

A System for Automated Built-In Self-Test of Embedded Memory Cores in System-on-Chip



Srinivas Garimella
Graduate Student
Department of ECE
Auburn University
Auburn, AL 36849, USA
garimsm@auburn.edu

Charles Stroud
Professor
Department of ECE
Auburn University
Auburn, AL 36849, USA
strouce@auburn.edu

Keywords: BIST, SoC Testing, Embedded Memory Testing, VHDL

Abstract—A system for automatic generation of Built-In Self-Test (BIST) for embedded memory cores in a system-on-chip (SoC) is presented. The BIST approach tests RAMs of any address and data bus widths and can test both single-port and dual-port RAMs operating in synchronous or asynchronous mode. A Field Programmable Gate Array (FPGA) independent BIST model is developed using VHDL. The parameterized VHDL model has been synthesized and used to test various sizes and types of embedded RAMs in SoCs and FPGAs.¹

1. INTRODUCTION

System on Chips (SoCs) typically incorporate many types of cores that are pre-designed blocks or models of complex functions that can be used in a large number of applications [1]. Field Programmable Gate Array (FPGA) cores are becoming an important component of recent SoCs because of the benefits offered through reconfiguration of the FPGA core. For example, an embedded FPGA core can be used for fixing logic design errors, for implementing different versions of the same application, and for implementing algorithms and protocols that are likely to change. Memory cores, on the other hand, form the most common components of a SoC. According to the International Technology Roadmap for Semiconductors (ITRS), memories will account for 90% of SoC die area by 2010 [2]. Memory cores in a SoC can be of any type: Static RAM (SRAM), Dynamic RAM (DRAM), or flash memory. Memory is also embedded into most current FPGAs in order to avoid the inefficient use of programmable logic blocks (PLBs) to implement large memories inside an FPGA. Memories are the densest components in SoCs and FPGAs; hence, they are more prone to defects than other components on the chip. With the number and size of memories increasing, it is essential to thoroughly test these embedded memories in SoCs and FPGAs.

It has been proposed that the programmable logic and interconnect resources of embedded FPGA cores in

SoCs can be used to test other embedded cores including memories [12]. However, as experienced during the previous development of BIST for FPGAs themselves [6][8][12][13][14], the actual BIST configurations required for complete testing are architecture dependent and must be developed for each family of FPGAs even though the overall BIST approach is architecture independent. This development can take many months. With the proliferation of various SoC and FPGA architectures that incorporate memory cores of varying sizes and functionality, it is important to be able to quickly develop and implement BIST for these cores.

The basic idea of this paper is the development of a system for the automatic generation of a BIST approach that implements the most efficient *march* algorithm (in terms of fault detection capability versus test time) for testing embedded memory cores in SoCs. The goal of the system is to generate a parameterized Hardware Description Language (HDL) like VHDL or Verilog which can easily be synthesized and downloaded into an FPGA, or the FPGA core in a SoC, for testing any number of sizes and types of embedded memories. The BIST circuitry includes a FSM-based Test Pattern Generator (TPG) and a number of comparison-based Output Response Analyzers (ORAs) along with test control and BIST results retrieval mechanisms. This FPGA independent BIST implementation can then be used to test memory components in any FPGA with minimal or no changes. Once the BIST circuitry has been downloaded in to the FPGA core and the embedded memories have been tested, the intended system function can be re-programmed into the FPGA core without incurring any area or performance penalties due to BIST circuitry.

The paper is organized as follows: Section 2 presents an overview of background information on the prior research in BIST for FPGAs and embedded memory cores. Section 3 describes the implementation of the BIST approach in FPGAs from Atmel and Xilinx. Advantages and limitations of this approach are discussed in Section 4. Details of the tool developed for automatically generating VHDL code for the *march* sequences are discussed in Section 5 and the paper is concluded in Section 6.

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

2. BACKGROUND

There has been considerable research and development in BIST for FPGAs which has typically been partitioned into BIST for the programmable logic resources (specifically the PLBs) and BIST for the programmable routing resources [6][8][12][13][14]. In most approaches to BIST for FPGAs, some of the PLBs are configured as Test Pattern Generators (TPGs) and Output Response Analyzers (ORAs). While both logic and routing resources are required to implement any FPGA BIST technique, the specific target of the testing is either the programmable logic (specifically the PLBs, referred to as blocks under test, or BUTs) or the programmable interconnect wire segments and switches. The TPGs for logic BIST are typically simple Linear Feedback Shift Registers (LFSRs) or counters to generate pseudo-exhaustive test patterns.

The most common logic BIST architecture is illustrated in Fig.1 where the BUTs and ORAs are arranged in alternating columns (or rows) [8]. Multiple, identical TPGs drive alternating columns (or rows) of BUTs with identical test patterns while the output responses of these identically programmed BUTs are compared in the neighboring columns (or rows) of ORAs. 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. Given the number of test configurations required to completely test a PLB, N_{BUT} , the minimum number of test configurations for a complete logic BIST is $2N_{BUT}$ when at least half of the PLBs are BUTs during any given test session. The value of N_{BUT} is typically on the order of 5 to 15 depending on the architecture and functionality of the PLB.

Extensive research has been done on the fault models of memories and various algorithms have been developed in literature to test memories for defects that have been observed in fabricated devices. These tests are called *march* tests and consist of similar and very regular *march* elements which are a sequence of memory read and write operations. The regularity of the *march* elements can be exploited in implementing the test algorithm as a finite state machine (FSM) [3][4]. In [3], an FSM based ap-

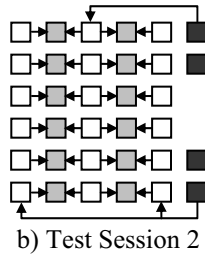
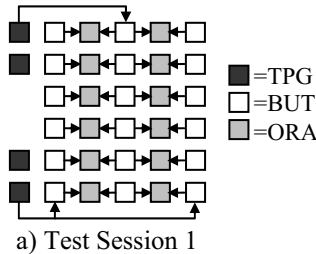


Fig. 1: Logic BIST architecture

proach is used for Built In Self Test (BIST) of memories, but transparency of the BIST approach made the implementation complex. When testing embedded memories in the FPGAs, such transparency is not needed since the desired system logic can be downloaded into the FPGA after testing. In [4], an FSM based approach is used, but the implementation required extra storage for loading instructions for controlling the flow of the algorithm.

3. RAM BIST IMPLEMENTATION

The BIST approach was originally implemented to test embedded memories in the Atmel AT94K series SoC which consists on an embedded FPGA core, embedded memory cores, and an embedded Advanced Virtual RISC (AVR) processor with various peripheral units [10]. There are two kinds of memories in the AT94K series SoC: 32x4-bit SRAMs (called *free RAMs* in Atmel terminology) dispersed over the entire array of the embedded FPGA. A *free RAM* is associated with each 4x4 array of PLBs. In addition to the *free RAMs*, there is a 36 Kbyte dual-port SRAM memory shared by both FPGA and AVR processor cores.

The *free RAMs* distributed throughout the FPGA core can be configured to operate in four different modes: single-port synchronous and asynchronous modes, dual-port synchronous and asynchronous modes. The *free RAMs* must be tested in all modes of operation. However, the *free RAMs* along the rightmost edge of the FPGA cannot operate in dual-port mode. Furthermore, it should be noted that the dual-port mode of operation is not a true dual-port RAM and also the read port is always asynchronous. As a result, these RAMs need not be tested in dual-port asynchronous mode if tested in the remaining three modes of operation.

The BIST architectures used for testing the *free RAMs* are illustrated in Fig. 2 and are similar to the one used for testing PLBs in [6] where TPGs and ORAs are constructed from PLBs in the FPGA array. A single TPG is used instead of the two TPGs used in BIST for PLBs for two reasons. First, the purpose of the two TPGs typically used in BIST for PLBs is to detect any faults in the PLBs used to construct the TPG. However, a single TPG is used in RAMBIST with an assumption that all the PLBs have been tested before testing the RAMs. Second, the TPGs for testing the RAMs are much more complicated than those used to test PLBs and, as a result, require a larger number of PLBs for implementation. Therefore, it

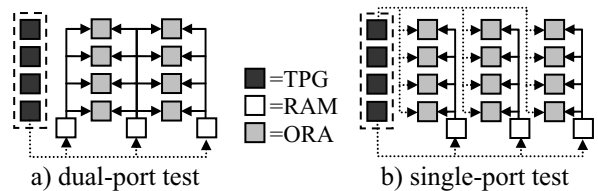


Fig. 2: RAM BIST architectures

is impossible to fit two RAM BIST TPGs in smaller FPGAs. The ORAs are comparison-based designs similar to those used in BIST for the PLBs [8]. For the dual-port RAM mode of operation, the ORAs compare the output data from two identical RAMs (shown in Fig. 2a). However, in single-port RAM configurations, the TPG can easily generate expected output data results such that each ORA compares data from a given RAM with expected data generated by the TPG (shown in Fig. 2b).

The TPG implements the March LR algorithm [5] with background data sequences (BDS) to test the RAMs in the single-port synchronous configuration. The dual-port RAM (DPR) test algorithm [6] without BDS is used for the dual-port synchronous configuration and a *March Y* algorithm without BDS is used for the single-port asynchronous configuration. Use of BDS in all configurations is redundant, because coupling faults and pattern sensitive faults detected once with BDS need not be tested for in all modes.

The 36 Kbyte SRAM that resides between the AVR and FPGA is accessible by both AVR and FPGA cores from their respective sides. However, the upper 20 Kbytes of the RAM is used as program memory and is accessed only by the AVR core. The lower 4 Kbytes of the RAM is accessed only by the FPGA core. The remaining 12 Kbytes of RAM can be configured to be used as data memory by both AVR and FPGA or as a program memory accessible only by the AVR. This 12 Kbyte of data SRAM along with the 4 Kbyte portion accessible only by the FPGA is tested as a single-port using FPGA logic. The March LR algorithm with BDS developed for *free RAMs* is used to implement the TPG by changing required parameters.

The BIST approach was developed as a parameterized VHDL model capable of testing multiple single-port and dual-port RAMs. The code was parameterized to facilitate selection of any number of RAMs to be tested as well as data and address bus widths associated with the RAMs. In addition, the parameterized VHDL provides the user with the ability to select of the active clock edge and the active level for control signals, such as the RAM write enable. The PLB counts obtained on synthesis of the various algorithms are given in Table I.

TABLE I: BIST PLB COUNT FOR ATMEL FPGA CORE

RAM BIST Algorithm	TPG PLBs	ORA PLBs
DPR Test w/o BDS	66	$N \times (N-2) \times 8$
<i>March Y</i> w/o BDS	18	$N \times (N-1) \times 8$
<i>March LR (free RAMs)</i>	123	$N \times (N-1) \times 8$
<i>March LR (Data RAM)</i>	230	16
$N = \text{number of free RAMs in one dimension of array}$		

The parameterized VHDL based BIST approach was used to test the embedded dual-port RAMs (called *block*

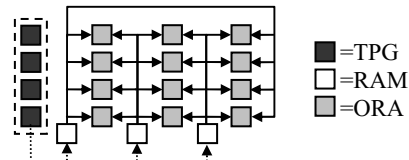


Fig. 3: Block RAM BIST architecture

RAMs in Xilinx terminology) distributed in Xilinx Virtex (I and II) and Spartan (II and III) series FPGAs in order to assess the flexibility and versatility of the VHDL-based BIST approach. The 4 Kbit *block RAMs* in the Virtex I and Spartan II FPGAs are true dual-port RAMs and can have five different sizes ranging from 4K×1-bit to 256×16-bit [11]. Therefore, five configurations are required to completely test *block RAMs* in a single port mode. In each configuration, both ports of the dual-port RAM are tested in turn using *March LR*. In the 4K×1-bit mode, the RAM core is tested for pattern sensitivity and coupling faults by the *March LR* algorithm and, as a result, BDS need not be applied to the other RAM configurations. However, to test for bridging faults in the write and read data busses of the RAM, *March LR* with BDS is used to test the 256×16-bit mode. Since the *block RAMs* are true dual-port RAMs, two additional march algorithms (*March d2pf* and *March s2pf* [7]) are used to implement the TPG to test dual-port access of the RAMs. A total of seven RAM BIST configurations (5 single-port and 2 dual-port) are required to completely test *block RAMs*.

The BIST approach was synthesized, downloaded, and verified in FPGAs to test these *block RAMs*. While all *block RAMs* are tested in parallel, a reconfiguration of the FPGA is required for each RAM BIST configuration. The same parameterized VHDL *March LR* algorithm developed for *free RAMs* is used to implement TPGs in all single-port tests. In all configurations, the ORAs are configured to compare data from identical RAMs. To increase diagnostic resolution an extra set of ORAs are added to compare the RAMs at the edges as illustrated in Fig. 3. The PLB counts obtained on synthesis of the various RAM sizes and test algorithms are given in Table I. Only the smallest Spartan II FPGA (the XC2S15 with an 8×12 array of PLBs and four *block RAMs*) has insufficient logic resources to fit the BIST circuitry and can only have half of its block RAMs tested without BDS during a given BIST configuration; all Virtex I and all other Spartan II FPGAs can easily fit the complete BIST circuitry.

TABLE II: BIST PLB COUNT FOR VIRTEX I & SPARTAN II

RAM BIST Algorithm	TPG PLBs	ORA PLBs
<i>March LR</i> w/o BDS	31	$N \times (D+1)$
<i>March LR</i> with BDS	100	$N \times (D+1)$
$N = \# \text{ of block RAMs}$		$D = \# \text{ of data bits}$

The 18 Kbit *block RAMs* in the Virtex II and Spartan III FPGAs are also true dual-port RAMs and can have six

different sizes ranging from 16K×1-bit to 512×36-bit [15]. As a result, total of eight RAM BIST configurations (6 single-port and 2 dual-port) are required to completely test these *block RAMs*. Due to the more complex PLBs in Virtex II and Spartan III FPGAs (about twice that of Virtex I and Spartan II), the PLB counts given in Table II can be divided by two to obtain the number of PLBs required for TPG and ORAs in each RAM BIST configuration for the Virtex II and Spartan III *block RAMs*.

4. APPLICABILITY AND LIMITATIONS

This system can be used to test single-port and dual-port RAMs embedded in any FPGA or SoC. In SoC applications, the ports of the embedded memory cores must be accessible from the FPGA core to be tested completely. Some support is needed from the synthesis tools to control placement of RAMs with respect to their ORA logic for diagnosis of faults in the embedded RAM cores. Placement control is even more important in smaller FPGAs where it might not be possible to test all the RAMs in a single phase due to limited programmable logic resources for implementing the TPG and ORA functions. Placement cannot be controlled from Atmel's synthesis tool (called Figaro) if VHDL modeling is used. Unless the placement of RAMs can be controlled during synthesis, faulty RAMs cannot be identified from BIST results. The solution is to either manually place the RAMs or maintain mapping information for identifying physical location of the faulty RAMs from BIST results. As design gets larger, manual placement becomes tedious and mapping information may change each time the design is synthesized.

As a result, a VHDL-only approach did not prove to be beneficial for Atmel FPGAs. Macro Generation Language (MGL) [9] is provided by Atmel for AT40K and AT94K devices. MGL has features similar to VHDL as well as features which allow placement of logic blocks and control of routing. As a result, a mixed approach was used by making use of both VHDL and MGL. While MGL is used to define placement of RAMs and ORAs and their interconnection, VHDL is used for the TPG. Xilinx synthesis tools allowed placement of RAMs with respect to their ORA logic via a constraint file. Therefore a VHDL-only approach was used for testing memory components in Xilinx devices. For FPGAs where the contents of the ORA flip-flops cannot be read by configuration memory read back, the ORA was designed to incorporate a shift register to enable reading of BIST results. If ORA data can be read back from the FPGA configuration memory, as is the case in Xilinx FPGAs, a simpler design can be used for the ORA.

The fact that embedded RAMs can operate in different modes affects the portability of VHDL. Furthermore, memories may have to be tested with different *march* sequences if the memory technology changes. With

changes in memory technology, the fault models adopted for testing may change and this results in redevelopment of VHDL code for the TPG. An attempt was made to avoid this limitation by developing a tool which is capable of automatically generating VHDL code for any given *march* sequence. Currently, this tool is capable of generating code for single-port *march* sequences only. Details of this tool are described in the next section.

5. MARCH TEST GENERATION TOOL

The tool, called RAMBISTGEN, automatically generates VHDL code for any size of memory, for any active edge of the clock, and any active levels for control signals. The graphical user interface (GUI) of the tool is shown in Fig. 4. The tool takes an input file containing the desired *march* sequence and automatically produces an output file containing the resulting VHDL code to generate that sequence. The format for the input file is as follows:

```
<u/d> <r/w> <data> [, <r/w> <data>]
<u/d> <r/w> <data> [, <r/w> <data>]
```

Each line of the input file represents a *march* element of the sequence. Each line starts with *u* or *d* indicating the addressing order in *up* or *down* direction, respectively. This is followed by *r* or *w* indicating *read* or *write* operation, respectively, followed by the data to be read or written. The input is not case sensitive. Address bus width is specified by the user in the GUI and data bus width is interpreted from the read/write data in the input file. Example input file formats for *March Y* and *March LR* algorithms are given below.

March Y: $\uparrow \downarrow (w0); \uparrow (r0, w1, r1); \downarrow (r1, w0, r0); \uparrow (r0);$

RAMBISTGEN Input File Format:

```
d w 0
u r 0 , w 1 , r 1
d r 1 , w 0 , r 0
u r 0
```

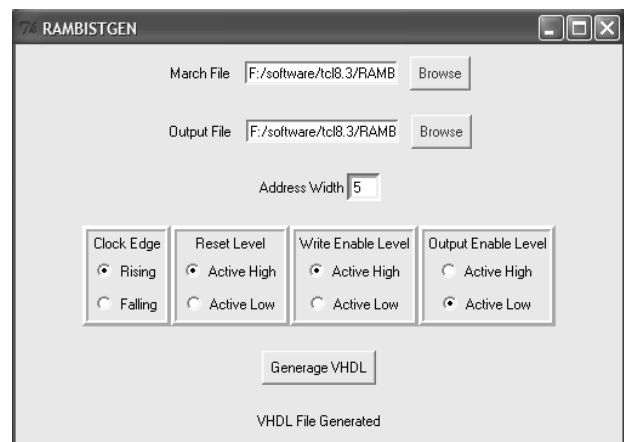


Fig. 4: RAMBISTGEN user interface

March LR with BDS : $\uparrow\downarrow(w0000); \downarrow(r0000; w1111);$
 $\uparrow(r1111; w0000; r0000; r0000; w1111); \uparrow(r1111;$
 $w0000); \uparrow(r0000; w1111; r1111; r1111; w0000);$
 $\uparrow(r0000; w0101; w1010; r1010); \downarrow(r1010; w0101; r0101);$
 $\uparrow(r0101; w0011; w1100; r1100); \downarrow(r1100; w0011;$
 $r0011); \uparrow(r0011);$

RAMBISTGEN Input File Format:

```

u w 0000
d r 0000 , w 1111
u r 1111, w 0000, r 0000, r 0000, w 1111
u r 1111, w 0000
u r 0000, w 1111, r 1111, r 1111, w 0000
u r 0000, w 0101, w 1010, r 1010
d r 1010, w 0101, r 0101
u r 0101, w 0011, w 1100 , r 1100
d r 1100, w 0011, r 0011
u r 0011

```

The RAMBISTGEN tool generates approximately 140 and 300 lines of VHDL code for implementing these March Y and March LR algorithms, respectively. The tool interprets the input file as follows:

1. Each line of the file is categorized as a phase.
2. All the words separated by a comma are treated as different elements of that phase. For e.g. in `u r 1111, w 0000` there are two elements: `r 1111` and `w 0000`.
3. During FSM implementation, each phase is treated as a separate state and each element of that phase forms a sub-state of that phase. VHDL code generated for this *march* element is as shown below.

```

process (Sensitivity List) begin
case phase is
...
when phase_no =>
case element is
when element1 =>
if (Address /= MaxAddress ) then
Data <= "1111";
OEN <= OEN_ACTIVE;
WEN <= not (WEN_ACTIVE);
Next_element <= element_no;
else
Data <= "0000";
OEN <= not (OEN_ACTIVE);
WEN <= WEN_ACTIVE;
Address <= Address + inc;
...
end process;

```

This tool was created using Tool Control Language/Tool Kit (Tcl/Tk) and is compatible with Windows and Linux environments.

6. CONCLUSION

An automatically generated BIST approach for testing embedded memory cores in FPGAs and SoCs was discussed. The feasibility and flexibility of this approach

was demonstrated by implementing the BIST and testing memories in different manufacturers' FPGAs with only minor modifications. The approach can be followed for testing other types of RAMs like DRAMs and Flash by changing the TPG algorithm. A similar approach can be used for testing other regular structure cores like multipliers embedded in FPGAs and SoCs.

REFERENCES

- [1] A. Milenkovic and D. Fatzer, "Teaching IP Core Development: An Example", *Proc. IEEE Conf. on Microelectronic Systems Education*, 2003, pp. 16-17.
- [2] ___, "International Technology Roadmap for Semiconductors, 2000 Update, Design", 2000, pp. 5-6.
- [3] B. Cockburn and Y. Sat, "Synthesized Transparent BIST for Detecting Scrambled Pattern-Sensitive Faults in RAMs", *Proc. IEEE Int'l Test Conf.*, 1995, pp. 23-32.
- [4] K. Zarrineh and S. Upadhyaya, "On Programmable Memory BIST Architectures", *Proc. Design Automation and Test in Europe*, 1999, pp. 708-713.
- [5] A. Van de Goor, G. Gaydadjiev, V. Jarmolik and V. Mikitjuk, "March LR: A Test for Realistic Linked Faults," *Proc. IEEE VLSI Test Symp.*, 1996, pp. 272-280.
- [6] C. Stroud, K. Leach and T. Slaughter, "BIST for Xilinx 4000 and Spartan Series FPGAs: A Case Study", *Proc. IEEE Int'l Test Conf.*, 2003, pp. 1258-1267.
- [7] S. Hamdioui and A. Van de Goor, "Efficient Tests for Realistic Faults in Dual-Port SRAMs", *IEEE Trans. on Computers*, vol. 51, no. 5, 2002, pp. 460-473.
- [8] M. Abramovici and C. Stroud, "BIST-Based Test and Diagnosis of FPGA Logic Blocks", *IEEE Trans. on VLSI Systems*, Vol. 9, No. 1, 2001, pp. 159-172.
- [9] ___, "Integrated Development System AT40K Macro Library Version 6.0," User's Manual, Atmel Corp., 1998.
- [10] ___, "AT94K Series Field Programmable System Level Integrated Circuit," Datasheet, Atmel Corp., 2001.
- [11] ___, "Virtex Field Programmable Gate Arrays," Data Sheet DS003-1, Xilinx, Inc., 2001.
- [12] M. Abramovici, C. Stroud, and J. Emmert, "Using Embedded FPGAs for SoC Yield Improvement," *Proc. ACM/IEEE Design Automation Conf.*, 2002, pp. 713-724.
- [13] C. Stroud, J. Harris, S. Garimella and J. Sunwoo, "BIST Configurations for Atmel FPGAs Using Macro Generation Language", *Proc. IEEE North Atlantic Test Workshop*, 2004, pp. 83-90.
- [14] C. Stroud, J. Nall, M. Lashinsky and M. Abramovici, "BIST-Based Diagnosis of FPGA Interconnect," *Proc. IEEE Int'l Test Conf.*, 2002, pp. 618-627.
- [15] ___, "Virtex II Platform FPGAs," Data Sheet DS031, Xilinx, Inc., 2003.