Built-In Self-Test of DSPs in Virtex-4 FPGAs

(Funded by NSA)

Charles Stroud
Dept. of Electrical & Computer Engineering
Auburn University
Outline of Presentation

- History of DSP Architectures in FPGAs
  - Overview of Virtex-4 DSP
- Prior Testing R&D vs. Our Analysis for:
  - Literature on DSP test not applicable
    - No papers published on DSPs in FPGAs
  - Literature on Multipliers and Adders
    - Application to Virtex-4 DSPs
- BIST for DSPs in Virtex-4
  - Architecture, Operation, and Implementation
  - Timing and Fault Injection Analysis
- Summary and Conclusions
  - Plans for application to Virtex-5
Xilinx FPGA Architectures

- **4000/Spartan**
  - \(N \times N\) array of unit cells
  - Unit cell = CLB + routing
  - Fast carry logic in CLBs for adders

- **Virtex/Spartan-2**
  - \(M \times N\) array of unit cells
  - Carry logic + AND gate for array multipliers
  - 4K block RAMs at edges

- **Virtex-2/Spartan-3**
  - 18K block RAMs in array
  - 18x18-bit multipliers with each RAM
  - “based on modified Booth architecture”

- **Virtex-4/Virtex-5**
  - Added 48-bit DSP cores w/multipliers

- **Altera includes 9x9 multipliers**
  - “based on modified Booth architecture”

C. Stroud 1/08
VLSI D&T Seminar
Virtex-4 DSP Architecture

- 2 DSP slices per tile
  - 16-256 tiles in 1-8 columns
- Each DSP includes:
  - 3-input, 48-bit adder/subtractor
    - P = Z±(X+Y+Cin)
    - Optional accum reg
  - 18x18-bit 2's-comp multiplier (w/o adder)
  - User controlled operational modes
    - For X, Y, & Z MUXs
  - Configuration bits control other MUXs
    - Pipelining registers
    - Accumulator register

Outputs w/ dedicated routing

Inputs for cascading

Outputs w/ dedicated routing

Inputs for cascading

C. Stroud 1/08
VLSI D&T Seminar
Multiplier and Adder Architectures

- Test algorithm depends on architecture
  - But architecture is not specified in data sheets
    - Eliminate sequential logic architectures
    - “Based on modified Booth”
- Adder choices include:
  - Ripple carry
  - Carry select
  - Carry save
  - Carry-look-ahead (CLA)
    - Our assumption based on area/performance analysis
    - But multiple types of CLA

- Multiplier choices include:
  - Array
  - Booth
  - Modified Booth
  - Wallace tree
  - Modified Booth/Wallace tree
    - Our assumption based on area/performance analysis

- Our goal: find/develop architecture independent test algorithm(s)
Array Multiplier Test Algorithm

- Kalyana Kantipudi’s MS thesis
  - 10 vectors give 100% fault coverage for C6288
    - a 16x16-bit array multiplier

- 18x18-bit array multiplier results
  - Only achieved ≈ 95% fault coverage
    - Pattern expansion required for 16x16-bit to 18x18-bit
      - Potential for mistakes if patterns not expanded properly

- Modified Booth multiplier results
  - ≈ 62% with carry-save adder
  - ≈ 37% with CLA

- Conclusion: array multiplier test vectors do not adequately test modified Booth multiplier

Chris Erickson’s Results

Note difference in FC wrt adder implementation
Modified Booth Test Algorithms

Two test algorithms using 8-bit counter (256 vectors)

- “Low Power BIST for Wallace Tree-based Fast Multipliers”
  - Bakalis, Kalligeros, Nikolos, Vergos & Alexiou
  - 5x3 connections with 5 inputs to Booth encoding
    - But which side is Booth encoding?
    - Our approach: run both 5x3 and 3x5 algorithms

- “Effective Built-In Self-Test for Booth Multipliers”
  - Gizopoulos, Paschalis & Zorian
  - 4x4 connections to multiplier inputs
    - Our approach: also include 4x4 if fault coverage improves

8-bit counter
MSB                  LSB
\[ 4 \times 3 \text{ algorithm} \]

\[ 5 \times 5 \text{ algorithm} \]

\[ 3 \times 3 \text{ algorithm} \]

\[ 8 \times 8 \text{ multiplier} \]

\[ n \times n \text{ multiplier} \]

Algorithm used in
Srinivas Garimella’s
MS thesis for
Virtex-2 multipliers
4x4 Booth Multiplier Test Algorithm

- 18x18-bit array multiplier results
  - \(\approx 99.99\%\) (1 undetected fault)

- Booth multiplier results
  - \(\approx 90\%\) with ripple-carry adder
  - \(\approx 90\%\) with carry-save adder
  - \(\approx 70\%\) with CLA

- Conclusion: modified Booth multiplier test vectors do test array multiplier
  - But Modified-Booth/Wallace-Tree appears to be most likely candidate for Virtex-4 DSP multiplier implementation
  - Also for Virtex-5 and Altera
Other Multiplier Results

- 4x4-bit implementations
  - Exhaustive test patterns
    - Undetected faults are undetectable
    - Same as 4x4, 5x3, & 3x5 algorithm for 4x4-bit multiplier
- Simulation results discrepancy for array multiplier
  - 4 undetected faults in 4x4-bit implementation
  - 1 undetected fault in 18x18 multiplier w/ 4x4 algorithm in Chris Erickson’s results

<table>
<thead>
<tr>
<th>Multiplier</th>
<th># faults</th>
<th># detected</th>
<th># undetected</th>
<th>FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>272</td>
<td>268</td>
<td>4</td>
<td>98.5%</td>
</tr>
<tr>
<td>Signed Array</td>
<td>337</td>
<td>320</td>
<td>17</td>
<td>95.0%</td>
</tr>
<tr>
<td>Wallace Tree</td>
<td>283</td>
<td>280</td>
<td>3</td>
<td>98.9%</td>
</tr>
</tbody>
</table>
### 8x8 Modified-Booth/Wallace-Tree

<table>
<thead>
<tr>
<th>Multiplier Version</th>
<th>Test Algorithm</th>
<th># Vectors</th>
<th>Total Faults</th>
<th>Faults Detected</th>
<th>Not Detected</th>
<th>Fault Coverage</th>
<th>Effective FC</th>
</tr>
</thead>
<tbody>
<tr>
<td>No reduction</td>
<td>Exhaustive</td>
<td>65,536</td>
<td>3,402</td>
<td>2,520</td>
<td>882</td>
<td>74.1%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>4x4</td>
<td>256</td>
<td></td>
<td>2,477</td>
<td>925</td>
<td>72.8%</td>
<td>99.0%</td>
</tr>
<tr>
<td>With reduction</td>
<td>Exhaustive</td>
<td>65,536</td>
<td>2,341</td>
<td>2,031</td>
<td>310</td>
<td>86.8%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>4x4</td>
<td>256</td>
<td></td>
<td>2,005</td>
<td>336</td>
<td>85.6%</td>
<td>98.7%</td>
</tr>
<tr>
<td></td>
<td>5x3</td>
<td>256</td>
<td></td>
<td>2,011</td>
<td>330</td>
<td>85.9%</td>
<td>99.0%</td>
</tr>
<tr>
<td></td>
<td>3x5</td>
<td></td>
<td></td>
<td>2,015</td>
<td>326</td>
<td>86.1%</td>
<td>99.2%</td>
</tr>
<tr>
<td></td>
<td>4x4 &amp; 5x3</td>
<td></td>
<td></td>
<td>2,015</td>
<td>326</td>
<td>86.1%</td>
<td>99.2%</td>
</tr>
<tr>
<td></td>
<td>4x4 &amp; 3x5</td>
<td>512</td>
<td></td>
<td>2,019</td>
<td>322</td>
<td>86.2%</td>
<td>99.4%</td>
</tr>
<tr>
<td></td>
<td>5x3 &amp; 3x5</td>
<td></td>
<td></td>
<td>2,029</td>
<td>312</td>
<td>86.7%</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

- **Fault simulation results:**
  - 5x3 plus 3x5 give best fault coverage
  - No additional faults detected with 4x4

*Chitanya Bandi’s Results (note: used ripple carry adder to sum partial products)*
Carry-Look-Ahead Adder

- Recall CLA was more difficult to test
- Basic CLA is 4-bits
  - 4-bit CLAs then combined to form larger adders
  - Ripple CLAs
  - 2 types based on Lookahead Carry Unit (LCU):
    - Ripple LCU
    - Multi-stage LCU

Gi = Ai \cdot Bi
Pi = Ai + Bi

C1 = G0 + P0 \cdot C0
C2 = G1 + G0 \cdot P1 + P1 \cdot P0 \cdot C0
C3 = G2 + G1 \cdot P2 + G0 \cdot P1 + P2 + P2 \cdot P1 \cdot P0 \cdot C0
C4 = G3 + G2 \cdot P3 + G1 \cdot P2 + P3 + G0 \cdot P1 + P2 \cdot P3 + P3 \cdot P2 + P1 \cdot P0 \cdot C0

PG = P0 \cdot P1 \cdot P2 \cdot P3
GG = G3 + G2 \cdot P3 + G1 \cdot P2 + P3 + G0 \cdot P1 + P2 \cdot P3 + P3 \cdot P2 + P1 \cdot P0 \cdot C0
CLA Test Algorithms

- "On the Adders with Minimum Tests"
  - Kajihara and Sasao
    - Proc. VLSI Test Symp, pp. 10-15, 1997 (VTS’97)
  - 10 vectors detect all single and multiple faults
  - In any size ripple CLA (not an LCU implementation)

- "Scalable Test Generators for High-Speed Datapath Circuits"
  - Al-Asaad, Hayes, and Murray
  - $2 \times (N+1)$ vector sequence (for an $N$-bit adder)
  - TPG implementation requires:
    - $N+1$-bit shift register
    - $N$ XOR gates, $N$ XNOR gates, and 1 inverter
CLA BIST Scheme

- Easy BIST circuit to implement
  - But we found a problem in design
    - 2 missing patterns needed for 100% FC
  - Replace inverter with flip-flop
    - $2 \times (N+2)$ vector sequence

Replace inverter with flip-flop to get $2 \times (N+2)$ vector sequence.

- $N+1$-bit Serial Shift Register

A. Stroud 1/08
Fault Simulation Results

- JETTA’98 approach gives best overall fault coverage regardless of adder implementation
  - Undetected faults in JETTA’98 approach can be detected
    - Results in “New BIST” column for $2 \times (N+2)$ vector sequence

- JETTA’98 also claims similar BIST approach for Modified-Booth multiplier
  - But description of test algorithm is very sketchy

<table>
<thead>
<tr>
<th>48-bit CLA Adder Implementation</th>
<th>Gate Delays</th>
<th># Faults</th>
<th>Test Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>VTS’97</td>
</tr>
<tr>
<td>Ripple CLA</td>
<td>28</td>
<td>1392</td>
<td>100%</td>
</tr>
<tr>
<td>Ripple LCU</td>
<td>12</td>
<td>1542</td>
<td>95.7%</td>
</tr>
<tr>
<td>Multi-stage LCU</td>
<td>10</td>
<td>1506</td>
<td>95.9%</td>
</tr>
</tbody>
</table>
Adder in Virtex-4 DSP

- **Adder has 3 input ports**
  - \( P = Z \pm (X+Y+C_{in}) \)
    - We interpret this as a 2-stage CLA adder/subtractor implementation

- **Apply test patterns to each stage in turn**
  - 2 clock cycles per vector
  - **OPMODE** control

  **Clock cycle #2**
  - **X test vector**

---

C. Stroud 1/08

VLSI D&T Seminar
DSP BIST Modes & Sequences

- **Test pattern sequence**
  - Four groups of 256 clock cycles (ccs) each
    - Allows control of operational modes (OPMODEs) of DSP

- **Test mode controlled by 4-bit shift register**
  - Bits include: Test Mode (2), Invert Control Signals, Reset
  - Contents loaded via Boundary Scan interface
    - Reduces the number of downloads to FPGA

---

<table>
<thead>
<tr>
<th>Mode (Test)</th>
<th>First 256 ccs</th>
<th>Second 256 ccs</th>
<th>Third 256 ccs</th>
<th>Fourth 256 ccs</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (multiply)</td>
<td>P = A×B</td>
<td>P = A×B</td>
<td>P = A×B+C</td>
<td>P = A:B+C</td>
</tr>
<tr>
<td>01 (adder)</td>
<td>P = Z(C)</td>
<td>P = Y(C)</td>
<td>P = Z(C)</td>
<td>P = Y(C)</td>
</tr>
<tr>
<td>Preg=1 only</td>
<td>P=X(P)+Y(C)</td>
<td>P=Y(C)+Z(P)</td>
<td>P=Y(C)+Z(P)</td>
<td>P=Y(C)+Z(ShiftP)</td>
</tr>
<tr>
<td>10 (cascade)</td>
<td>P₁ = A:B+Z(PC)</td>
<td>P₁=A:B+Z(ShiftPC)</td>
<td>P₁ = Z(C)</td>
<td>P₁ = Z(C)</td>
</tr>
<tr>
<td>P₀ = Z(C)</td>
<td>P₀ = Z(C)</td>
<td>P₀ = A:B+Z(PC)</td>
<td>P₀ = A:B+Z(ShiftPC)</td>
<td></td>
</tr>
</tbody>
</table>

- **Constant Control Signals**
- **Pseudo-Random Control Signals**
BIST Architecture

- 2 TPGs drive alternate rows of DSPs tiles
  - TPG drives both DSPs in tile
  - Prevents faulty TPG from escaping detection
- DSPs driven by different TPGs compared by ORAs
  - Like DSPs compared
    - Slice 0 compared to slice 0
    - Slice 1 compared to slice 1
  - Top DSPs compared to bottom DSPs in circular comparison
TPG Architecture

- Counter ⇒ $5 \times 3$ and $3 \times 5$ multiplier test to ports A&B
- Shift register ⇒ $2 \times (N+2)$ vector adder test to port C
- FSM ⇒ OPMODE control for 4 group sequences
- LFSR ⇒ pseudo-random patterns to other control inputs during last two groups of 256 clock cycles
ORA Implementation

- Old comparison-based ORA
  - Logic 1 latched in FF due to mismatches
  - Configuration memory readback used to get results

- CLBs have dedicated carry chain for fast adders and counters
  - New ORA latches logic 0 due to mismatch
  - Carry chain performs iterative OR function
  - Single pass/fail indication at end of BIST sequence
  - Only read configuration memory to get failing results for diagnosis
# BIST Configurations

- **5 downloads to FPGA**
  - 1 compressed download (<50% of full config)
  - + 4 partial reconfigurations (<0.5% of full config)
    - only change DPS configuration bits

- **7 BIST sequences**
  - BIST configurations #2 & #3 ran twice
    - different control register values for multiplier/adder test algorithms

<table>
<thead>
<tr>
<th>BIST Config</th>
<th>Pipeline Registers</th>
<th>Signals Active Level</th>
<th>B Input Source</th>
<th>Test Modes Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slice0</td>
<td>Slice1</td>
</tr>
<tr>
<td>1</td>
<td>All Regs=0</td>
<td>High</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>2</td>
<td>All Regs=1</td>
<td>High</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>3</td>
<td>A&amp;Breg=2 Other Regs=1</td>
<td>Low</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>4</td>
<td>All Regs=1</td>
<td>High</td>
<td>Direct</td>
<td>Cascade</td>
</tr>
<tr>
<td>5</td>
<td>All Regs=1</td>
<td>Low</td>
<td>Cascade</td>
<td>Direct</td>
</tr>
</tbody>
</table>

*bottom row failures due to unconnected cascade inputs*
Cascade Mode Testing

- One slice from pair put in cascade mode at a time
  - Circular comparison of slices sees identical behavior
- Cascade inputs to bottom DSP are not connected
  - Expected failures in ORAs comparing that DSP’s outputs
DSP BIST Implementations

- Circular comparison per DSP column
  - Each slice in tile compared with its counterpart
    - slice0-to-slice0
    - slice1-to-slice1

- CLB carry chain used to provide pass/fail indication
  - Only read config memory contents to get results for diagnosis
Automated BIST Configurations

- C program generates .XDL file
- .XDL to .NCD
  - `xdl -xdl2ncd bist.ncd`
- FPGA Editor
  - Design Rule Check
  - Route design
- .NCD to .BIT
  - BitGen
  - Download into FPGA
- .NCD to .XDL
  - Modification program for generating remaining 4 BIST configurations
DSP BIST Implementations

Brad Dutton
Generated BIST configurations for all Virtex-4 FPGAs and verified BIST on LX25, LX60, SX35, & FX12 via download and execution.
BIST Timing Analysis

David Baumann’s results

Bogus timing analysis by Xilinx tools due to unused cascade path with no pipeline registers
BIST Timing Analysis

Based on configuration #3

F_{max} function of #DSPs & size of array

4 TPGs might improve F_{max}
Physical Fault Injection

- Faulty FPGAs are difficult to find
  - 1 ORCA with faulty PLB & 2 ORCAs with faulty routing
- Physical fault insertion
  - Etch package down to bare die and “zap”
- We use fault injection emulation
  - Modify configuration bits before or after download (RMW)
  - Can inject single and/or multiple faults
    - Stuck-at faults & bridging faults
    - Faults limited effects of configuration bits

System or BIST configuration file

Stuck-at values

Fault mask

Download file

FPGA

Mustfa Ali’s work

C. Stroud 1/08

VLSI D&T Seminar
Fault Injection Emulation Results

1) Download BIST configuration
2) Manipulate configuration bit via read-modify-write
3) Run BIST sequence
4) Get BIST results

# BIST configs detecting fault

<table>
<thead>
<tr>
<th>Config</th>
<th>stuck-at-0</th>
<th>stuck-at-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Csel0ib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Csel1ib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op0ib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op1ib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op2ib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op3ib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op4ib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Op5ib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ceaib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cebib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cemib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cepib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cecfib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cecinib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rstaib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rstbib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rstmbb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rstpib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rstctib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rstcinib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aregb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Areg2b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breg0b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breg2b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mreg0b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preg0b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cinreg0b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cselreg0b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opreg0b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subreg0b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cikib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Casco</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creg0b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cecib&amp;t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nocfgb</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Investigated known test algorithms for multipliers and adders
- Looked for architecture independent tests with highest fault coverage
- JETTA’98 approach easy to implement
  - Needs modification for 100% FC
- 7 DSP BIST sequences with 5 downloads
  - New ORA eliminates config memory readback
  - Total testing time < 52% of 1 full download
    - Using compressed and partial reconfiguration
      - Only DSP configuration bits need to be changed

Application to Virtex-5 DSPs
BIST Approach for Virtex-5 DSP

Optional regs like V4 but data sheets have less info

Larger multiplier but same test algorithm

Logical operations but 48-bit cascade of A:B allows direct testing

Pattern detect but known algorithm for = comparator

*These signals are dedicated routing paths internal to the DSP48E column. They are not accessible via fabric routing resources.