Built-In Self-Test for System-on-Chip: A Case Study

Charles Stroud, Srinivas Garimella, John Sunwoo, Jonathan Harris, and Jared Phillips

Dept. of Electrical and Computer Engineering
Auburn University
Purpose

• Investigate SoC BIST approach using the programmable logic core in a commercial SoC
  – Test FPGA core first
  – Then use FPGA to test other cores

• Results using Atmel AT94K series SoC:
  – FPGA provides limited testing of other cores
  – Processor core provides more effective testing of cores including FPGA core
  – Processor core is primary testing resource instead of FPGA
Outline

- Overview of FPGAs
- History of BIST for FPGAs
- Intended SoC BIST approach
- AT94K series SoC architecture
- Actual SoC BIST approach
  - FPGA
  - RAM cores
  - Processor
- Experimental Results
- Conclusion & Demo
Field Programmable Gate Arrays

- Configuration Memory
- Programmable Logic Blocks (PLBs)
- Programmable Input/Output Cells
- Programmable Interconnect

Typical Complexity = 5M - 100M transistors
Basic FPGA Operation

Load Configuration Memory

- Defines system function
  - Input/Output Cells
  - Logic in PLBs
  - Connections between PLBs & I/O cells

Changing configuration memory => changes system function

- Can change at anytime
  - Even while system function operating
  - Run-time reconfiguration
Programmable Logic Blocks

- PLBs can perform any digital logic function
  - Look-Up Tables (LUTs)
    - Combinational logic
    - Memory (RAM)
  - Flip-flops
    - Sequential logic
  - Special logic
    - Add, subtract, multiply
    - Count up and/or down
    - Dual port RAM
- Must be tested in all modes of operation
- #PLBs/FPGA: 100 to 50,000
Programmable Interconnect

Wire segments & Programmable Interconnect Points (PIPs)
- Cross-point PIPs – connect/disconnect wire segments
  - To turn corners
- Break-point PIPs – connect/disconnect wire segments
  - To make long or short signal routes
- Multiplexer (MUX) PIPs select 1 of many wires for output
  - Used at PLB inputs
  - Primary interconnect media for new FPGAs
FPGA BISTory for PLBs

- Basic idea: program FPGA to test itself (1985)
- 1st off-line BIST (1993 – 1st BIST for FPGA PLBs)
  - Schematic entry & routing problems
- Iterative Logic Array BIST (1995)
  - 3 test sessions & sequential logic test problems
- Hybrid BIST (1996)
  - 2 test sessions, automatic generation & diagnosis
- Boundary scan-based hybrid BIST (1998)
  - Better diagnostic resolution & accuracy
  - Off-line diagnostic procedure
    - Identifies any combination of multiple faulty PLBs
- 1st on-line BIST & diagnosis (1999)
- BIST for SoC (2003)
FPGA BISTory for Interconnect

- **1st off-line BIST** (1997 – 1st BIST for FPGA routing)
  - Tested whole chip => poor diagnostic resolution
- **1st on-line BIST** (1999)
  - Used small Self-Test AREas (STARs)
- **STAR-based off-line BIST** (2001)
  - Better diagnostic resolution & accuracy
- **1st BIST-based adaptive diagnosis** (2001)
  - Good diagnostic resolution
    - Identifies any faulty wire segments or switches
- **BIST for SoC** (2003)
Logic BIST for ORCA 2C FPGA

- 9 test phases per test session
  - Each test phase performs pseudo-exhaustive testing
  - 4 test LUTs
  - 5 test FFs
- Typical usage
  - PLBs: 90%-95%
  - Routing: 60%
- Test requirements
  - Memory: 2 Mbytes
  - Test time: 260 msec
  - ORCA 2C15
    - 20x20 array of PLBs
BIST for Programmable Logic Blocks

Program rows (or columns) of PLBs as:
Test Pattern Generators (TPGs)
Output Response Analyzers (ORAs)
Blocks Under Test (BUTs)

Test BUTs in all modes of operation
Reverse rolls of PLBs in 2\textsuperscript{nd} test session
**Diagnostic Procedure**

**Step 1:** Record ORA results

**Step 2:** Mark known good BUTs

2 ORAs = 0 => BUT between them is good

**Step 3:** Mark implied good BUTs

BUT=ORA=0 => next BUT is good

**Step 4:** Mark known faulty BUTs

BUT=0, ORA=1 => next BUT is faulty

**Step 5:** Look for ORA inconsistencies

BUT=0, ORA=1, BUT=0

=> implies ORA or interconnect faults

**Step 6:** If every PLB has been identified as fault-free or faulty, the group of faulty PLBs has been uniquely diagnosed

=> otherwise mark as unknown

<table>
<thead>
<tr>
<th>Test #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>O2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>O4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>O6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>O8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>O10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>O12</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B13</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>O14</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>O16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>O18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
BIST for Programmable Interconnect

- Program PLBs as
  - Test Pattern Generators
  - Output Response Analyzers
- Program groups of wires under test
  - Wire segments
  - Programmable Interconnect Points (PIPs)
- Fill FPGA with these BIST circuits
  - Tests run concurrently
- Two types of routing BIST
  - Parity-based (Sun, et. al., ITC’00)
  - Comparison-based (Stroud, et. al., ITC’98)
FPGA Based SoC BIST

- Originally proposed at DAC’02
  - in “Using Embedded FPGAs for SoC Yield Improvement”
  - by Abramovici, Stroud & Emmert
- Apply BIST for FPGAs to FPGA core(s)
  - BIST for Programmable Logic Blocks (PLBs)
  - BIST for programmable interconnect
  - Bypass defects in FPGA core(s) to improve yield
- Program FPGA core(s) to test other cores
  - No area or performance penalties for DFT
  - Low-cost testers for SoCs
Atmel AT94K Series SoC Architecture

- Three types of embedded cores
  - FPGA (40x40 array of PLBs)
  - SRAM cores
    - 32x4 bit RAMs distributed in FPGA
    - Program memory
      - Single port to processor
    - Data RAM
      - Dual port to FPGA & processor
  - 8-bit RISC microprocessor
    - With various peripherals
    - Processor can write to FPGA configuration memory
      - Partial dynamic reconfiguration
FPGA and Distributed RAM Cores

• Arranged in 4x4 arrays of PLBs
• 1 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 connections between x8 and x4 lines
Applying DAC’02 SoC BIST Scheme?

- FPGA core can test itself and:
  - 32x4-bit RAMs distributed in FPGA core
  - 1 port of Data RAM

- Processor core can test itself and:
  - Peripherals
  - Program memory
  - Both ports of Data RAM (with FPGA assistance)
  - With dynamic partial reconfiguration of FPGA:
    - BIST for FPGA core
    - RAMs distributed in FPGA core (with FPGA assistance)

**Conclusion:** Processor core is primary test resource
- Insufficient FPGA core access to other cores
PLB Architecture

- **Very small**
  - 4 data inputs
    - 1 clock & 1 set/reset
  - Two 3-input LUTs
  - One D flip-flop
    - Async set/reset
  - Bank clock & set/reset
    - Columns of 4 PLBs

- **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 make shift register
Bank Clocking & Reset Scheme

- Banks of 4 PLBs in column
  - Clock
    - From column clock or x8 line
      - Column clock from 8 global clock lines
    - Rising (non-inverted) or falling (inverted) edge-triggered
  - Set/Reset
    - From global reset or x8 line
    - Active high (non-inverted) or active low (inverted)
    - Flip-flops can be individually set or reset
FPGA Logic BIST Architecture

- Column-based BIST architecture
  - Due to bank clocking & set/reset
- Logic blocks tested in all modes of operation
  - 4 configurations required
- BIST architecture flipped vertically for 2nd test session
  - All PLBs tested in 2 test sessions
- ORA can observe X from one BUT and Y from another BUT
  - 2 routing schemes to observe both X and Y outputs of BUTs
Fault Simulation Model

• Accurate gate level model for PLB & local routing
  – Based on reverse engineering (by us)
  – Based on NDA (by Atmel)

• Uses configuration bits directly from download file
  – No one has been this accurate before
    • Other than FPGA manufacturers

• Used to evaluate BIST configuration fault coverage
  – Logic BIST
  – Routing BIST
    • Can do MUX-based routing
      – Local routing & repeaters
    • Cannot yet do transmission gate-based routing
      – Bi-directional cross-point PIPs
Logic BIST Configurations

# BIST configurations per test session

- **3 is absolute minimum**
  - 100% fault coverage
- **4 with bit manipulation**
  - 99.7% fault coverage
  - 1 potential detected fault
  - Post process bit files, or
  - AVR partial reconfiguration
- **5 with MGL**
  - 98.8% fault coverage
  - 5 potential detected faults
FPGA Logic BIST Observability

**Low fault coverage for edge PLBs**

- Run Logic BIST phases twice
  - Swap X and Y BUT-to-ORA connections along edges
  - Let AVR reverse connections during BIST
- Rotate Logic BIST sessions
  - Allows additional local routing testing
FPGA Logic BIST Diagnosis

• ORA observes X from one BUT and Y from another
  – zigzag routing schemes to observe both X & Y outputs
• Diagnostic procedure works on column or row based
  – Assumes single output observed by 2 ORAs
• Solution:
  – Translate ORAs to columns before diagnosis
  – Translate ORAs & BUTs back to original positions after diagnosis
  – ORA inconsistency indicates difference between X and Y
FPGA Routing BIST Architecture

- Comparison-based BIST not good for small PLBs
  - Only compares 2 wires in a 1 PLB ORA
- Used modified parity-based approach
  - 2-bit up-counter w/ even parity, and
  - 2-bit down-counter w/ odd parity
    - Gives opposite logic values for
      - Stuck-on PIPs & bridging faults
    - Parity used as test pattern
      - 3 wires under test in 1 PLB ORA
      - Good for small PLB-based FPGAs
FPGA Routing BIST Example

- Global wire segments under test: shorts & opens
- Global cross-point PIPs: stuck-on & stuck-off
- PIPs & wires shift down each configuration
- 16 configurations: 8 for each 8x8 array of x8 lines

Stuck-Off Test
Stuck-On Test
Stuck-Off Test

Opposite logic values from adjacent BIST circuit
Opposite logic values from adjacent BIST circuit
Routing BIST

Repeater tests

• 3 sets of tests
  – Loopback
  – Criss-cross
  – Straight through

• 4 configurations each
  – Vertical E & A sets
  – Horizontal E & A sets

• Applied second time to reverse direction

Swap TPGs/ORAs for directional test
Small RAM Core BIST Architecture

- **Dual-Port RAM mode**
  - Synchronous
  - Custom March algorithm
    - Not a true DP RAM
  - Outputs compared to other RAMs
- **Single-Port RAM mode**
  - Synchronous
    - March LR w/ BDS
  - Asynchronous
    - March Y w/o BDS
  - Outputs compared to expected results from TPG
ORA Implementations

- Logic BIST ORA
  - Cannot fit ORA and shift register in 1 PLB
  - Partial reconfiguration to create shift register after BIST
- RAM BIST ORA
  - 2 PLBs
    - ORA in 1 PLB
    - Shift register in 1 PLB
  - No reconfiguration needed
Small RAM Core BIST Diagnosis

- **Dual-Port RAM mode**
  - Outputs compared to other RAMs
  - Same diagnostic program as PLBs
    - No translation required

- **Single-Port RAM mode**
  - Outputs compared to expected results from TPG
  - Diagnosis very simple

- **Diagnostic resolution to faulty RAM and faulty bit(s)**
  - Requires more ORAs but reduces ambiguities
## FPGA RAM BIST Configurations

<table>
<thead>
<tr>
<th>RAM/Mode</th>
<th>Test algorithm</th>
<th>F_{C_{\text{IND}}}</th>
<th>F_{C_{\text{CUM}}}</th>
<th>TPG PLBs</th>
<th>ORA PLBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync Dual-Port</td>
<td>DPR test</td>
<td>75.4%</td>
<td>75.4%</td>
<td>66</td>
<td>8N^2 - 16N</td>
</tr>
<tr>
<td>Sync Single-Port</td>
<td>March-LR w/BDS</td>
<td>81.8%</td>
<td>99.8%</td>
<td>123</td>
<td>8N^2</td>
</tr>
<tr>
<td>Async Single-Port</td>
<td>March-Y w/o BDS</td>
<td>75.6%</td>
<td>100%</td>
<td>18</td>
<td>8N^2</td>
</tr>
<tr>
<td>Data RAM Single-Port</td>
<td>March-LR w/BDS</td>
<td>-</td>
<td>-</td>
<td>209</td>
<td>16</td>
</tr>
</tbody>
</table>

\[ N = \text{number of RAMs in one dimension of array} \]

- ORAs require many PLBs but structure is very regular
- PLB-based TPG is very irregular structure
- Fewer TPG PLBs needed when TPG performed by processor
<table>
<thead>
<tr>
<th>Function Tested</th>
<th># Configs</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM cores</td>
<td>3</td>
<td>distributed RAM in FPGA</td>
</tr>
<tr>
<td>PLBs</td>
<td>16</td>
<td>Data RAM – single port only</td>
</tr>
<tr>
<td>FPGA Routing</td>
<td>16</td>
<td>also tests most local routing</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>x4 &amp; x8 lines, cross-point PIPs</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>repeaters</td>
</tr>
<tr>
<td>Total</td>
<td>68</td>
<td>local routing not tested by LB</td>
</tr>
</tbody>
</table>
Processor RAM BIST Architecture

- **Small RAM cores**
  - Processor: TPG & test control
  - FPGA: ORAs

- **Data RAM (16Kbyte dual-port)**
  - March-LR w/BDS both ports
  - March s2pf and March d2pf
  - Assistance from FPGA resources

- **Program memory**
  - March-LR w/BDS
  - Must move BIST program to complete program memory test
FPGA Logical Memory Map

Tag defines page being written
X,Y define PLB array location
Z defines which byte at a given X,Y location is written

4 Dimensional Memory Map

Core Cells
Vertical Repeaters
Horizontal Repeaters
North/South I/O
And so on .......

TAG Addressed PAGES

PLB 0,0
PLB 0,1
PLB 1,0
PLB 1,1
Reconfiguration from AVR

Assembly language programming example:

EXT_INTERRUPT_FLAG:
```assembly
ldi rTemp, (Column# - 1) ; PLB Horizontal Coordinate
out FPGAX, rTemp
ldi rTemp, (Row# - 1) ; PLB Vertical Coordinate
out FPGAY, rTemp
ldi rTemp, 0bttttcccc ; Tag and Byte Coordinate
out FPGAZ, rTemp
ldi rTemp, 0bd/ddddddd ; New Byte Contents
out FPGAD, rTemp
```

*Write to FPGAD register writes the configuration memory byte*

C programming:

- Less efficient in Program Memory storage requirements and execution clock cycles
  - But only about 6% less efficient
- Faster and less error-prone development
Processor BIST Reconfiguration

- Generate BIST configurations from processor
  - Download compiled program(s)
  - Processor core does the rest
    - Configures FPGA
    - Runs BIST
    - Retrieves ORA results
    - Reconfigure FPGA for next BIST
- Need very regular BIST structures for efficient algorithmic generation of configuration
  - Reduces reconfiguration time
  - Reduces size of programs in memory
    - Reduces download time
    - Faster test time
Other advantages:

• **Complete control over FPGA configuration bits**
  - Higher fault coverage

• **Can do additional testing not possible otherwise**
  - Enable RAM operation during logic BIST to test Tgates stuck-on at outputs of RAM
  - Test initial write data in flip-flops and clearing of RAM data

• **No external test equipment required**

• **Faster BIST development time**

• **Problem:** Debugging BIST configurations
### Experimental Results

**Example:** Improvements in logic BIST test time and memory storage requirements

<table>
<thead>
<tr>
<th>Logic BIST Approach</th>
<th>Config File Size</th>
<th>Total Test Time</th>
<th>Speed-up</th>
<th>Memory Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 configuration</td>
<td>44 Kbytes x 16 files</td>
<td>5.6 sec</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 test session</td>
<td>45.5 Kbytes x 4 files</td>
<td>379 msec</td>
<td>3.7</td>
<td>3.9</td>
</tr>
<tr>
<td>4 test sessions</td>
<td>52.4 Kbytes x 1 file</td>
<td>447 msec</td>
<td>12.6</td>
<td>13.5</td>
</tr>
<tr>
<td>Processor only</td>
<td>9.4 Kbytes x 1 file</td>
<td>162 msec</td>
<td>34.9</td>
<td>75</td>
</tr>
</tbody>
</table>
### AVR RAM BIST reconfiguration

<table>
<thead>
<tr>
<th>RAM test algorithm</th>
<th>Configuration bytes</th>
<th>Processor execution cycles</th>
<th>Test time (msec)</th>
<th>NCL</th>
<th>Program memory bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual-Port</td>
<td>60,651</td>
<td>72,264</td>
<td>63.5</td>
<td>144</td>
<td>664</td>
</tr>
<tr>
<td>March-LR</td>
<td>59,661</td>
<td>76,355</td>
<td>62.7</td>
<td>196</td>
<td>1,192</td>
</tr>
<tr>
<td>March-Y</td>
<td>58,815</td>
<td>49,400</td>
<td>60.8</td>
<td>138</td>
<td>646</td>
</tr>
<tr>
<td>Combined</td>
<td>65,983</td>
<td>398,091</td>
<td>81.9</td>
<td>383</td>
<td>1,860</td>
</tr>
</tbody>
</table>

TPG function performed by AVR:
- makes RAM BIST structure in PLBs very regular
- makes it faster and easier to reconfigure by AVR
- requires smaller Program Memory storage
# SoC BIST Summary

<table>
<thead>
<tr>
<th>Function</th>
<th># Tests</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM cores</td>
<td>3</td>
<td>distributed RAMs in FPGA</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Data RAM</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Program Memory</td>
</tr>
<tr>
<td>PLBs</td>
<td>16</td>
<td>also tests most local routing</td>
</tr>
<tr>
<td>FPGA Routing</td>
<td>16</td>
<td>global lines and cross-point PIPs</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>repeaters</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>local routing not tested by LB</td>
</tr>
<tr>
<td>Total</td>
<td>68</td>
<td>excludes processor &amp; peripherals</td>
</tr>
</tbody>
</table>

- 68 downloads to FPGA
  - FPGA cannot test Data and Program memories
- Versus 1 download to processor
  - Processor can perform all these tests
Fault Injection

• Good fault injection capabilities with
  – Download of individual BIST configurations
  – Knowledge of configuration memory bits
    • Manipulate download bits or
    • Use AVR partial reconfiguration to insert faults

• AVR BIST reconfiguration
  – Download bit manipulation lost during reconfiguration
  – Need AVR subroutine to re-inject faults after any partial reconfiguration to ensure faults are present
Demo Unit

- power supply connector
- parallel port connector
- 244 buffer
- 244 buffer
- 144-pin surface mount AT95K40
- 84-pin socket AT94K10 or AT40K10
- LED display
- clock oscillator
- dip switch
- LED display

Node connections:
- Power supply to Vcc
- Vcc to 244 buffer
- 244 buffer to 144-pin surface mount AT95K40
- 144-pin surface mount AT95K40 to 84-pin socket AT94K10 or AT40K10
- 84-pin socket AT94K10 or AT40K10 to LED display
- LED display to clock oscillator
- Clock oscillator to dip switch
- Dip switch to LED display

Circuit components:
- C_{f1-20}
- C_{p1-4}
- Vcc
Demo Unit

- AT94K evaluation board only works in master serial configuration mode (mode 0)
  - Must first download to *Atmel configurator*
    - Uses same parallel port we use to interface to BIST
    - Very slow!!! & need switch for parallel port
- Demo unit will work in slave serial configuration mode (mode 1)
  - PC has control of download
    - No contention for parallel port
    - No parallel port switch and much faster
- Demo unit will also support synchronous RAM configuration mode (mode 4)
  - One SoC reads/writes configuration memory of other SoC
Conclusion

• DAC’02 BIST approach was good idea but…
  – Must have sufficient interface resources between FPGA and other cores to be tested

• Processor’s ability reconfigure FPGA core
  – Processor is primary testing resource
    • FPGA is only secondary resource
    – Order of magnitude reduction in test time
    – 2 orders of magnitude reduction in memory storage for BIST configurations

• Key to efficient FPGA reconfiguration by processor core is regular BIST structure
Today’s Demo