A Test Pattern Generator for DSP BIST in Virtex-4

Chris Erickson
Outline of Presentation

- Overview of Digital Signal Processor
  - Test configurations
  - Test problems
- Multiplier Implementation Comparison
  - Array multiplier
  - Modified Booth multiplier
- Test Algorithm Comparison
  - Applying each algorithm to all multipliers
  - Experimental Results
- Summary and Conclusions
Outline of DSP

- **Primary Inputs**
  - A – 18 bits
  - B – 18 bits
  - C – 48 bits
  - Cin
  - Subtract
- **Cascade Inputs**
  - BCin – 18 bits
  - PCin – 48 bits

- **Primary Outputs**
  - P – 48 bits

- **Cascade Outputs**
  - PCout – 48 bits
  - BCout – 18 bits
  - PCin – 48 bits
Test Configurations & Problems

- Test configurations = 3 minimum
- Test Problems – no implementation details
  - 18x18-bit Multiplier
  - 48-bit (3-input) Adder/Subtactor
Test Pattern Generator

- Need TPG to complete DSP BIST development
- Goals for multiplier and adder/subtractor
  - Develop a set of test patterns that will work regardless of design implementation used by Xilinx
  - Develop a minimum vector set that will provide 100% fault coverage
- Previous multiplier work includes:
  - Minimum test vectors for a 16x16 array multiplier
    - 10 test vectors for C6288 (Kalyana Kantipudi’s MS Defense)
  - Test algorithm for a 16 x 16 Modified Booth multiplier
    - 256 test vectors
      - Gizopoulos, et.al., IEEE Design & Test 1998
Multiplier Comparison

- **Array Multiplier**
  - Uses full adders
  - Easy to implement
  - Low area overhead
    - 18x18-bit
      - # gates = 1800
  - Slow
- **Booth Multiplier**
  - Uses carry-save adders
  - Complex to implement
  - Large area overhead
    - 18x18-bit
      - # gates = 2600
  - Fast

- Implementation choice for small area applications
- Implementation choice for high performance applications
- Use in Virtex-II
  - Probably used in Virtex-4
Ripple Carry

Easy and efficient to build but not fast. Every bit calculation is done in series. Not an option for circuits requiring speed.
Carry-Save Adder

Only more efficient than ripple-carry when the number of inputs $\geq 3$
Carry Look-Ahead

Requires significantly more area and logic. Speeds up calculation time immensely since calculations can be completed in parallel.
Modified Booth Multiplier

A7  A6  A5  A4  A3  A2  A1  A0
   r   pp  r   pp  r   pp  r   pp
   3,0 3,1 3,2 3,3
   2,0 2,1 2,2 2,3
   1,0 1,1 1,2 1,3
   0,0 0,1 0,2 0,3
   B0  B1  B2  B3
Modified Booth Multiplier

A7 → F → F → F → F → F → F → F → F → F → P11
A5 → F → F → F → F → F → F → F → F → P10
A3 → F → F → F → F → F → F → F → F → P9
A1 → H → H → H → H → H → H → H → H → P8
   ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓
   3,0    3,1    3,2    3,3    3,4  2,2    2,3    2,4  3,4
   ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓
   2,0    2,1    2,2    2,3    2,4  1,2    1,3    1,4  2,4
   ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓
   1,0    1,1    1,2    1,3    1,4  0,2    0,3    0,4  3,2
   ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓
   0,0    0,1    0,2    0,3    0,4  0,1    0,2    0,3  0,4
   ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓
   P0     P1     P2     P3     P4

2/7/07
Fault Simulation Models

- ASL models developed for 18x18-bit
  - Array multiplier
  - Modified Booth multiplier
- Applied previous test algorithms to both multipliers
  - Single stuck-at gate-level fault model with AUSIM
Vector create/edit/compare Program

- Will take a vector list and expand it
- Will take the results of simulation and compare to its own generated results
- Will indicate where errors occurred
- Capable of segmenting and parsing through long lines of binary data to produce a readable, understandable format
Vector Program Example

The image shows a visual representation of a vector program example. It includes a form with input fields for 'Input A', 'Input B', 'Input C', 'File Output', and 'Actual Output'. There are options to select operations such as DSP, Multiplier, and Adder. The form also includes buttons for 'Start', 'Clear All', and 'Exit'. The input fields display binary values, indicating the program's functionality or data processing. The right side of the form has similar input fields and buttons.
Pattern Expansion

Minimum 10 vector set for 16-bit Pattern

| 1101101101101101 | 1101111111111111 |
| 0110110110110110 | 1111111111111111 |
| 0000000000000000 | 0010111111111111 |
| 1011011011011011 | 1101111111111111 |
| 1111111111111111 | 1101010101010101 |
| 1111111111111111 | 0110101010101010 |
| 0011111111111101 | 1101010101010101 |
| 0011111111111101 | 1010101010101010 |
| 1101101101101100 | 0010111111111111 |
| 1101101101101100 | 1010101010101010 |

Expanded 18-bit Pattern

| 110110110110110110 | 110111111111111111 |
| 011011011011011011 | 011011011011011111 |
| 000000000000000000 | 001011111111111111 |
| 101101101101101111 | 101101101101101111 |
| 111111111111111111 | 110101010101010101 |
| 111111111111111111 | 011010101010101010 |
| 001111111111111110 | 001111111111111110 |
| 001111111111111110 | 001111111111111110 |
| 110110110110110110 | 110110110110110110 |
| 110110110110110110 | 110110110110110110 |
Array Multiplier Test Algorithm

- 10 vectors total for C6288
- 18x18-bit array multiplier results
  - ≈ 95%
  - Pattern expansion required
    - ✔ To go from 16x16-bit for C6288 to 18x18-bit for DSP

- Booth multiplier results
  - ≈ 37% with carry-look-ahead implementation
  - ≈ 62% with carry-save implementation
    - ✔ Booth multiplier models may have had bugs

- Array multiplier test vectors do not test Booth multiplier
Booth Multiplier Test Algorithm

- Uses an 8-bit counter or LFSR
- Is based on pseudo-exhaustive testing
- Reported to provide >99% fault coverage on any size N-bit modified Booth multiplier

Test Patterns

<table>
<thead>
<tr>
<th>Groupings of 4-bits</th>
<th>Allows for recode circuitry to be symmetrically aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0000 0000</td>
<td></td>
</tr>
<tr>
<td>0010 0010 0010 0010</td>
<td></td>
</tr>
<tr>
<td>0011 0011 0011 0011</td>
<td></td>
</tr>
<tr>
<td>0100 0100 0100 0100</td>
<td></td>
</tr>
<tr>
<td>0101 0101 0101 0101</td>
<td></td>
</tr>
<tr>
<td>0110 0110 0110 0110</td>
<td></td>
</tr>
<tr>
<td>0111 0111 0111 0111</td>
<td></td>
</tr>
<tr>
<td>1000 1000 1000 1000</td>
<td></td>
</tr>
<tr>
<td>1001 1001 1001 1001</td>
<td></td>
</tr>
<tr>
<td>1010 1010 1010 1010</td>
<td></td>
</tr>
<tr>
<td>1011 1011 1011 1011</td>
<td></td>
</tr>
</tbody>
</table>

Recode Results

<table>
<thead>
<tr>
<th>$X_{2i+1}X_{2i}X_{2i-1}$</th>
<th>Sign</th>
<th>One</th>
<th>Two</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+0Y</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>+1Y</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>+1Y</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>+2Y</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-2Y</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1Y</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1Y</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-0Y</td>
</tr>
</tbody>
</table>
Modified Booth Multiplier

A7 A6 A5 A4 A3 A2 A1 A0
B0 B1 B2 B3

0 3,0 0 3,1 0 3,2 0 3,3
0 2,0 0 2,1 0 2,2 0 2,3
0 1,0 0 1,1 0 1,2 0 1,3
0 0,0 0 0,1 0 0,2 0 0,3

Electrical and Computer Engineering
Auburn University
Samuel Ginn College of Engineering

2/7/07
Modified Booth Multiplier

A7 A6 A5 A4 A3 A2 A1 A0
B0 B1 B2 B3

00 01 0 0 0 0 0 0
1 0 0 0 0 0 0 0

+ 00001

000001 00000 00000 00000

000000010001
Booth Multiplier Test Algorithm

- Tried two algorithms
  - 256 vectors total for 4-bit grouping
    - 8-bit counter
  - 64 vectors total for 3-bit grouping
    - 6-bit counter

- 18x18-bit array multiplier results
  - ≈ 99.99%
  - 1 undetected fault

- Booth multiplier results
  - ≈ 70% with carry-look-ahead implementation
  - ≈ 90% with carry-save implementation
  - ≈ 90% with ripple-carry implementation
  - Undetected faults primarily in recode circuitry
  - Booth multiplier models may have had bugs

- Booth multiplier test vectors do test Array multiplier
Planned TPG Development

- Continue debugging gate-level multiplier
- Incorporate multiplier into the DSP model
- Validate and debug gate-level DSP design by comparing to VHDL/Verilog simulation results
- Similar TPG development for 3-input adder/subtractor in DSP
- Additional test patterns for remaining logic in DSP
  - If not tested by multiplier and adder/subtractor test algorithms
Summary

- Compared previously proposed test algorithms for
  - Array multipliers - small vector set
  - Modified Booth multiplier - larger but reasonable vector set

- Array multiplier test vectors do poor job of testing Booth multiplier

- Booth multiplier test vectors do good job of testing multipliers
  - Independent of implementation

- Similar approach should feasible for development of TPG for 48-bit adder/subtractor