
{qxu,ybzhang}@cse.cuhk.edu.hk; krish@ee.duke.edu

ABSTRACT

The test time for core-external interconnect shorts/opens is typically much less than that for core-internal logic. Therefore, prior work on test infrastructure design for core-based system-on-a-chip (SOC) has mainly focused on minimizing the test time for core-internal logic. However, as feature sizes shrink for newer process technologies, the test time for interconnect signal integrity (SI) faults cannot be neglected. We investigate the impact of interconnect SI tests on SOC test architecture design and optimization. We present a compaction method for SI faults and algorithms for test architecture optimization. Experimental results for the ITC'02 benchmarks show that the proposed approach can significantly reduce the overall testing time for core-internal logic and core-external interconnects.

Categories and Subject Descriptors

B.7.3 [**Integrated Circuits**]: Reliability and Testing

General Terms

Reliability, Design, Algorithms.

Keywords

Signal Integrity, Interconnects, Test Architecture Optimization

1. INTRODUCTION

As feature sizes shrink and clock frequencies increase for high-performance system-on-a-chip (SOC) designs, signal integrity (SI), the ability of an input signal to generate correct responses in a circuit [9], has become a major concern for the interconnects between embedded cores. Signal integrity problems, caused by cross-coupling capacitance and inductance between interconnects, include overshoot, undershoot, glitches, oscillation, excessive signal delay and even signal speedup [13]. If the noise-induced voltage swing and/or timing skew depart from the noise-immune region, functional error may occur. In addition, process variation and manufacturing defects may aggravate the coupling effects between interconnects [18]. A number of physical design and fabrication solutions (e.g., [3]) have been proposed in the literature to tackle signal integrity problems. Since it is unacceptable to over-design the circuit to tolerate all possible process variations and it is impossible to predict the occurrence of defects, manufacturing test strategies are essential for detecting SI-related errors [5, 22, 23].

Various signal integrity fault models (e.g., [5, 13, 24]) and associated test methodologies (e.g., [2, 23]) have been proposed in the literature. The test time for SI faults is high because of the need to exercise a large number of signal-state combinations for the interconnects [22, 23]. SI-related problems are aggravated in core-based SOC designs because interconnects carrying signals between embedded cores tend to be long and hence they suffer more from parasitic effects [19]. Despite this problem, most prior work on the testing of core-based SOCs has focused on core-internal test (InTest) only [26] and neglected the problem posed by core-external interconnect SI faults. For nanometer SOCs running at speeds in the range of hundreds of MHz and higher, the test time for SI faults can be as high as or even exceed the test time for the embedded cores. Therefore, the goal of this paper is to study, for the first time, the impact of SI faults on SOC test architecture design. The main contributions of this paper are as follows:

- We present a two-dimensional SI test compaction strategy to reduce the interconnect SI test data volume.
- We develop algorithms for SOC test architecture optimization to minimize the overall SOC testing time for both SI faults and core-internal faults.

The remainder of this paper is organized as follows. Section 2 reviews prior work in this domain and motivates the work described in this paper. Section 3 presents the proposed test compaction method for SI faults. In Section 4, SOC test architecture optimization techniques for handling both core-internal faults and core-external SI faults are introduced. Experimental results for benchmark SOCs [16] are presented in Section 5. Finally Section 6 concludes this paper.

2. RELATED WORK AND MOTIVATION

Early attempts for testing SI-related problems modeled crosstalk at the circuit level [1, 4]. Although more accurate than gate-level models, the complexity of the test pattern generation procedures limits its usefulness for SOC interconnects. Cuviello et al. [5] proposed a behavioral level SI fault model, called *maximal aggressor (MA)* model. This approach assumes that all aggressors make the same simultaneous transition in the same direction and act collectively to generate the glitch when the victim is quiescent or the delay error when the victim makes an opposite transition. (An interconnect on which the error effect takes place is defined as the *victim*, while the affecting interconnects are referred to as its *aggressors*.) Therefore, only $6N$ test vector pairs are needed to detect SI faults for a set of N interconnects. If all the physical defects are capacitive or resistive, all MA faults can be targeted using a pattern count that is linear in the number of interconnects. When inductance is considered, however, such test patterns may not be able to generate maximum noise/delay on the victim line [4, 17], hence Tehranipour et al. [24] presented a *multiple transition (MT)* fault model that covers all transitions on victim and multiple transitions on aggressors. The number of test patterns for this MT fault model, however, is exponential in the number of interconnects under test. To address this problem, an empirically-determined locality factor k showing how far the effect of aggressors remains significant, was introduced. For a set of N interconnects, the number of test patterns for the reduced MT fault model is roughly $N \cdot 2^{2k+2}$.

Built-In Self-Test (BIST) has been the primary test method used to detect SI-related errors [20, 24]. At the driver side, test generators

DAC 2007, June 4–8, 2007, San Diego, California, USA.

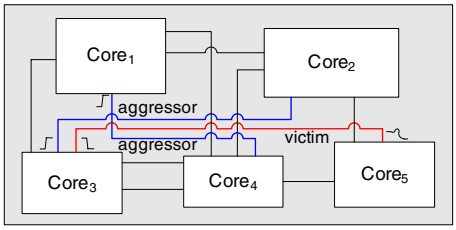

Figure 1: An arbitrary SOC interconnect topology.

are embedded to generate transitions on aggressors and the victim. At the receiver side, various types of integrity loss sensor (ILS) cells (e.g., [2, 23]) are designed to detect SI-related errors. Hardware-based test generators, however, may cause over-testing and/or under-testing since not all test patterns generated in the test mode are valid in the normal functional mode of the SOC. In addition, because the SOC interconnect topology can be arbitrary (see Fig. 1), the interconnects between several cores may be close enough to result in SI error [22]. It is very difficult, if not impossible, to take this into account for these hardware-based techniques. As a result, in this work, the test stimuli are assumed to be loaded from an external tester.

Most prior work in SOC test architecture optimization [26] only considers core internal testing. This is mainly because testing interconnect shorts/opens requires little time, hence core-external (ExTest) testing can be ignored for test architecture optimization. However, when high SI fault coverage is desired, the testing time for SOC interconnects can be comparable to the testing time for the core-internal logic. To understand this issue, let us estimate the SI testing time for an on-chip 32-bit functional bus, based on the popular MA and reduced MT fault models, when serial ExTest is used. Suppose ten cores connect to the bus, and let us assume that on average, each core sends data to two other cores on the bus. Hence, the number of victim interconnects under test is $N = 2 \times 10 \times 32 = 640$. Based on the above discussion, without test set compaction, 3840 test vector pairs are needed for the MA fault model; while roughly 163840 test vector pairs are needed for the reduced MT fault model before test set compaction when the locality factor $k = 3$. Since the sum of the numbers of all the core I/Os for a typical SOC is in the range of several thousand, the test time for MA faults is in the range of millions of clock cycles for serial ExTest, while the test time for reduced MT faults is two orders of magnitude higher. As reported in [7], the test time for a representative video processing SOC is less than 2 million clock cycles when the total number of test access mechanism (TAM) wires is 140, which in turn is less than the testing time for the above SI faults. Moreover, with shrinking feature sizes of deep-submicron technology, short interconnects may also suffer from SI problems [19]. Therefore, it is likely that we need to detect SI faults for hundreds or even thousands of interconnects in the SOC. Prohibitively high test time is needed if the SOC test architecture is not optimized for both core-internal logic test and interconnect SI test.

Three conclusions can be drawn from the above discussion:

- Effective test set compaction strategy should be utilized to reduce the volume of test data for SI faults;
- Parallel external testing is required in order to reduce the test time for interconnect SI faults;
- The SOC test architecture needs to minimize the overall testing time for both core-internal logic and core-external interconnects.

The above observations motivate the work presented in this paper. We use the TestRail architecture [14] because, in contrast to the Test Bus architecture [25], it naturally supports parallel external testing. To apply SI test at the core-level, the wrapper output cell (WOC) should be able to provide the necessary consecutive transitions. The wrapper input cell (WIC) needs to include a signal integrity loss sensor (e.g., [2, 23]), to capture the signal with noise and/or delay. This paper considers, as a starting point, that every core in the SOC uses

	$core_1$ WOC	...	$core_i$ WOC	...	$core_n$ WOC	Bus
p_1	... x ↑ x x ↓	...	x 1 ↑ ... x	...	x x x ...	x x 1 ...
p_2	... ↑ x ↓ x x	...	x x x ... ↑	...	x x ↑ ...	x x 1 ...
...		...		...		
p_x	... x x x x x	...	0 x ↓ ... x	...	↓ x x ...	1 x x ...

Table 1: Format of the SI test patterns.

such wrappers [23]. These wrappers are compatible with the IEEE 1500 standard [10] — some additional hardware is added to the wrappers for signal integrity test. In addition, the user-defined logic is also treated as a wrapped core. In other words, the SOC is assumed to contain wrapped cores and interconnect wires (or simple glue logic) that are affected by signal integrity faults, without large blocks of logic between the embedded cores.

3. TWO-DIMENSIONAL SI TEST SET COMPACTION

Since a victim interconnect is mainly affected by its neighboring aggressors [13, 22], the SI test patterns typically feature a large number of don't-care bits [24]. The format for test patterns is shown in Table 1, where 'x' represents the "don't-care" bit; '0' or '1' indicates that the corresponding core output terminal stays at '0' or '1' in consecutive cycles; while ↑ and ↓ represent a positive transition and a negative transition respectively. For each test pattern, we also add a postfix to denote whether this test pattern utilizes a shared bus line (as discussed in the following paragraph). A '1' indicates that the specific bus line is utilized while 'x' represents that the choice of the bus line is a "don't-care".

Test Pattern Count Reduction. Multiple test patterns are compacted into one pattern if they are compatible (i.e., their intersection is non-empty) to reduce the pattern count. Since bus lines are test resources that are shared by multiple cores, it is possible that several SI test patterns trigger the same bus line from different core boundaries and these patterns should not be compacted into one pattern. The problem of finding the minimum compacted test set for a given test set can be formulated as the clique covering problem [12]. A graph is created such that each vertex corresponds to a test pattern and an edge is added between two vertices if the corresponding two test patterns are compatible. Then a set of compatible SI test patterns form a clique in this graph which represents a compacted pattern. The objective is to find minimum number of cliques covering all the vertices in the graph. The clique covering problem is NP-complete [6]. To reduce computational complexity, we use a greedy heuristic which merges the first uncompacted pattern with its following compatible patterns in each cycle. Experiments show that we can achieve similar compaction ratios as approximation algorithms for the clique covering problem with significantly less computation time.

Test Pattern Length Reduction. The above compaction scheme to reduce test pattern count can be viewed as reducing the volume of the test data in a vertical manner. As each SI test pattern involves only a few cores' terminals (denoted as *care cores* of the SI test pattern), we can bypass the boundaries of those *don't-care cores* (e.g., core 1 for p_x in Table 1) and reduce the length of this test pattern. This strategy can be viewed as compacting the test pattern in a horizontal manner. That is, instead of compacting all the test patterns as a whole and hence the length of every compacted test pattern remains the sum of all cores' WOCs, we propose to partition the entire SI test set into several groups and compress them separately so that the test pattern length for each group can be less.

We classify the SI test patterns in such a way that, the test patterns whose care cores are all within the same core group form a SI test group, in which the length of each test pattern is reduced to be the sum of the WOCs of this core group. For the remaining test patterns whose care cores fall into multiple core groups, we simply group them as a whole and their length remains the sum of the lengths of the WOCs for all the cores. To achieve maximum compression, the objective is to minimize the number of remaining patterns and at the same time each partition has roughly balanced test pattern length. This problem

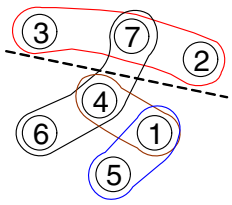

Figure 2: Hypergraph partitioning for SI test pattern length reduction.

can be formulated as a hypergraph partitioning problem, with each vertex in the hypergraph corresponding to a core (the weight of each vertex is the number of WOCs for the core). A hyperedge is added for each test pattern that connects all its care cores (vertices). Since there might be multiple test patterns having the same care cores, we use the weight of each hyperedge to represent this information.The hypergraph partitioning problem has been well-researched in the literature and we reuse the *hMetis* package [21] to solve our problem. As shown in Fig. 2, for the horizontal SI test pattern compaction of a hypothetical SOC containing seven cores, the patterns corresponding to the cut hyperedge 7-4-6 need to load the WOCs for all the cores, while the other patterns can be applied with shorter pattern lengths (vertex and edge weights are not shown in the figure). Our experiments show that the proposed two-dimensional SI test set compaction strategy is able to reduce test data volume significantly.

4. TEST ARCHITECTURE OPTIMIZATION

As discussed in Section 2, the testing time for interconnect SI faults can be comparable to or even higher than the testing time for the core-internal logic. Therefore, system integrators need to optimize the SOC test architecture, i.e., design the TAM, for both types of tests in order to reduce the overall testing time. The TAM optimization problem can be stated as follows:

Problem P_{SI_opt}: Given the maximum TAM width W_{max} for the SOC, and

- the test set parameters for each embedded core, including the number of input and output terminals, the number of test patterns for core internal logic, the number of scan chains and the length of each scan chain;
- the test set parameters for each group of compacted SI tests, including the set of cores involved and the number of interconnect SI test patterns;

Determine the wrapper design for each core, the TAM resources assigned to each core and a test schedule for the entire SOC such that: (i) the sum of the TAM width used at any time does not exceed W_{max}; (ii) the total SOC testing time T_{soc} is minimized.

One of the subproblems of P_{SI_opt} is to design and optimize the test wrapper for each core. Since the test application time of a core is dependent on the length of the maximum wrapper scan chain[1], the main objective in wrapper design and optimization is to build balanced wrapper scan chains. This is a well-researched problem [15, 11], and we use the *Combine* procedure from [15] for solving it in *InTest* mode. For a core wrapper in SI test mode, wrapper scan chains contain wrapper cells only and we assume without loss of generality that balanced wrapper input/output scan chains are achieved.

As the same wrapper cells are used for both core-internal logic test and core-external interconnect SI test, these two kinds of tests are scheduled at different times to avoid test resource conflict. Therefore, $T_{soc} = T^{in}_{soc} + T^{si}_{soc}$, where T^{in}_{soc} and T^{si}_{soc} denote the core-internal testing time for all the cores and the total core-external interconnects test for all the SOC, respectively. For a given TAM architecture, T^{in}_{soc} is the maximum internal logic testing time on any partition of the TAM. The schedule for the core-internal tests, i.e., the sequence in which

[1]Wrapper scan chains are built by concatenating core internal scan chains and WBR cells for InTest.

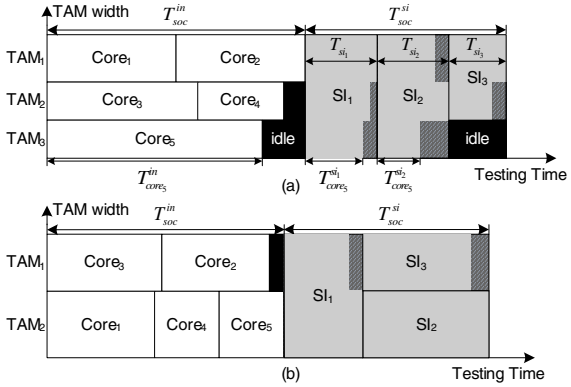

Figure 3: Example TAM designs and their corresponding test schedules.

the cores are tested for a given TAM architecture, does not affect T^{in}_{soc}. However, the calculation of T^{si}_{soc}, is complicated by the fact that multiple TAM partitions may be involved in one SI test. Before introducing our data structure and algorithms, let us use the following example to calculate the SI testing time for two TAM designs:

Example 1: Two possible TAM designs and their corresponding test schedules for the same SOC are shown in Fig. 3. The SI test has been grouped into three groups, in which SI_1 group involves all the five embedded cores, SI_2 group involves $Core_1$, $Core_4$ and $Core_5$, and SI_3 group involves $Core_2$ and $Core_3$. $T^{in}_{core_i}$ stands for the time for the internal logic test time of core i, while $T^{si_j}_{core_i}$ denotes the interconnect testing time due to core i for SI test group j. The value of T_{si_j} for each SI test group j is determined by the bottleneck TAM (the TAM with the maximum test time) for this SI test. Therefore, for the TAM design and test schedule shown in Fig. 3(a),

$$\begin{aligned} T_{si_1} &= \max\{T^{si_1}_{core_1} + T^{si_1}_{core_2}, T^{si_1}_{core_3} + T^{si_1}_{core_4}, T^{si_1}_{core_5}\} \\ &= T^{si_1}_{core_1} + T^{si_1}_{core_2}; \end{aligned}$$

while for the TAM design and test schedule shown in Fig. 3(b),

$$\begin{aligned} T_{si_1} &= \max\{T^{si_1}_{core_1} + T^{si_1}_{core_4} + T^{si_1}_{core_5}, T^{si_1}_{core_2} + T^{si_1}_{core_3}\} \\ &= T^{si_1}_{core_1} + T^{si_1}_{core_4} + T^{si_1}_{core_5} \end{aligned}$$

It can be easily seen that, under different TAM architectures, even for the same SI test involving the same TAM resources (i.e., SI_1 in Fig. 3(a) and Fig. 3(b) using all TAM wires), their SI testing times can be different. Therefore, based on TestRail architecture, we propose to solve Problem P_{SI_opt} in two steps: First, in Section 4.1, we describe how to schedule SI tests for a given TAM design. Next in Section 4.2, we describe our solution for the general problem of designing and optimizing the SOC test architecture based on [8].

4.1 SI Test Scheduling for a Given TAM Design

Data Structure. The data structures that we use to store the SI test group information and the TestRail configuration are presented in Fig. 4. The two data structures are updated whenever the SOC TAM design is changed. In particular, in data structure for *TestRail* r, we use $time_{in}(r)$, $time_{si}(r)$ and $time_{used}(r)$ to denote the internal testing time, the SI testing time and the utilized testing time on TAM r, respectively.

For example, for TAM_3 shown in Fig. 3(a),

- $time_{in}(r) = T^{in}_{core_5}$
- $time_{si}(r) = T^{si_1}_{core_5} + T^{si_2}_{core_5}$
- $time_{used}(r) = time_{in}(r) + time_{si}(r) = T^{in}_{core_5} + T^{si_1}_{core_5} + T^{si_2}_{core_5}$

We use $time_{used}(r)$ to compare the actual utilization of TAM resources for different TAM partitions.

Calculating Testing Time for Individual SI Test. We use a procedure, called *CalculateSITestTime*, to calculate $time_{si}(s_i)$ of each SI test group s_i, given the TestRail architecture R_{soc} and all the SI test

Data structure SI Test s

1. $C(s)$; /* The set of cores involved */
2. $pattern(s)$; /* Number of patterns */
3. $begin(s)$; /* Schedule begin time */
4. $end(s)$; /* Schedule end time */
5. $time_{si}(s)$; /* SI Testing time */
6. $R_{tam}(s)$; /* The set of TAMs involved */
7. $r_{btn}(s)$; /* The bottleneck TAM for this SI test */

Data structure TestRail r

1. $C(r)$; /* The set of cores on TestRail r */
2. $width(r)$; /* TAM width of r */
3. $time_{in}(r)$; /* Internal testing time */
4. $time_{si}(r)$; /* Utilized SI testing time */
5. $time_{used}(r)$; /* Utilized total testing time */

Figure 4: Data structures for SI test and TestRail

groups S_{soc}. For each SI test s_i, the SI testing time $time_{si}(s_i)$ is the longest SI testing time of all the TAMs involved in s_i while the testing time of each TAM is the sum of the SI testing time of each core in that TAM involved in s_i.

Scheduling of SI Tests. The procedure to schedule SI tests and determine the SOC SI testing time T^{si}_{soc} is shown in Fig. 5. It takes the TestRail architecture R_{soc} and all the SI test groups S_{soc} as inputs. Line 1 calculates the testing time of each SI test (as in Example 1). Line 2 initializes *unSchedSI*, the unscheduled SI tests and *currSchedTAMs*, the TAMs that are utilized by the SI tests currently under schedule. Line 3 initializes *currTime*, i.e., the begin time for to-be-scheduled SI test. After initialization, the loop that follows schedules SI test one by one (Lines 4-17). Inside the loop, we first try to find a SI test s^* that can be scheduled with begin time *currTime*, that is, s^* does not utilize any TAM in *currSchedTAMs* (Line 5). If such s^* can be found, we schedule it by updating $begin(s^*)$, $end(s^*)$, *currSchedTAMs*, and *unSchedSI* (Lines 7-10). If s^* is the last scheduled SI test, we shall update T^{si}_{soc} as the end time of s^*, $end(s^*)$. If all the unscheduled SI tests utilize the TAM resources in *currSchedTAMs* and hence cannot be scheduled with begin time *currTime*, we find *nextTime*, i.e., the end time for the first SI test that is expected to end after *currTime* (Line 14). We then update the begin time of the to-be-scheduled SI tests (Line 15) and *currSchedTAMs* based on the SI tests still under schedule (Line 16). Finally the procedure returns the SOC SI testing time T^{si}_{soc} and the SI tests with updated schedule information.

4.2 TAM Design and Optimization

The above discussion for calculating T^{si}_{soc} is based on a given TAM architecture. What makes Problem P_{SI_opt} more difficult is that $time_{si}(s)$, the testing time for an SI test, is unknown until the final SOC test architecture is in place. This is fundamentally different from the problem optimizing the SOC test architecture for core-internal logic test only, in which the test time for each core can be predetermined for a given TAM width [26]. Unlike many test scheduling algorithms that schedule cores one after another, and stop after all cores are scheduled, the $TR-Architect$ algorithm proposed in [8] generates an initial test architecture with all cores assigned to TAMs and then optimizes this architecture. This strategy is particularly attractive for interconnect SI test as we are able to calculate the SI testing time during the optimization process. Therefore, we adapt this algorithm for solving Problem P_{SI_opt} in this paper.

Identifying Bottleneck TAMs. The basic idea of the $TR-Architect$ algorithm is to optimize T^{in}_{soc} at the TAM level by merging TAMs and/or distributing free TAM wires to the bottleneck TAM, i.e., the TAM with the longest T^{in}_{tam}. Either two *non-bottleneck* TAMs are merged with less TAM width to release freed TAM resources to the bottleneck TAM, or the bottleneck TAMs is merged with another TAM to decrease T^{in}_{soc}.

Algorithm 1 - ScheduleSITest

INPUT: R_{soc}, S_{soc}
OUTPUT: S'_{soc}, T^{si}_{soc}

1. $S'_{soc} = CalculateSITestTime(R_{soc}, S_{soc})$;
2. initialize $unSchedSI = S'_{soc}$; $currSchedTAMs = \emptyset$;
3. initialize $currTime = 0$;
4. **while** $unSchedSI \neq \emptyset$ {
5. find $s^* \in unSchedSI$ for which $R_{tam}(s^*) \cap currSchedTAMs = \emptyset$;
6. **if** found {
7. set $begin(s^*) = currTime$;
8. set $end(s^*) = begin(s^*) + time_{si}(s^*)$;
9. $currSchedTAMs = currSchedTAMs \cup R_{tam}(s^*)$;
10. $unSchedSI = unSchedSI \setminus \{s^*\}$;
11. **if**($unSchedSI == \emptyset$) {
12. $T^{si}_{soc} = end(s^*)$; }
13. } **else** {
14. calculate $nextTime = end(s')$ such that
. $end(s') > currTime$ and $end(s')$ is the minimum;
15. set $currTime = nextTime$;
16. update $currSchedTAMs$;
. }
. }
17. **return** S'_{soc}, T^{si}_{soc};

Figure 5: Procedure for scheduling SI tests.

However, since we are minimizing $T_{soc} = T^{in}_{soc} + T^{si}_{soc}$, it is possible that multiple bottleneck TAMs exist at the same time. That is, in addition to the bottleneck TAM for core internal logic test, each SI test has its own bottleneck TAM, which may affect the total SOC testing time T_{soc}. As a result, we define the *bottleneck TAMs* of the SOC to be those which are critical to the test time; T_{soc} is reduced if extra wires are assigned to them. The remaining TAMs are referred to as *non-bottleneck TAMs* of the SOC. For the schedule shown in Fig. 3(a), TAM_1 and TAM_2 are bottleneck TAMs and TAM_3 is a non-bottleneck TAM. For the schedule shown in Fig. 3(b), TAM_2 is a bottleneck TAM and TAM_1 is a non-bottleneck TAM. During the TAM optimization procedure, whenever there are extra free TAM wires, we should distribute them only to the bottleneck TAMs.

Algorithm for Problem P_{SI_opt}. As in $TR-Architect$, we first create an initial TestRail architecture and optimize it by merging TAMs and distributing free TAM wires afterwards. There are two key questions during the optimization process, namely, "How to find out the merging candidate and merge them?" and "How to distribute free TAM wires?". Because there may exist multiple bottleneck TAMs at the same time in our problem, the answers to these two questions highlight the main differences between our algorithm and $TR-Architect$.

In our procedure for merging TAMs, referred to as *mergeTAMs*, with the given TestRail architecture R_{soc}, all the SI tests S_{soc} and one candidate TAM r_1 to be merged as inputs, we look for another TAM candidate in $R_{find} = R_{soc} \setminus \{r_1\}$, which leads to the lowest testing time after merging with r_1. We enumerate every TAM r_i in R_{find} as a candidate for merging, and attempt to merge r_i and r_1 with different TAM widths in the range of $width_{min} = \max\{width(r_i), width(r_1)\}$ and $width_{max} = width(r_i) + width(r_1)$ and distribute these leftover free TAM wires. The intuition behind this is that we may be able to merge two TAMs with TAM width less than the sum of their initial TAM width and the freed TAM wires can be assigned to other bottleneck TAMs to reduce T_{soc}. The procedure outputs the TestRail architecture with the lowest testing time after merging. It is also possible that we cannot find a merging plan to reduce T_{soc}. In such case, the original TestRail architecture is returned.

The procedure for distributing free TAM wires, namely *distributeFreeWires*, takes the given TestRail architecture R_{soc}, all the SI tests S_{soc} and the number of free TAM wires as inputs. Each of these free TAM wires is distributed iteratively to the bottleneck TAMs. Since

Algorithm 2 - TAM_Optimization

INPUT: C_{soc} /* Set of embedded cores */
W_{max} /* Given SOC TAM width */
S_{soc} /* SI test groups */
OUTPUT: R_{soc} /* TestRail architecture */

1. initialize $R_{soc} = \emptyset$;
. /* Create a start solution */
2. **for** all $c_i \in C_{soc}$ {
3. create TAM r_i such that $C(r_i) = \{c_i\}$;
4. $width(r_i) = 1$; $time_{in}(r_i) = time_{in}(c_i)$;
5. $R_{soc} = R_{soc} \cup \{r_i\}$;
. }
6. calculate $time_{si}(r)$ and $time_{used}(r)$ for all $r \in R_{soc}$;
7. **if** ($W_{max} < |R_{soc}|$) {
8. **for**($i = W_{max} + 1$ to $|R_{soc}|$) {
9. **sort** R_{soc} such that $time_{used}(r_1) \geq$
. $time_{used}(r_2) \geq ... \geq time_{used}(r_{|R_{soc}|})$;
10. find $r_i (1 \leq i \leq W_{max})$ such that
. T_{soc} is the minimum when merging with $r_{W_{max}+1}$;
11. $r_i = r_i \cup r_{W_{max}+1}$; update $time_{in}(r_i)$;
12. $R_{soc} = R_{soc} \setminus r_{W_{max}+1}$;
13. update $time_{si}(r)$ and $time_{used}(r)$ for all $r \in R_{soc}$;
. }
14. } **else if** ($|R_{soc}| < W_{max}$) {
15. set $numFreeWires = W_{max} - |R_{soc}|$;
16. $R_{soc} = distributeFreeWires(R_{soc}, S_{soc}, numFreeWires)$;
. }
. /* Optimize the TestRail architecture from bottom up */
17. set ISIMPROVED=TRUE;
18. **while**(ISIMPROVED **AND** $|R_{soc}| > 1$) {
19. set $initTestingTime = T_{soc}$;
20. **sort** R_{soc} such that
. $time_{used}(r_1) \geq time_{used}(r_2) \geq ... \geq time_{used}(r_{|R_{soc}|})$;
21. $R_{soc} = mergeTAMs(R_{soc}, S_{soc}, r_{|R_{soc}|})$;
22. **if**($T_{soc} == initTestingTime$) {
23. ISIMPROVED=FALSE; }
. }
. /* Optimize the TestRail architecture from top down*/
24. set ISIMPROVED=TRUE;
25. **while**(ISIMPROVED **AND** $|R_{soc}| > 1$) {
26. set $initTestingTime = T_{soc}$;
27. **sort** R_{soc} such that
. $time_{used}(r_1) \geq time_{used}(r_2) \geq ... \geq time_{used}(r_{|R_{soc}|})$;
28. $R_{soc} = mergeTAMs(R_{soc}, S_{soc}, r_1)$;
29. **if**($T_{soc} == initTestingTime$) {
30. ISIMPROVED=FALSE; $R_{skip} = \{r_1\}$; }
. }
31. **while**($R_{skip} \neq R_{soc}$) {
32. set $R_{temp} = R_{soc} \setminus R_{skip}$; $initTestingTime = T_{soc}$;
33. find $r^* \in R_{temp}$ such that $time_{used}(r^*)$ is the maximum;
34. $R_{soc} = mergeTAMs(R_{soc}, S_{soc}, r^*)$;
35. **if**($T_{soc} = initTestingTime$) {
36. $R_{skip} = R_{skip} \cup \{r_1\}$; }
. }
37. $coreReshuffle(R_{soc}, S_{soc})$;
38. **return** R_{soc}, T_{soc};

Figure 6: Overall procedure for solving P_{SI_opt}.

we may have multiple bottleneck TAMs at the same time, we select one TAM based on the criterion that T_{soc} is the minimum after this TAM obtains the extra TAM wire. Because R_{soc} is changed whenever a free TAM wire is assigned, $time_{si}(r)$ and $time_{used}(r)$ for every $r \in R_{soc}$ should also be updated. Finally the procedure outputs the new TestRail architecture R'_{soc} with all free TAM wires assigned.

The pseudocode for our algorithm *TAM_Optimization* for Problem P_{SI_opt} is presented in Fig. 6. First, we create a start solution (Lines 1-16). This mainly consists of three steps. In Step 1 (Lines 2-5), we assign each core to a one-bit wide TAM and we calculate the testing time of the core internal logic $time_{in}(r)$, the SI testing time of the interconnects $time_{si}(r)$ and the actual utilized testing time $time_{used}(r)$ for every $r \in R_{soc}$. In case $W_{max} < |R_{soc}|$, we do not have enough TAM wires and hence we need to merge TAMs together (Lines 7-13). We first sort R_{soc} based on the total utilized testing time in each TAM (Line 9), then $r_{W_{max}+1}$ is merged iteratively with another TAM r_i. We select this merging candidate r_i based on the criteria that T_{soc} is the minimum after merging with $r_{W_{max}+1}$ (Line 10). Since R_{soc} is changed after merging, $time_{si}(r)$ and $time_{used}(r)$ for every $r \in R_{soc}$ are updated (Line 13). In the case $W_{max} > |R_{soc}|$, we have extra free TAM wires left and procedure *distributeFreeWires* is called to distribute them.

Next, we optimize the TAM architecture by merging the TAM with the lowest $time_{used}$ with another TAM (Lines 17-23). We first sort R_{soc} in non-increasing order and we select $r_{|R_{soc}|}$ as one of the merging candidate r_1, then we call procedure *mergeTAMs* to search for another TAM to merge with r_1 and possibly redistribute TAM resources to reduce T_{soc}. This is an iterative procedure and it stops when no reduction in T_{soc} can be achieved (Lines 22-23). Afterwards, we try to further optimize the TAM architecture by trying to merge the TAM with the longest $time_{used}$ with another TAM (Lines 25-30) and merging other TAMs (Lines 31-36). Finally, the algorithm tries to minimize T_{soc} by iteratively moving one core from bottleneck TAMs of the SOC to another TAM, if possible (Line 37).

5. EXPERIMENTAL RESULTS

To analyze the effectiveness of the proposed solution, experiments are carried out for two ITC'02 benchmark SOCs from [16], namely, p34392 and p93791. Without loss of generality, we do not consider hierarchy in the testing of core-internal logic. Since the benchmarks do not include any functional interconnect information, it is not possible to generate tests for SI faults for them. Therefore, we generate random test patterns for our experiments in the following manner. In each test pattern there are one victim and N_a $(2 \leq N_a \leq 6)$ random aggressors, where at most two aggressors are outside of the victim core boundary. The SOC-level TAM width W_{max} is varied from 8 to 64, in increments of 8. In addition, we assume that a 32-bit functional bus is utilized in both SOCs. The probability that the bus is used by a test pattern is set at 50%. If the bus is decided to be used in a particular pattern, we randomly generate $1 \sim N_a$ occupied bits in the postfix of the pattern (see Section 3).

Table 2 and Table 3 show the SOC overall test application time T_{soc} for our method compared to $TR-Architect$. We vary the initial interconnect test pattern count N_r and the SI test grouping strategy in the experiments. The value for $T_{[8]}$ is obtained by optimizing the SOC TAM architecture for core-internal testing time T^{in}_{soc} only, and then computing the total testing time T_{soc} for both core-internal test and core-external test. The testing times are reported in terms of the number of clock cycles ('cc' in the tables). The parameter T_{g_i} denotes the test time T_{soc} obtained using the proposed *TAM_Optimization* algorithm when the SI tests are partitioned into i parts, and we determine $T_{min} = \min_i\{T_{g_i}\}$, which corresponds to the the test architecture the we use. The parameters $\Delta T_{[8]}$ and ΔT_g are computed as $\Delta T_{[8]} = \frac{T_{[8]} - T_{min}}{T_{[8]}} \times 100\%$ and $\Delta T_g = \frac{T_{g_1} - T_{min}}{T_{g_1}} \times 100\%$, respectively. Note that ΔT_g quantifies the benefit derived from our two-dimensional compaction strategy over the one-dimensional compaction scheme that reduces only the test-pattern count. We can see that test time reduction is of over 10 percent in several cases (e.g., for SOC p93791, when $W_{max} = 16$ and $N_r = 100,000$). The magnitude of this reduction depends on the initial SI test set and the core configurations.

From Tables 2 and 3, we note that obliviously optimizing SOC test architectures without considering interconnect SI faults leads to much higher test time. This gap grows with an increase in the pattern count for the SI faults and the associated percentage of SI testing time in T_{soc}. We can also see that when W_{max} is small, there is no significant advantage in using proposed algorithm; in a few cases, worse results

	SOC p34392															
	$N_r = 10,000$								$N_r = 100,000$							
W_{max}	$T_{[8]}$ (cc)	T_{g_1} (cc)	T_{g_2} (cc)	T_{g_4} (cc)	T_{g_8} (cc)	T_{min} (cc)	$\Delta T_{[8]}$ (%)	ΔT_g (%)	$T_{[8]}$ (cc)	T_{g_1} (cc)	T_{g_2} (cc)	T_{g_4} (cc)	T_{g_8} (cc)	T_{min} (cc)	$\Delta T_{[8]}$ (%)	ΔT_g (%)
8	2128642	2473012	2217506	**2137599**	2159634	2137599	-0.42	13.56	3316258	3303307	2999506	3034656	**2845502**	2845502	14.20	13.86
16	1197929	**1120560**	1124839	1174501	1166925	1120560	6.46	0	2487849	1643720	1595247	1650343	**1567503**	1567503	36.99	4.64
24	812192	**816661**	892412	821588	866507	816661	-0.55	0	1519424	1141199	**1110321**	1143510	1145184	1110321	26.92	2.71
32	693458	**615984**	633155	656570	641076	615984	11.17	0	1787666	853502	845838	**835448**	845474	835448	53.27	2.12
40	685291	607404	568070	**566093**	566156	566093	17.39	6.80	1779499	747495	757767	**719065**	720219	719065	59.59	3.80
48	685291	565157	564173	**560084**	561620	560084	18.27	0.90	1779499	731255	683135	683832	**676463**	676463	61.99	7.49
56	685291	563741	559948	558712	**557692**	557692	18.62	1.07	1779499	711369	669799	665245	**658664**	658664	62.99	7.41
64	685291	563741	559642	557004	**556896**	556896	18.74	1.21	1779499	700834	666176	**635880**	650688	635880	64.27	9.27

N_r: Initial interconnect test pattern count; W_{max}: Given SOC TAM width; $T_{[8]}$: Test time obtained by optimizing the SOC TAM architecture for InTest only; T_{g_i}: Test time obtained using the proposed *TAM_Optimization* algorithm when the SI tests are partitioned into i parts; $T_{min} = \min_i\{T_{g_i}\}$; $\Delta T_{[8]} = \frac{T_{[8]} - T_{min}}{T_{[8]}} \times 100\%$; $\Delta T_g = \frac{T_{g_1} - T_{min}}{T_{g_1}} \times 100\%$.

Table 2: Test application time comparison for SOC p34392

	SOC p93791															
	$N_r = 10,000$								$N_r = 100,000$							
W_{max}	$T_{[8]}$ (cc)	T_{g_1} (cc)	T_{g_2} (cc)	T_{g_4} (cc)	T_{g_8} (cc)	T_{min} (cc)	$\Delta T_{[8]}$ (%)	ΔT_g (%)	$T_{[8]}$ (cc)	T_{g_1} (cc)	T_{g_2} (cc)	T_{g_4} (cc)	T_{g_8} (cc)	T_{min} (cc)	$\Delta T_{[8]}$ (%)	ΔT_g (%)
8	4226884	4179655	**3979541**	4066751	4054977	3979541	5.85	4.79	9279984	7157553	6629333	6482537	**6308163**	6308163	32.02	11.87
16	2191881	2073979	2056323	**2034337**	2036416	2034337	7.19	1.91	4935934	3650655	3382864	3231137	**3170750**	3170750	35.76	13.15
24	2150865	1413121	1373548	**1370751**	1391513	1370751	36.27	3.00	9532785	2459100	2258253	2190633	**2187138**	2187138	77.06	11.06
32	1076516	1040041	1050669	1061650	**1026305**	1026305	4.66	1.32	2177213	1882999	1732000	1668109	**1650127**	1650127	24.21	12.37
40	1016416	841403	838764	**824816**	829521	824816	18.85	1.97	2879472	1523711	1422586	1351591	**1338896**	1338896	53.50	12.13
48	1212780	708625	702850	**696281**	737059	696281	42.59	1.74	5969285	1256464	1176390	1169794	**1160866**	1160866	80.55	7.61
56	853904	606648	606572	**594544**	604312	594544	30.37	2.00	3479319	1071698	1008224	995447	**985120**	985120	71.69	8.08
64	784973	528843	539587	521242	**520286**	520286	33.72	1.62	3410388	923350	856743	**848786**	867541	848786	75.11	8.08

Table 3: Test application time comparison for SOC p93791

are obtained compared to SI-oblivious TAM optimization (e.g., for SOC p34392, when $W_{max} = 8$ and $N_r = 10,000$). This is mainly because the TAM design solution space is small for smaller values of W_{max}, therefore similar TAM architectures are obtained with different optimization criterion. When W_{max} is higher, we have more freedom during the TAM design process and hence the improvement offered by the new optimization procedure is more noticeable.

6. CONCLUSION

We have presented a TAM optimization flow for core-based SOCs, which considers test times for both core-internal logic and core-external signal integrity faults on interconnects. This is in contrast to prior work on test infrastructure design for core-based system-on-a-chip (SOC), which has mainly focused on only minimizing the test time for core-internal logic. As feature sizes shrink for newer process technologies, the test time for interconnect signal integrity (SI) faults cannot be neglected. We have investigated the impact of interconnect SI tests on SOC test architecture design and optimization. We have also presented a compaction method for SI test sets such that the test data volume can be reduced. Experimental results for the ITC'02 benchmarks show that the proposed approach can significantly reduce the overall testing time for core-internal logic and core-external interconnects. The test times obtained using this approach are noticeably less than that obtained by the *TR_Architect* tool, which only considers the core-internal test time during optimization.

7. ACKNOWLEDGEMENT

This work was supported in part by the Hong Kong SAR RGC Earmarked Research Grant 2150503. The authors thank Prof. Nicola Nicolici of Mcmaster University for motivating discussions and insightful comments.

8. REFERENCES

[1] A. Attarha and M. Nourani, "Test Pattern Generation for Signal Integrity Faults on Long Interconnects," in *Proc. IEEE VLSI Test Symp.*, pp. 336–341, 2002.

[2] X. Bai, S. Dey, and J. Rajski, "Self-Test Methodology for At-Speed Test of Crosstalk in Chip Interconnects," in *Proc. Design Automation Conf.*, pp. 619–624, 2000.

[3] M. Becer, et. al, "Crosstalk Noise Control in An SoC Physical Design Flow," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 23(4):488–497, Apr. 2004.

[4] W.-Y. Chen, S. K. Gupta and M. A. Breuer, "Test Generation for Crosstalk-Induced Delay in Integrated Circuits," in *Proc. International Test Conf.*, pp. 191–200, 1999.

[5] M. Cuviello, et al, "Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects," in *Proc. International Conf. on Computer-Aided Design*, pp. 297–303, 1999.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman Publishers, 1979.

[7] S. K. Goel, et al, "Test Infrastructure Design for the Nexperia™ Home Platform PNX8550 System Chip," in *Proc. Design, Automation and Test in Europe*, Vol. 3, pp. 108–113, 2004.

[8] S. K. Goel and E. J. Marinissen, "Effective and Efficient Test Architecture Design for SOCs," in *Proc. International Test Conf.*, pp. 529–538, 2002.

[9] M. Guler and H. Kilic, "Understanding the Importance of Signal Integrity," in *IEEE Circuits and Devices Magazine*, 15(6):7–10, Nov. 1999.

[10] IEEE Standard for Embedded Core Test - IEEE Std. 1500-2004. *IEEE*, 2004.

[11] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, "Co-Optimization of Test Wrapper and Test Access Architecture for Embedded Cores," in *Springer Journal of Electronic Testing: Theory and Application*, 18(2):213–230, Apr. 2002.

[12] N. Jha and S. Gupta, *Testing of Digital Systems*, Cambridge University Press, 2003.

[13] S. Kundu, et al, "On Modeling Crosstalk Faults," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 24(12):1909–1915, Dec. 2005.

[14] E. J. Marinissen, et al, "A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores," in *Proc. International Test Conf.*, pp. 284–293, 1998.

[15] E. J. Marinissen, S. K. Goel, and M. Lousberg, "Wrapper Design for Embedded Core Test," in *Proc. International Test Conf.*, pp. 911–920, 2000.

[16] E. J. Marinissen, V. Iyengar, and K. Chakrabarty, "A Set of Benchmarks for Modular Testing of SOCs," in *Proc. International Test Conf.*, pp. 519–528, 2002.

[17] S. Naffziger, "Design Methodologies for Interconnects in GHz+ ICs," in *Proc. IEEE International Solid State Circuits Conf.* short course, Feb. 1999.

[18] S. Natarajan, M. A. Breuer, and S. K. Gupta, "Process Variations and Their Impact on Circuit Operation," in *Proc. IEEE International Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 73–81, 1998.

[19] P. Nordholz, et al, "Signal Integrity Problems in Deep Submicron Arising from Interconnects between Cores," in *Proc. IEEE VLSI Test Symp.*, pp. 28–33, 1998.

[20] K. Sekar and S. Dey, "LI-BIST: A Low-Cost Self-Test Scheme for SoC Logic Cores and Interconnects," in *Proc. IEEE VLSI Test Symp.*, pp. 417–422, 2002.

[21] N. Selvakkumaran and G. Karypis, "Multi-Objective Hypergraph Partitioning Algorithms for Cut and Maximum Subdomain Degree Minimization," in *Proc. International Conf. on Computer-Aided Design*, pp. 726–733, 2003.

[22] W. Sirisaengtaksin and S. K. Gupta, "Enhanced Crosstalk Fault Model and Methodology to Generate Tests for Arbitrary Inter-core Interconnect Topology," in *Proc. Asia Test Symp.*, pp. 163–169, 2002.

[23] M. H. Tehranipour, N. Ahmed, and M. Nourani, "Testing SoC Interconnects for Signal Integrity Using Bounary Scan," in *Proc. IEEE VLSI Test Symp.*, pp. 158–163, 2003.

[24] M. H. Tehranipour, N. Ahmed, and M. Nourani, "Testing SoC Interconnects for Signal Integrity Using Extended JTAG Architecture," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 23(5):800–811, May 2004.

[25] P. Varma and S. Bhatia, "A Structured Test Re-Use Methodology for Core-Based System Chips," in *Proc. International Test Conf.*, pp. 294–302, 1998.

[26] Q. Xu and N. Nicolici, "Resource-Constrained System-on-a-Chip Test: A Survey," in *IEE Proc. Computers and Digital Techniques*, Vol. 152, No. 1, pp. 67–81, 2005.