Built-In Self-Test Configurations for Atmel FPGAs Using Macro Generation Language

Charles Stroud, Jonathan Harris, Srinivas Garimella, and John Sunwoo
Dept. of Electrical and Computer Engineering
Auburn University
Auburn, AL 36849-5201
strouce/harri34/garimsm/sunwojo@auburn.edu

ABSTRACT: The development and automatic generation of Built-In Self-Test (BIST) configurations for Atmel AT40K series Field Programmable Gate Arrays (FPGAs) are described. These BIST configurations completely test the programmable logic and routing resources in the core of the FPGA along with the dedicated Random Access Memories (RAMs) dispersed within the array. The BIST configurations are generated using Atmel's Macro Generation Language (MGL) for any size FPGA. The advantages and limitations of this approach are discussed.[1]

1. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) consist of a two-dimensional array of Programmable Logic Blocks (PLBs) interconnected by a programmable routing network and surrounded by programmable input/output (I/O) cells. In recent FPGAs, additional dedicated Random Access Memories (RAMs) are provided within the array for data storage functions. The system function performed by the FPGA is established by the contents of the configuration memory such that an FPGA can perform a wide variety of system functions by rewriting the configuration memory. The PLBs consist of sections of the configuration memory that can be used as Look-Up Tables (LUTs) to perform combinational logic functions. In addition, the PLBs contain flip-flops to perform sequential logic functions.

While attractive to designers, the programmability of FPGAs poses serious testing problems due to the large number of configurations required to test all possible modes of operation. One solution is to reprogram the FPGA with BIST circuitry to allow the FPGA to test itself without the need for expensive external test equipment [1]. When completely tested and found to be fault-free, the FPGA can be safely reprogrammed to perform the intended system function(s) prior to normal system operation. In the event that a fault is detected by the BIST, additional BIST-based diagnostic configurations can be downloaded into the FPGA to facilitate identification of the faulty resource(s) and reconfiguration of the FPGA to avoid the faults during on-line operation [1]. BIST for FPGAs can be used at all levels of testing (from wafer to system-level testing) without the area overhead and performance penalties associated with BIST approaches for Application Specific Integrated Circuits (ASICs) [2].

One problem with BIST for FPGAs is that while the basic BIST architecture is generic, the specific test configurations are not [3]. The BIST configurations for the PLB and interconnect resources must be developed for each specific FPGA architecture. Once developed, these BIST configurations can be applied to all devices of the same size and type. Scaling the BIST configurations to fit different size FPGAs in the same family is relatively straight forward. However, developing BIST configurations for a new family of FPGAs requires development of new test configurations specific to that architecture. Therefore, the challenge is to develop a mechanism to automatically generate the BIST configurations for a given FPGA series architecture. Fortunately, Atmel's Macro Generation Language (MGL), provided as part of their Integrated Development System (IDS), is a programming language specifically for the development of regular and parameterized circuits to be implemented in their AT40K series FPGA. As a result, MGL has the potential for supporting automatic generation of BIST configurations for these FPGAs.

In this paper, we describe development of programs in MGL that are capable of automatic generation of all BIST configurations needed to test the core of any Atmel AT40K series FPGA as well as the FPGA core in AT94K series System-on-Chips (SoCs). We begin with a presentation of background material in Section 2 including the architecture and operation of BIST approaches for FPGAs, an overview of the architecture of Atmel FPGAs, and an overview of MGL. We then describe development of MGL-based generation of BIST configurations, including fault coverage and problems encountered, for programmable logic, RAMs, and interconnect resources in Sections 3, 4 and 5, respectively. The paper is summarized in Section 6.

2. BACKGROUND

2.1. Overview of FPGA BIST

In most approaches to BIST of programmable logic and interconnect resources in 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 an FPGA BIST technique, the specific target of the testing is either the logic blocks under test (BUTs) or intercon-

[1] This work is sponsored by National Security Agency.

nect wires under test (WUTs) [1]-[10]. The TPGs for logic BIST are typically Linear Feedback Shift Registers or counters to generate pseudo-exhaustive test patterns [1] while TPGs for routing BIST typically generate exhaustive test patterns [9] or walking test patterns [7]. Parity check ORAs have been used for routing BIST [6]. Due to the replication of the PLBs and their associated routing in the FPGA, comparison-based ORAs are effective and offer good diagnostic resolution for both logic and routing BIST [1][9].

The most common logic BIST architecture is to arrange the BUTs and ORAs in alternating columns (or rows) [1]. 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. The BUTs are repeatedly reconfigured in their various modes of operation until the BUTs are completely tested. The roles of the PLBs are then reversed by flipping the architecture such that the PLBs that were previously tested as BUTs become the TPGs and ORAs in order to test the PLBs that were previously TPGs and ORAs [1]. 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 a function of the architecture and functionality of the PLB and is typically on the order of 10 to 15 for PLBs such as those in the Lattice Semiconductor ORCA 2C/2CA [1] and the Xilinx 4000 and Spartan series FPGAs [4]. The only prior work in logic BIST for Atmel AT40K series FPGAs was for the FPGA core of the AT94K series SoC (or Field Programmable System Level Integrated Circuit – FPSLIC, in Atmel terminology [12]) where the microprocessor core in the SoC was used for on-line testing of the LUTs of the PLBs using two test configurations per LUT [8]. A total of $2N^2$ test configurations (where N is the number of PLBs in one dimension of the array) were required since the PLBs were tested one at a time. It should also be noted that the PLB was not completely tested in this approach.

A number of routing BIST architectures have been developed including a comparison-based approach [9], a parity-based approach [6], and an oscillation-based delay-fault testing approach [11]. In the most commonly implemented routing BIST approach, small Self-Test AReas (STARs) are constructed with PLBs used to implement TPGs to source identical test patterns (either walking [7] or exhaustive [9] patterns) to two sets of WUTs that are then compared by the ORA at the other end of the WUTs. The FPGA is filled with these STARs to facilitate concurrent testing of as many routing resources as possible in order to minimize the total number of test configurations for complete testing of the interconnect network for the various faults that can occur. These faults include shorts and opens in the wire segments, wire segments stuck-at-1 and stuck-at-0, stuck-on and stuck-off faults in the various types of Programmable Interconnect Points (PIPs) or switches that are used to program the interconnect network as well as stuck-at faults in the configuration memory bits used to program the PIPs [2]. Vertical and horizontal STARs test vertical and horizontal interconnect resources, respectively. It should be noted that the STAR-based technique can be used with the parity-based BIST approach in [6]. Since the programmable routing resources typically account for a larger portion of the FPGA than the PLBs, the number of test configurations is generally larger as well. The total number of test configurations for complete testing is a function of the interconnect network architecture and have ranged from approximately 45 for the Lattice ORCA 2C and 2CA series FPGAs [9] to over 200 for the Xilinx 4000 and Spartan series FPGAs [4]. We are not currently aware of any prior work in routing BIST for Atmel FPGAs.

2.2. Overview of Atmel's FPGAs and MGL

The Atmel FPGA consists of a symmetrical array of identical cells throughout the architecture, as illustrated in Figure 1. Each PLB has two 3-input LUTs, a Set/Reset D Flip-Flop (FF), and several multiplexers that provide a variety of functionality. For every 4×4 array of logic cells, there is one 32×4 RAM. The RAM can be operated in single port or dual port mode and

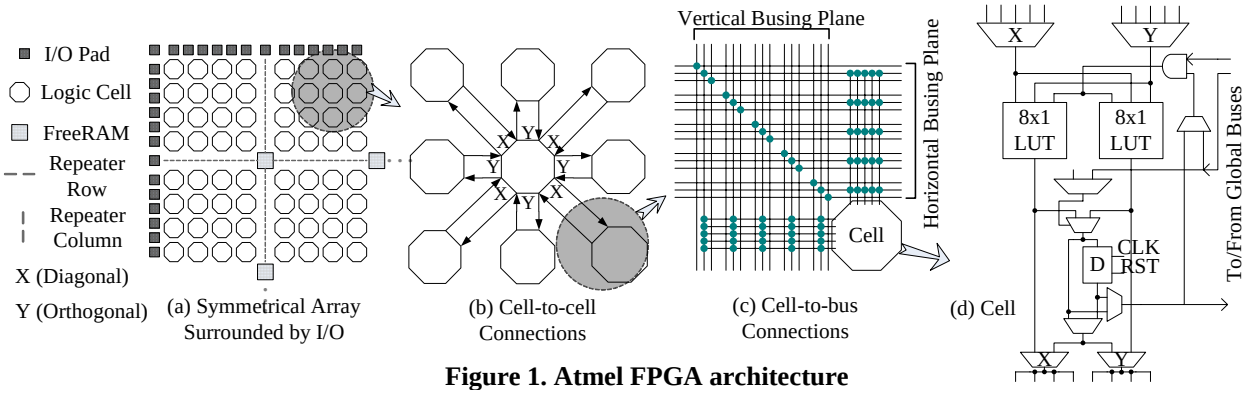

Figure 1. Atmel FPGA architecture

there is no use of logic resources in any PLB when accessing the SRAM. Dedicated local routing resources allow direct horizontal, vertical, and diagonal PLB-to-PLB connections without using global routing resources. Five vertical and five horizontal bus planes are associated with each PLB, where each plane consists of two express lines and one local line that can be accessed by any of the four inputs to the PLB or by the output from the PLB. Bus repeaters are placed within the global routing resources every four cells to prevent signal degradation for long and/or heavily loaded nets.

Given the time involved in developing BIST configurations for a given FPGA architecture and the large number of BIST configurations required to completely test both the logic and routing resources, it is desirable to develop a mechanism that facilitates automatic generation of the BIST configurations for all sizes of FPGAs in a given series. This has been done previously for Xilinx 4000 and Spartan series FPGAs using Xilinx Design Language (XDL) [4] and for Lattice ORCA 2C and 2CA series FPGAs using a language similar to XDL [3]. These languages are a textual netlist-like description of the programming and interconnection of the PLBs for a given configuration of the FPGA. Independent programs were developed in C and Perl to automatically generate the XDL description for each BIST configuration based on the array size of the particular FPGA [4]. Each XDL file was then converted to a configuration bit file through programs provided in the FPGA tool suite.

MGL, on the other hand, provides users of the AT40K FPGAs and AT94K FPGA cores with an integrated method for creating parameterized, user-defined circuits to meet desired design specifications. In order to configure a PLB, MGL requires the use of either predefined macros (gates, multiplexers, flip-flops, etc.) or dynamic macros. Dynamic macros give the most versatility in defining the functionality of a PLB ranging from any 2-input to 4-input logic function in various combinations of registered or non-registered outputs and feedback [13]. The two look-up tables (LUTs) in the PLB are programmed through Boolean expressions declared within the dynamic macro instantiation. However, user control over the PLBs is limited to the dynamic macros and no further control is provided by MGL. These dynamic macros can be instantiated through the available graphical interface in IDS, or through the use of MGL. The former must be instantiated each time a design is created whereas the latter is easily automated since MGL is similar to a programming language and may be parameterized and used to create designs in any of the Atmel AT40K FPGAs and AT94K FPGA cores.

MGL is structured such that basic constructs typical to most programming or hardware description languages, such as VHDL, may be utilized. As in VHDL, the language in MGL is strongly typed, meaning that all objects must be assigned a specific type [14]. In contrast to VHDL, however, MGL is case-sensitive, such that *function()* and *Function()* are distinct and independent [14]. In order to instantiate a design using MGL in a particular FPGA array, three components must exist in the MGL code: user-defined functions (i.e., BUTs, ORAs, TPGs), the target FPGA device (i.e., AT40K*xx*), and the inputs and outputs to the circuit (i.e., clock input, reset input, pass/fail output) [14]. Pre-processor directives, global variables and constants may also be defined.

To illustrate the format of MGL and its similarities to programming and hardware description languages, the example given in Figure 2 was taken from our logic BIST MGL code. This example MGL code instantiates a column of ORAs which receive inputs from the outputs of the BUTs to the left (denoted *BL*) and right (*BR*) of the ORA, in addition to clock and reset inputs. To begin, the function is declared and a target FPGA device is selected, in this case an AT40K40 FPGA in a 208-pin package. To parameterize the instantiation of the column of ORAs, a variable, *ARRAY_SIZE*, is declared and is determined by the built-in MGL *sizeof()* function which returns the size of one dimension of the PLB array. Two *for* loops are next used to create all of the respective I/O ports and then all of the ORA instances which are placed in column 2 of the FPGA. The *connec-*

```
function ORAs(name : string) : macro
begin
fpga := setdevice( "AT40K40-2DQI");
ARRAY_SIZE := sizeof(fpga);
interface ORAsIO of ORAs is
  inputports("CLK", "RST");
  for i in 0 to ARRAY_SIZE loop
   inputports("BL"{0}"_"{i}, "BR" "{0}"_"{i});
   outputports("P/F" + i);
  end loop;
end interface;
contents of ORAs is
for i in 0 to ARRAY_SIZE loop
instance "ORAcell" + i of ORAcell is
   location(2,i);
   connections(
       "CLK" -> "CLK",
       "RST" -> "RST",
       "BL" -> "BL"{1}"_"{i},
       "BR" -> "BR"{3}"_"{i},
       "P/F" -> "P/F");
    placeports("P/F" -> "Y");
end instance;
end loop;
end contents;
return(ORAs);
end;//ORAs function complete
```

Figure 2. Example MGL program

tions statement defines the I/O connections of the respective ORA PLB.

The focus of this paper is the development of parameterized MGL programs that will automatically generate BIST configurations for any size AT40K FPGA or AT94K FPGA core. In the following sections, we describe the use of MGL and the resultant BIST configurations for the logic, RAM, and routing resources. In addition, we discuss MGL capabilities and limitations observed during this development.

3. LOGIC BIST

The logic BIST configurations developed for the Atmel AT40K series FPGAs and the FPGA core in the Atmel AT94K series SoCs consisted of five test phases, each with a different dynamic macro instantiated as the BUTs. These BIST configurations are automatically generated with parameterized MGL code. With the adjustment of a few parameters in the MGL code, the BIST configurations can be generated for any desired size FPGA or FPGA core. The architecture for the logic BIST test sessions are illustrated in Figure 3a and 3b with a 5-bit up-counter used for each TPG to drive the five inputs to each BUT. The logic BIST configurations are oriented along columns due to column-based bank clocks and resets for every group of four PLBs in the column. For example, in a row-based BIST architecture, the reset to the BUTs cannot be tested without resetting the ORAs and losing detected fault information. The X output of a BUT connects to any of four diagonally adjacent ORAs while the Y output connects any of four orthogonally adjacent ORAs. In order to observe both outputs of the BUTs, an alternating routing scheme was devised (illustrated in Figure 3c and 3d) in which connections between the BUTs and the comparison-based ORAs are alternated between each BIST configuration.

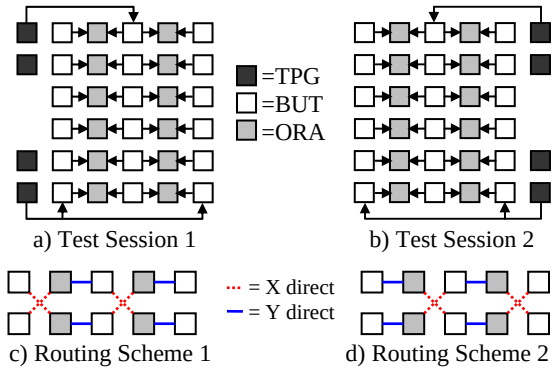

Figure 3. Logic BIST architecture and connections

Five BUT configurations were chosen from the dynamic macros available in MGL that yielded the maximum fault coverage. Fault simulations were performed considering the alternating BUT-to-ORA connection scheme shown in Figure 3c and 3d to produce the plot in Figure 4 showing both individual and cumulative fault coverage through the sequence of the five BUT configurations. The cumulative single stuck-at gate-level fault coverage was 98% for BUTs in the middle of the array. For those BUTs along the edges of the FPGA array, fault coverages of 97.3% and 97.8% were obtained for the two left-most and right-most columns, respectively. The macros used in the five BUT configurations are summarized in Figure 4 and Table 1. Limitations of MGL dynamic macros prevent control of all configuration bits in the BUTs such that only certain combinations of configuration bits exist within the instantiation of dynamic macros; this limits the maximum fault coverage to less than 100% as a result of the *invisible logic problem* described in [1].

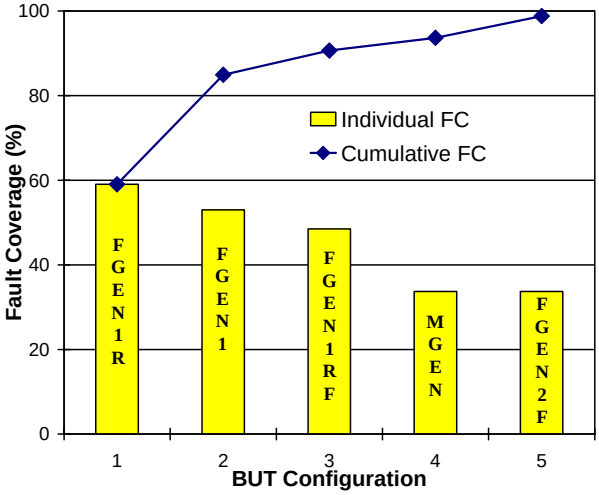

Figure 4. Logic BIST fault coverage

Table 1. Logic BIST Configurations

Config	*PLB Mode of Operation*	*Macro*
1	4-input LUT, rising-edge FF, active-high reset	FGEN1R
2	4-input LUT	FGEN1
3	4-input LUT, falling-edge FF with sequential feedback, active-low set	FGEN1RF
4	Multliplier-based LUTs	MGEN
5	Two 3-input LUTs with combinational feedback	FGEN2F

Due to the small size of the PLBs in the AT40K series FPGA, it is not possible to implement a comparison-based ORA with a shift register capability for retrieving the results at the end of the BIST sequence. One option is to have a 2-PLB ORA implementation, one PLB for comparison of four BUT outputs and the other PLB for latching mismatches and shifting out results, as illustrated in Figure 5a, but this reduces diagnostic resolution. However, since the device can be partially reconfigured without affecting the values held within the PLB flip-flop using the synchronous RAM configuration mode [12], a 1-PLB ORA which compares two BUT

outputs and latches mismatches (Figure 5b) can be partially reconfigured as a shift register (Figure 5c) upon completion of the BIST sequence to retrieve the ORA results. We use this dynamic partial reconfiguration approach for creating the shift register while maintaining maximum diagnostic resolution.

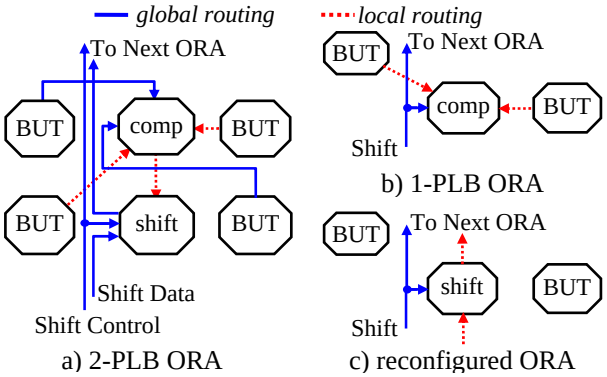

Figure 5. ORA/shift register implementations

An evaluation of the propagation delay for each of the five logic BIST configurations was performed to determine the maximum clock frequency for the various array sizes in the AT40K series FPGAs. This was done by using the Atmel Figaro IDS software to evaluate the worst case delays for each BUT configuration and array size. The various sizes of the AT40K*xx* FPGAs are: 05 (16×16 PLBs), 10 (24×24 PLBs), 20 (32×32 PLBs), and 40 (48×48 PLBs). Figure 6 gives the timing analysis results in terms of the maximum BIST clock frequency. As indicated, the maximum BIST clock frequency for the AT40K40 is approximately 27MHz, while the AT40K05 device can be operated at a clock frequency of greater than 60MHz for all five BIST configurations.

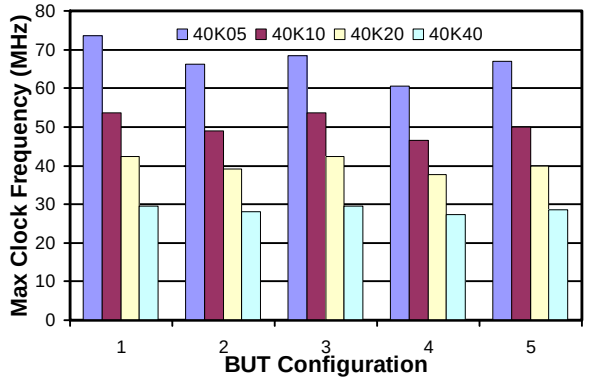

Figure 6. Maximum clock frequency for logic BIST

4. RAM BIST

The embedded RAMs (referred to as "free RAMs" in Atmel terminology) inside the FPGA core are distributed over the entire array. Each 4×4 array of PLBs share a 32×4-bit RAM. All these RAMs except those in the rightmost column can be configured as both single-port and dual-port RAMs and can operate in either synchronous or asynchronous mode. The dual-port RAMs are arranged in such a way that they have a common read address with one of their adjacent RAMs and a common write address with the other. The dual-port RAMs are not true dual-port RAMs in that they have one port dedicated for reading and another for writing. Reading and writing the dual-ports of the RAM are independent of each other and reading the RAM is completely asynchronous. Therefore, complex March algorithms used to test true dual-port RAMs, such as described in [17], need not be used. The dual-port RAM test algorithm used in [4] was modified to test the Atmel free RAMs. This algorithm (denoted *DPR test*) is given in Table 2 in terms of the operations performed on each port (*write*:*read*); '*n*' denotes no operation on that port.

Table 2. RAM BIST Configurations and Algorithms

Mode	***Config***	***Algorithm***
Sync Dual-Port	1) DPR test	⇑⇓(w0:n); ⇓(n:r0); ⇑(w1:⇓r1); ⇓(w0:⇑r0);
Sync Single-Port	2) March-LR w/BDS	⇑⇓(w0000); ⇓(r0000; w1111); ⇑(r1111; w0000; r0000; r0000; w1111); ⇑(r1111; w0000); ⇑(r0000; w1111; r1111; r1111; w0000); ⇑(r0000;w0101;w1010; r1010); ⇓(r1010; w0101; r0101); ⇑(r0101; w0011; w1100; r1100); ⇓(r1100;w0011; r0011);⇑(r0011);
Async Single-Port	3) March-Y w/o BDS	⇑⇓(w0); ⇑(r0;w1;r1); ⇓(r1;w0;r0); ⇑(r0);

Pattern sensitivity and intra-word coupling faults associated with bit-oriented memories are tested in the single-port RAM mode. This is because all RAMs are tested in the single-port mode, including the RAMs along the right hand column of the array. The March-LR algorithm described in [19] is used to test the RAMs in the synchronous single-port mode. The March-LR algorithm is modified as described in [20] to test word-oriented memories by including background data sequences (BDS). The March-LR algorithm used to test the single-port RAMs in synchronous mode is given in Table 2. The March-Y algorithm without background data sequences is used to test the single-port RAMs in asynchronous mode and is also given in Table 2 where r0/w0 indicates reading or writing all 0s in the word.

All of the RAMs in the FPGA are tested in parallel using the three test configurations summarized in Table 2. The total fault coverage obtained is 99.9%. The dual-port RAMs are not tested in asynchronous mode of operation because the port which is dedicated for reading is always asynchronous and is tested during synchronous mode. In all three test configurations, a single TPG is used to produce the read and write addresses as well as the write data for the RAMs. A comparison-based ORA similar to the one used for logic BIST is used. In the dual-port RAM test, the ORAs compare output data

from two adjacent RAMs as illustrated in Figure 7a. In the single-port RAM tests, the ORAs compare output data from each RAM with expected results generated by the TPG as illustrated in Figure 7b. All ORAs are connected in the form of a shift register to retrieve the results from the ORAs at the completion of the BIST sequence. Diagnostic resolution with this approach can identify the faulty RAM as well as the faulty data bit associated with that RAM.

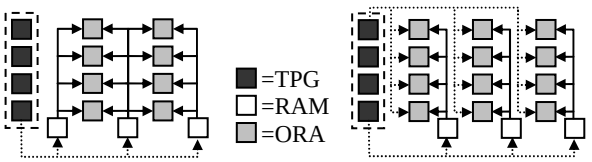

Figure 7. RAM BIST architectures

Unlike logic BIST, VHDL was initially used to implement the RAM BIST including the TPG and ORAs. A parameterized VHDL model is developed to test RAMs associated with any array size. This approach gives a degree of freedom in implementing logic at a higher level instead of using the PLBs and dynamic macros of MGL as basic elements. As a result, the parameterized VHDL could be ported to test the RAMs in other FPGAs, such as the block RAMs in Xilinx Virtex I/II and Spartan II/III series FPGAs [15]. However, this VHDL-based approach poses problems during synthesis in the Atmel FPGA. Specifically, the VHDL-based approach requires that the RAMs be interactively placed at desired locations so that the faulty RAMs can be identified directly from diagnostic results. The ORAs must also be manually placed to resolve routing contentions due to the heavy use of routing resources by the RAM BIST. Therefore, the MGL-based approach used for logic BIST is better in this aspect. Using MGL, the regular structure portion of the RAM BIST logic consisting of the RAMs and the ORAs can be exploited to eliminate contention for routing resources and also to place the RAMs and ORAs at desired locations for more efficient diagnosis based on failing BIST results.

Benefits of MGL over VHDL were more evident when testing the RAM in single-port mode. A tri-state buffer is required to read and write data to and from RAMs in single-port mode. Synthesis of the VHDL model used a single PLB for implementing a tri-state buffer and two PLBs for the ORA requiring twelve out of sixteen PLBs in every 4×4 array and leaving only four PLBs in every array for the TPG. Alternatively, MGL was used to place and route the ORA along with the tri-state buffer in eight PLBs. The TPG was then modeled using VHDL with automatic placement and routing of the TPG PLBs performed by the synthesis tool. In the single-port RAM BIST, the TPG also generates the expected data. Therefore, instead of comparing data from adjacent RAMs, the ORA compares data from a RAM with the expected data from the TPG as shown in Figure 7b. Therefore, the use of MGL not only eliminated manual placement of ORAs and RAMs but also helped in using both logic and routing resources more efficiently. Furthermore, the same MGL program is used to generate all single-port RAM BIST algorithms with the simple insertion of the TPG VHDL model associated with the desired test algorithm.

The March-LR algorithm with background data sequences is used the test the RAMs in single-port, synchronous mode of operation and requires 153 PLBs for the TPG implementation. The March-Y algorithm (without background data sequences) is used for testing the single-port RAMs in the asynchronous mode of operation and requires only 18 PLBs for the TPG. The smallest AT40K device (AT40K05) has only 16×16 PLBs and, as a result, cannot fit the March-LR TPG when all RAMs are being tested in a single BIST configuration. One alternative is to apply the single-port RAM synchronous mode test configuration twice, testing half of the RAMs in each configuration. Another alternative is to use a TPG implementing a March-Y algorithm with background data sequences, which uses only 67 PLBs, to test all RAMs concurrently in single-port synchronous mode of operation in the smallest Atmel device. In this March-Y approach, some pattern sensitivity and intra-word coupling fault detection capability is traded off for reduced testing time.

5. ROUTING BIST

The size of the ORA that can be implemented in the Atmel AT40K PLB is limited to 3-inputs to allow for a feedback to latch any incorrect responses to the test patterns applied. As a result, the comparison-based ORA used for routing BIST in [4] and [9] can only be constructed to compare two wires in an Atmel AT40K PLB such that each set of WUTs in the comparison-based BIST approach consists of a single wire. Therefore, for fine-grain FPGA architectures like the Atmel AT40K series FPGA [18], we find that a modified version of the parity-based BIST approach originally proposed in [6] facilitates concurrent testing of more routing resources than the comparison-based routing BIST approach. The primary modification is the use of a small counter (also more compatible with fine-grain FPGA architectures) and using the parity bit as an additional test pattern.

Since the PLB can implement only one cell of a counter or LFSR, a TPG consists of two PLBs used to implement a 2-bit up-counter with an additional PLB to generate even parity over the 2-bit count value. Another TPG consists of a 2-bit down counter with an odd parity generator. In both cases, the generated parity bit is used as a third test pattern sent over the set of WUTs since between any two bits in the 3-bit test pattern sequence

there exist at least two vectors with opposite logic values (0,1) and (1,0). This is necessary for the detection of shorts (bridging faults) in the WUTs. Note that stuck-on and stuck-off PIPs behave as shorts and opens, respectively, in WUTs and are detected as long as the necessary test conditions are set up on wire segments not under test [9]. The ORA then consists of a 3-input exclusive-OR or exclusive-NOR to check for even or odd parity, respectively, on the 2-bit count value plus the parity bit depending on the TPG (up-count with even parity or down-count with odd parity). Any detected errors are latched by the feedback and OR gate of the ORA, as illustrated in Figure 8. In most configurations, a single TPG drives multiple sets of WUTs and ORAs in order to maximize the routing resources under test during any given BIST configuration and, therefore, helping to minimize the number of routing BIST configurations.

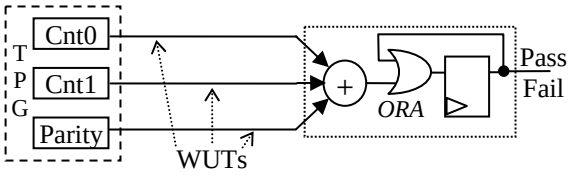

Figure 8. Routing BIST architecture

The local routing resources at the inputs of the Atmel PLB can be considered as multiplexers with up to nine inputs (five horizontal/vertical bus connections and four direct connections). This requires a minimum of nine routing BIST configurations for complete testing. The five vertical bus connections to the input multiplexers are completely tested during logic BIST by cycling through the five vertical bus TPG-to-BUT connections during each of the five logic BIST configurations. The five horizontal bus connections can be tested by rotating two of the logic BIST configurations by 90 degrees, each configuration testing three of the five horizontal bus connections. As in the case of logic BIST, two test sessions consisting of two test configurations each must be applied to test the horizontal bus connections to all PLBs in the array for a total of four test configurations. The direct Y connections to the PLBs are also completely tested by these same test configurations along with the set of logic BIST configurations. The direct X connections must be tested with additional test configurations using the architecture shown in Figure 8 where the WUTs zig-zag through the PLB array between the TPG and ORA. This basic structure is rotated through the four directions for a total of four test configurations. As a result, a total eight test configurations are required, in addition to the logic BIST configurations, to completely test the local routing resources.

In the global routing resources, the vertical and horizontal repeater cells can be considered as miniature versions of the switch boxes in Xilinx FPGAs which have been shown to require a minimum of three routing BIST configurations [6]. However, it took us four configurations for vertical repeaters and another four configurations for horizontal repeaters. Two examples of these repeater tests are illustrated in Figure 9a where alternating up-count/even-parity and down-count/odd-parity TPGs are used to drive alternating sets of WUTs in order to detect stuck-on faults in some of the PIPs in the repeater while other PIPs are being tested for stuck-off faults. As the four repeater BIST configurations are applied, all PIPs in the repeater cell are tested for stuck-on and stuck-off faults.

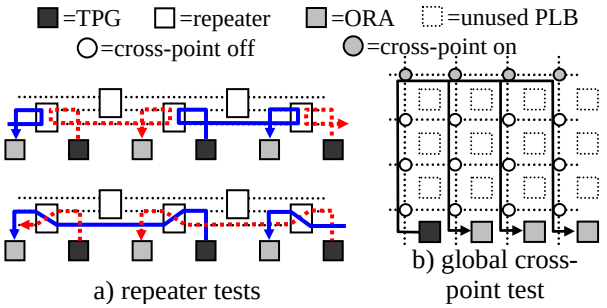

Figure 9. Example repeater BIST configuration

A total of eight configurations are required to test the cross-point PIPs interconnecting a given set of express busses. This includes both stuck-off and stuck-on cross-point PIP faults. An example of the cross-point PIP BIST configurations is shown in Figure 9b. During subsequent BIST configurations, the *on* cross-point PIPs that are would shift down through the rows until all cross-point PIPs are tested.

Therefore, a total of 32 BIST configurations are needed, in addition to the logic BIST configurations, to completely test the routing resources in the core of the FPGA, as summarized in Table 3. The STAR size is given in terms of the size of the PLB array needed to implement the BIST circuitry and, as a result, the diagnostic resolution associated with a failing routing BIST ORA indication. However, the faults can usually be located to an area smaller than the STAR size given. For better diagnostic resolution, additional diagnostic BIST configurations must be developed and applied in order to locate a faulty wire segment or PIP [9]. Note that the routing resources associated with the free RAMs in the array are tested as a part of the RAM BIST since these resources are dedicated for access to/from the RAMs.

Table 3. Routing BIST Configurations

Routing Resource	***# Configs***	***STAR***
Multiplexed PLB inputs	8	4×4
Vertical repeater cells	4	1×16
Horizontal repeater cells	4	16×1
Express bus cross-points	16	8×8

6. SUMMARY AND CONCLUSIONS

We have presented an MGL-based approach for the automatic generation of 45 total BIST configurations that completely test the logic, RAM, and interconnect resources in Atmel AT40K series FPGAs as well as the FPGA core in AT94K series SoCs. The user simply specifies the size of the array and/or the target device to obtain the complete set of BIST configurations for that device. MGL proved to be reasonably effective in the development and automatic generation of BIST configurations despite some minor limitations and bugs we encountered. Table 4 summarizes the number of BIST configurations produced by each MGL program along with the approximate number of non-commented lines of MGL source code (NCL) in each program.

Table 4. MGL code and BIST Configurations

MGL program	***NCL***	***# Configs Generated***
Logic BIST	1700	10 logic + 4 local routing
RAM BIST	120	3 SPRAM tests
	140	1 DPRAM test
Local Routing	1000	4 X direct connections
Repeaters	1300	8 (4 vertical + 4 horizontal)
Express bus cross-points	1360	8
	735	8
Total	6,355	46 configurations

These BIST configurations produce very near 100% fault coverage. In the case of logic BIST, 100% fault coverage could not be obtained due to minor limitations of control provided by MGL that limit the fault coverage to 98%. However, the download bitstream files for the logic BIST configurations can be modified to produce the testing conditions necessary to obtain 100% fault coverage. RAM BIST fault coverage is 99.9% with eight undetected faults residing in the clock and address decoding circuitry; we are currently investigating modifications to the RAM BIST algorithms to detect those faults. Routing BIST fault coverage is also very near 100%. However, at this point we have only completed fault simulations on the local routing resources and in the process of performing fault simulations on the global routing resources to verify this fault coverage.

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

[3] C. Stroud, J. Nall, A. Taylor and L. Charnley, "A System for Automated Generation of Built-In Self-Test Configurations for Field Programmable Gate Arrays," *Proc. Int'l Conf. on Systems Engineering*, pp. 437-443, 2002.

[4] 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.

[5] C. Hamilton, G. Gibson, S. Wijesuriya and C. Stroud, "Enhanced BIST-Based Diagnosis of FPGAs via Boundary Scan Access," *Proc. IEEE VLSI Test Symp.*, pp. 413-418, 1999.

[6] 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.

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

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

[9] 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.

[10] C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-In Self-Test for Programmable Logic Blocks in FPGAs," *Proc. IEEE VLSI Test Symp.*, pp. 387-392, 1996.

[11] M. Abramovici and C. Stroud, "BIST-Based Delay-Fault Testing in FPGAs," *J. Electronic Testing: Theory & Applications.*, Vol. 19, N0. 5, pp. 549-558, 2003.

[12] Atmel, Corp., www.atmel.com/products.

[13] __, *Integrated Development System AT40K Macro Library Version 6.0*, Atmel Corp., Oct. 1998.

[14] __, *Integrated Development System Technical Reference and Release Notes Version 6.0*, Atmel Corp., Oct. 1998.

[15] Xilinx, Inc., www.xilinx.com/products.

[16] Lattice Semiconductor Corp., www.latticesemi.com/products.

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

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

[19] 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.

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