Built-In Self-Test for System-on-Chip: A Case Study

Charles Stroud, John Sunwoo, Srinivas Garimella, and Jonathan Harris

Dept. of Electrical and Computer Engineering
Auburn University, Alabama USA

ABSTRACT - We describe the development of Built-In Self-Test (BIST) for a generic SoC consisting of a Field Programmable Gate Array (FPGA) core for application specific logic along with a processor and several memory cores. Our target device was the Atmel AT94K series System-on-Chip (SoC), also known as a Field Programmable System Level Integrated Circuit (FPSLIC). The original goal for this project was to develop BIST configurations to completely test the programmable logic and routing resources of the FPGA core and then to use the FPGA core to test the other cores. We found that the FPGA can provide only limited testing of the some memory cores and even less testing of the processor. The processor, on the other hand, provides more effective testing of some memory cores than the FPGA core. In addition, the ability of the processor to write the FPGA configuration memory provides an improved and more efficient method of testing the FPGA core. As a result, the processor core was the primary testing resource instead of the FPGA.[1]

1. INTRODUCTION

A new approach to Built-In Self-Test (BIST) for System-on-Chip (SoC) devices that contain one or more Field Programmable Gate Array (FPGA) cores was proposed in [1]. The basic idea is to use BIST approaches developed for FPGAs to first completely test and diagnose the FPGA core found in many generic SoC architectures. These FPGA BIST approaches reprogram the FPGA logic and routing resources with BIST circuitry to allow the FPGA to test itself while off-line without the need for external test equipment or dedicated circuitry for BIST [1]. Once tested and diagnosed, the fault-free portion of the FPGA core can then be used to test and diagnose other cores in the SoC. As proposed in [1], the FPGA core provides the primary resource for testing the SoC, complete with the advantages associated with BIST for FPGAs, reducing the need for expensive external test equipment and the need for dedicated BIST or other design-for-testability (DFT) circuitry. While this appears to be a potentially good approach to SoC testing, it has not been applied to a production SoC in practice. As a result, we set out to apply this idea to a generic, commercial SoC.

The Atmel AT94K series Field Programmable System Level Integrated Circuit (FPSLIC) is a generic SoC architecture that consists of three main types of core functions: an 8-bit processor core, an FPGA core, and different types of Random Access Memory (RAM) cores [2]. Due to the large FPGA core, this SoC architecture appears to be a good candidate for the application of the BIST approach proposed in [1]. During the course of our development, we found that a number of practical architectural issues prevented the direct application of the SoC testing approach as originally proposed. For example, the limited interfaces between the FPGA core and the other cores significantly restrict the ability of the FPGA core to test the other cores. However, the architecture provides a number of features and capabilities that offer unique opportunities for SoC testing in a different manner than that proposed in [1]. For example, the ability of the processor core to write the FPGA configuration memory allows the application of BIST for the FPGA core without the need for time-consuming downloads of the BIST configurations needed to test the FPGA core.

In this paper, we present our investigation of the application of the BIST approach for SoCs proposed in [1] to the Atmel AT94K series SoC. We describe obstacles that prevent direct application of the approach as originally proposed as well as architectural features and capabilities that lead to new methods for testing SoCs containing FPGA cores. We begin in Section 2 with an overview of the BIST approach proposed in [1] followed by an overview of the AT94K series SoC architecture in Section 3. In Section 4 we describe the architectural limitations that prevent the complete application of the BIST approach proposed in [1] and describe those portions of the SoC that can be tested with that BIST approach. We then describe the approaches we used to test the FPGA core, the various RAM cores, and the processor core in Sections 5, 6, and 7 respectively. In Section 8, we show the improvements obtained when testing the FPGA core via the processor compared to the more traditional approaches to BIST for FPGAs and we conclude the paper in Section 9.

2. PREVIOUSLY PROPOSED BIST FOR SOCS

The basic idea of the SoC BIST proposed in [1] is to test the FPGA core first and then use the FPGA core to test other cores. The underlying idea in BIST for FPGAs is to configure some of the programmable logic blocks

[1] This work was sponsored by the National Security Agency under contract H98230-04-C-1177 and by the Dept. of the Army, SMDC, under grant W9113M-04-1-0002.

(PLBs) as Test Pattern Generators (TPGs) and as Output Response Analyzers (ORAs). These are then used to detect faults in PLBs under test (BUTs) in logic BIST or wires under test (WUTs) in routing BIST.

Typically, in logic BIST, the BUTs and ORAs are arranged in alternating columns (or rows) and multiple identical TPGs are used to drive the alternating columns (or rows) of BUTs [3]. The output responses of the identically programmed BUTs are compared by ORAs in neighboring columns (or rows). During a given test session, the BUTs are repeatedly reconfigured in their various modes of operation until they are completely tested. During the next test session, this architecture is flipped and the roles of the PLBs are then reversed such that those previously configured as TPGs and ORAs become BUTs and vice versa. The PLBs can be tested in only two test sessions when at least half the PLBs are configured as BUTs during a given test session.

Two types of routing BIST approaches have proven to be effective in testing the programmable interconnect resources in FPGAs. The first is a comparison-based approach in which the TPG drives exhaustive test patterns over two sets of WUTs that are compared at the other end by a comparison-based ORA [4]. The second approach is parity-based where the TPG sources exhaustive test patterns over a set of WUTs and produces a parity-bit that is sent to the ORA [5]. The ORA generates parity over the data observed on the WUTs and compares the generated parity-bit with that sent by the TPG. While both approaches have been shown to be effective in detecting faults, the comparison-based approach was extended to diagnosis of faults in the programmable interconnect network [4].

Once the FPGA core is tested, it can be configured to test other cores [1]. A common core could be a RAM where the FPGA is programmed with the TPG and ORA functions to perform RAM BIST. The TPG produces a March test while the ORA compares the output responses of the RAM with expected output responses produced by the TPG. An important underlying assumption in the approach proposed in [1] is that the FPGA has complete access to the RAM inputs/outputs. Similarly, the FPGA could be used to test other cores that would normally be tested by external means. While it was assumed that these cores may or may not have embedded design-for-testability circuitry such as scan-chains, it was assumed that the design-for-testability circuitry and the inputs/outputs of the core to be tested were accessible by the FPGA core [1]. Once the other cores are tested, the FPGA can be reconfigured with the desired system function such that there is no BIST area or performance penalty other than the memory needed to store the BIST configurations needed to test the FPGA and other cores.

3. Overview of AT94K Series SoC

The AT94K series SoC architecture (illustrated in Figure 1) consists of three major components: 1) an FPGA core, 2) three types of SRAM cores, and 3) an 8-bit Advanced Virtual RISC (AVR) processor core [2]. The processor core includes a variety of peripheral units such as 16-bit timer/counters and Universal Asynchronous Receiver-Transmitters (UARTs). The thee types of SRAM cores include: 1) small 128-bit RAMs dispersed throughout the FPGA core, 2) a 20-Kbyte to 32-Kbyte processor program memory, and 3) a 4-Kbyte to 16-Kbyte dual-port data RAM shared between the FPGA and processor cores with one port interfacing to the FPGA and the other to the processor.

The total RAM capacity between the program and data memories is 36-Kbytes [2]. The variation in the sizes of each RAM results from a 12-Kbyte RAM partition that can be swapped between program and data memories in 4-Kbyte pieces. While the FPGA can read all of the address locations in both program and data memories, it can only write to the data memory portion of the complete 36-Kbyte RAM space. The FPGA interface to the data and program memories consists of a 16-bit address bus, an 8-bit write data bus, an 8-bit read data bus, and a write enable. The processor core, on the other hand, can write and read the entire RAM space with the exception of a small portion of the data RAM that can only be accessed by FPGA. The processor interface to the data and program memories consists of a 16-bit address bus, an 8-bit bi-directional data bus, and control lines for read and write enables.

The small RAM cores (called *free RAMs* in Atmel terminology) are distributed evenly through the FPGA core with one RAM for every 4×4 array of PLBs [2]. Each free RAM is a 32×4-bit memory and can operate as a synchronous or asynchronous single-port or dual-port RAM. It should be noted that the dual-port RAM mode simply has separate write and read ports and is not a true dual-port RAM. The single-port RAM mode has a single address bus and a bi-directional data bus.

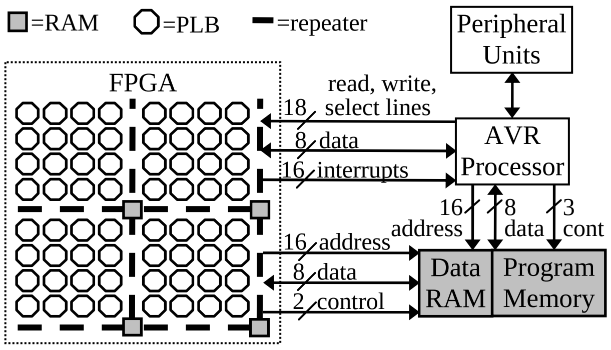

Figure 1. AT94K series Soc architecture

The interface between the processor and FPGA cores (excluding the shared data RAM) consists of sixteen decoded select lines from the processor to the FPGA, an 8-bit bi-directional data bus, a read enable and a write enable from the processor to the FPGA, and sixteen interrupt lines from the FPGA to the processor [2]. In addition, the processor core can write the FPGA core configuration memory such that the FPGA can be dynamically reconfigured (fully or partially) by the processor core during normal system operation. This configuration memory access is facilitated through a 24-bit address bus and an 8-bit data bus. The address bus is partitioned into three 8-bit components (called FPGAX, FPGAY, and FPGAZ) that specify the address of the target configuration memory byte of the FPGA to be reconfigured. The FPGA is PLB addressable where the X and Y address values correspond to the horizontal and vertical location, respectively, of the PLB to be reconfigured. The Z address corresponds to specific logic and routing resources within the specified PLB.

The FPGA core consists of an Atmel AT40K series FPGA comprised of an $N \times N$ array of PLBs [2]. As illustrated in Figure 2, each PLB consists of two 3-input look-up tables (LUTs), a set/reset D flip-flop, a number of multiplexers, and an AND gate to provide a variety of logic functions. This PLB is small compared to most other FPGAs, only about one-fourth the size of the ORCA 2C or Xilinx Virtex/Spartan II PLBs. This small PLB size presents some interesting problems in BIST implementations, as will be discussed in Section 5.

The X-outputs and Y-outputs of each PLB connect diagonally and orthogonally via dedicated local routing resources to its neighboring PLBs, as illustrated in Figure 3a [2]. In addition, there are five vertical and five horizontal global busing planes associated with each PLB as shown in Figure 3b. Each busing plane consists of two outside wire segments (referred to as express wires) and a middle wire segment. The four inputs to the PLB as well as the output from the PLB can connect to the middle wire segment of any of the global busing planes associated with the PLB. The X and Y inputs to the PLB (shown in Figure 2) can be selected via non-decoded multiplexers from any of their respective four direct local connections or from any of the five vertical or five horizontal global connections. The W and Z inputs to the PLB can only be selected from any of the five vertical or five horizontal global connections. Programmable interconnect points (PIPs) of the cross-point type are located at all intersections of the vertical and horizontal global busses. For every 4×4 array of PLBs, buffered bus repeaters in the five global busing planes prevent signal degradation in lengthy and/or heavily loaded nets. The repeaters also provide all possible combinations of connections between express and middle wire segments connected to the repeater as illustrated in Figure 3c.

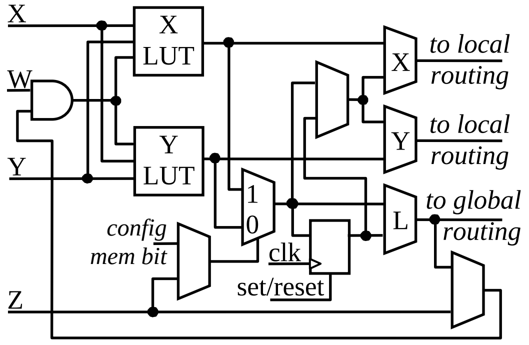

Figure 2. PLB architecture

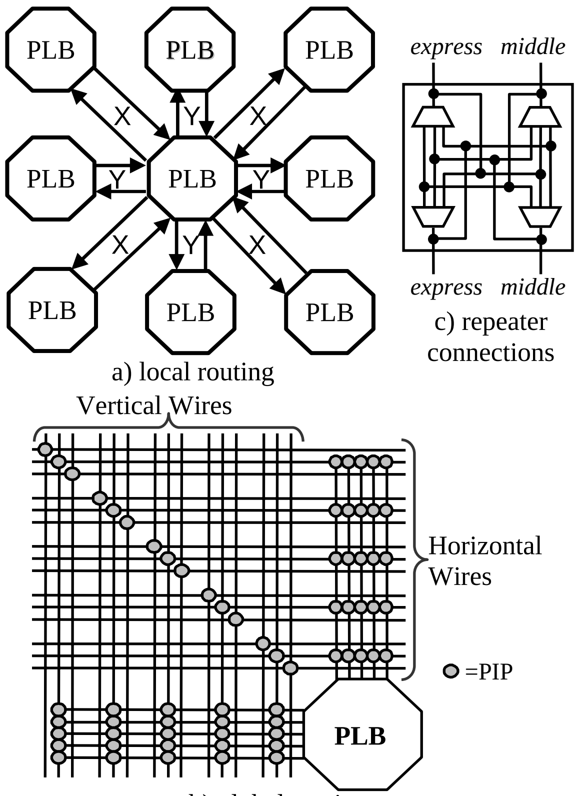

Figure 3. Routing resources

4. Applying SoC BIST to AT94K Series SoC

In our attempt to apply the BIST ideas proposed in [1] to the Atmel AT94K series SoC, we found that some of the ideas could be applied but many could not. The cores that can be tested as proposed include the FPGA core, the *free RAM* cores, but only a single port of the shared data RAM. The cores that cannot be tested as proposed in [1] include the dual-port of the data RAM, the program memory, and the processor core. The inability to test these cores from the FPGA is primarily due to the limited interfaces and interconnect access to/from the FPGA core. The processor core, on the other hand, has better access to most other cores than the FPGA core. Most significant is the processor core write access to the FPGA configuration memory which, as will be shown, allows the processor core to provide the primary test access and control resource for BIST of the SoC. The FPGA core becomes a secondary, although very important, testing resource since it can be easily reconfigured and controlled by the processor.

5. FPGA CORE BIST

The BIST for the programmable logic and routing resources in the FPGA core is similar to the BIST described in [1]. Due to the small size of the PLBs, a number of modifications were required to both logic and routing BIST as will be discussed. In logic BIST, a 5-bit binary up-counter is used for each TPG while comparison-based ORAs are used for their fault detection effectiveness and good diagnostic resolution [3]. The test patterns are routed from the TPGs to the BUTs via global interconnected resources while the BUT-to-ORA connections are made using local routing resources. However, the local routing architecture of the FPGA and the PLB architecture allow only a single X output and a single Y output from the adjacent BUTs to be observed by the ORA. Therefore, an alternating routing scheme was devised which allows for complete testing of the PLBs. The two routing schemes shown in Figure 4 are alternated during successive test configurations in order to provide observability of both X and Y outputs from each PLB. The BIST architecture is column-based due to bank clocks and set/resets in the array of PLBs.

Our first set of logic BIST configurations for the FPGA core were developed using Atmel's Macro Generation Language (MGL) [6]. The MGL provides a method of creating design-specific implementations in the FPGA through the use of dynamic macros. The MGL code can be utilized to create functions in which logical specification and physical layout of dynamic macros and their respective interconnect can be specified in a programming-like environment. However, the dynamic macros provide the user with limited control of the configuration bits associated with the PLBs [6]. As a result, five BIST configurations were required in each test session in order to obtain 98.8% single stuck-at gate level fault coverage of the PLBs.

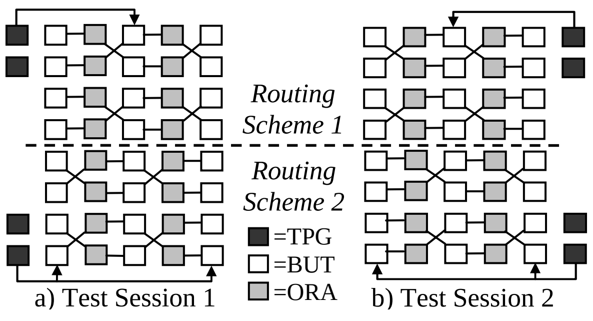

Figure 4. Logic BIST architecture

We determined through fault simulations that only three BIST configurations are needed to obtain 100% fault coverage ***only if*** complete control of the configuration bits for the BUTs is available ***and*** all outputs of the BUTs are simultaneously observable (including X and Y outputs as well as the L output to global routing). However, with only complete control of the configuration bits while observing only the X and Y outputs of the BUTs, four BIST configurations of the BUTs are required to obtain a fault coverage of 99.7%. With these four configurations, only one potentially detected fault is left and this fault is detected during routing BIST such that 100% fault coverage is obtained with the complete set of BIST configurations. The individual (FC_{IND}) and cumulative (FC_{CUM}) fault coverage obtained with these four BIST configurations are summarized in Table 1 along with individual and cumulative fault coverage obtained for the five BIST configurations obtained with MGL and the theoretical best-case three BIST configurations assuming complete control of configurations bits and observability of all PLB outputs.

Table 1. Logic BIST configurations fault coverage

BUT Config	MGL [6]		X-Y Outputs		Theoretical	
	FC_{IND}	FC_{CUM}	FC_{IND}	FC_{CUM}	FC_{IND}	FC_{CUM}
1	59.6%	59.6%	59.6%	59.6%	62.3%	62.3%
2	53.6%	85.5%	50.6%	89.8%	59.6%	95.8%
3	49.1%	91.3%	59.5%	97.9%	57.5%	100%
4	43.4%	95.5%	35.5%	99.7%		
5	31.3%	97.9%				

Complete control of the configuration bits for the BUTs is available through the processor core access to the FPGA configuration memory. As a result, the dynamic partial reconfiguration of the FPGA core by the processor not only reduces the number of logic BIST configurations but also provides higher fault coverage. Furthermore, it was observed that for the BUTs located on the edge of the array, the fault coverage was much lower (only about 83%) than that of the middle BUTs since both X and Y outputs in the BUTs along the edge of the array are not observable in all BIST configurations. By utilizing the dynamic partial reconfiguration of the FPGA by the processor, the BUT-to-ORA connections at the edges of the array can be changed during each BUT configuration to obtain the same fault coverage as in the middle of the array. This results in minimal increase in testing time since only the BUT-to-ORA connections along the edges of the array need to be reconfigured and the FPGA core can be partially reconfigured without affecting the flip-flop values present in the PLBs. An alternative technique is to rotate the floor plan of the two test sessions illustrated in Figure 4 by 90° such that the BIST architecture is now row-based and to apply all four test sessions. While this doubles the total number of logic BIST configurations, it not only overcomes the lower fault coverage along the edge of the array but also provides nearly complete testing of all local routing resources during logic BIST. In addition, the four test sessions provide improved multiple fault detection capabilities and diagnostic resolution [3].

Dynamic partial reconfiguration overcomes another problem associated with the small PLB size. The PLB does not have sufficient logic resources to implement a comparison-based ORA with a shift register for retrieving the BIST results at the end of the BIST sequence. This can be seen in Figure 5 where there is a conflict for the Y input to the PLB by the BUT output and the shift register input. In addition, the contents of the PLB flip-flop cannot be read by the processor core. Therefore, we use partial reconfiguration via the processor core to reconfigure the ORAs into a shift register chain in order to retrieve the BIST results and diagnose the location of the faulty PLB(s) at the end of each BIST configuration.

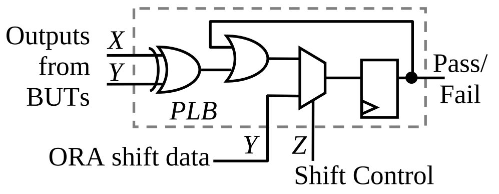

Figure 5. Logic BIST ORA

The BIST for the programmable interconnect requires more configurations than that for the logic BIST due to the fact that most of the area in FPGAs is consumed by the routing resources and their configuration memory bits [7]. A modified version of the parity-based routing BIST approach in [5] was found to be more efficient in terms of the total number of BIST configurations than the comparison-based BIST approach in [4] due to the limited ORA implementations that can be obtained with the small PLB. In this modified architecture, a 2-bit binary counter is used in conjunction with even or odd parity generators to provide test patterns to the set of WUTs with the generated parity bit used as an additional test pattern along with the 2-bit count value. The ORA consists of the corresponding parity check circuit as illustrated in Figure 6. The TPG, ORA and WUTs are confined to as small an area as possible for the particular routing resources under test. This area is called a self-test area (STAR) [4]. The FPGA is populated with multiple STARs that execute their BIST sequences concurrently. Diagnostic resolution obtained from a failing ORA indication is to that STAR [4].

The local routing resources associated with each PLB consist of the X and Y direct connections (Figure 3a) and four PLB input multiplexers with up to fourteen connections: five from the horizontal global buses, five from the vertical global busses, and four from the direct connections. These local routing resources are nearly completely tested during the four logic BIST sessions. The exception is the direct diagonal X connections which require four additional routing BIST configurations.

Within the global routing resources there are horizontal and vertical repeaters placed throughout the array for every four PLBs. Each repeater consists of four 3-input non-decoded multiplexers as shown in Figure 3c. Eight BIST configurations are used to completely test the vertical repeaters along with another eight BIST configurations for the horizontal repeaters. Two examples of these repeater tests are illustrated in Figure 7 where alternating up-count/even-parity and down-count/odd-parity TPGs drive alternating sets of WUTs in order to detect bridging faults and stuck-on faults in some of the PIPs in the repeater while other PIPs are being tested for stuck-off faults. As the eight BIST configurations are applied, all multiplexer PIPs in the repeater cell are tested for stuck-on and stuck-off faults.

Eight configurations are used to test the cross-point PIPs interconnecting a given set of global express busses for both stuck-off and stuck-on faults. A simplified example of the cross-point PIP BIST configurations is illustrated in Figure 8. During subsequent BIST con-

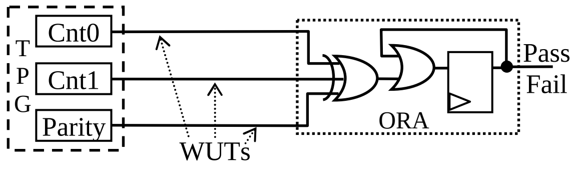

Figure 6. Routing BIST architecture

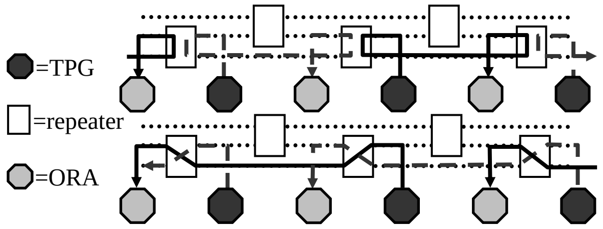

Figure 7. Routing BIST for repeaters

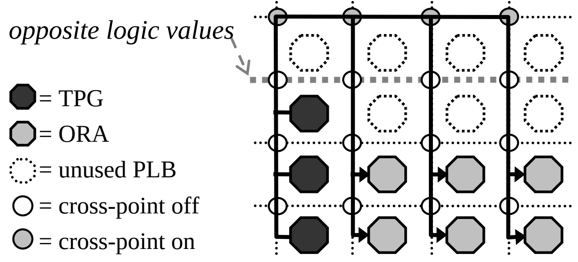

Figure 8. Routing BIST for cross-point PIPs

figurations, the *on* cross-point PIPs that are being tested for stuck-off faults would shift down through the rows until all cross-point PIPs are tested. Opposite logic values are driven from the neighboring STAR on one bus per BIST configuration in order to test the PIPs for stuck-on faults. Since there are two sets of express busses, a total of sixteen BIST configurations are used.

A total of 36 routing BIST configurations are required to test all of the programmable routing resources, not including those local routing resources tested by the 16 logic BIST configurations. These are summarized in Table 2. The STAR size is given in terms of the size of the PLB array needed to implement the BIST circuitry and, as a result, the diagnostic resolution associated with a failing routing BIST ORA indication. For better diagnostic resolution, additional diagnostic BIST configurations can be developed and applied in order to locate a faulty wire segment or PIP [4]. As in the case of logic BIST, dynamic partial reconfiguration from the processor core is used to reconfigure the ORAs into a shift register for BIST results retrieval. Note that the routing resources associated with the free RAMs in the array are tested as a part of the RAM BIST since these routing resources are dedicated for access to/from the RAMs.

Table 2. Routing BIST Configurations

Routing Resource	# Configs	STAR size
direct X connections	4	4×4
vertical repeater cells	8	1×16
horizontal repeater cells	8	16×1
express bus cross-points	16	8×8
Total Routing BIST	36	

6. RAM CORE BIST

The *free RAMs* dispersed over the entire array of the FPGA core and located in every 4×4 array of PLBs have all inputs/outputs accessible by the PLBs and routing resources of the FPGA core. Therefore, these RAMs can be tested by the FPGA core with PLBs configured to function as TPGs and ORAs as proposed in [1]. Except for those RAMs in the rightmost column of the array, all free RAMs can operate as either a single-port or dual-port RAM in synchronous or asynchronous modes. The RAMs along the rightmost column can only operate in single-port RAM mode. A total of three RAM BIST configurations are required to completely test these RAMs and all of the RAMs are tested in parallel.

The RAMs are tested in their synchronous dual-port mode using a test algorithm similar to the dual-port RAM test described in [8]. The March-LR algorithm [9] is used to test the RAMs in synchronous single-port mode and the March Y algorithm as described in [10] is used to test the asynchronous single-port mode. Background data sequences (BDS) are used to detect neighborhood pattern sensitive and intra-word coupling faults [11]. The three RAM BIST configurations are summarized in Table 2 along with individual and cumulative stuck-at fault coverage obtained via fault simulation. The number of TPG and ORA PLBs used in each BIST configuration is also included in the table.

Table 3. RAM BIST Configurations

RAM/ Mode	Test algorithm	FC_{IND}	FC_{CUM}	TPG PLBs	ORA PLBs
Sync Dual-Port	DPR test	75.4%	75.4%	66	N×(N-2)×8
Sync Single-Port	March-LR w/BDS	81.8%	99.8%	123	N×(N-1)×8
Async Single-Port	March-Y w/o BDS	75.6%	100%	18	N×(N-1)×8
Data RAM	March-LR w/BDS	-	-	209	16
N = number of free RAMs in one dimension of array					

When testing the dual-port mode, the RAM BIST architecture employed (illustrated in Figure 10a) is similar to the one used for logic BIST. A 2-PLB ORA, similar to the one in Figure 5, compares bit outputs of two adjacent RAMs and latches any mismatches. When testing the single-port modes, the TPG also generates the expected results and the ORA compares the expected results with the output response of a single RAM (Figure 10b). The active-low output enable (OEN) for the RAM causes the ORA, illustrated in Figure 11, to compare the expected results with the data from the RAM when the output enable is asserted. When the output enable is not asserted, the data from the TPG is written into the RAM with the ORA comparing the test data

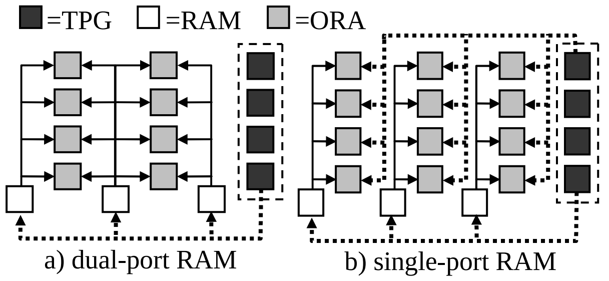

Figure 10. RAM BIST architectures

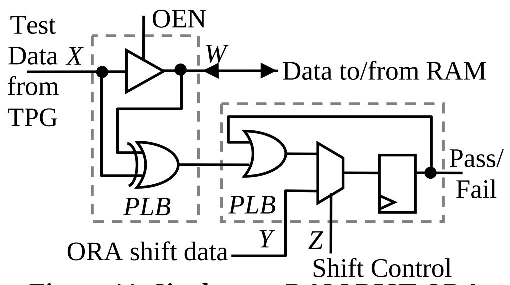

Figure 11. Single-port RAM BIST ORA

from the TPG with the data being written into the RAM. Both types of ORAs (dual-port and single-port) are connected to form a shift register in order to retrieve the BIST results after each test sequence. Since the PLB array utilization is only about 50% (compared to over 98% in logic BIST) the shift register for retrieving BIST results is incorporated in the ORA such that there is no need for the processor core to reconfigure the ORAs into a shift register at the end of the BIST sequence.

An alternative to implementing the TPG in PLBs of the FPGA core is to implement the TPG functionality as a program executed by the processor core. This is particularly important when using the processor core to partially reconfigure the FPGA core for the various RAM BIST sequences. The RAM and ORA portion of the three free RAM BIST architectures given in Table 3 and illustrated in Figure 10 are very similar and also very regular, much like the logic BIST architecture. However, the TPG implementations are not similar or regular, as indicated by the difference in the number of PLBs used to implement the RAM test algorithms in Table 3. For more efficient partial reconfiguration of the RAM BIST sequences by the processor core, it is important to maintain regular structures for algorithmic generation of the configuration by the processor core. This can be accomplished by implementing the TPG functions as programs executed from the processor, leaving only the RAMs and ORAs in the FPGA core.

The data RAM core shared by both processor and FPGA cores is a true dual-port RAM with each core having access to one port. As a result, the dual-port data RAM cannot be tested by the FPGA core alone; the FPGA core can only perform a single-port RAM test. Therefore, the strategy used to test the dual-port data RAM is as described in [12]. First the single-port tests are applied from the two ports separately to test port-related faults as well as some cell-related faults. The March-LR test algorithm [9] with background data sequences [11] is applied from both sides in single-port mode. When testing from the FPGA core, the TPG can be implemented inside the FPGA core along with comparison-based ORAs to detect mismatches between the output responses from the RAM and the expected results produced by the TPG. As noted in Table 3, this approach requires 209 PLBs for the TPG and 16 PLBs for the ORAs. An alternative when testing from the FPGA core is to implement most of the TPG functionality as a program to be executed by the processor core and applied to the data RAM via the processor-FPGA interface. This reduces the number of PLBs required for the TPG to 30 PLBs.

When testing the other port from the processor core, the TPG (March-LR with background data sequences) and ORA functions are implemented as a program executed in the processor core. If the shared data RAM passes both single-port tests, then the dual-port operation is tested with the March s2pf- and March d2pf- applied from both ports [12][13]. In this case, both the processor and FPGA cores are used interactively to perform the dual-port RAM test under the control of the processor core. Hence, the alternative FPGA port BIST implementation, where most of the TPG is implemented as a program in the processor core, is used in order to allow the processor core to maintain control over the dual-port RAM BIST sequence. In addition, both the processor and FPGA cores are driven by the same clock to ensure simultaneous access of memory locations.

The program memory is accessible only by the processor core and, therefore, cannot be tested by the FPGA core. The program memory is tested by applying March-LR single-port tests from the processor core with the TPG and ORA functions implemented as a program and executed by the processor. The portion of the program memory where the RAM test program is not stored is first tested. The RAM test program is then relocated to facilitate testing the remaining part of the program memory.

7. PROCESSOR CORE BIST

Since the processor core is an important resource for testing the other cores, it is critical that the processor core be tested, and probably should be tested first [17]. It was suggested in [1] that the FPGA core could serve as TPG and ORA functions for other cores besides RAMs. For example, if a core were to incorporate scan chains, the FPGA could be configured to provide the circuitry necessary to perform scan-based BIST on that core. On the other hand, if the core did not contain scan chains or other design-for-testability circuitry, the FPGA core could be used to provide non-intrusive TPG and ORA functions such as pseudorandom test patterns and signature analysis, respectively. This latter approach assumes adequate access of the core inputs/outputs to facilitate sufficient controllability and observability of the core for adequate fault coverage.

In the case of the processor core, there are no scan chains that we know of and there is only limited and insufficient access to the processor inputs/outputs from the FPGA core. Since this is a programmable processor core, we are able to use the software-based self-testing techniques that have been used for other microprocessors, such as those proposed in [14-17]. The approaches described in [14] and [15] seem to be based on detailed structural knowledge of the target processor to facilitate high fault coverage in those processors. We are limited in knowledge to the high level architectural information given in the datasheets and additional information that can be derived from the instruction set and descriptions of the various registers and units within the processor core [2]. Therefore, we choose to use a more traditional functional testing approach to exercise the processor core and its various constituents as in [16] and [17]. As opposed to providing a test circuitry resource, the FPGA core in this case is only used as a routing resource to test the interfaces between the cores.

8. Improvements to FPGA and SoC BIST

In no previous work in BIST for FPGAs, such as [3]-[8], has an embedded processor core been present that can write the FPGA configuration memory to utilize when implementing BIST methods for FPGAs and FPGA cores. The first prior work in utilizing the processor core to test the FPGA core was described in [18]. That approach also used the AT94K series SoC but only tests the two LUTs in the PLBs. The LUTs are tested one PLB at a time, rather than concurrently, and the processor seems to be used for TPG and ORA functions.

When utilizing the processor core for dynamic partial reconfiguration of the FPGA core, only one BIST configuration needs to be downloaded along with a program to be executed by the processor core for the reconfiguration of subsequent BIST configurations. This provides an improvement to total test time when compared to downloading individual BIST configurations. The download time dominates the total testing time since the configuration clock runs at a much lower frequency (1 MHz) compared to the processor clock frequency (25 MHz). In addition, the total memory required to store BIST configurations is reduced. We should also note that we have performed timing analysis on all of our BIST configurations and all are capable of maximum clock frequencies greater than the specified 25 MHz maximum processor clock frequency.

To illustrate the reduction in total test time and BIST configuration storage requirements, we begin with logic BIST. Each of the four BIST configurations associated with each of the four test sessions contains approximately 44 Kbytes of configuration data including the program for reconfiguration of the ORAs into a shift register at the end of the BIST sequence for retrieving BIST results. Therefore, a total of approximately 704 Kbytes of memory is needed to store all sixteen logic BIST configurations. A single configuration for a given logic BIST session with a program to reconfigure the subsequent three BIST configurations requires only 45.5 Kbytes of configuration and program data, giving a factor of 3.9 reduction in memory storage for four test sessions. A single download that reconfigures all 16 logic BIST configurations requires 52.4 Kbytes, giving a factor of 13.5 reduction in memory storage requirements for the complete set of 16 logic BIST configurations. In these cases, the program performs the following steps:

1. Execute the BIST sequence for the current BIST configuration.
2. Reconfigure the ORAs into a shift register at the end of the BIST sequence.
3. Retrieve the BIST results.
4. Reconfigure the shift register back to ORAs for the next BIST configuration.
5. Reconfigure BUTs for the next BIST configuration.
6. Repeat steps 1 through 5 until all of the BIST configurations have been executed.

The test time is determined by the total time to download the BIST configuration and the time for the processor to execute the steps listed above. At the maximum download and processor clock frequencies, it takes 350 milliseconds for a single logic BIST configuration, 1.41 seconds for a test session of four logic BIST configurations, and a total of 5.6 seconds for the complete set of 16 logic BIST configurations. Using the processor core for reconfiguration of the four BUT configurations within a given test session, it takes a total of 379 milliseconds per test session, giving a speed-up of 3.7. The complete set of 16 BIST configurations takes a total 447 milliseconds with processor core control of the BIST sequence and reconfiguration for all test sessions, giving an order of magnitude speed-up of 12.6. If we algorithmically generate the initial BIST configuration from the processor, then only the program for reconfiguration and execution of the 16 logic BIST configurations needs to be downloaded. This requires a download file of only 9.4 Kbytes with a total download and execution time of 162 milliseconds. This yields a speed-up of 34.9 with a factor of 75 reduction in memory storage for the complete set of 16 logic BIST configurations.

Table 4 summaries the number of processor clock cycles required to perform the various functions associated with reconfiguration and execution of logic BIST. The number of non-commented lines of C code and the number of bytes of program memory storage required for the compile program is also given. In logic BIST of the AT94K40 (a 48×48 array) there are 1,152 BUTs and 1,104 ORAs in each BIST configuration. Therefore, BUT reconfiguration requires about 43 cycles per BUT while reconfiguration of the ORAs into a shift register requires about 23 cycles per ORA and reconfiguration back to ORAs after retrieval of the BIST results requires about 34 cycles per ORA.

A similar reduction in test time is obtained with routing BIST and BIST for the free RAMs. The three BIST configurations for the free RAMs were developed with the processor core performing the TPG function with only the ORAs implemented in the FPGA core along with the RAMs and routing for the test patterns produced by the processor. Table 5 summaries the three individual BIST configurations as well as a single configuration that combines all three RAM test algorithms along with partial reconfiguration of the FPGA between RAM BIST sequences. All of these configurations include execution of the BIST sequence and retrieval of BIST results by the processor core along with a communication protocol for transmitting the BIST results to an external processor for application of diagnostic procedures. Table 5 give the number of bytes in the configuration download files which includes FPGA core configuration data as well as the programs that are stored in the program memory and executed by the processor core. The number of processor clock cycles required to execute the programs is given along with the total test time, non-commented lines of C source code, and the number of bytes of program memory storage required for the compiled program. Downloading and executing the RAM BIST sequences individually requires 187 milliseconds while a speed-up of about 2.3 times is obtained when all three BIST sequences are downloaded and executed by the processor core with partial reconfiguration between BIST sequences in the combined configuration file. Notice that the number of processor execution cycles for the combined download file is greater than the sum of the three individual RAM BIST configurations due to the additional reconfiguration of the FPGA core by the processor core. The single combined download file reduces the storage requirements for the RAM BIST configurations by a factor of almost 3.

Table 4. Logic BIST Reconfiguration

Reconfiguration function		Processor execution cycles	NCL	Program memory bytes
Per BIST configuration	Step 2. ORA to shift register	25,570	127	764
	Step 4. shift register to ORA	37,220	102	328
	Step 5. BUTs	49,050	119	436
Per test session	BUTs	53,658	155	691
	ORAs	58,194	255	756
	TPGs + routing	65,814	482	847

Table 5. RAM BIST Configurations

RAM test algorithm	Configuration bytes	Processorexecution cycles	Test time (msec)	NCL	Program memory bytes
Dual-Port	60,651	72,264	63.5	144	664
March-LR	59,661	76,355	62.7	196	1,192
March-Y	58,815	49,400	60.8	138	646
Combined	65,983	398,091	81.9	383	1,860

The BIST configuration for the shared dual-port data RAM can be integrated in the combined RAM BIST configuration for the free RAMs. In this case, the FPGA core is reconfigured by the processor to the data RAM BIST architecture after the free RAMs have been tested. This would include single-port tests of the data RAM from both the FPGA and processor ports as well as the dual-port RAM test controlled by the processor. Finally, the program to generate a March-LR test of the program memory can be incorporated into the same configuration download file such that all RAM cores are tested in one continuous sequence. Each of these single-port RAM test programs are approximately 200 non-commented lines of C code and require about 1,200 bytes of the program memory when compiled.

9. Conclusions

We have presented the results of our investigation of BIST for a commercial generic SoC using the Atmel AT94K with FPGA core, RAM cores, and a processor core. Our original intent was to apply the BIST methods proposed in [1] to this device; specifically to program the FPGA core to test itself and then use the FPGA core to test the other cores in the device. The FPGA core could be configured for BIST of the programmable logic and routing resources. In addition, the FPGA core could be configured for BIST of the small RAM cores dispersed throughout the FPGA. However, the limited interfaces and access between the FPGA core and the remaining cores prevented BIST of those cores using the FPGA core as the primary test resource. Partial BIST of the dual-port data RAM could be performed by the FPGA core in the form of a single-port RAM BIST with background data sequences for pattern sensitivity and coupling faults. A complete dual-port RAM test could not be performed without the use of the processor core. Furthermore, the processor and program memory cores could not be tested by the FPGA core. While the basic ideas proposed in [1] are good, the ability to realize the ideas in practice is a function of the interconnect and accessibility between the FPGA core and the other cores to be tested. An FPGA core with complete access to all I/O of a core to be tested can provide BIST resources for that core. Therefore, it is important to maximize the interconnect between an FPGA core and any other core to be tested by the FPGA in an SoC implementation.

In our development, the processor core turned out to be the primary testing resource instead of the FPGA core. Furthermore, the ability of the processor core to perform dynamic partial reconfiguration of the FPGA core leads to more than an order of magnitude improvement in the total testing time for the FPGA core when compared to downloading individual BIST configurations. Only one BIST configuration must be downloaded into the FPGA along with the program executed by the processor for the partial reconfiguration for subsequent BIST configurations, execution of the BIST sequence and retrieval of the BIST results. Currently we use one download for logic BIST, one for each type of routing BIST test session (repeaters and global cross-point PIPs), and one download for each type of RAM core (free RAMs and data/program memory). As a result, only five BIST configurations need to be downloaded for complete testing of the FPGA core and RAM cores to execute a total of 59 BIST configurations. This greatly reduces the amount of storage needed to hold the BIST configurations for system-level testing.

The most important key to efficient reconfiguration of the FPGA core by the processor core appears to be a regular BIST structure that can be algorithmically generated and reconfigured by the processor core. This includes all BIST structures implemented in the FPGA core. Irregular functions, such as the TPG circuitry for the RAM BIST sequence for example, should be implemented and executed as programs in the processor core whenever possible.

We are currently exploring the possibility of performing all testing of the FPGA core and RAM cores with a single processor program for the generation and reconfiguration of the various BIST configurations. This will result in further reductions in testing time and memory storage required for the system-level application of BIST for the SoC. In addition, we intend to incorporate diagnostic algorithms in the program for on-chip diagnosis to facilitate fault-tolerant operation of the FPGA core and RAM cores in SoC.

ACKNOWLEDGEMENTS

The content of the information in this paper does not necessarily reflect the position or the policy of the federal government, and no official endorsement should be inferred.

REFERENCES

[1] M. Abramovici, C. Stroud, and J. Emmert, “Using Embedded FPGAs for SoC Yield Improvement,” *Proc. ACM/IEEE Design Automation Conf.*, pp. 713-724, 2002

[2] __, “AT94K Series Field Programmable System Level Integrated Circuit,” Datasheet, Atmel Corp., 2001 (available at *www.atmel.com*)

[3] M. Abramovici and C. Stroud, “BIST-Based Test and Diagnosis of FPGA Logic Blocks,” *IEEE Trans. on VLSI Systems*, Vol. 9, No. 1, pp. 159-172, 2001

[4] C. Stroud, J. Nall, M. Lashinsky and M. Abramovici, “BIST-Based Diagnosis of FPGA Interconnect,” *Proc. IEEE Int’l Test Conf.*, pp. 618-627, 2002

[5] X. Sun, J. Xu, B. Chan and P. Trouborst, “Novel Technique for BIST of FPGA Interconnects,” *Proc. IEEE Int’l Test Conf.*, pp. 795-803, 2000

[6] C. Stroud, J. Harris, S. Garimella and J. Sunwoo, “BIST Configurations for Atmel FPGAs Using Macro Generation Language”, *Proc. IEEE North Atlantic Test Workshop*, pp. 83-90, 2004

[7] D. Fernandes and I. Harris, “Application of Built-In Self-Test for Interconnect Testing of FPGAs”, *Proc. IEEE Int’l Test Conf.*, pp. 1248-1257, 2003

[8] C. Stroud, K. Leach, and T. Slaughter, “BIST for Xilinx 4000 and Spartan Series FPGAs: A Case Study”, *Proc. IEEE Int’l Test Conf.*, pp. 1258-1267, 2003

[9] A. van de Goor, G. Gaydadjiev, V.N. Jarmolik and V.G. Mikitjuk, "March LR: A Test for Realistic Linked Faults", *Proc. IEEE VLSI Test Symposium*, pp. 272-280, 1996

[10] C. Stroud, A Designer’s Guide to Built-In Self-Test. Kluwer Academic Publishers, Boston MA, 2002.

[11] A. van de Goor, I. Tlili and S. Hamdioui, ”Converting March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories,” *Proc. IEEE Int. Workshop on Memory Technology, Design and Testing*, pp. 46-52, 1998

[12] S, Hamdioui and A.J. van de Goor, “Efficient Tests for Realistic Faults in Dual-Port SRAMs” *IEEE Trans. on Computers*, Vol. 51, No. 5, pp. 460-473, 2002

[13] A. van de Goor and S. Hamdioui, “Fault Models and Tests for Two Port Memories,” *Proc. IEEE VLSI Test Symp.*, pp. 401-410, 1998

[14] L. Chen and S. Dey, “Software-Based Self-Testing Methodology for Processor Cores,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20, No. 3, pp. 369-380, March 2001

[15] A. Krstic, L. Chen, W-C. Lai, K-T Cheng, and S. Dey, “Embedded Software-Based Self-Test for Programmable Core-Based Designs,” *IEEE Design & Test of Computers*, pp. 18-27, Jul-Aug, 2002

[16] D. Brahme and J. Abraham, “Functional Testing of Microprocessors,” *IEEE Trans. on Computers*, Vol. 33, No. 6, pp. 475-484, June 1984

[17] R. Rajsuman, “Testing a System-on-Chip with Embedded Microprocessor,” *Proc. IEEE Int’l Test Conf.*, pp. 499-508, 2003

[18] S. Pontarelli, G. Cardarilli, A. Malvoni, M. Ottavi, M. Re, and A. Salsano, “System-on-Chip Oriented Fault-Tolerant Sequential Systems Implementation Methodology”, *Proc. IEEE Int’l Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 455-460, 2001