Embedded Processor Based Built-In Self-Test and Diagnosis of FPGA Core in FPSLIC

John Sunwoo (Logic BIST)
Srinivas Garimella (RAM BIST)
Sudheer Vemula (I/O Cell BIST)
Chuck Stroud (Routing BIST)
Jonathan Harris (original Logic and Routing BIST)
Atmel AT94K FPSLIC Architecture

- **Field Programmable Gate Array**
  - up to 48x48 array of Programmable Logic Blocks (PLBs)

- **RAM cores**
  - 32x4 bit RAMs distributed in FPGA
  - Program memory (up to 32 Kbyte)
    - Single port to processor
  - Data RAM (up to 16 Kbyte)
    - Dual port to FPGA & processor

- **8-bit RISC processor core**
  - Various peripherals
  - Processor can write (but not read)
    - FPGA configuration memory
  - Dynamic partial reconfiguration
FPGA Core Architecture

- Arranged in 4x4 arrays of PLBs
- One 32x4-bit RAM per 4x4 array
  - Single- or Dual-Port operation
  - Sync or async operation
- Local routing to adjacent PLBs
  - 4 directs (Y)
  - 4 diagonals (X)
- Global routing – 5 planes having:
  - Two x8 lines/plane - spans 8 PLBs
  - One x4 line/plane - spans 4 PLBs
    - connects to/from PLB
  - Repeaters provide buffering and interconnections between x8 & x4 lines
General FPGA Logic BIST Architecture

- Columns or rows of PLBs configured as:
  - Test Pattern Generators (TPGs)
    - Counter or LFSR: pseudo-exhaustive test
  - Blocks Under Test (BUTs)
    - Configured in all modes of operation
    - 2 test sessions to test all PLBs as BUTs
  - Output Response Analyzers (ORAs)
    - Comparison-based monitoring adjacent BUTs
  - Routing resource usage:
    - Global routing for TPG signals to BUTs
    - Local routing for BUT outputs to ORAs

- Architecture forms basis for diagnostic procedure
  - MULTICELLO: Multiple Faulty Cell Locator

Test Session 1
Test Session 2

- Yellow = TPG
- Orange = BUT
- Blue = ORA
FPSLIC PLB Architecture

- Relatively small
  - 4 data inputs
    - Plus Clock and Set/Reset
  - Two 3-input LUTs
  - One D flip-flop
    - Async Set/Reset

- Issues for BIST
  - ORAs need 5 LUT inputs
    - Compare 2 outputs
    - Latch any mismatches
    - feedback to LUTs
    - Shift out results
    - 2 inputs to LUTs
  - Dynamic partial reconfiguration to create shift register
Logic BIST

- Column-based BIST architecture
  - Logic blocks configured as TPGs, ORAs, and BUTs
- BUTs tested in all modes
  - 4 test configurations
- BIST architecture flipped vertically for 2nd test session
- ORA can only observe X from one BUT & Y from other BUT
  - 2 routing schemes needed to observe both X and Y outputs of all BUTs
Pathological Case for Fault Detection

To escape detection all of the following must be true:

- X & Y have same position in both TPGs in column 1
- W & Z have same position in both TPGs in column 8
- X & Y have equivalent faults
- W & Z have equivalent faults
- X & Y cause TPGs in column 1 to skip patterns that detect W & Z
- W & Z cause TPGs in column 8 to skip patterns that detect X & Y

Rotating architecture by 90° will detect these faults!
Conventional FPGA BIST

Test Phase 1

Test Phase 2

- External downloads consume major portion of testing time
  - More test phases means more external storage is required
  - To find faulty blocks, ORA results are retrieved after each test phase
    - Download of the next test phase causes FPGA core to be reset

- External controller is needed
  - BIST clocks from external pin
  - ORA results are retrieved outside through external pin
  - Embedded processor is not a main BIST component
Embedded Processor Based BIST

- Use embedded processor core to
  - Reconfigure FPGA for BIST
  - Execute BIST and retrieve BIST results
  - Perform diagnostic procedure
  - Perform fault injection emulation
    - Methodical verification of BIST configurations
  - Processor must access configuration memory
    - Read access desirable for read-modify-write

- Implemented in Atmel AT94K
  - 8-bit AVR microcontroller
  - Configuration memory write access only

- Currently implementing in Virtex-4&5 for NSA
  - PowerPC (hard), MicroBlaze & PicoBlaze (soft)
  - Configuration memory write and read access
AVR-generated FPGA BIST

- **Test Phase 1**
  - No download to FPGA core
  - Single AVR program contains algorithmic reconfiguration routines for configuring FPGA core for each test phase

- **Test Phase 2**
  - TPG and ORA configurations are reusable for next phase
    - Until architecture flips to next test session
    - AVR reconfigures BUTs for each test phase
  - ORA results can be retrieved after multiple phases
    - FF values in ORAs remain throughout subsequent phases
Dynamic Reconfiguration Sequence

- Clear FPGA core
  - PLBs, repeaters, clk, FFs, freeRAMs, IOBs
  - No need for chip reset
- Configure ORAs
  - Comparison-based ORA
  - Reset ORA FFs
  - Routing scheme 1 or 2
- Configure BUTs
  - Change BUT configurations
  - Write FFs from AVR (FF set/reset test)
    - Cannot be tested by external download
- Configure TPGs
  - Two 5-bit counters
  - Route TPG signals to BUTs
    - Need care when configuring repeaters
  - Reset TPG FFs

ELECTRICAL AND COMPUTER ENGINEERING
AUBURN UNIVERSITY
SAMUEL GINN COLLEGE OF ENGINEERING

1/19/09 Embedded Systems Lecture
Dynamic Reconfiguration Sequence

- Route BIST clock
  - BIST clock driven by AVR
    - Configure AVR-FPGA interfaces that cannot be called from MGL
    - Route from FPGAIOWE to GCK4
    - Route clock signal to all BUTs, ORAs and TPGs
  - Run BIST clock

- Reconfigure ORAs to shift reg
  - Dynamic partial reconfiguration of ORAs to shift register without destroying ORA FF contents

- Route scan out path
  - Shift reg output to 1 of 8-bit bus between AVR and FPGA
    - AVR Data In (ADIN)
  - Retrieve ORA results
    - On-chip diagnosis by AVR
## Configuration Routines for logic BIST

<table>
<thead>
<tr>
<th>BIST Reconfiguration Subroutines</th>
<th>Program Memory Size (Bytes)</th>
<th>Number of Lines of C Code (Approx.)</th>
<th>Processor Execution Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear FPGA</td>
<td>492</td>
<td>150</td>
<td>K10: 59,664</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instantiate BUT</td>
<td>834</td>
<td>300</td>
<td>K40: 215,128</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instantiate ORA</td>
<td>220</td>
<td>70</td>
<td>K10: 25,829</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instantiate TPG</td>
<td>1,486</td>
<td>600</td>
<td>K40: 100,360</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Route BIST clock</td>
<td>234</td>
<td>40</td>
<td>K10: 14,844</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORA/shift register</td>
<td>282</td>
<td>80</td>
<td>K40: 60,686</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate clocks</td>
<td>32</td>
<td>6</td>
<td>K10: 14,866</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Route scan out</td>
<td>402</td>
<td>45</td>
<td>K40: 4,911</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retrieve results</td>
<td>306</td>
<td>35</td>
<td>K10: 24,791</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>388</td>
<td>2,659</td>
<td>K40: 24,791</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>4,676</td>
<td>4,000</td>
<td>K10: 157,957</td>
</tr>
</tbody>
</table>

- Largest subroutine (program memory & lines of code) is for configuring TPGs
  - Due to irregular structures
- Clearing FPGA and configuring BUTs/ORAs need less program memory size
  - Due to regular structure of BIST architecture
On-Chip BIST and Diagnosis

- Atmel SoCs contain:
  - Program & Data RAMs
  - Processor core
  - FPGA core

- Use processor to:
  - Configure FPGA for BIST
  - Run BIST
  - Get BIST results
  - Perform diagnosis
    - Reduces test and diagnosis time by a factor of 36.9
    - Store only one BIST and diagnostic program on-chip
Diagnosis Based on BIST Results

Step 1: Record ORA results
Step 2: Mark BUTs good between consecutive ORAs with 0s
Step 3: Mark BUTs good for every two adjacent 0s followed by empty cell
Step 4: Mark BUTs bad for every consecutive 0 and 1 followed by empty cell
Step 5: Inconsistencies mean fault in ORA or in routing resources from BUTs
Step 6: Unique diagnosis if all BUTs marked faulty or fault-free

Note:
Row 4: BUTs 1 & 2 have equivalent faults
Row 5: BUTs 1 & 2 may be fault-free or faulty
Row 6: BUT 6 may be faulty or fault-free

Ambiguities:
Row 3: BUT 5 and/or BUT 6 is faulty
Row 4: BUT 6 may be faulty or fault-free
Row 5: BUT 6 may be faulty or fault-free

=Core Under Test
=TPG PLBs
=ORA PLBs
=Processor Core

Step 2: Mark BUTs good between consecutive ORAs with 0s
Step 3: Mark BUTs good for every two adjacent 0s followed by empty cell
Step 4: Mark BUTs bad for every consecutive 0 and 1 followed by empty cell
Implementation of Diagnostics

<table>
<thead>
<tr>
<th>Logic Resource</th>
<th>Array Size</th>
<th>Execution Clock Cycles</th>
<th>Program Memory (bytes)</th>
<th>Data Memory (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMs</td>
<td>6×6</td>
<td>2,400</td>
<td>1,130</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>12×12</td>
<td>9,700</td>
<td>1,130</td>
<td>73</td>
</tr>
<tr>
<td>PLBs</td>
<td>24×24</td>
<td>32,000</td>
<td>1,330</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>48×48</td>
<td>110,000</td>
<td>1,330</td>
<td>720</td>
</tr>
</tbody>
</table>

- Diagnostic program is independent of array size
  - Extra 200 bytes for ORA/BUT translation
- Data memory requirement and execution time are a function of array size
  - $O(N^2)$ where $N$ is array size in one dimension
## BIST and Diagnostic Programs

<table>
<thead>
<tr>
<th>Test Function</th>
<th>Execution Clock Cycles</th>
<th>Program Memory (bytes)</th>
<th>Data Memory (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM BIST</td>
<td>398,100</td>
<td>1,860</td>
<td>73</td>
</tr>
<tr>
<td>Logic BIST</td>
<td>998,560</td>
<td>3,380</td>
<td>138</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>110,000</td>
<td>1,330</td>
<td>720</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1,506,660</strong></td>
<td><strong>6,570</strong></td>
<td><strong>930</strong></td>
</tr>
</tbody>
</table>

- BIST and diagnostic programs easily fit in AT94K™
  - ≈ 20% of 32Kbyte Program Memory
  - ≈ 20% of 4Kbyte Data RAM
- Same diagnostic program used for PLBs & RAMs
  - PLB diagnosis dictates memory requirements
Routing BIST

- Modified parity-based routing BIST
  - 3-bit TPG = 2-bit counter + parity
    - Drive all five x4 global buses
  - Detects all
    - stuck-at faults, bridging faults and opens in wire segments
    - stuck-on/off faults in repeaters

<table>
<thead>
<tr>
<th>Count-up</th>
<th>Count-down</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>C1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Configuration Routines for Routing BIST

<table>
<thead>
<tr>
<th>BIST Subroutine</th>
<th># Configs</th>
<th>Memory (bytes)</th>
<th>Lines of C Code</th>
<th>Processor Cycles K40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-points</td>
<td>16</td>
<td>3,442</td>
<td>519</td>
<td>1,906,745</td>
</tr>
<tr>
<td>Repeater s</td>
<td>40</td>
<td>4,412</td>
<td>636</td>
<td>6,671,973</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>56</strong></td>
<td><strong>7,854</strong></td>
<td><strong>1,155</strong></td>
<td><strong>8,578,718</strong></td>
</tr>
</tbody>
</table>

- Routing BIST generation and reconfiguration program are similar to logic BIST shown previously.
- Single AVR program generates all 56 routing BIST configurations.

1/19/09 Embedded Systems Lecture
### Results: Speed-up

<table>
<thead>
<tr>
<th>Resource</th>
<th>Function</th>
<th>Download</th>
<th>Processor</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic BIST</td>
<td>Download</td>
<td>7.680 sec</td>
<td>0.101 sec</td>
<td>76.0</td>
</tr>
<tr>
<td></td>
<td>Execution</td>
<td>0.016 sec</td>
<td>0.085 sec</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>Total time</td>
<td>7.696 sec</td>
<td>0.186 sec</td>
<td>41.4</td>
</tr>
<tr>
<td>Routing BIST</td>
<td>Download</td>
<td>20.064 sec</td>
<td>0.110 sec</td>
<td>182.4</td>
</tr>
<tr>
<td></td>
<td>Execution</td>
<td>0.026 sec</td>
<td>0.343 sec</td>
<td>0.075</td>
</tr>
<tr>
<td></td>
<td>Total time</td>
<td>20.090 sec</td>
<td>0.453 sec</td>
<td>44.3</td>
</tr>
<tr>
<td><strong>Total Test Time</strong></td>
<td><strong>27.786 sec</strong></td>
<td><strong>0.639 sec</strong></td>
<td></td>
<td><strong>43.5</strong></td>
</tr>
</tbody>
</table>

- Configuration download time dominates total test time for conventional download approach.
- Longer execution time on processor-generated BIST approach than download approach.
Results: Memory Reduction

<table>
<thead>
<tr>
<th>Resource Tested</th>
<th>Download</th>
<th>Processor</th>
<th>Memory Reduction Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average File Size</td>
<td># Files</td>
<td>File Size</td>
</tr>
<tr>
<td>Logic</td>
<td>60 Kbyte</td>
<td>16</td>
<td>12 Kbyte</td>
</tr>
<tr>
<td>Routing</td>
<td>57 Kbyte</td>
<td>44</td>
<td>14 Kbyte</td>
</tr>
<tr>
<td>Combined</td>
<td>58 Kbyte</td>
<td>60</td>
<td>22 Kbyte</td>
</tr>
</tbody>
</table>

- Download approach needs 60 configuration downloads to the chip
- Processor-generated BIST approach requires only one (relatively small) download to the program memory of AVR processor
Physical Fault Injection

- Faulty FPGA are difficult to find
  - 1 ORCA with faulty PLB & 2 ORCAs with faulty routing
- Physical fault insertion
  - Etch package and “zap” with laser
- Fault Injection Emulation
  - Modify configuration bits
    - Prior to download, or
    - Read-modify-write of configuration memory
      - External or internal embedded processor
  - Fault Emulator can create single & multiple faults in:
    - PLBs: LUTs, flip-flops, etc.
    - Interconnect: PIPs stuck-on & stuck-off
Automated Fault Injection Analysis

- Fault injection emulation used for debugging, analysis & verification of BIST configurations & diagnosis
  - Embedded processor based fault injection gives faster and more thorough analysis

![Graph showing fault detection rates for different configurations](image)

- Reading modify write of configuration bits makes analysis more efficient!