Built-In Self-Test of Configurable Cores in SoCs Using Embedded Processor Dynamic Reconfiguration

John Sunwoo
Digital Home Research Division
Electronics and Telecommunications Research Institute
Daejeon, Korea
bistdude@etri.re.kr

Charles Stroud
Electrical and Computer Engineering
Auburn University
Auburn, Alabama, USA
strouce@auburn.edu

Abstract –*Built-In Self-Test (BIST) provides an effective way to test configurable cores in System-on-Chip (SoC) implementations. We present a case study of the use of dynamic reconfiguration from an embedded processor core to implement BIST for the programmable logic and routing resources in configurable cores in commercially available SoCs. Experimental results from actual implementations include speed-up and memory savings obtained and compared to traditional BIST approaches for configurable cores.* [1]

Keywords: Built-In Self-Test (BIST), Field Programmable Gate Array (FPGA), Embedded Microcontroller.

1 Introduction

Testing time and cost for System-on-Chips (SoCs) is high due to the large scale integration ratio in a single chip [1]. Typical commercially available microcontroller-based SoCs consist of peripherals for the microcontroller, associated Random Access Memory (RAM) cores to implement program/data memory space, and Field Programmable Gate Array (FPGA) based configurable cores, along with other cores. Testing configurable cores requires numerous configuration downloads to completely test the various modes of operation of the programmable logic and routing resources. The basic approach to Built-In Self-Test (BIST) of FPGAs and FPGA-based configurable cores is to program some of the programmable logic blocks (PLBs) as Test Pattern Generators (TPGs) and Output Response Analyzers (ORAs) to test remaining programmable logic and routing resources. As a result, this imposes no area or performance penalties on the normal system function [2]. BIST eliminates the need for expensive external test equipment but still requires a large number of configuration downloads. Furthermore, the size of each configuration download file is large due to large amount of programmable resources. As a result, the download time for testing configurable cores in SoCs dominates the total test time of the SoC and, hence, the total test cost [3].

In this paper, we investigate the improvements in the total test time required to completely test embedded configurable cores by the algorithmic generation of BIST configurations using an embedded microcontroller followed by dynamic partial reconfiguration of the FPGA-based core for subsequent BIST configurations. As a result, all external configuration downloads are eliminated since the embedded processor programs the FPGA-based core for BIST, executes the BIST sequence, retrieves the BIST results, and executes diagnostic procedures to locate and identify faults detected by the BIST. The paper begins in Section 2 with an overview of the Atmel AT94K series Field Programmable System Level Integrated Circuit (FPSLIC) [4] used as the target SoC for this case study. However, we emphasize that the technique can be used in any SoC with an embedded processor capable of dynamic partial reconfiguration of embedded configurable cores. We present the BIST architectures developed for the programmable logic and routing resources in Section 3 along with experimental results from actual implementations in various size AT94K series SoCs in terms of the improvements in test time and memory storage requirements compared to traditional external downloads of BIST configurations. The paper concludes in Section 4 with an overview of other applications for this approach.

2 Overview of SoC Architecture

The Atmel AT94K series SoC consists of an FPGA core, various RAM cores, and an 8-bit Advanced Virtual RISC (AVR) microcontroller core [4]. The FPGA core is based on a fine-grain architecture that has a large number of small PLBs [5]. The FPGA core consists of a symmetrical $N \times N$ array of PLBs, where N=48 for the AT94K40 device (the largest AT94K series SoC) [4]. Each PLB contains two 3-input Look-Up Tables (LUTs), a D flip-flop, and additional multiplexers/gates. Every PLB has dedicated diagonal (X) and orthogonal (Y) local routing resources to its neighboring PLBs, as shown in Figure 1a. As shown in Figure 1b, vertical and horizontal global routing resources are associated with each PLB that traverse four PLBs (×4) and eight PLBs (×8). Vertical and horizontal bus repeaters are placed at the boundaries of every 4×4 array of PLBs (as shown in Figure 1c for the horizontal case) to prevent signal degradation in lengthy and/or heavily loaded signal nets. Bank clock and set/reset lines are associated with the vertical repeaters running to groups of four PLBs in each column within a repeater boundary.

[1] This work was sponsored by the Dept. of the Army, SMDC, under grant W9113M-04-1-0002.

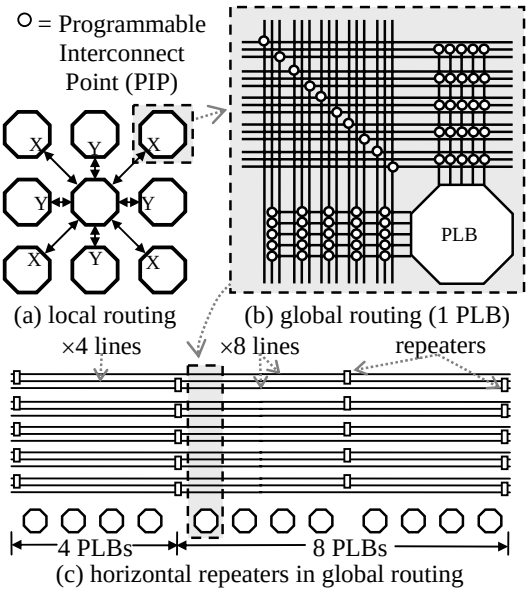

Figure 1. Configurable Core Architecture

The various RAM cores consist of three types of memory resources [4]: 1) many small 32×4-bit RAMs distributed throughout the FPGA core, 2) a 4-Kbyte to 16-Kbyte dual-port data RAM shared by AVR microcontroller and the FPGA core, and 3) a 20-kbyte to 32-Kbyte program memory accessible only by the AVR microcontroller and used for executing AVR programs.

The AVR core is an 8-bit RISC architecture with 32 general purpose registers including a number of peripherals like watchdog timer, UART, etc [4]. There are two 8-bit bi-directional general purpose I/O ports referred to as PORTD and PORTE. An 8-bit bi-directional data bus between the FPGA and AVR provides communications between the two cores. Whenever 8-bit data is written to (or read from) the data bus by the AVR, a strobe signal to the FPGA is generated on FPGAIOWE (or FPGAIORE) along with one of 16 decoded select lines to the FPGA. There are up to four external interrupts to the AVR along with 16 interrupts from the FPGA.

The AVR microcontroller core can write to (but not read from) the FPGA core configuration memory such that the FPGA can be dynamically reconfigured (either fully or partially) by the AVR core during normal system operation [4]. The FPGA 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 referred to as FPGAX, FPGAY, and FPGAZ. FPGAX and FPGAY correspond to horizontal, vertical location of the programmable resource in the array while FPGAZ corresponds to specific logic/routing resources within the specified programmable resource. A write to the 8-bit data bus, FPGAD, results in a write cycle to the FPGA core configuration memory.

Prior work in BIST for the AT94K series SoC resulted in a total of 68 BIST configurations for complete testing of the SoC with the exception of the AVR microcontroller, as summarized in Table 1 [3]. The FPGA core requires the vast majority of the configurations to be downloaded into the SoC and, hence, testing the FPGA core accounts for the majority of the total testing time as a result of those downloads.

Table 1. BIST Configurations for AT94K SoC [3]

Core	# Configs	Resources Tested
RAMs	2	program memory
	3	dual-port data RAM
	3	distributed RAMs in FPGA
FPGA	16	PLBs - also tests local routing
	16	Routing - global cross-point PIPs
	28	Routing - repeaters
Total	68	*not including AVR processor*

3 Processor Reconfiguration of BIST

A significant reduction in testing time is possible by developing programs to algorithmically generate BIST configurations from the AVR along with dynamic partial reconfiguration of subsequent BIST configurations for the FPGA core. BIST for FPGAs is typically partitioned into BIST for the programmable logic resources and BIST for the programmable routing resources [2].

3.1 Logic BIST

The BIST architecture for testing the PLB resources configures a column of PLBs to function as two or more identical TPGs that drive identical test patterns to alternating columns of identically configured blocks under test (BUTs) whose outputs are monitored by comparison-based ORAs located in adjacent columns between the BUTs [6]. Since a PLB cannot be configured to have more than one X-input and one Y-input selected at a time, the BIST architecture as shown in Figure 2a is used wherein each ORA monitors a diagonal X-output and an orthogonal Y-output from their neighboring BUTs [3]. The BUTs are reconfigured in various modes of operation until they are completely tested. The BIST architecture is then flipped about the vertical axis, as illustrated in Figure 2b, to test the PLBs that were previously TPGs and ORAs for the complete test of all PLBs in the array as BUTs.

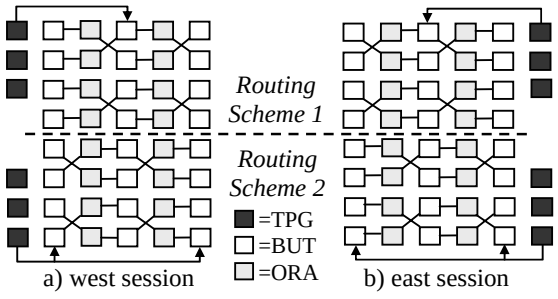

Figure 2. Logic BIST Architecture

The algorithmic BIST configuration generation and reconfiguration program for the embedded AVR core was developed in C. The execution sequence of the subroutines in the program is as follows:

1. *Clear FPGA* – clears FPGA configuration memory by writing default inactive values; also executed when there are transitions between test sessions.

2. *Instantiate ORAs* – places, routes, and configures ORAs with connections to BUTs; the ORA flip-flops are also initialized.

3. *Instantiate/reconfigure BUTs* – places, routes (to global routing resources), and configures BUTs; during reconfiguration for subsequent BIST configurations only the BUT mode of operation and local routing connections are modified.

4. *Instantiate TPGs* – places, routes, and configures two 5-bit counters in the TPG column of the PLB array; global routing from TPGs to BUTs is also performed along with initialization of the TPG flip-flops.

5. *Route BIST clock from AVR interface* – connects the FPGA Write Enable (FPGAIOWE) line from the AVR interface to a global clock input line and routes the clock to TPGs, BUTs, and ORAs.

6. *Generate BIST clock* – generates clock cycles to FPGA core (via FPGAIOWE) to execute the BIST sequence during which TPGs generate test patterns and ORAs latch any mismatches observed in BUTs due to faults.

7. *Reconfigure ORAs as shift register* – ORAs are dynamically reconfigured as a shift register without affecting BIST result contents of the ORA flip-flops.

8. *Route shift register output to AVR* – routes output of last ORA in shift register to data input of the AVR core.

9. *Retrieve ORA results* – shifts contents of ORA shift register into the AVR and stores results in data RAM.

The subroutines are summarized in Table 2 in terms of the number of non-commented lines of C source code, the number of bytes of program memory storage required, and the number of processor execution cycles to execute each subroutine in an AT94K10 (24×24 PLB array) and an AT94K40 (48×48 PLB array). Note that the execution time in terms of the number of processor cycles is a function of the array size with the exception of the generating the BIST clock cycles since the length of the BIST sequence is independent of array size. In addition to the subroutines described, there are additional miscellaneous subroutines used to communicate and transfer BIST and diagnostic results to a higher computing resource (a PC in our case); the processor execution time for these miscellaneous subroutines is not considered in the total time.

Table 2. Logic BIST Configuration Routine Analysis

BIST Subroutine	Memory (bytes)	Lines of C Code	Processor Cycles	
			K10	K40
Clear FPGA	492	150	59,664	215,128
Instantiate BUT	834	300	25,829	100,360
Instantiate ORA	220	70	14,844	60,686
Instantiate TPG	1,486	600	4,652	14,866
Route BIST clock	234	40	1,923	4,911
ORA/shift register	282	80	6,371	24,791
Generate clocks	32	6	456	456
Route shift out	402	45	24,879	97,370
Retrieve results	306	35	19,339	75,859
Miscellaneous	388	2,659	N/A	N/A
Total	4,676	4,000	157,957	594,427

The FPGA core is typically reset during external downloads of BIST configurations and, as a result, the BIST results must be retrieved after each BIST configuration has been executed. Dynamic partial reconfiguration, on the other hand, does not affect the contents of the ORAs and, as a result, the ORA contents can be retrieved after each BIST configuration or after each test session consisting of a set of BIST configurations to attain faster test time. In the latter case, there is some loss in diagnostic resolution but not in fault detection capabilities. Faulty BUTs can still be identified with the loss in diagnostic resolution being the ability to identify the failing mode of operation of the BUT.

3.2 Routing BIST

The routing BIST architecture shown in Figure 3 is a modified parity-based BIST approach [7]. Here a 2-bit binary count value is used in conjunction with a parity bit to supply a 3-bit test pattern to a group of wires under test. The 3-bit test pattern is applied to all five ×4 wires of the bus structure associated with each PLB with the parity bit applied to the middle wire segment and the two count values are applied to both pairs of outer wire segments. The TPG can be count-up (initialized to all 0s) with even parity or count-down (initialized to all 1s) with odd parity. The test pattern sequences produced by these two TPGs produce opposite logic values on any possible pair of bits for at least two cases such that both 0-1 and 1-0 combinations exist. As a result, this set of test patterns is effective in detecting stuck-at faults, bridging faults, and opens in wire segments as well as stuck-on and stuck-off faults in PIPs and multiplexers.

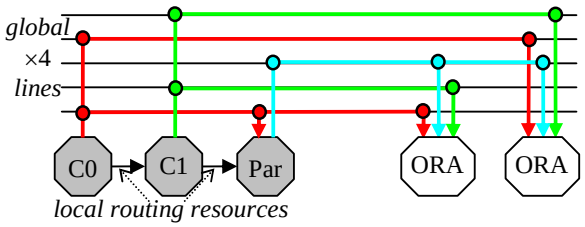

Figure 3. Parity-based Routing BIST Architecture

The subroutines used to construct the routing BIST generation and reconfiguration program are similar to those for logic BIST. Two routing BIST routines are used to test the cross-point PIPs and repeaters as summarized in Table 3 in terms of the number of configurations required for complete testing, the number of bytes of program memory storage required, and the number of processor execution cycles to execute each routine in an AT94K40. Similar to logic BIST, the 48×48 PLB array of the AT94K40 requires about four times the number of processor execution cycles compared to the AT94K10. Note that the original download repeater BIST configurations in Table 1 required only 28 configurations while a total of 40 configurations were needed to completely test the repeaters in Table 3. Efficient algorithmic BIST configuration generation and reconfiguration requires very regular BIST structures and we found that more regularity in the BIST structure (compared to [3]) could be obtained at the expense of additional BIST configurations.

Table 3. Routing BIST Configuration Routine Analysis

BIST Subroutine	# Configs	Memory (bytes)	Lines of C Code	Processor Cycles K40
Cross-points	16	3,442	519	1,906,745
Repeaters	40	4,412	636	6,671,973
Total	56	7,854	1,155	8,578,718

3.3 Experimental Results

The programs for algorithmic BIST configuration generation and reconfiguration for subsequent logic and routing BIST configurations were implemented and verified on both AT94K10 and AT94K40 SoCs. The total test time is given in Table 4 for the AT94K40 device. The total test time is calculated by adding the download time and BIST execution time, including BIST results retrieval time. External download uses a maximum clock frequency of 1MHz in order to check for download errors via a check-sum function [4]. Since the FPGA core can operate at maximum clock frequency of 25 MHz, BIST execution time is calculated assuming that the BIST clock runs at 25 MHz. This data was obtained both from simulation of the programs in AVR Studio and from measuring actual download and execution times in several AT94K40 devices. As can be seen, a speed-up of almost a factor of 45 in total testing time is obtained.

Table 4. Total Test Time and Speed-up

Resource	Function	Download	Processor	Speed-up
Logic BIST	Download	7.680 sec	0.101 sec	76.0
	Execution	0.016 sec	0.085 sec	0.2
	Total time	7.696 sec	0.186 sec	41.4
Routing BIST	Download	20.064 sec	0.110 sec	182.4
	Execution	0.026 sec	0.343 sec	0.075
	Total time	20.090 sec	0.453 sec	44.3
Total Test Time		27.786 sec	0.639 sec	43.5

The combined programs for logic and routing BIST configurations require about 12.6 Kbytes of the total 32 Kbytes available in the program memory of the AT94K series SoC. An additional 2.5 Kbytes of program memory space is required for diagnostic procedures and for communications with higher computing resources to report BIST and diagnostic results. This amounts to almost 50% of the available program memory space and, as a result, may not be desirable for permanent residence in the program memory. Therefore, the total test time data included in Table 4 includes download of the BIST configuration generation program into the program memory. When downloading a program into the AT94K series device, additional configuration data is required for functions such as setting control registers and directing the machine language code to the program and data memories.

The external memory requirements in terms of the number and sizes of files for storing the complete BIST generation and execution programs with diagnostic and communication procedures are summarized in Table 5 and compared to download BIST configuration approach used in [3]. The memory reduction by a factor of 158 for the combined logic and routing BIST is significant in that it makes the use of BIST at the system level more feasible.

Table 5. Total Memory Reduction

Resource Tested	Download		Processor		Memory Reduction Factor
	Average File Size	# Files	File Size	# Files	
Logic	60 Kbyte	16	12 Kbyte	1	80
Routing	57 Kbyte	44	14 Kbyte	1	179
Combined	58 Kbyte	60	22 Kbyte	1	158

4 Conclusions

We have developed a single program to algorithmically generate BIST configurations on-chip via the embedded processor core. This program includes the complete reconfiguration, execution, and retrieval of test results during BIST of the programmable logic and routing resources in the FPGA core of the Atmel AT94K series configurable SoC. We have demonstrated the improvements in the total test time and in BIST configuration memory storage requirements that result based on actual implementation and verification in AT94K10 and AT94K40 devices. The ability to perform dynamic partial reconfiguration of embedded FPGA core from the embedded processor core within the chip boundary provides a major improvement to testing capability with a speed-up in total testing time by a factor of 43.5 and a reduction in external memory storage requirements by a factor of 158. As a result, this program can easily be used for manufacturing testing and/or incorporated into the system for on-demand BIST and diagnosis of the FPGA core for fault-tolerant applications.

Acknowledgement

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] K. Yeom, J. Song, P. Min, and S. Park, “A Reconfigurable Test Access Mechanism for Embedded Core Test,” *Proc. IEEK International SoC Design Conf.*, pp. 396-399, 2004

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

[3] C. Stroud, J. Sunwoo, S. Garimella, and J. Harris, “Built-In Self-Test For System-on-Chip: A Case Study,” *Proc. IEEE International Test Conf.*, pp. 837-846, 2004

[4] __, “AT94K Series Field Programmable System Level Integrated Circuit,” Datasheet, Atmel Corp., 2001

[5] S. Donthi and R. Haggard, “A Survey of dynamically reconfigurable FPGA devices,” *Proc. Southeastern Symp. on System Theory*, pp. 422-426, 2003

[6] 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

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