

Self-Test Methodology for At-Speed Test of Crosstalk in Chip Interconnects*

Xiaoliang Bai
Department of ECE
University of California, San Diego
xibai@ece.ucsd.edu

Sujit Dey
Department of ECE
University of California, San Diego
dey@ece.ucsd.edu

Janusz Rajski
Mentor Graphics Corporation
Wilsonville, OR
janusz_rajski@mentorg.com

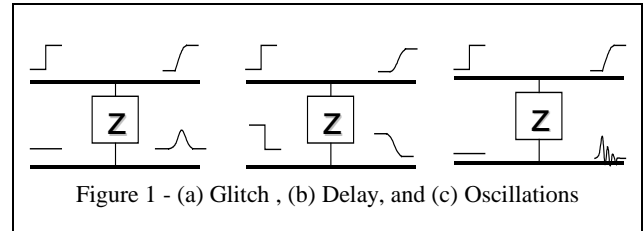
Abstract

The effect of crosstalk errors is most significant in high-performance circuits, mandating at-speed testing for crosstalk defects. This paper describes a self-test methodology that we have developed to enable on-chip at-speed testing of crosstalk defects in System-on-Chip interconnects. The self-test methodology is based on the Maximal Aggressor Fault Model [13], that enables testing of the interconnect with a linear number of test patterns. To enable self-testing of the interconnects, we have designed efficient on-chip test generators and error detectors to be embedded in necessary cores; while the test generators generate test vectors for crosstalk faults, the error detectors analyze the transmission of the test sequences received from the interconnects, and detect any transmission errors. We have also designed test controllers to initiate and manage test transactions by activating the appropriate test generators and error detectors, and having error diagnosis capability. We have developed, simulated, and synthesized parameterized HDL models of the self-test structures. We have applied the self-test methodology to test crosstalk defects in the buses of a DSP chip. Using a new high-level crosstalk simulation technique, we have validated the self-test methodology, including the self-test structures inserted in the DSP chip.

1. Introduction

Use of nano-meter technologies in System-on-Chips (SoC) increases cross-coupling capacitance and inductance between interconnects, leading to severe crosstalk effects that may result in improper functioning of the chip. Several design techniques, including physical design [1][2] and analysis tools [3][4][5], are being developed to help design for margin and minimize crosstalk problems. However, the amount of over design may be prohibitive. Moreover, it is impossible to anticipate in advance the full range of process variations, and manufacturing defects may significantly aggravate the cross-coupling effects. Hence, there is a critical need to develop testing techniques for manufacturing defects that may produce crosstalk effects.

Cross-coupling between a pair of interconnects can result in two different crosstalk effects: a glitch, or a delayed transition, depending on the nature of signal transitions at the interconnects, shown in Figure 1(a) and 1(b) respectively. In addition to glitches and delays, presence of significant coupling inductance can result



in damped voltage oscillations superimposed on top of a glitch or delay, as illustrated in Figure 1(c). If the damping is large enough, the effects of this third case may be approximated by one of the first two cases. In this paper, we focus on testing for coupling capacitance induced glitch and delay errors.

Since the crosstalk effects will be most evident in high-frequency chips, it is important to be able to test such chips at the operational speed of the system. However, as described in the 1999 ITRS [6], while ASIC speeds have improved at 30% per year, external testers' accuracy for timing signal resolution at ASIC pins have improved only at a rate of 12 % per year. Moreover, due to demands for higher speed, increased pin count, increased memory, and greater accuracy along with mixed digital and analog signals, the test equipment cost can rise toward US \$20 M. Hence, to facilitate at-speed testing, as well as circumvent the prohibitively rising cost of external ATEs, we address the problem of self-testing for crosstalk in System-on-Chip interconnects.

Empirical data has shown that crosstalk effects are most significant in long interconnects [3][13]. With the sharply increasing number of components and cores in a single chip, global interconnects will dominate future System-on-Chips. Figure 2 shows an example system chip. It comprises of several cores, C1-C12, and several bus structures, both internal to cores as well as at the system-level. Inside each component, like the CPU core C1 shown in Figure 2, the blocks like the ALU, multiplier, shifter, instruction decode unit and program counter communicate through numerous buses. At the system-level, the cores communicate using a hierarchy of buses, having different bandwidth, speed, power and other characteristics, as well as numerous core-core interconnects, like the interconnects between the MPEG decoder and interface core shown in Figure 2. Consequently, we focus on enabling at-speed testing of System-on-Chip interconnects, like buses and other inter-core wires. Our self-test methodology involves inserting self-test structures, like test generators and error detectors in appropriate cores, and a global test controller, to coordinate the self-test of the SoC interconnects, as shown in Figure 2.

Several crosstalk extraction and analysis methods [4][5] have been recently developed; while useful for design validation, they cannot be used for manufacturing testing, and the generation of the simulation vectors is not clear. Research has also started in test pattern generation for crosstalk noise [7][8]. The techniques have focused on test generation for glitches and delay faults introduced by cross-coupling capacitance in gate level circuits, and are not applicable to test SoC interconnects. Moreover, while several self-testing techniques, including Built-In Self-Test (BIST), have been developed for other manufacturing defects and

* This work is supported by the Semiconductor Research Corporation under Contract #98-TJ-648

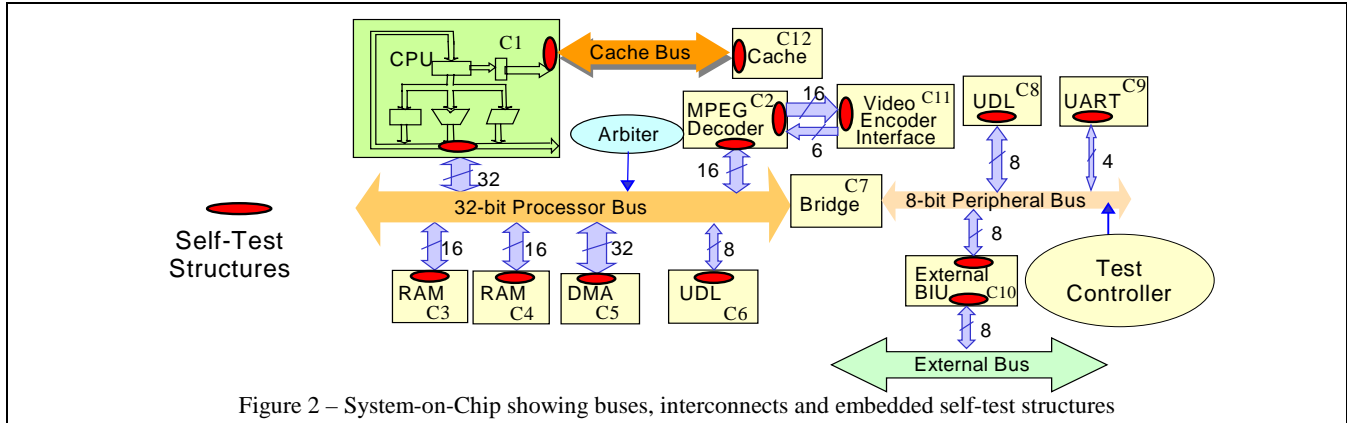


Figure 2 – System-on-Chip showing buses, interconnects and embedded self-test structures

faults, like stuck-at and delay faults, no effort has been reported to enable at-speed testing of cross-coupling defects, including any self-test techniques for crosstalk.

Our self-test methodology of SoC interconnects is based on the Maximal Aggressor Fault Model that was reported and validated in [13]. We will next briefly review the interconnect fault model, and the corresponding test transitions that are required to test the interconnect faults.

1.1 Interconnect Crosstalk Fault Model and Tests

If the cross-coupling problem has to be considered at the process level, the number of possible process variations and defects that need to be considered can be extensive even for a pair of interconnects. For wide buses, considering all such variations explicitly is clearly prohibitive. At the circuit level, a coarser mesh of lumped circuit elements can describe the cumulative effect of process variations behaviorally, but the resulting fault space is still too large. Hence, the need for an abstract fault model which will cover all the crosstalk defects with a small number of faults.

The Maximal Aggressor Fault Model (MAFM), is a high level representation of all the physical defects and process variations that lead to one of the four crosstalk errors, positive glitch (g_p), negative glitch (g_n), rising delay (d_r), falling delay (d_f), on the victim wire of the set of interconnects under test [13]. All the other wires are designated aggressors, and act collectively to generate the glitch or delay error on the victim. Figure 3 shows the transitions needed on the aggressor/victim interconnects to produce the four error types on victim wire Y_i . These are the unique Maximal Aggressor tests that are needed for the corresponding four crosstalk faults for victim Y_i .

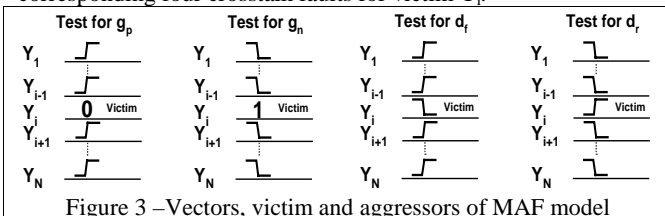


Figure 3 – Vectors, victim and aggressors of MAF model

For a set of N interconnects to be tested, a total of $4N$ faults need to be tested, requiring $4N$ 2-pattern tests. Note that these $4N$ faults cover all the possible physical defects and process variations that can lead to any crosstalk error effect on any of the N interconnects. Hence, a Maximal Aggressor (MA) test set for all the MAFM faults will cover all the crosstalk defects affecting the N interconnects. Since the required MA tests are known a-priori, if self-test structures can be inserted in the SoC to generate all the required MA tests, the resulting self-test methodology will be able to perform on-chip at-speed testing of SoC interconnects, with 100% coverage of crosstalk defects.

1.2 Paper Outline

In Section 2, we explain the requirements and selection of test transactions to test the global interconnects of system chips. In Section 3, we will describe the self-test methodology that we have developed, which involves the insertion of test generators, error detectors, and global test controllers in the system chip, as shown in Figure 2. In Section 4, we report on the application of our self-test methodology on a DSP chip, and the validation of the self-test structures using a high-level crosstalk simulation method. Section 5 concludes the paper.

2. Test Requirements and Test Transactions

Testing the bus and other core-core interconnects in a SoC for crosstalk imposes several requirements on the test vectors applied. The following issues have to be addressed while deciding on a set of test transactions, which should covers testing of all the normal core-core transactions.

Bi-directional transactions: Because of transmission line effects and variations in drivers and loads connected to buses and core-core interconnects, crosstalk may vary when interconnects are driven from different directions. For buses that are bi-directional in normal operation, testing has to be conducted in both directions.

Core-to-Core transactions: Every core-to-core transaction used in normal operation must be tested, since every core-core transaction involves a unique set of interconnects, drive strengths, and loads. Consider the system shown in Figure 4 consisting of 3 cores, sharing some interconnects (bus) between them. Though testing the inter-core communication $C1 \leftrightarrow C3$ does exercise the shared interconnects/bus, the above test is not enough to test $C2 \leftrightarrow C3$, since the latter involves a different set of interconnects, drive strengths, and loads. If all the communications $C1 \leftrightarrow C3$, $C2 \leftrightarrow C3$, and $C1 \leftrightarrow C2$ are part of the normal system operation, then all three pairs need to be tested separately.

Dynamic bus sizing: For general-purpose buses shared among several cores, the entire bus width may not be utilized for some transactions. Suppose in the system shown in Figure 4, the $C2 \leftrightarrow C3$ transaction requires only 16 bits of the 32-bit bus. Consideration must be given to the state of the unused lines during testing. They may be biased to a logic value, floating, or in transition. The difference between these cases is illustrated in Figure 5. This diagram illustrates a positive glitch test on a three-wire bus of which only two wires are used during normal operation. Depending on the state of the unused line during testing, the energy in the generated glitch may change dramatically. For testing specific bus transactions with reduced bus width, the state of the unused lines must reflect their actual operational state. Otherwise, the test may lead to overly pessimistic or optimistic testing in which a glitch (or delay) may be generated that is too large or too small.

Split bus transactions: Split bus transactions occur when a bus is partitioned between two simultaneous but unassociated bus transactions, each with reduced width. Using the same reasoning

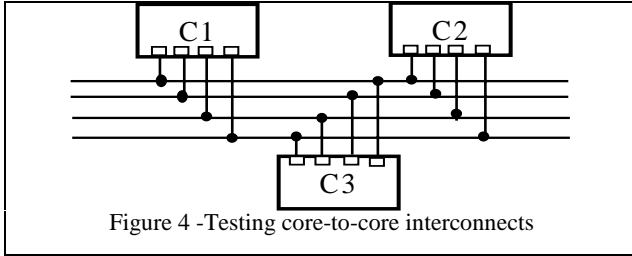


Figure 4 - Testing core-to-core interconnects

given for dynamic bus sizing, the transactions must be considered simultaneously when testing. For example, in Figure 2, consider a situation during normal operation when the processor bus is split for a C2 to C3 transaction and a simultaneous C4 to C6 transaction. When testing a victim line in the C2 to C3 transaction, along with the aggressor lines in the C2 to C3 path, all C4 to C6 lines must also be considered aggressors in order to generate the same amount of crosstalk as that seen during normal operation.

Defined by the function of the system, there is a prescribed set of valid inter-core transactions during normal operation. As shown above, the consideration of bi-directional, core to core, and split transactions, as well as dynamic bus sizing requires that the set of inter-core transactions for testing mimic those in the normal operational transaction set. In determining the set of test transactions, one way to ensure correct and complete coverage is to perform the entire set of valid operational transactions. This, however, may be more testing than necessary. For an operational set of transactions, certain transactions in the set may cover the fault situation of other transaction in the set. Additionally, a number of transactions in the set may be testable simultaneously.

3. Self-Test Structures

For each core to core test transaction, our self-testing methodology requires a test generator in the bus/interconnects interface of the source core and an error detector in the bus/interconnects interface of the destination core. For example, in the SoC shown in Figure 2, if we plan to test the transaction from core C1 (CPU) to core C3 (RAM), a test generator is inserted at the output of the CPU core, and an error detector is inserted at the inputs of the RAM core. The test vector is launched on the bus under test by the test generator(s) on one clock edge from the source core(s), and then measured for logical consistency at the other end of the bus by the error detector(s) in the destination core(s) on the subsequent clock edge with the clock running at operational speed. Since drivers and loads of the cores play a critical role in crosstalk noise, the test generators/error detectors must be located before the core's buffer connections to the bus. Additionally, to select and activate the appropriate test generators and error detectors, a global test controller is also

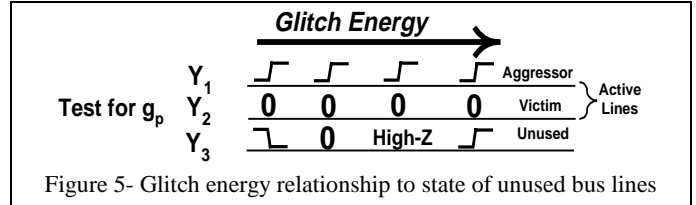


Figure 5- Glitch energy relationship to state of unused bus lines

required. Next, we will describe our designs for the test generator, error detector, and test controller.

3.1 Test Generator

Each possible fault g_p , g_n , d_r , and d_l on a victim wire of an n -bit wide bus requires a two-vector test sequence, as shown in Figure 3. However, the test vectors can be overlapped such that a six-vector test sequence can test all the four possible faults on each victim wire of the bus/global interconnection, as shown in Figure 6 (a). The test sequence has several interesting properties, enabling very efficient designs of the test generators and error detectors. For example, for each n -bit test vector, there are only two distinct values needed: the value on the victim wire, and a value that is identical on each aggressor wire.

Figure 6 (b) shows our test generator design based on the above property. The finite state machine generates the two distinct values, victim and aggressor, in each state, s_1 to s_6 . A state traversal from s_1 to s_6 corresponds to generation of the six test vectors, during which one of the bus lines will be configured as a victim (receive the victim value), and the rest of the bus wires will be configured as aggressors (receive the aggressor value). The configuration is achieved by the victim counter, which counts from victim 0 to victim $n-1$, and the decoder, which activates the select signal q_i of the appropriate multiplexer to enable bus wire b_i to be the victim, and every other wire to be an aggressor. The victim counter is reset by the reset signal when the FSM receives the start signal T_{enable} from the global test controller, and makes a transition to state s_1 to start generating the 1st vector. At the end of every s_1 - s_6 traversal, having generated all the tests for the victim wire, the FSM returns to state s_1 , setting the enable signal to advance the victim counter. After the completion of n 6-vector sequences (upon receiving signal q_{n-1} in state s_6 , the FSM returns to the idle state s_0 , and stays there until T_{enable} is received again.

3.2 Error Detector

Error detection at the destination consists of verifying the correctness of the two-vector transition pairs. Figure 7 illustrates the possible design of an error detector. It utilize an XOR network that compares the incoming vectors to the test vectors generated by a local test generator that is identical to the source test generator. The configuration shown in Figure 7 allows the test generator to be utilized for both test generation and error

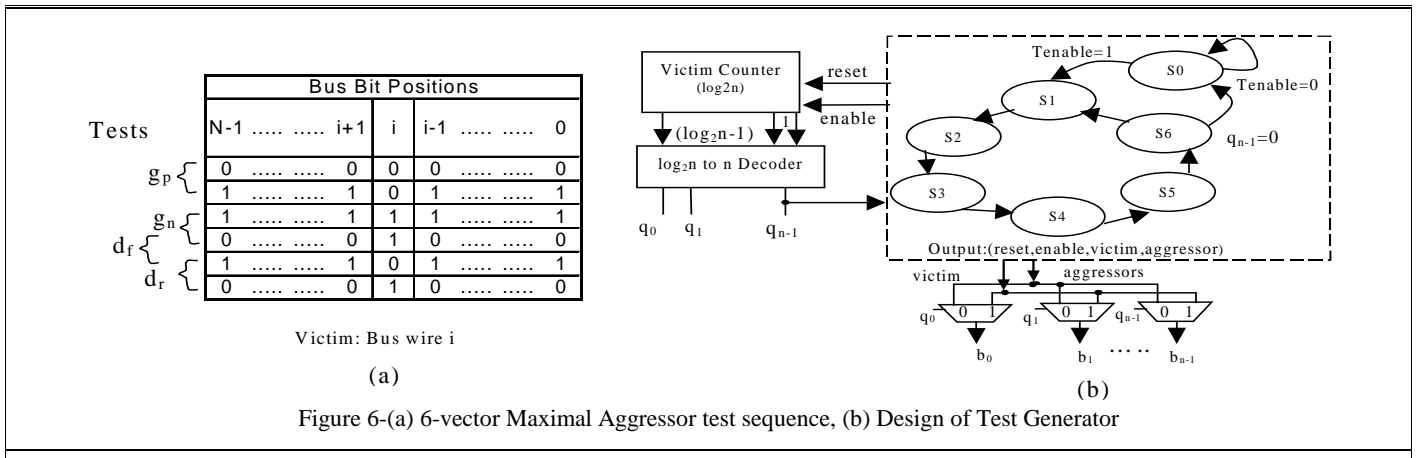


Figure 6-(a) 6-vector Maximal Aggressor test sequence, (b) Design of Test Generator

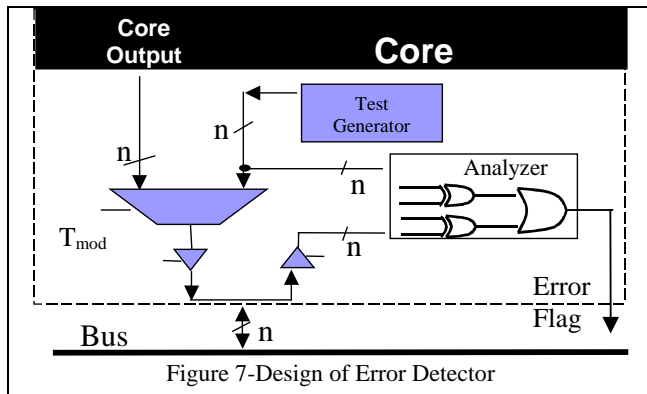


Figure 7-Design of Error Detector

detection. As shown in Figure 7, T_{mode} multiplexes the test generator output with the normal core output for driving test vectors onto the bus.

For cores with bi-directional transactions, we combined the test generator and error detector as one testing component, named TG/ED. The testing component uses an additional bit to receive mode information, which informs it to act as either a test generator or an error detector as needed.

3.3 Implementation of Parameterized Test Generator/Error Detector

We have developed parameterized HDL simulation and synthesis models for the proposed test generator and error detector designs described above. The models are parameterized in terms of the number of interconnects ending with a generator/detector. Hence, they can be easily instantiated to fit the different width requirements of the cores/interconnects of a system chip. The models have been synthesized and verified for various bit widths, using Leonardo synthesis tool (Mentor Graphics), and a 0.35 μm technology library. In Table 1, we report the area overhead (in terms of 2-input NAND gates) of the test generators, error detectors, and combined test generator-error detector (TG/ED) synthesized for different bit widths. For example, a 32-bit Test

Width	Test Generator	Error Detector	TG/ED
8-Bit	219	224	250
16-Bit	334	325	394
32-Bit	525	547	647

Table 1- Overhead of Test Generators, Error Detectors

Generator needs 525 2-input NAND gates. Compared to the size of typical cores, like the ARM7TDMI processor core with 59K transistors, and the NEC V850 micro-controller core with 72K transistors, the test generator and error detector overheads are very nominal.

3.4 Global Test Controller

For the BIST hardware described above, system level testing requires global synchronization of generators and error detectors and selection of cores involved in each test. The design of the global test controller depends extensively on the architecture of the system and hence, optimal implementation will vary from system to system. The processor bus in Figure 2 for instance, may have distributed arbitration where selection of cores is accomplished through address decode logic contained in each core. In this case, test control that is also distributed and shares the same decode logic may be optimal. In contrast, bus arbitration may be centralized into one arbiter that controls cores with select lines. Here, a test controller that is combined with the arbiter logic may be optimal. In the following part, we describe our design of a generic global test controller that disables normal bus arbitration and conducts the tests independently. Figure 8 illustrates the controller.

The controller conducts a sequence of core to core test transactions by enabling a set of generators and error detectors via select lines and monitoring for errors through flags set by the error detectors. The controller is centered around a look up table (LUT) that contains information on which enable lines are activated and how many test vectors are run for each test transaction. It also contains information for bus splitting. The LUT is a result of selecting and scheduling test transactions. To start the test, the normal bus arbitration logic is disabled and the state machine in the test controller is turned on. During the test, the state machine advances a transaction counter that sequences through each test transaction by pointing to a row in the LUT that contains information on the current test transaction. The appropriate selecting signals are driven by the LUT and then a vector counter begins to count the number of test vectors clocked. The vector counter is compared to the vector count also contained in the LUT, and when the correct number of vectors have been run, the controller advances to the next test transaction and repeats until all test transactions are completed. During testing, if an error is detected, the current transaction and vector count are logged into a buffer for post testing diagnosis. From the transaction number and the vector number, the exact line on which the error occurred can be determined.

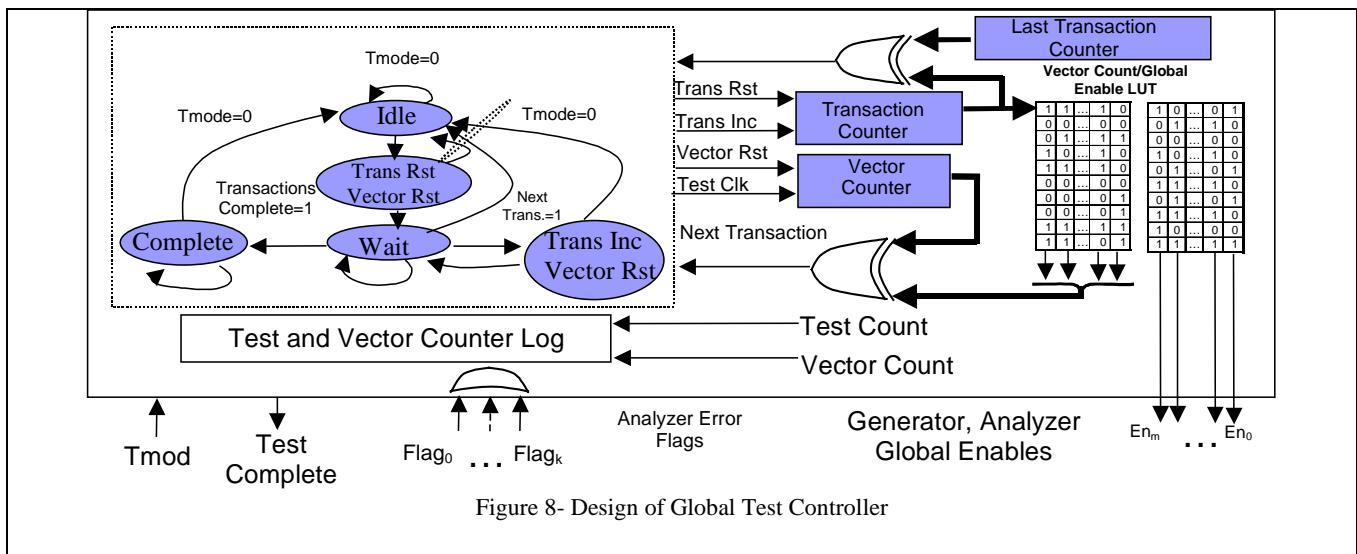


Figure 8- Design of Global Test Controller

4. Application of the Self-Test Methodology to a DSP chip

To validate the self-test methodology, including the design of test generator, error detector and test controller, we have applied the methodology to a Digital Signal Processor chip, CMUDSP [10], which corresponds to Motorola DSP56002. We first briefly describe the architecture and structure of the CMUDSP chip. Next, we discuss how we apply our self-test methodology for testing crosstalk in the buses of the chip, including insertion of the appropriate test generators, error detectors, and test controller, and the overhead involved. We also discuss how validation of the self-test methodology is performed.

As shown in Figure 9, CMUDSP consists of four units: Arithmetic and Data Logic Unit (ALU), Address Generation Unit (AGU), Bus Switch, and Program Control Unit (PCU). The ALU contains the X, Y, A and B registers along with the multiply-accumulate and adder units. The AGU generates the addresses for accessing the data memories. The PCU contains program counter and flag bits for controlling the whole DSP core. The PCU also performs program address generation and instruction decoding. The Bus Switch is used to control data flow between buses. CMUDSP consists of three sets of separate data buses (XDB, YDB, PDB) and address buses (XAB, YAB, PAB). The X and Y buses are connected with data memory, and the P buses are connected with program memory.

From the CMUDSP architecture, and the bus transactions that are allowed during normal operation, we first identify the test transactions that need to be supported. Based on the test transactions needed, we inserted self-test components in each core selectively. For testing output buses to a core, a test generator is inserted, and for testing input buses, a test detector is used. For bi-directional buses to a core, we inserted a combined test generator/error detector structure, which shares some common logic to minimize hardware overhead (see Table 1 in Section 3.3). A test controller is inserted to control the test transactions as shown in Figure 9.

Besides having to test single bus transactions, if a set of multiple buses are involved in normal operation, then that set of multiple buses need to be tested simultaneously to activate the worst-case crosstalk effect. For example, the Bus Switch unit has

	ALU	AGU	Bus Switch	PCU	Test Controller
Original Circuit	11204	7820	1041	3946	N/A
With Test Component	12576	8888	2987	5394	826

Table 2 - Hardware overhead (2-input NAND gates)

four separate 24-bit buses - three of them, XDB, YDB and GDB are tested with self-test structures inserted, as shown in Figure 9. If all the three buses need to be tested together (say from the XDB, YDB at ALU and GDB at AGU to Bus Switch unit), the test controller has to be configured so that the test generators in AGU and ALU combine properly to generate test vectors for the multiple buses (as discussed in section 2), and the TG/ED component of the Bus Switch act as an error detector simultaneously. This is done from the configuration of Look Up Table (LUT) of the test controller.

We have synthesized the original DSP chip, as well as the chip with self-test structures inserted, using Mentor Graphics Leonardo synthesis tool [11]. Table 2 shows the synthesis results, in terms of the number of 2-input NAND gates for each original component and after test structures are inserted. The total hardware overhead (measured in 2-input NAND gate) for the self-test methodology is about 22%. As shown by Table 2, the high overhead is due to the relatively small size of some of the components (like PCU and the Bus Switch). We expect that the area overhead will be much smaller when we compare the area of the physical layouts of the original and final circuits, as this will consider the area due to the bus components, which constitute a significant part of the circuit, and for which there is no extra test overhead. Also, application of the self-test methodology on System-on-Chips, consisting of processor and other cores, will also show relatively small test overhead since the overhead of the test generator and error detector components will be nominal compared to the size of processor and other cores, as shown in Section 3.3.

4.1 Validation of the Self-Test Methodology

To validate the self-test methodology, including the proper design and functioning of the test generators, error detectors and test controllers embedded in the DSP chip, we will have to inject and simulate crosstalk defects in the bus structures, and ensure that the test vectors generated by the self-test structures detect all the defects injected. The most accurate way of simulating the chip is spice-level simulation; however, spice-level simulation of the entire chip is not feasible. Hence, we have developed a high-level crosstalk defect simulation method, which allows simulation of the crosstalk effects on bus interconnects, together with the HDL models of the rest of the chip components, including the self-test structures. While we will not get into the details of the high-level crosstalk fault simulation methodology here, we will describe it briefly, and demonstrate its use in validating the self-test methodology applied to the CMUDSP chip.

To enable injection of crosstalk defects in bus interconnects, and simulation of the behavior of the defect effects (glitch or delay) at the high-level, we also inserted behavioral-level interconnect DSM error models corresponding to the bus interconnects that are under test. The resulting high-level crosstalk

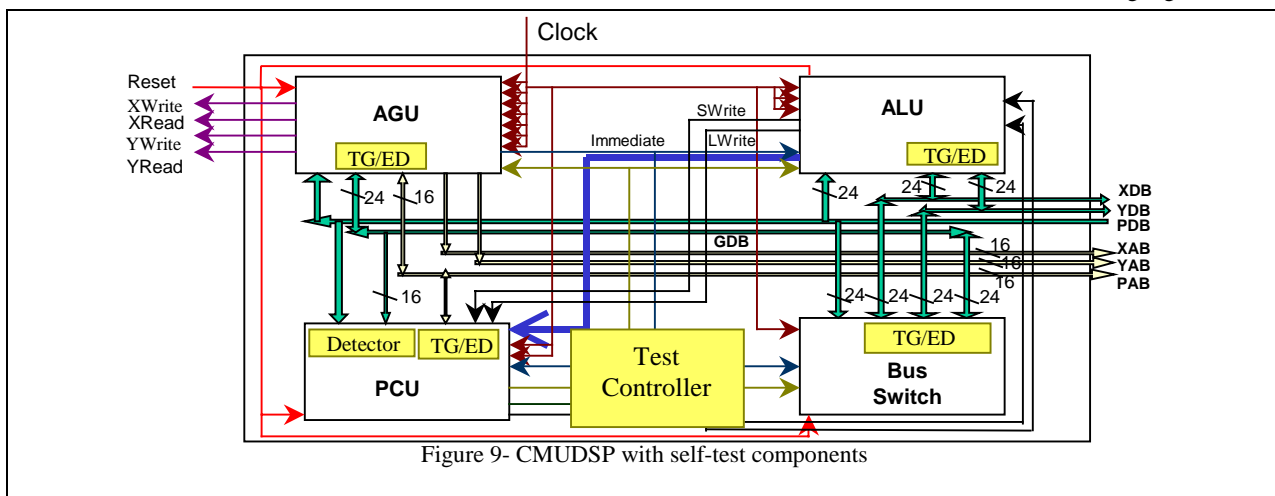
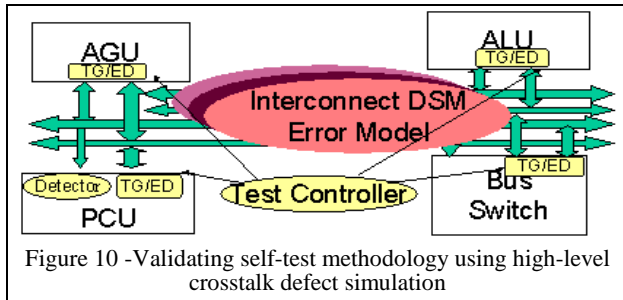


Figure 9- CMUDSP with self-test components

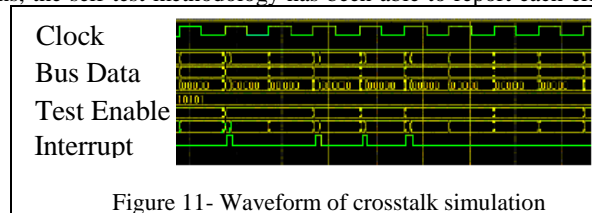


defect simulation environment for the CMUDSP chip is shown in Figure 10. A behavior level interconnect DSM error model generates corresponding errors such as positive glitch, negative glitch, rising delay and falling delay [13] according to the cross-coupling parameters and the test vectors transmitted on a bus. A crosstalk defect can be injected by perturbing the values and distribution of cross-coupling parameters beyond a threshold value [13] to reflect process variations beyond design margins.

During crosstalk defect simulation, the flow of test vectors through the circuit components is as following: An HDL test generator generates the MAF test vectors for a bus, which are input to the corresponding DSM error model. Depending on the test vector transitions, and the coupling parameters of the bus, the DSM error model generates output vectors, which may or may not contain (digitally encoded) glitch or delay errors. Finally, the HDL error detector model at the end of the bus analyzes the output of the DSM error model, and determines whether an error has occurred.

We have used the high-level crosstalk defect simulation environment, and ModelSim HDL simulator[12], to validate the MAFM-based self-test methodology used for the DSP chip. A parameter file, containing randomly perturbed cross-coupling values between the various interconnects, is used during crosstalk simulation. The coupling values lead to randomly generated crosstalk defects, which are expected to give rise to errors on randomly selected victim lines. Simulation starts with the activation of the test controller, which schedules the required test transactions and initiates the corresponding test generators and error detectors by issuing appropriate Test Enable signals. Depending on the use of the test vectors and the parameter file, the interconnect error models may generate crosstalk errors, which are then captured by the error detector and communicated back to the test controller, leading to the activation of the Interrupt signal from the controller.

Figure 11 shows a snap-shot of the crosstalk simulation of the CMUDSP chip. The signals shown include the system clock, the bus data (test vectors transmitted on buses), the test enable signals, and the interrupt signal from the controller. By analyzing the simulation waveform, and checking with the expected errors (which can be determined a-priori from the parameter file), we can determine the correctness of the self-test methodology, including the self-test structures. We have performed crosstalk simulation with several parameter files corresponding to several sets of randomly perturbed coupling values. For all the simulation runs, the self-test methodology has been able to report each error



correctly, indicating the correctness of the designs and functioning of test generators, error detectors, and the test controller.

5. Conclusion

We have investigated and developed a self-test methodology for at-speed testing of crosstalk errors in global interconnects and buses in System-on-Chips. The self-test methodology involves the insertion of self-test structures, such as test generators, error detectors, and a test controller in the SoC. The self-test structures generate tests according to the MAFM [13], and hence guarantees that any crosstalk error, due to coupling capacitances between interconnects, are detected. The parameterized test structures (test generator, error detector, test controller) have been simulated and synthesized. We have also applied our self-test method to a DSP chip, and validated the proper design and functioning of the self-test structures using a novel high-level crosstalk simulation framework.

In the future, we will apply this technique to other circuits, including SoCs, and perform physical design to assess the true cost of the technique. Use of the proposed self-test structures to generate crosstalk test for aggressors may be costly for some applications. We would like to investigate alternative techniques to enable the interconnect crosstalk testing including the generation of test patterns using on-chip cores such as processors.

6. References

- [1] H. Zhou, and D. F. Wang, "Global Routing with Crosstalk Constraints", *Proceedings 1998 design and Automation conference 35th DAC*, page 374-377, 1998.
- [2] Z. Chen, and I. Koren, "Crosstalk Minimization in Three-Layer HVH Channel Routing", *Proceeding IEEE International Symposium on Defect and Fault Tolerance in VLSI System*, pages 38-42, 1997.
- [3] P. Nordholz, D. Treytnar, J. Otterstedt, H. Grabinski, D. Niggemeyer, and T.W. Williams, "Signal Integrity Problems in Deep Submicron arising from Interconnects between Cores", *Proceedings IEEE VLSI Test Symposium*, pages 28-33, 1998.
- [4] K. Rahmat J. Neves, J. Lee, "Methods for calculating coupling noise in early design: a comparative analysis". *Proceeding International Conference on Computer Design VLSI in Computers and Processors*, pages 76-81, 1998.
- [5] H. Kawaguchi and T. Sakurai, "Delay and Noise Formulas for Capacitively Coupled Distributed RC Lines", *Proceedings of the Asian and South Pacific Design Automation Conference*, Pages 35-43, 1998.
- [6] Semiconductor Industry Association (SIA), International Technology Roadmap for Semiconductor, 1999 Edition.
- [7] K. T. Lee, C. Nordquist, and J. Abraham, Automatic Test "Pattern generation for Crosstalk Glitches in Digital Circuits", *Proceedings IEEE VLSI Test Symposium*, pages 34-39, 1998.
- [8] N. Itazaki, Y. Matsumoto, and K. Kinoshita, An Algorithmic "Test Generation Method for crosstalk Faults in Synchronous Sequential Circuits", *Proceedings Sixth Asian Test Symposium*, pages 22-27, Nov. 1997.
- [9] W. Chen, S. K. Gupta, and M. A. Breuer, "Test Generation in VLSI Circuits for Crosstalk Noise", *Proceedings IEEE International Test Conference*, pages 641-650, 1998.
- [10] C. Inacio, The CMU DSP Team, "The Carnegie Mellon Synthesizable Digital Signal Processor Core", 1999.
- [11] Leonardo Spectrum, V1999.1d, Exemplar Logic Inc., Fremont, CA.
- [12] ModelSim EE Vsim 5.3a Simulator, Model Technology Inc. Portland, OR.
- [13] M. Cuvillo, S. Dey, X. Bai, Y. Zhao, "Fault Modeling and Simulation for Crosstalk in System-On-Chip Interconnects", *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 297-303, 1999.