On-Chip BIST-Based Diagnosis of Embedded Programmable Logic Cores in System-on-Chip Devices

Charles Stroud, Srinivas Garimella, and John Sunwoo
Department of Electrical and Computer Engineering
Auburn University
Auburn, Alabama 36849-5201, USA
emails: strouce/garimsm/sunwojo@auburn.edu

Abstract

On-chip Built-In Self-Test (BIST) based diagnosis of the embedded Field Programmable Gate Array (FPGA) core in a generic System-on-Chip (SoC) is presented. In this approach, the embedded processor core in the SoC is used for reconfiguration of the FPGA core for BIST, initiating the BIST sequence, retrieving the BIST results, and for performing diagnosis of faulty programmable logic blocks, memory cores, programmable interconnect resources within the FPGA core based on failing BIST results. These BIST and BIST-based diagnostic procedures have been implemented and verified on a commercial SoC with fault injection emulation. Diagnostic resolution is achieved to the faulty logic or memory block and can be used for on-chip reconfiguration to bypass faulty resources for fault-tolerant applications.[1]

1. Introduction

Built-In Self-Test (BIST) approaches have been developed for Field Programmable Gate Arrays (FPGAs) [1]-[6]. The basic idea is to reprogram the FPGA logic and routing resources with BIST circuitry to allow the FPGA to test itself without the need for expensive, external test equipment or on-chip dedicated circuitry for BIST. Since FPGA cores have been embedded in System-on-Chip (SoC) architectures, these BIST approaches can be applied to embedded FPGA cores to reduce the testing cost associated with SoCs [7]. As proposed in [7], 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. When applied to a generic, commercial SoC, however, it was found that 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 [6]. Yet, some SoC architectures provide features that offer unique opportunities for SoC testing. For example, the ability of the embedded processor core to write the FPGA configuration memory.

The ability for an embedded processor to reconfigure the FPGA core facilitates reconfiguration of the FPGA core for BIST without the need for the time-consuming downloads of BIST configurations needed to test the FPGA core [6]. Furthermore, the embedded processor should also be capable of executing the BIST sequence, retrieving the BIST results, and performing diagnostic procedures based on those BIST results for the identification of fault resources in the FPGA core, including programmable logic and memory blocks as well as programmable interconnect resources. Once identified, the faulty resources could be bypassed for fault-tolerant applications. Collectively, these features provide for on-chip test, diagnosis and repair.

In this paper, we describe the development of on-chip BIST and diagnosis by the embedded processor of the FPGA core in the Atmel AT94K series SoC. We begin with an overview of the AT94K series SoC architecture in Section 2 and an overview of the BIST approach used to test the FPGA core as well as static Random Access Memory (RAM) cores distributed throughout the FPGA core in Section 3. This is followed by a detailed discussion of the diagnostic procedures used to identify faulty FPGA core resources in Section 4. Experimental results from our implementation and verification of the approach in actual SoCs are presented in Section 5 and we conclude in Section 6.

2. Overview of AT94K Series SoC

The Atmel AT94K series SoC architecture, illustrated in Figure 1, consists of three major components: 1) an FPGA core, 2) three types of RAM cores, and 3) an 8-bit Advanced Virtual RISC (AVR) processor core [8]. The processor core includes a variety of peripheral units including 16-bit timer/counters and Universal Asynchronous Receiver-Transmitters (UARTs). The three types of RAM cores include: 1) a large number of 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

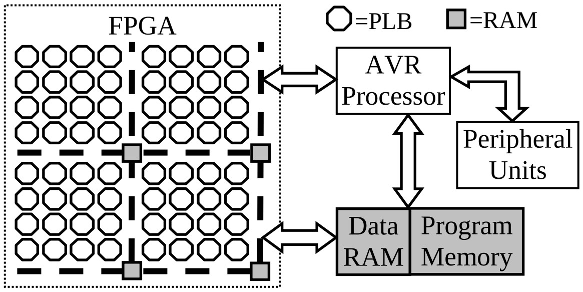


Figure 1. AT94K Series SoC Architecture

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

]]>

the FPGA and processor cores. The small RAM cores (referred to as *free RAMs* in Atmel terminology) are distributed evenly through the FPGA core with one RAM for every 4×4 array of programmable logic blocks (PLBs) [8]. Each RAM is a 32×4-bit memory that can operate as synchronous or asynchronous, single-port or dual-port RAM. The dual-port RAM mode has separate write and read address and data ports while the single-port RAM mode has a single address bus with a bi-directional data bus. These RAMs will be tested and diagnosed along with the rest of the FPGA core.

The remainder of the FPGA core consists of an *N*×*N* array of programmable logic blocks (PLBs), where *N*=48 for the largest AT94K device. Each PLB contains two 3-input look-up tables (LUTs), a set/reset D flip-flop, and additional multiplexers/gates illustrated in Figure 2 [8]. Local programmable routing resources connect the Y and X outputs of each PLB to the inputs of direct and diagonal adjacent PLBs, respectively (Figure 3a). Global routing resources consist of five vertical and five horizontal routing busses associated with each PLB (Figure 3b). The W, X, Y, and Z inputs as well as the L output of the PLB can connect to the global routing resources. In addition to the global routing resources, the X and Y inputs to the PLB (Figure 2) can also connect to any of their respective four direct local routing resources. Programmable Interconnect Points (PIPs) are located at all intersections of the vertical and horizontal global routing resources. For every 4×4 array of PLBs, buffered repeaters in the five global routing busses prevent signal degradation in lengthy or heavily loaded nets. The repeaters also provide connections between the wire segments of each global routing bus.

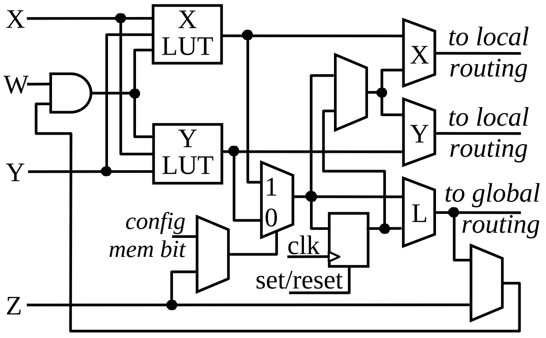
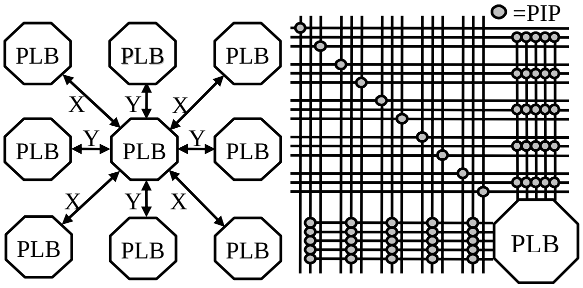


Figure 2. Programmable Logic Block



a) local routing b) global routing

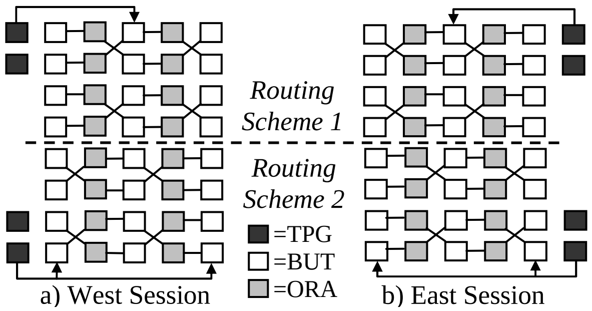
Figure 3. Programmable Routing Resources

The embedded 8-bit AVR processor core can write (but not read) 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 via a 24-bit address bus and 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 FPGAX and FPGAY address values correspond to the horizontal and vertical PLB location to be reconfigured. The FPGAZ address corresponds to specific logic and/or routing resources within the specified PLB.

3. BUILT-IN SELF-TEST

The BIST approach for the embedded FPGA core of the AT94K series SoC is described in [6]. The basic idea is to program some of the PLBs as Test Pattern Generators (TPGs) and Output Response Analyzers (ORAs) to test the remaining programmable logic, RAM, and routing resources. In logic BIST, a 5-bit binary up-counter is used for each TPG while comparison-based ORAs are used for their effective fault detection and good diagnostic resolution [1]. The test patterns are routed from the TPGs to the PLBs under test (BUTs) via global routing resources while the BUT-to-ORA connections are made using local routing resources. The BIST architecture, shown in Figure 4 is column-based due to bank clocking and set/reset in the array of PLBs.

Partial reconfiguration of the FPGA core by the embedded processor is used to reconfigure the BUTs in their various modes of operation. A total of four BIST configurations are required to completely test the BUTs [6]. Since the contents of the PLB flip-flop cannot be read by the processor core, dynamic partial reconfiguration by the processor core is used to transform the ORAs into a shift register to retrieve the BIST results at the end of each set of four BIST configurations, referred test session. The logic BIST architecture is then flipped about the vertical axis to test those PLBs not tested during the first test session. As a result, all PLBs are tested in two test sessions. However, the local routing architecture of the FPGA core and the PLB architecture allow only a single X output and a single Y output from adjacent BUTs to be observed by a



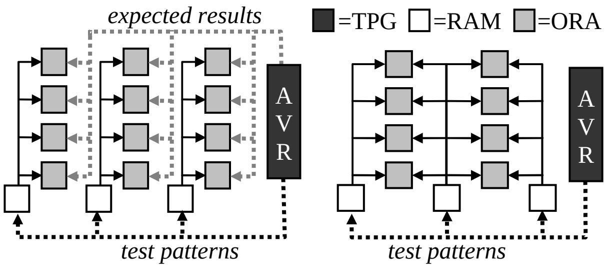
a) West Session b) East Session

Figure 4. Logic BIST Architecture

given ORA. Therefore, an alternating routing scheme was devised which allows complete observability of the outputs of the BUTs [6]. In the original logic BIST implementation described in [6], the two test sessions shown in Figure 4 were rotated by 90°, for a total of 16 BIST configurations, to overcome low fault coverage along the edges of the array. However, when the processor core is used for dynamic partial reconfiguration of the FPGA core for BIST, a more efficient procedure is to execute each of the test sessions shown in Figure 4 twice, once for each of the two routing schemes. In this improved method, the BIST results are retrieved from the ORAs at the end of each test session for a given routing scheme, while in the original approach, BIST results had to be retrieved at the end of each BIST configuration.

The small 128-bit RAM cores dispersed throughout the FPGA core and located in every 4×4 array of PLBs have all inputs and 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 [6]. However, we found that a more efficient method for partial reconfiguration of the FPGA core by the embedded processor is to implement the TPG functionality as a program executed by the processor core. This minimizes the number of unique reconfigurations of PLBs in the FPGA core by the processor core and, as a result, the size and runtime of the program to be executed by the processor core for BIST of the RAMs in the FPGA core.

The RAM BIST architectures used are illustrated in Figure 5. 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 [5]. The March-LR algorithm [9] is used to test the RAMs in their 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 are used with the March-LR algorithm to detect neighborhood pattern sensitive and intra-word coupling faults [11]. For the dual-port RAM, the BIST architecture (Figure 5b) is similar to that of logic BIST where the outputs of neighboring RAMs are compared by a set of ORAs. In the single-port RAM modes, however, the AVR processor core can easily generate the expected results as part of its TPG functionality. These expected results are then compared in the ORA with the outputs of the RAM under test (Figure 5a).



a) single-port RAM b) dual-port RAM

Figure 5. RAM BIST Architectures

4. BIST-BASED DIAGNOSIS

Two diagnostic procedures for RAM BIST were developed as a result of the two different BIST architectures. In the single-port RAM BIST architecture (Figure 5a), the TPG function performed by the processor core also produces the expected read data results along with the test patterns sent to the RAMs. The expected read data results are sent to the ORAs and are compared to the actual read data from each RAM under test with any mismatches encountered latched in the ORA until retrieval of the BIST results at the end of each BIST sequence. The ORAs incorporate a shift register mode of operation to facilitate shifting the BIST results from each ORA through the shift register to the processor core of the SoC. Each ORA corresponds to a single bit of the 4-bit words of the RAMs. The position of the ORA in the PLB array, and the corresponding RAM with which it is associated, is determined by the ORA's position in the shift register. As a result of the ORA comparison of the RAM under test outputs with the expected read results produced by the TPG, the diagnostic procedure for the single-port RAM modes of operation is straight forward. The diagnostic procedure looks for ORA failure indications (logic 1s) and translates the positions based on the shift register order to identify not only which RAMs are faulty but also which bits in a given RAM are faulty. A faulty ORA can mimic a fault in its corresponding RAM, but since the PLBs used to construct the ORAs are tested and diagnosed during BIST of the PLBs, the faulty ORA can be identified.

The diagnostic procedure for the dual-port RAM mode of operation is more complicated since the outputs of two RAMs are compared to detect mismatches that result from faults in one or both of the RAMs. It is possible that equivalent faults in two RAMs being compared by the same ORA will go undetected. However, RAMs in the middle of the FPGA are being observed by two sets of ORAs and being compared to a different RAM in each set of ORAs such that the few combinations of faulty RAMs that can go undetected by the BIST approach is highly improbable; for example, all RAMs in a row of the FPGA would have to have equivalent faults to go undetected. The diagnostic procedure is based on the Multiple Faulty Cell Locator (MULTICELLO) algorithm originally developed for diagnosing faulty PLBs in FPGAs [1]. The procedure assumes a column (or row) based BIST architecture where columns (or rows) of BUTs (or RAMs in this case) are observed by ORAs in adjacent columns. The procedure also assumes there are at most two consecutive BUTs (or RAMs) with equivalent faults. The steps of the diagnostic procedure, as applied to RAMs with an example of a 7×7 array of dual-port RAMs, are as follows:

Step 1. *Record the ORA results and initialize the faulty/fault-free status of all RAMs under test as unknown, indicated by an empty entry in the table.* This is illustrated in the Step 1 example below where a 7×7 array of RAM cores is used. The columns of RAMs are denoted as R_1 to R_7 with the ORAs denoted as O_{ij} where i and j are the RAM columns to the left and right of the ORA, respectively. A ‘1’ in an ORA column entry indicates that a failure was observed by at least one of the four ORAs associated with each output of the 4-bit word RAM for that row. A ‘0’ indicates that no failure was observed by the ORA.

Step 1 Example

row	**R_1**	**O_{12}**	**R_2**	**O_{23}**	**R_3**	**O_{34}**	**R_4**	**O_{45}**	**R_5**	**O_{56}**	**R_6**	**O_{67}**	**R_7**
1		**0**		**0**		**0**		**1**		**1**		**0**	
2		**0**		**0**		**1**		**1**		**0**		**0**	
3		**1**		**1**		**0**		**0**		**1**		**1**	
4		**0**		**0**		**0**		**0**		**0**		**0**	
5		**0**		**0**		**0**		**1**		**1**		**1**	
6		**1**		**0**		**0**		**0**		**0**		**0**	
7		**0**		**0**		**1**		**0**		**0**		**0**	

Step 2. *In each row, for every two consecutive ORAs with 0s, enter a 0 for the RAM under test between them to indicate that the RAM is fault-free.* This is illustrated in the Step 2 example below where new entries are marked in bold while entries from the previous step are shown in non-bold text. At this point in the example, we have determined that 19 of the 49 RAMs are fault-free.

Step 2 Example

row	**R_1**	**O_{12}**	**R_2**	**O_{23}**	**R_3**	**O_{34}**	**R_4**	**O_{45}**	**R_5**	**O_{56}**	**R_6**	**O_{67}**	**R_7**
1		0	**0**	0	**0**	0		1		1		0	
2		0	**0**	0		1		1		0	**0**	0	
3		1		1		0	**0**	0		1		1	
4		0	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0	
5		0	**0**	0	**0**	0		1		1		1	
6		1		0	**0**	0	**0**	0	**0**	0	**0**	0	
7		0	**0**	0		1		0	**0**	0	**0**	0	

Step 3. *In each row, for every two adjacent 0s followed by an unknown RAM, enter a 0 in the empty cell to indicate that the RAM is fault-free.* This is illustrated in the Step 3 example below where new entries are shown in bold. At this point in the example, we have determined that 37 of the 49 RAMs are fault-free.

Step 3 Example

row	**R_1**	**O_{12}**	**R_2**	**O_{23}**	**R_3**	**O_{34}**	**R_4**	**O_{45}**	**R_5**	**O_{56}**	**R_6**	**O_{67}**	**R_7**
1	**0**	0	0	0	0	0	**0**	1		1		0	
2	**0**	0	0	0	**0**	1		1	**0**	0	0	0	**0**
3		1		1	**0**	0	0	0	**0**	1		1	
4	**0**	0	0	0	0	0	0	0	0	0	0	0	**0**
5	**0**	0	0	0	0	0	**0**	1		1		1	
6		1	**0**	0	0	0	0	0	0	0	0	0	**0**
7	**0**	0	0	0	**0**	1	**0**	0	0	0	0	0	**0**

Step 4. *In each row, for every adjacent 0 and 1 followed by an unknown RAM, enter a 1 in the empty cell to indicate that the RAM is faulty.* This is illustrated in the Step 4 example where new entries are shown in bold. At this point we have determined that 6 of the 49 RAMs are faulty and 37 are fault-free.

Step 4 Example

row	**R_1**	**O_{12}**	**R_2**	**O_{23}**	**R_3**	**O_{34}**	**R_4**	**O_{45}**	**R_5**	**O_{56}**	**R_6**	**O_{67}**	**R_7**
1	0	0	0	0	0	0	0	1	**1**	1		0	
2	0	0	0	0	0	1	**1**	1	0	0	0	0	0
3		1	**1**	1	0	0	0	0	0	1	**1**	1	
4	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	**1**	1		1	
6	**1**	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0	0	0	0	0

Step 5. *Consistency check: If an ORA indicates a failure but the RAMs on both sides of the ORA are determined to be fault-free, then there is a fault either in the ORA or in the routing resources between one of the RAMs and the ORA.* In the Step 4 example above, there is an ORA inconsistency in row 7 of the array indicating a fault in the ORA (or its associated routing) located between RAM columns 3 and 4.

Step 6. *If all RAMs have been marked as faulty or fault-free then a unique diagnosis has been obtained; otherwise, any RAM that remains marked as unknown may be faulty.* In the Step 4 example above, we see that the remaining 6 of the 49 RAMs are unknown in terms of their faulty/fault-free status. In row 1, the RAMs in columns 6 and 7 could fault-free or they could have equivalent faults. In row 3, the RAMs in columns 1 and 7 could be faulty or fault-free. In row 5, one (or both) of the two RAMs in column 6 or 7 is faulty, we just don’t know which one.

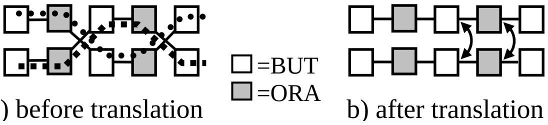
As can be seen in the Step 4 example, the RAMs with unknown status are located near the edges of the array where diagnostic resolution is lower due to the RAMs along the edge being observed by only one ORA. These ambiguities in the diagnosis can be removed by rotating the RAM BIST architecture by 90° where rows of ORAs are comparing rows of RAMs, such that the sets of RAMs being compared are orthogonal, and reapplying the diagnostic procedure to the new BIST results. This improves diagnostic resolution at the cost of doubling the testing time; however, a unique diagnosis can be obtained for almost any combination of faulty RAMs. It should also be noted that the examples shown here assume a single ORA for the complete 4-bit word RAM. In reality, the diagnostic algorithm is applied to the four ORAs associated with the 4-bit word RAMs such that it is more likely that unique diagnosis will be obtained without the need for rotation since any known faulty bit in a RAM would indicate that the RAM is faulty. In addition, diagnostic results from the single-port RAM BIST can be used to remove ambiguities in some cases.

The diagnostic procedure for the PLBs in the FPGA core is complicated by the routing scheme used to observe both X and Y outputs of the BUTs by the ORAs. The MULTICELLO diagnostic procedure, as described in [1], only works on rows or columns of alternating BUTs and ORAs, and as a result, cannot be directly applied to the "zigzag" pattern of the BUT-to-ORA connections across the rows, as shown in Figure 4 and, more specifically, by the dotted and dashed lines in Figure 6a. This zigzag connection is required as a result of the local routing architecture of the FPGA core in conjunction with the PLB architecture allowing only a single X output and a single Y output from adjacent BUTs to be observed by a given ORA. One solution is to translate the positions of the BUTs and ORAs with respect to the BIST results such that the translated BUTs and ORAs lie in the same row (as shown in Figure 6b), apply the MULTICELLO diagnostic procedure, and then translate the BUTs and ORAs back to their true position for identification of the faulty and fault-free PLBs in the FPGA core. This translation is relatively straight forward, particularly when pairs of rows of PLBs are grouped together to form the zigzag connection pattern as illustrated in Figures 4 and 6a. Furthermore, this allows the same basic diagnostic program used in the processor core for the dual-port RAM diagnosis to be used for faulty PLB diagnosis as long as the appropriate BUT and ORA positions are translated before Step 1 and again after Step 5 of the diagnostic procedure given above. As in the case of RAM BIST, potentially faulty PLBs determined as unknown by the diagnostic procedure during logic BIST can be determined to be faulty or fault-free in most cases by rotating the logic BIST architecture illustrated in Figure 4 by 90° and applying the diagnostic procedure to the new BIST results.

5. EXPERIMENTAL RESULTS

The diagnostic algorithms described above have been implemented and verified in compiled C programs that have been downloaded and executed in the embedded processor core of the Atmel AT94K series SoC. The programs reconfigure and execute BIST, and retrieve the results of each BIST sequence. At the conclusion of each BIST sequence, the processor core applies the appropriate diagnostic procedure for that particular BIST architecture (logic or RAM) to the BIST results that have been retrieved and stored in the Data RAM of the SoC.

Upon completion of the diagnostic procedure, the processor core reports the diagnostic results to a higher level controlling processor (a PC in our environment). Results fall into any one of three possible categories: 1) faulty, 2) unknown, or 3) ORA inconsistency. In all three cases, the row and column number of each faulty or suspected faulty (unknown) resource is reported to facilitate reconfiguration around the faulty resources for fault tolerant applications. In the first two cases for RAM BIST architectures, the faulty bit or bits are also reported. If the faulty bits are not being used by the current system application, then the system configuration can be loaded into the FPGA without any reconfiguration to avoid the fault(s). On the other hand, if the bits of the RAM that are being used are determined to be faulty, reconfiguration of the interconnection to and from the RAM can be applied to allow the system function to avoid the faulty bit(s). If the system application uses all bits of a RAM that has been determined to be faulty, then reconfiguration to a fault-free RAM site is required.



a) before translation b) after translation

Figure 6. Logic BIST BUT-to-ORA Connections

The on-chip diagnostic procedure for RAMs requires a total of 1.1 Kbytes of Program Memory independent of the array size as shown in Table 1. Only about one-third of the total code is used to implement the MULTICELLO algorithm, which is also used for diagnosing faulty PLBs during logic BIST. The rest of the code is used for translating the diagnostic results to the row and column coordinates of the faulty resources and for reporting and transferring the diagnostic results to the controlling processor. Table 1 gives the worst-case processor execution clock cycles for running diagnostics and interpreting the obtained results. The amount of BIST results information to be processed by the diagnostics varies with the size of the array and, as a result, the Data RAM memory required also varies, as indicated in Table 1. When implementing diagnostics for logic BIST, the pre- and post-processing performed before and after MULTICELLO, to account for the zigzag BUT-to-ORA connections, requires an additional 200 bytes of Program Memory. The larger array of PLBs (compared to the array of RAMs) to be diagnosed also requires additional Data RAM storage and longer execution time. Therefore, the logic BIST diagnosis determines the maximum Program Memory and Data RAM storage requirements since the same diagnostic program is used for both RAMs and PLBs.

Table 1. Implementation of Diagnostics

Logic Resource	**Array Size**	**Execution Clock Cycles**	**Program Memory (bytes)**	**Data Memory (bytes)**
RAMs	6×6	2,400	1130	20
	12×12	9,700	1130	73
PLBs	24×24	32,000	1330	180
	48×48	110,000	1330	720

The logic BIST described in [6] relied on external reconfiguration of the FPGA core for BIST as well as external control for execution of the BIST sequence and retrieval of the BIST results. The more efficient approach of dynamic partial reconfiguration of the FPGA core by the embedded processor core is illustrated in Table 2 in terms of the number of processor execution clock cycles and Program Memory size required for reconfiguration of BIST, execution of the BIST sequence, and retrieval of the BIST results for diagnosis in the processor core.

As shown in Table 2, using the processor core to reconfigure and control logic BIST produces a 33.7% reduction in the average number execution clock cycles per test configuration and a 47% reduction in Program Memory storage requirements. This is due in part to the fact that ORA results can be retrieved after each group of four test configurations without lost of fault detection information instead of after every test configuration as is the case in the externally controlled logic BIST approach. Another factor is that the externally controlled logic BIST approach in [6] required running four test sessions (west, east, south, and north) for complete testing while the processor core controlled logic BIST approach only requires running two test sessions (west and east), twice each. Thus, the processor core controlled logic BIST requires less reconfiguration clock cycles to completely test the PLBs in the FPGA core.

Table 2. Logic BIST Reconfiguration Comparison

Compared Features	External Control [6]	Processor Control
Number of Test Configurations	16	16
Program Memory Size (bytes)	6,372	3,380
Execution Clock Cycles	1,483,644	998,560
Average Cycles per Test Config.	92,728	62,410

The total resources required for on-chip BIST and diagnosis of the PLB and RAM resources of the embedded FPGA core are summarized in Table 3. The programs for all three functions easily fit into the 32 Kbyte Program Memory of the AT94K10 and AT94K40 while the Data RAM memory requirements use only a small portion of the total 16 Kbytes in available in these SoCs. As a result, the BIST and diagnostic programs can be stored on-chip for on-demand test and diagnosis of the logic and RAM resources in the embedded FPGA core. These programs have been downloaded and verified on actual AT94K10 devices (with a 24×24 array of PLBs and a 6×6 array of RAMs) as well as on AT94K40 devices (with a 48×48 array of PLBs and a 12×12 array of RAMs).

Table 3. BIST and Diagnostics Summary

Testing Function	Execution Clock Cycles	Program Memory (bytes)	Data Memory (bytes)
RAM BIST	398,100	1,860	72
Logic BIST	998,560	3,380	138
Diagnostics	110,000	1,330	720
Total	1,506,660	6,570	930

6. CONCLUSIONS

We have described the next step in the evolution of built-in self-test, self-diagnosis, and self-repair of embedded FPGA cores in SoC implementations. In this case, we have moved on-chip, via the embedded processor core, the reconfiguration of the FPGA core for BIST, the control of the execution of the BIST sequence, the retrieval of the BIST results, and more importantly, the on-chip diagnosis of the FPGA core based on the failing BIST results. Furthermore, the BIST and diagnostic programs can be efficiently implemented in the program and data memories associated with the processor core, as demonstrated by our experimental results. As a result, all testing and diagnosis can be performed on-chip and on-demand, as needed by the mission of the system application. The next step in the evolutionary process is the integration of on-chip algorithms to reconfigure the system function intended for implementation in the embedded FPGA core to avoid the faulty resources identified by the diagnostic procedure for fault tolerant applications and operation.

7. ACKNOWLEDGEMENTS

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

8. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[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] 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.

[6] C. Stroud, J. Sunwoo, S. Garimella, and J. Harris, “Built-In Self-Test for System-on-Chip: A Case Study”, Proc. IEEE Int’l Test Conf., pp. 837-846, 2004.

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

[8] __, “AT94K Series Field Programmable System Level Integrated Circuit,” Datasheet 1138D, Atmel Corp., 2001 (available at www.atmel.com).

[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 Symp., 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’l Workshop on Memory Technology Design and Testing, pp. 46-52, 1998.