On Embedded Processor Reconfiguration of Logic BIST for FPGA Cores in SoCs

John Sunwoo, Srinivas Garimella, and Charles Stroud
Dept. of Electrical and Computer Engineering
200 Broun Hall, Auburn University, AL 36849-5201
email: sunwojo/garimsm/strouce@auburn.edu

Abstract: Due to the limited access to the individual embedded cores in System-on-Chips (SoCs), testing is more time consuming and costly than testing stand-alone Field Programmable Gate Arrays (FPGAs). However, the ability for an embedded processor core to reconfigure FPGA cores in SoC applications opens new opportunities for Built-In Self-Test (BIST) of the FPGA cores themselves. This paper discusses a number of implementation issues in BIST for FPGA cores using partial dynamic reconfiguration from an embedded processor including efficient ordering of the reconfiguration process, actual speed-up and memory savings associated with logic BIST, and the resulting affect on diagnosis of the logic resources in the FPGA core.[1]

1. Introduction

A configurable System-on-Chip (SoC) typically consists of a microprocessor core and peripherals, Field Programmable Gate Array (FPGA) cores, program/data memory, and other cores as needed. Since SoCs have a highly integrated structure with limited number Input/Output (I/O) pins, it may not be possible to test all the embedded cores in a SoC using test patterns from external sources. Built-In Self-Test (BIST) could be a better approach for testing SoCs as it does not require any external test equipment and test patterns are generated by the embedded core itself, thus eliminating the problem of core access. By eliminating external test equipment, the BIST approach reduces the testing cost.

BIST approaches have been developed for FPGAs by programming some of the programmable logic blocks (PLBs) as Test Pattern Generators (TPGs) and Output Response Analyzers (ORAs) to test the remaining programmable logic and interconnect resources [1]. However, these techniques typically require downloading a large number of BIST configurations into the FPGA one at a time, executing of each BIST sequence, and retrieving of the BIST results at the end of each BIST sequence. While this problem can be reduced by minimizing the total number of BIST configurations and/or by taking advantage of the partial reconfiguration capabilities provided in recent FPGAs, the total test time and memory storage requirements are still dominated by the download process. For SoC testing, the embedded microprocessor cores in SoCs can be programmed to test other accessible cores such as FPGA cores. Dynamic, partial, and full reconfiguration from embedded processor cores to the FPGA cores between each test phase can reduce the total test time. After completion of BIST, the embedded processor can retrieve the test results, perform diagnosis, and report the faults and their locations to a higher controlling device such as a PC.

We have examined different implications on BIST for FPGA cores in configurable SoCs that support dynamic partial reconfiguration of the FPGA core by an embedded processor. The SoC targeted for this work is the Atmel AT94K series Field Programmable System Level Integrated Circuit (FPSLIC) [2]. The idea of using the embedded microprocessor core as the main BIST component was first proposed in [3], which achieved a 12.6 speed-up in total test time and a factor of 13.5 reduction in memory required for storing BIST configurations for the complete testing of the logic resources in the FPGA [3]. The main idea was to eliminate external downloads for each BIST configuration and have single download file which includes the initial FPGA BIST configuration and algorithmic procedures for the embedded processor to reconfigure the FPGA core for subsequent BIST configurations. Test time was improved further by changing the order of the BIST sequence since the ORA results can be retrieved after multiple BIST configurations with little loss in diagnostic resolution [4]. This resulted in improvements in test time as well as the program memory size requirements.

In this paper, we improve the previous work by eliminating all external downloads to the FPGA and replacing those downloads with a single program. The program contains algorithmic routines to reconfigure the FPGA core for every BIST configuration. This requires only a single download to the program memory, but if the program is sufficiently small, it can reside in the program memory without the need for any download. The key point in this approach is to have an algorithmic routine to reconfigure the FPGA for different BIST configurations [3]. The paper begins with an architectural overview of the Atmel AT94K series SoC as well as the BIST architecture in Section 2. Next, the idea of using the embedded processor core to reconfigure for BIST is presented in Section 3 followed by detailed illustrations of algorithmic processor routines including an efficient reconfiguration ordering scheme in Section 4. The experimental results that show test time speed-up factor

[1] This work was sponsored by the National Security Agency under contract H98230-04-C-1177.

and improved memory storage requirements over the conventional BIST method are presented in Section 5. The details of how this approach is applied to testing RAM cores distributed in the FPGA are described in Section 6 and the paper concludes in Section 7.

2. Overview of SoC and BIST Architectures

The Atmel AT94K series SoC architecture consists of an FPGA core, RAM cores, and an 8-bit Advanced Virtual RISC (AVR) processor core [2]. The FPGA core is based on a fine-grain architecture that has a large number of small PLBs (about the one-fourth size of the Xilinx Virtex/Spartan II series PLB [3]) [5]. It consists of a symmetrical $N\times N$ array of PLBs, where N=48 for the largest AT94K device. Each PLB contains two 3-input look-up tables (LUTs), a D flip-flop, and additional multiplexers/gates [2]. Figure 1 shows X-outputs and Y-outputs of each PLB that connect diagonally and orthogonally via dedicated local routing resources to inputs of its neighboring PLBs, respectively. As illustrated in Figure 2, vertical and horizontal bus repeaters are placed at the boundaries of every 4×4 array of PLBs to prevent signal degradation through the buses. Bank clock and set/reset lines run to groups of four PLBs in a single column within a repeater boundary.

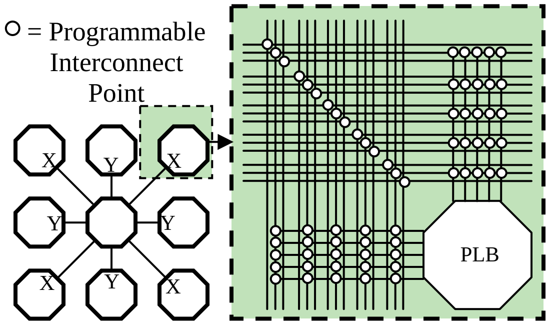

Figure 1. Routing Resources

Two types of RAM resources are present in the AT94K series devices: dedicated embedded 32x4 *free* RAMs in the FPGA and embedded SRAM memory shared by both AVR and the FPGA. Each 4×4 array of PLBs share a *free* RAM which can operate as a single-port or dual-port RAM in synchronous or asynchronous modes. The 36-Kbyte shared SRAM memory can be partitioned into different sizes of data memory and program memory spaces. The program memory is used for executing AVR programs and cannot be accessed by the FPGA core.

The AVR core is an 8-bit RISC architecture that has 32 general purpose registers including a number of peripherals like watchdog timer, UART, etc. [2]. In addition, there are two 8-bit bi-directional general purpose I/O ports called PORTD and PORTE [2]. The AVR and the FPGA cores interact through an 8-bit bi-directional data bus. The dynamic reconfiguration capability of the AVR core enables the embedded processor core to function as a main test resource can provide advantages in testing due to its accessibility to the other cores [3].

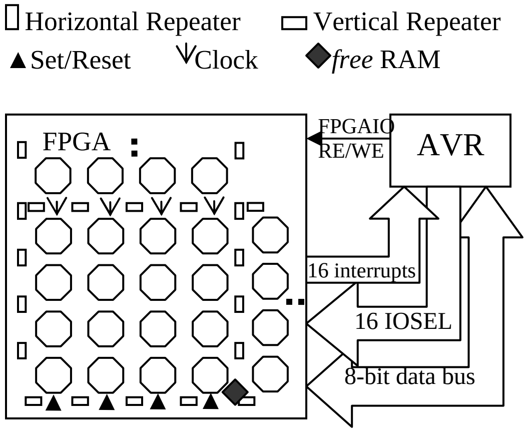

Figure 2. FPGA Core Architecture

The embedded AVR processor core can write into (but not read from) the FPGA core configuration memory such that the FPGA can be dynamically reconfigured (either fully or partially) by the processor core during normal system operation. In later sections, some advantages and disadvantages associated with this feature that can write into any byte of the configuration memory is discussed. As illustrated in Figure 3, 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 (called FPGAX, FPGAY, and FPGAZ) that specify the address of the target configuration memory byte 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. The 8-bit data bus is called FPGAD and any writes into FPGAD cause a configuration clock cycle to the FPGA configuration memory [2]. In addition to this cache logic feature, the AVR

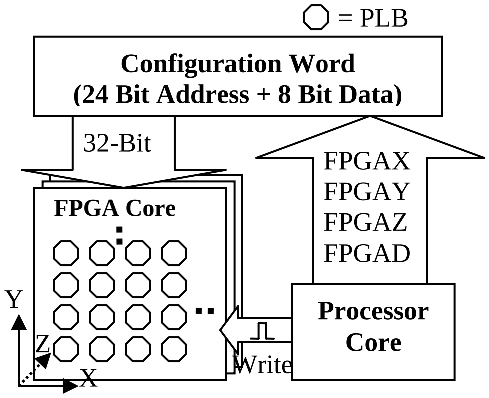

Figure 3. FPGA-AVR Cache Logic Interface

core can directly access the FPGA core through an 8-bit bi-directional data bus and a 16-bit decoded address bus available from AVR and directly connected to the FPGA global routing resources [2]. The FPGA has 16 prioritized interrupt lines that connect to the AVR.

The BIST architecture for testing the PLB resources configures a column of PLBs to function as two or more identical TPGs that drive 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 4 is used wherein each ORA monitors a diagonal X-output and a direct Y-output from their neighboring BUTs. The BUTs are reconfigured in various modes of operation until they are completely tested. The BIST architecture is then flipped about the vertical axis to test the PLBs that were previously TPGs and ORAs for the complete test of all PLBs as BUTs [3].

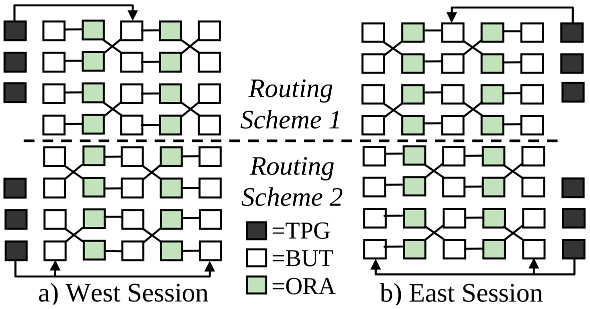

Figure 4. Logic BIST Architecture [3]

3. Processor Core Reconfiguration of BIST

The dynamic partial reconfiguration capability of the embedded processor core was previously used in [3][4][6]. However, all of these approaches needed to download one or more BIST configurations into the FPGA core. The work presented in this paper is an extension of the previous work in [3][4][6] and is primarily targeted at further improving the test time and memory requirements by avoiding any and all downloads into the FPGA. By programming the embedded processor core to execute algorithmic reconfiguration routines, the amount of memory required for storing BIST configurations is reduced since no configuration data is downloaded into the FPGA. The fine-grain architecture in conjunction with the PLB addressable configuration memory of the AT94K series SoCs helps to configure the BIST structures without the need for excessive configuration clock cycles. If small enough, the BIST program can remain resident in the program memory for on-demand reconfiguration and execution of BIST, requiring no download at all. If fast enough, the BIST program can be more frequently used during idle intervals in system operation for high reliability, high availability applications.

To accomplish the first goal, minimizing the size of the program, the BIST architecture must be regular to facilitate an efficient reconfiguration algorithm. In addition, the order of the configuration process must be efficient. The configuration order also impacts the second goal, minimizing test execution time. The test execution time can also be reduced by not retrieving test results from the ORAs after each BIST configuration but instead, using dynamic partial reconfiguration to execute many BIST configurations before retrieving test results. There is some loss of diagnostic resolution in that the faulty functionality within the PLB can no longer be identified. However, there is no loss in diagnostic resolution in that the faulty PLB(s) can still be identified [4].

In [6], the BIST configurations for the FPGA cores in AT94K series SoCs were developed using Atmel's Macro Generation Language (MGL) and were parameterized to handle devices of various sizes [6]. Since BIST configurations generated from the MGL test all logic resources in the BUTs with total of 4 BIST configurations, our first goal was to program the embedded processor to perform the same tests by mimicking the BIST structures which were generated by the MGL, replacing all the BIST configuration downloads with a single AVR program.

One of the limitations of this approach, of having single processor program to test all the resources in an FPGA, is that the time required for developing and debugging the program can be significant. Most of the FPGA design tools provide a graphical representation of the design to be implemented in the FPGA since this can help in debugging the design. Atmel provides a tool called Figaro which graphically represents how the design is mapped onto the FPGA provided the original design is described using MGL, VHDL or Verilog. On the other hand, if the entire BIST configuration is generated through partial reconfiguration by the AVR, debugging the design without any tool support can become quite tedious and error-prone. If the AT94K series SoCs were capable of the dynamic configuration read-back via the AVR processor core (which is not the case), BIST development time would be greatly reduced by facilitating read-modify-write operations to the configuration memory. Instead, the BIST configurations previously developed and verified using MGL as described in [6] must serve as a baseline for developing and debugging the desired program for the AVR processor core.

In order to develop the AVR program, we must determine the BIST configuration that has to be generated initially and also the proper order of subsequent configurations so as to minimize the configuration time from the AVR. We use the BIST configurations originally developed using MGL [6] to help determine these two issues. While the graphical representation of the design helps in planning the reconfiguration routines as to how the different resources (logic, routing, repeaters, and clocks) have to be configured, the MGL generated bit-stream helps in determining the order in which to write various configurations bytes for different resources so as to make the algorithmic reconfiguration routines efficient in terms of speed and size as well as power dissipation during reconfiguration.

After developing and verifying the routines for the initial configuration, routines have to be developed for reconfiguring the BUTs to test the different modes of operation. The BIST reconfiguration order has to be carefully considered and arranged since, if different resources are configured independently, there is possibility of destroying the previously configured bytes as some of the configuration bytes are shared by different resources. For example, if connecting a single programmable interconnect point at the input to a PLB requires writing a logic 1 on the least significant bit location of a specific byte, then writing 00000001 may turn off existing activated programmable interconnect points that are needed for BIST.

4. Processor Core Program Development

The algorithmic reconfiguration program for the embedded AVR core was developed in C. The program's subroutines and reconfiguration sequence is arranged in the following order:

1. *Clear the FPGA* – Instead of the chip reset, this subroutine clears the FPGA configuration memory contents to ensure the BIST components will be configured into an empty FPGA. It clears all configuration memory bytes associated with PLBs, repeaters, clocks/resets, flip-flops, *free* RAMs, and I/O buffers [2]. This routine is also executed when there are transitions between test sessions as shown in Figure 4.

2. *Initialize the ORAs* – This subroutine configures the local routing resources associated with each ORA and its LUTs to function as a comparison-based ORA. It configures the ORAs to either routing scheme 1 or 2 as shown in Figure 4 as well as resets the ORA flip-flop contents to logic 0.

3. *Initialize/reconfigure the BUTs* – This subroutine first configures the cross points where the TPG signals and the BUT inputs are crossed along the very top and the bottom of the FPGA array. When the routine is used to reconfigure the BUTs for the next test phase, depending on the test session and the BIST configuration, it changes the local routing connecting the BUTs as well as the programmable logic resources inside the BUTs. The BUTs are also reset through this subroutine meaning the flip-flops in all of the BUTs are initialized to either logic 0 or logic 1 (depending on the BUT configuration) to ensure correct BIST operation. In fact, this feature provides additional testing of the flip-flops that cannot be tested by downloading individual BIST configurations into the FPGA core and illustrates the improved controllability obtained with partial reconfiguration from the embedded processor core.

4. *Initialize the TPGs* – This subroutine programs two 5-bit counters in the TPG column of the PLB array. It also performs all local, global, and repeater routing between the TPG PLBs as well as the TPG to BUT signal connections as shown in Figure 4. When configuring repeaters in this step, writing to some of the repeater bytes needs extra attention because some of the bytes in repeaters also include clock and set/reset control bits. This subroutine also initializes the TPG flip-flops to logic 0 to ensure that the TPGs are synchronized prior to execution of the BIST sequence.

5. *Route BIST clock controlled by the AVR interface* – This subroutine connects the FPGA Write Enable line (FPGAIOWE) from the AVR interface to one of the global clock input lines of the FPGA core so that the BIST clock signal can be distributed throughout the PLBs. FPGAIOWE is a strobe line which is activated when the AVR writes onto the 8-bit bi-directional data bus and is used to generate and control the BIST clock from the AVR. Since the AVR-FPGA interface cannot be called from the MGL, careful attention was required while developing this routine. Finally, this subroutine configures the clock control settings such as clock invert bits for the TPGs, BUTs, and ORAs which is the last steps before running the BIST.

6. *Run the BIST clock* – In this subroutine, the embedded AVR processor generates the BIST clocks to the FPGA core to run the complete BIST sequence. The TPGs generate the test patterns and any ORAs that see mismatches in the outputs of their two neighboring BUTs will latch up a logic 1.

7. *Reconfigure the ORAs as a scan chain* – At the completion of the BIST sequence, the ORAs will hold the test results to be read by the AVR. During this subroutine, all of the ORAs are dynamically reconfigured as a scan chain without affecting the contents of the ORA flip-flops.

8. *Route the scan out data to the AVR interface* – When ORA results are scanned out to the AVR core, the bi-directional data bus between the AVR and FPGA core must be used to shift the ORA results to AVR for storage in the data memory. This subroutine routes a signal path from the output of the last ORA in the shift register to one of the 8-bit data bus lines to the AVR core shown in Figure 2

9. *Retrieve ORA results and store in the data memory for fault detection and/or diagnosis.* – According to the instruction given to the embedded processor by a higher computing source (a PC in our case), the AVR can retrieve the ORA results after every BIST configuration or after multiple BIST configurations. In the latter case, there is some loss in diagnostic resolution but it does not degrade any fault detection capabilities. Thus, it still detects any faulty PLBs while attaining faster test time. The AVR can either return the actual test results (the contents of the ORAs) or it can perform an on-chip diagnostic procedure [4] as instructed by the higher computing source. In the event that the AVR is instructed to perform diagnosis, it returns a list of all faulty PLBs and their locations in the array for the BIST configuration(s) just executed.

To minimize program size, most of the configuration routines for the first (west) test session are parameterized so that they can be reused for the second (east) test session (see Figure 4). The main difference between the two test sessions is the direction of the TPG signal flow across the top and the bottom of the array which correspond to horizontal repeaters on the top and bottom rows. The rest of the configuration subroutines for the BUTs and ORAs are reused simply by applying offsets to the column locations. TPG configuration routines are also reused by changing the TPG column location from FPGAX = 0 for the first (west) test session to FPGAX = *ArraySize*-1 for the second (east) test session. Thus, most of the configuration subroutines described above take two parameters: directions of the TPG signal flow to the BUTs (west or east) and the BIST configuration for the particular BUT mode of operation to be tested.

To find an efficient configuration sequence when reconfiguring the FPGA core from scratch, a primary goal is to avoid the risk of overwriting a configuration bit that has been previously written and, as a result, inadvertently injecting errors into a BIST configuration. The following considerations help to minimize this risk. First, do not configure more than what is needed when configuring the FPGA for the test. For example, the clock need not be configured until the other BIST components are configured and ready for the BIST clock. When BIST clock is ready to be applied for the BIST sequence, the scan out path from the ORAs is not needed and should only be configured right before the BIST results have to be retrieved. Second, keep track of configuration bytes that have more than one kind of programmable component (such as repeaters and global clocks and resets, for example). Third, configure resources that are regular and repeat over the entire array first (such as the BUTs and ORAs, for example) and then configure the resources that are local to a specific area in the FPGA array (such as the clock, scan out signal, and TPGs).

Atmel’s MGL and Figaro IDS tools help, to a certain extent, in speeding up the development and debugging process for the AVR program which consists of the various configuration subroutines. In order to use an MGL program in debugging, a completely developed and verified MGL-based BIST configuration from [2] was modified to omit certain configurations of the BIST components in the FPGA core as illustrated in Figure 5. An AVR program was then developed to write the configuration of the original components missing in the modified MGL configuration. The MGL generated bit stream and the compiled AVR code are then combined into a single bit stream using Atmel System Designer and downloaded into the SoC. The MGL-based BIST configuration with missing BIST components will report failures on running BIST. However, if the BIST runs correctly after the execution of the AVR configuration routine, then we have verified, at least to a certain extent, that the configuration subroutine correctly replaces the missing BIST component. Each BIST component is removed, one at a time, from the MGL code and combined with appropriate AVR configuration subroutine to verify all of the AVR configuration subroutines for all BIST components. In this manner, we are essentially using the BIST architecture to test itself for design verification. A fault injection emulation technique is then used by reconfiguring certain PLBs to have faults and to verify that the BIST accurately detects and diagnoses these faults [7]. When there is no MGL-generated configuration data to be downloaded into the FPGA core, we are left with one AVR program which consists of all the logic BIST reconfiguration subroutines to be downloaded to the program memory of the SoC.

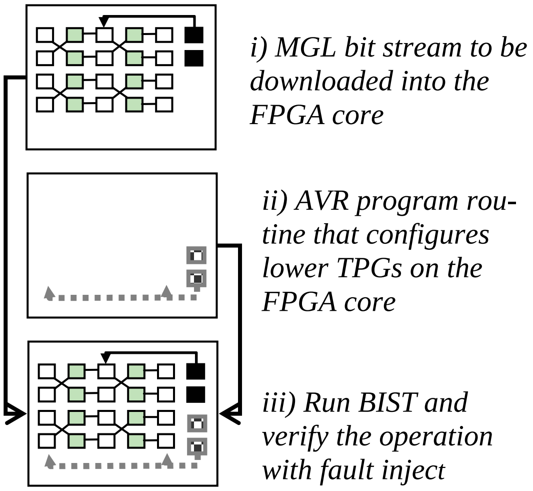
i) MGL bit stream to be downloaded into the FPGA core
ii) AVR program routine that configures lower TPGs on the FPGA core
iii) Run BIST and verify the operation with fault inject

Figure 5. Use of MGL to Verify AVR Routines

5. Experimental Results

The AVR program consisting of the various subroutines described in the previous section are summarized in Table 1 in terms of individual program memory storage requirements, number of lines of source code, and the number of processor execution cycles for each configuration subroutine. Note that Table 1 contains the detailed functional level analysis of the final program which compiles to an Intel HEX file format to be downloaded to the program memory of the chip to run all of the west and east test sessions which are equivalent to BIST configurations studied in [4]. Almost all of the subroutines developed for west test session are reused in east session to reduce the program memory size.

Table 1. Total Configuration Routine Analysis

BIST Reconfiguration Subroutines	Program Memory Size (KBytes)	Number of Lines of Code (Approx.)	Processor Execution Cycles	
			K10	**K40**
Clear FPGA	0.492	150	59664	215128
Place/config BUT	0.834	300	25829	100360
Place/route ORA	0.22	70	14844	60686
Place/route TPG	1.486	600	4652	14866
Route BIST clock	0.234	40	1923	4911
ORA/shift reg	0.282	80	6371	24791
Route scan out	0.402	45	24879	97370
Misc.	0.726	2700	*	*
Total	4.676	4000	138162	518112

** Ignored in the total value.*

Due to the irregular structure of the TPG and associated routing, the subroutine for configuring the TPG PLBs and the TPG to BUT routing occupies a large portion of the program memory. The second biggest subroutine is the placement and reconfiguration of the BUTs since this contains 16 different combinations of BUT test configurations as well as the flip-flop and set/reset tests in half of the BIST configurations. The complete AVR program occupies 4.7 KBytes of program memory which corresponds to only about 14% of the total 32 KByte program memory space of the AT94K series SoC.

In contrast to the program memory size or the number of C source code line, processor execution cycles listed in Table 1 shows a different aspect of the BIST reconfiguration program. For example, more execution cycles are required in the routines for the clearing the FPGA, for placement and reconfiguring of the BUTs, and for placement and routing the ORAs. Fewer execution cycles are required for placing and routing the TPGs. This is because the first three subroutines contain extensive loops which travel along every X and Y locations of the chip. This illustrates how the regular and algorithmic structure of the BIST architecture helps to reduce the program memory storage requirements. The K10 notation in Table 1 denotes AT94K10 devices which have an array size of 24×24 PLBs while the K40 denotes AT94K40 devices which have a 48×48 PLB array. The column showing processor execution cycles for K40 is greater by a factor of approximately four indicating that the increase in reconfiguration time and retrieval of results is linear with the device size.

Subroutines for applying the diagnostic procedure to the BIST results and for communicating with the higher controlling source also increase the program memory storage requirements. Also, due to the additional bits added from the tool that generates the final bit-stream, the actual file size to be downloaded to the program memory of the AVR increases from 4.7 Kbytes to 12.6 Kbytes as summarized in Table 2.

Table 2. Actual Download File Size (KBytes)

All Configs	On-Chip Diagnosis + other	Added by System Designer Bit Generation	Total
4.676	2.5	5.419	12.6

With the internal BIST reconfiguration process executed by the AVR core, we achieve much better external memory storage requirements and faster testing time when compared to downloading individual BIST configurations into the FPGA. This is summarized in Table 3 for external memory storage and in Table 4 for total test time. The data shown in these tables are for a AT94K40 device with a 48×48 PLB array.

Table 3. Total Memory Reduction

Approach	Total Configuration File Size	Memory Reduction
Conventional	65 Kbytes × 16 files	1
Processor Only	12.6 Kbytes × 1 file	83.1

Table 4. Total Test Time and Speed up

	Download	Processor	Speedup Factor
Download (1MHz)	8.371 sec.	0.101 sec.	83.077
Run-time (25MHz)	0.016 sec.	0.085 sec.	0.193
Total BIST Time	8.387 sec.	0.186 sec.	45.125

The total test time is calculated by adding the download time and BIST execution time (or run-time as listed in Table 4). The external download is done using a maximum clock speed of 1MHz since all external downloads which involve a check for download errors (the check-sum function) at the FPGA can run at a maximum configuration clock frequency of 1 MHz [2]. Since the AVR can run at 25 MHz clock speed, BIST execution time is calculated assuming that the BIST clock runs at 25 MHz. This data was obtained from simulation on the Code Vision AVR software program for both conventional and processor-only cases and was also verified against actual download and execution in several AT94K40 devices.

As a result of the single AVR program for BIST reconfiguration, we obtain a factor of 45 speed-up in total test time and a factor of 83 reduction in external memory requirements for storing BIST configurations. It is interesting to note that the run-time in Table 4 increases for AVR BIST reconfiguration. This is due to the fact that the embedded processor core is doing all the reconfiguration, execution, and retrieval of BIST results while in the download of BIST configurations, the processor core is only used to reconfigure the ORAs into shift registers as the end of the BIST sequence for retrieval of the test results. With this consideration, the increase in run-time seems surprisingly small.

6. PROCESSOR-BASED BIST FOR *FREE* RAMS

The same idea of a single program to reconfigure the FPGA for BIST, execute the BIST sequence, and retrieve the BIST results can also be applied to other resources like RAMs. The embedded *free* RAMs in AT94K series SoCs can operate in four modes: single-port synchronous and asynchronous, or dual-port synchronous and asynchronous modes. However, three BIST configurations are sufficient to completely test the *free* RAMs [3]. These three modes of operation and their respective test algorithms are summarize in Table 5. Note that Background Data Sequences (BDS) which detect coupling and pattern sensitive faults need only be used in one of the BIST configurations. The addition of BDS in all configurations not only adds redundancy to fault detection but also increases the total test-time.

Table 5. Test Algorithms *free* RAMs

Mode of Operation	**Test Algorithm**
Single-port synchronous	March LR w/ BDS [8]
Single-port asynchronous	March Y w/o BDS [7]
Dual-port synchronous	DPR test w/o BDS [9]

Unlike logic BIST, wherein the TPG is a counter in all configurations, the TPGs in three RAM BIST configurations are irregular, dissimilar and much more complex taking up a large number of PLBs. In order to simplify the reconfiguration process, the TPG is moved into AVR leaving RAMs, ORAs, and the interconnection between them inside the FPGA core. The TPG data is registered inside the FPGA core as the data bus over which the AVR and the FPGA communicate is only 8-bits wide. Testing the *free* RAMs in all the configurations involves registering the TPG signals from the AVR and then clocking the FPGA core from the AVR. This sequence of registering TPG signals and clocking the FPGA core is repeated until the respective test algorithm (Table 5) is completed. At the end of the BIST sequence, the ORA results are scanned out for faulty/fault-free status determination and/or diagnosis.

In order to ease the reconfiguration process when moving from one RAM BIST configuration to another, some redundancy is added in the logic and routing of the BIST circuitry as illustrated in Figure 6. The design of the ORA used in single-port BIST configurations is shown in Figure 6 (a). The scan chain is integrated in the ORA, unlike logic BIST, due to availability of sufficient logic resources. In single-port mode, there is one port which is used for reading and writing data. As a result, an active-high tri-state buffer is used which is controlled by an active-low read enable (OEN) that also goes to the RAM. The TPG data is written into the RAM during a write operation and the data from the RAM is compared with the TPG data (expected data generated by the TPG) during a read operation. There are two additional data lines as indicated in Figure 6 (a) which are routed but not connected to any PLB and thus are redundant.

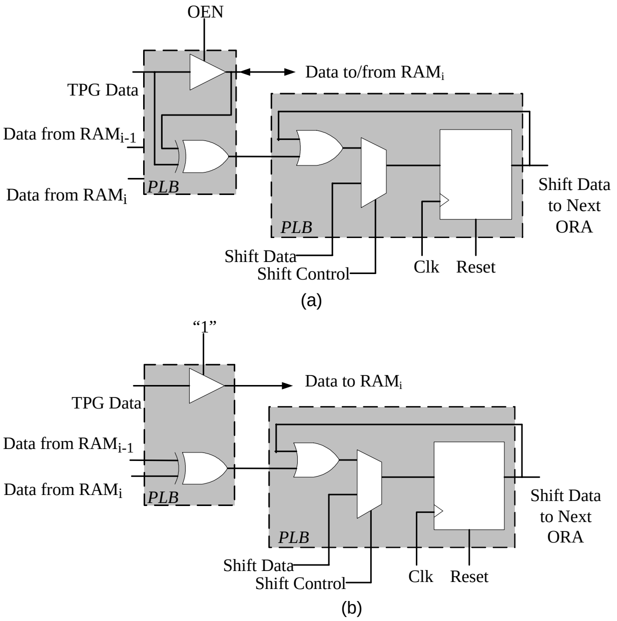

Figure 6. RAM BIST ORA Reconfiguration a) Single-port Modes b) Dual-port Mode

The design of the ORA used to test the dual-port RAM mode is illustrated in Figure 6(b). Comparison of the outputs of adjacent RAMs, similar to the BUTs in logic BIST, is used in this mode instead of using comparison with expected data as in the single-port RAM BIST. The tri-state bus is always enabled and connects TPG data to the write port of dual-port *free* RAM. The data read from the RAM is compared with the data from an adjacent RAM and a logic 1 is latch in case of any mismatch due to faulty RAM(s). While the tri-state buffer is redundant in dual-port mode, routing of read data from two free RAMs in redundant in single-port modes. Also, the read address that comes from the registered TPG is routed in all configurations and is redundant in single-port RAM BIST modes. However, the redundancy makes the reconfiguration by the embedded processor core much easier since the reconfiguration from one RAM BIST mode to another requires only modification of internal routing of the PLBs used to construct the ORAs, as illustrated in Figure 6. No other changes are required, since the TPG signals are supplied by the AVR and since the signals are already routed to all RAMs and ORAs.

The approach used to develop the program and debug the subroutines that generate the initial configuration is similar to the one used for logic BIST. The bitstreams generated using a mixed MGL-VHDL approach [3][4] serve as a baseline in generating the initial BIST configuration. Combining all the three configurations into a single program improves both the test-time and memory required to store the BIST configurations by about a factor of 9 when compared to three BIST configurations which are downloaded individually to completely test the *free* RAMs [3]. While this result is not as impressive as in the case of logic BIST, we are replacing only three RAM BIST configurations with a single program versus replacing 16 logic BIST configurations with a single program. However, the program and subroutines developed for RAM BIST reconfiguration and testing can be combined with those developed for logic BIST to achieve an even better speed-up in testing time and overall reduction in external memory storage requirements.

7. Summary and Conclusions

We have presented the improvements in the total test time and reductions in BIST configuration memory storage requirements as a result of the development of a single program that executes on the embedded processor core for the complete reconfiguration, execution, and retrieval of test results during BIST of the programmable logic resources in the FPGA core of the Atmel AT94K series configurable SoC. The ability to perform dynamic partial reconfiguration of embedded FPGA core from the embedded processor core provides a major testing capability, while the non-existent configuration memory read-back capability makes the SoC testing (and test development) much more difficult. By having a single program downloaded into the program memory of the embedded processor to reconfigure the FPGA core algorithmically, downloads to the FPGA core are eliminated, resulting in significant reduction in the total testing time (a factor of 45 in this case) as well as the configuration memory required (a factor of 83 in this case) compared to the previous work done in [3][4][6]. The single AVR BIST and diagnostic program is sufficiently small to reside on-chip for on-demand BIST and diagnosis of the programmable logic resources, including PLBs and RAMs, in the FPGA core of the SoC.

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

[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] C. Stroud, S. Garimella, J. Sunwoo, “On-Chip BIST-Based Diagnosis of Embedded Programmable Logic Cores in System-on-Chip Devices,” *Proc. ISCA International Conf. on Computers and Their Applications*, pp. 308-313, 2005

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

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

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

[8] A. Van de Goor, G. Gaydadjiev, V. Jarmolik and V. Mikitjuk, “March LR: A Test for Realistic Linked Faults,” *Proc. IEEE VLSI Test Symp.*, pp. 272-280, 1996

[9] C.E. Stroud, K.N. Leach and T.A. Slaughter, “BIST for Xilinx 4000 and Spartan Series FPGAs: A Case Study,” *Proc IEEE International Test Conf.*, pp. 1258-1268, 2003