Testing of Logic Circuits

- Fault Models
- Test Generation and Coverage
- Fault Detection
- Design for Test

Fault Model

- Stuck-At Model
  - Assume selected wires (gate input or output) are “stuck at” logic value 0 or 1
  - Models certain kinds of fabrication flaws that short circuit wires to ground or power, or broken wires that are floating
    - Wire w stuck-at-0: w/0
    - Wire w stuck-at-1: w/1
  - Often assume there is only one fault at a time—even though in real circuits multiple simultaneous faults are possible and can mask each other
  - Obviously a very simplistic model!
Fault Model

- Simple example:

  \[ \begin{array}{c}
  0 \quad w_1 \quad a/1 \quad f \\
  0 \quad w_2 \quad b \quad d \\
  0 \quad w_3 \quad c \quad 0
  \end{array} \]

  See 1

- Generate a test case to determine if \( a \) is stuck at 1
  - Try 000
  - If \( a \) stuck at 1, expect to see \( f = 0 \), but see 1 instead

### Fault Model

- Simple example

\[
\begin{array}{ccccccccc}
\text{Test} & w_1 & w_2 & w_3 & a/0 & a/1 & b/0 & b/1 & c/0 & c/1 & d/0 & d/1 & f/0 & f/1 \\
\hline
000 & X & X & X & X & X \\
001 & X & X & X & X & X \\
010 & X & X & X & X & X \\
011 & X & X & X & X & X \\
100 & X & X & X & X & X \\
101 & X & X & X & X & X \\
110 & X & X & X & X & X \\
111 & X & X & X & X & X \\
\end{array}
\]
Problems with Fault Model

- In general, n-input circuits require much less than $2^n$ test inputs to cover all possible stuck-at-faults in the circuit.
- However, this number is usually still too large in real circuits for practical purposes.
- Finding minimum test cover is an NP-hard problem too.

Path Sensitization

- Wire-at-time testing too laborious.
- Better to focus on wiring paths, enabling multi-wire testing at the same time.
- “Activate” a path so that changes in signal propagating along the path affects the output.
Path Sensitization

Simple Example:

To activate the path, set inputs so that \( w_1 \) can influence \( f \).

E.g., \( w_2 = 1, w_3 = 0, w_4 = 1 \)
- AND gates: one input at 1 passes the other input
- NOR gates: one input at 0 inverts the other input

To test: \( w_1 \) set to 1 should generate \( f = 0 \) if path ok
- faults \( a/0, b/0, c/1 \) cause \( f = 1 \)
- \( w_1 \) set to 0 should generate \( f = 1 \) if path ok
- faults \( a/1, b/1, c/0 \) cause \( f = 0 \)

One test can capture several faults at once!

Path idea can be used to “propagate” a fault to the output to observe the fault
- Set inputs and intermediate values so as to pass an internal wire to the output while setting inputs to drive that internal wire to a known value
- If propagated value isn’t as expected, then we have found a fault on the isolated wire

Path Sensitization

Good news: one test checks for several faults
- Number of paths much smaller than number of wires
- Still an impractically large number of paths for large-scale circuits
Fault Propagation

\[
\begin{align*}
\text{Fault Propagation} & \\
\begin{array}{c}
\text{Input A} \\
\text{Input B} \\
\text{Input C} \\
\end{array}
& \\
\begin{array}{c}
\text{Fault}
\end{array}
& \\
\begin{array}{c}
\text{Output F}
\end{array}
& \\
\text{Fault}
& \\
\text{Output F}
& \\
\end{align*}
\]
Tree Structured Circuits

To test inputs stuck-at-0 at given AND gate:
1. Set inputs at other gates to generate AND output of zero
2. Force inputs at selected gate to generate a one
3. If f is 1 then circuit ok, else fault

To test inputs stuck-at-1 at given AND gate:
1. Drive input to test to 0, rest of inputs driven to 1
2. Other gates driven with inputs that force gates to 0
3. If f is 0 then fault, else OK

Tree Structured Circuits

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1 \overline{w}_3 w_4$</td>
<td>1110</td>
</tr>
<tr>
<td>$w_1 w_3 w_4$</td>
<td>1111</td>
</tr>
<tr>
<td>$w_1 w_2 w_3$</td>
<td>1010</td>
</tr>
<tr>
<td>$w_1 w_2 w_3 w_4$</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overline{w}_1 w_3$</td>
<td>1111</td>
</tr>
<tr>
<td>$\overline{w}_1 w_3$</td>
<td>1111</td>
</tr>
<tr>
<td>$\overline{w}_1 w_2 w_3$</td>
<td>1010</td>
</tr>
<tr>
<td>$\overline{w}_1 w_2 w_3 w_4$</td>
<td>0000</td>
</tr>
</tbody>
</table>

000

Stuck-at-0
0
Tree Structured Circuits

Product Term | Test
---|---

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$\bar{w}_3$</th>
<th>$\bar{w}_4$</th>
<th>$w_2$</th>
<th>$\bar{w}_3$</th>
<th>$w_4$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Tree Structured Circuits
Tree Structured Circuits

Product Term | Test
---|---
$w_1 \overline{w}_3 \overline{w}_4 \overline{w}_2 \overline{w}_3 \overline{w}_4 w_1 w_2 w_3 w_4$ | 
1 | 1 1 1 0 1 0 0 0 0 1 0 0 0
2 | 0 1 0 1 1 1 1 0 0 1 0 1
3 | 0 0 0 1 0 1 1 1 1 0 1 1
4 | 0 1 1 1 1 0 1 1 0 0 1 0 0
5 | 1 0 1 1 0 0 0 1 1 1 1 0 1
6 | 1 1 0 0 1 1 0 0 0 1 0 0 1
7 | 1 0 0 1 0 1 0 1 1 1 1 1 1
8 | 0 0 0 0 0 1 1 0 1 0 0 1 1

Tree Structured Circuits

Product Term | Test
---|---
$w_1 \overline{w}_3 \overline{w}_4 \overline{w}_2 \overline{w}_3 \overline{w}_4 w_1 w_2 w_3 w_4$ | 
1 | 1 1 1 0 1 0 0 0 0 1 0 0 0
2 | 0 1 0 1 1 1 1 0 0 1 0 1
3 | 0 0 0 1 0 1 1 1 1 0 1 1
4 | 0 1 1 1 1 0 1 1 0 0 1 0 0
5 | 1 0 1 1 0 0 0 1 1 1 1 0 1
6 | 1 1 0 0 1 1 0 0 0 1 0 0 1
7 | 1 0 0 1 0 1 0 1 1 1 1 1 1
8 | 0 0 0 0 0 1 1 0 1 0 0 1 1
Tree Structured Circuits

Product Term | Test
---|---
$w_1 \bar{w}_3 \bar{w}_4 \bar{w}_2 \bar{w}_3 \bar{w}_1 \bar{w}_2 \bar{w}_3 \bar{w}_4$ |
1 | 1 1 1 0 1 0 0 0 0 | 1 0 0 0
2 | 0 1 0 1 1 1 1 1 0 | 0 1 0 1
3 | 0 0 0 1 0 1 1 1 1 | 0 1 1 1
4 | 0 1 1 1 1 0 1 1 0 | 0 1 0 0
5 | 1 0 1 0 0 0 1 1 1 | 1 1 1 1
6 | 1 1 0 0 1 1 0 0 0 | 1 0 1 0
7 | 0 0 0 0 1 1 0 1 0 | 1 0 0 1
8 | 0 0 0 0 1 1 0 1 0 | 0 0 1 1

CS 150 - Spring 2007 – Lec. #20: Testing - 18
Tree Structured Circuits

![Tree Structured Circuit Diagram]

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Test</th>
</tr>
</thead>
</table>
| $w_1 \overline{w}_3 \overline{w}_4 \overline{w}_2 \overline{w}_3 w_2 w_3 \overline{w}_1 w_2 w_3 \overline{w}_4$ | $1 1 1 1 0 1 0 0 0 0 1 0 0 0$
| $w_1 \overline{w}_3 \overline{w}_4 w_2 \overline{w}_3 w_2 w_3 \overline{w}_1 w_2 w_3 \overline{w}_4$ | $2 0 1 0 1 1 1 1 1 0 0 1 0 1$
| $w_1 \overline{w}_3 \overline{w}_4 w_2 \overline{w}_3 w_2 w_3 \overline{w}_1 w_2 w_3 \overline{w}_4$ | $3 0 0 0 1 0 1 1 1 1 0 1 1 1$
| $w_1 \overline{w}_3 \overline{w}_4 w_2 \overline{w}_3 w_2 w_3 \overline{w}_1 w_2 w_3 \overline{w}_4$ | $4 0 1 1 1 1 0 1 1 0 0 1 0 0$
| $w_1 \overline{w}_3 \overline{w}_4 w_2 \overline{w}_3 w_2 w_3 \overline{w}_1 w_2 w_3 \overline{w}_4$ | $5 1 0 1 1 0 0 1 1 1 1 1 1 0$
| $w_1 \overline{w}_3 \overline{w}_4 w_2 \overline{w}_3 w_2 w_3 \overline{w}_1 w_2 w_3 \overline{w}_4$ | $6 1 1 0 0 1 1 0 0 1 1 1 0 1$
| $w_1 \overline{w}_3 \overline{w}_4 w_2 \overline{w}_3 w_2 w_3 \overline{w}_1 w_2 w_3 \overline{w}_4$ | $7 1 0 0 1 0 1 0 1 1 1 1 1 1$
| $w_1 \overline{w}_3 \overline{w}_4 w_2 \overline{w}_3 w_2 w_3 \overline{w}_1 w_2 w_3 \overline{w}_4$ | $8 0 0 0 0 1 1 0 1 0 0 1 1$

Any other stuck-at-1 cases covered?
Tree Structured Circuits

Any other stuck-at-1 cases covered? Was that case already covered?

All inputs stuck-at-1's covered now
**Random Testing**

- So far: deterministic testing
- Alternative: random testing
  - Generate random input patterns to distinguish between the correct function and the faulty function

```
<table>
<thead>
<tr>
<th>Number of Tests</th>
<th>Probability Fault Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small number of tests has reasonable probability of finding the fault</td>
<td></td>
</tr>
</tbody>
</table>
```

**Announcements**

- Discussion Section Friday 10-11 AM CANCELLED due to low attendance (still have Thursday 4-5 and Friday 11-noon though)
- Checkpoint #4 Lab Lecture this Friday
- No Lab Lecture during Week #11
- Specification of Final Report during Week #12
Project

- Week #11 (this week): Finish Checkpoint #3
  - Hard deadline in your lab week of April 9
  - 20% late penalty after that
- Week #12 (next week): Start Checkpoint #4
  - Early Demo Hard Deadline Monday, 16 April
  - 10% Extra credit for early completion!
- Week #13: Finish Checkpoint #4
  - Standard Demo Deadline Monday, 23 April
  - 20% Late Penalty plus partial credit for partial functionality
  - 20% Extra credit cap
- Week #14: Final Report
  - Hard Deadline Monday, 30 April
  - There will be no late reports

Sequential Testing

- Due to embedded state inside flip-flops, it is difficult to employ the same methods as with combinational logic
- Alternative approach: design for test
  - Scan Path technique: FF inputs pass through multiplexer stages to allow them to be used in normal mode as well as a special test shift register mode
Scan Path Technique

- Configure FFs into shift register mode (red path)
- Scan in test pattern of 0s and 1s
- Non-state inputs can also be on the scan path (think synchronous Mealy Machine)
- Run system for one clock cycle in "normal" mode (black path)—next state captured in scan path
- Return to shift register mode and shift out the captured state and outputs

Scan Path Example

- w,y1,y2 test vector 001
- Scan 01 into y1, y2 FFs
Scan Path Example

- w,y1,y2 test vector 001
- Scan 01 into y1, y2 FFs

Scan Path Example

- w,y1,y2 test vector 001
- Scan 01 into y1, y2 FFs
Scan Path Example

- $w, y_1, y_2$ test vector 001
  - Scan 01 into $y_1, y_2$ FFs
  - Normal $w=0$

Scan Path Example

- $w, y_1, y_2$ test vector 001
  - Scan 01 into $y_1, y_2$ FFs
  - Normal $w=0$
  - Output $z=0, Y_1=0, Y_2=0$
Scan Path Example

- **w,y1,y2 test vector 001**
  - Scan 01 into y1, y2 FFs
  - Normal w=0
  - Output z=0, y1=0, y2=0
  - Observe z directly

Scan Path Example

- **w,y1,y2 test vector 001**
  - Scan 01 into y1, y2 FFs
  - Normal w=0
  - Output z=0, y1=0, y2=0
  - Observe z directly
  - Scan out Y1, Y2
Scan Path Example

- w, y1, y2 test vector 001
  - Scan 01 into y1, y2 FFs
  - Normal w=0
  - Output z=0, Y1=0, Y2=0
  - Observe z directly
  - Scan out Y1, Y2

Built-in Self-Test (BIST)

- Test Vector Generator
  - Pseudorandom tests with a feedback shift register
  - Seed generates a sequence of test patterns
  - Outputs combined using the same technique
  - Generates a unique signature that can be checked to determine if the circuit is correct
Linear Feedback Shift Register

!Starting with the pattern 1000, generates 15 different patterns in sequence and then repeats

Pattern 0000 is a no-no
Linear Feedback Shift Register
- Multi-input Compressor

Complete Self-Test System
- Normal Inputs
- Random Test Sequences
- PRBSG
- MUX
- Combinational Circuit
- FFs and Muxes
- PRBSG
- Random Test Sequences
- Pseudo Random Binary Sequence Generator
- MIC
- Multi-input Compressor
- SIC
- Single-input Compressor
Built-in Logic Block Observer (Bilbo)

- Test generation and compression in a single circuit!
  - \( M_1, M_2 = 11 \): Regular mode
  - \( M_1, M_2 = 00 \): Shift register mode
  - \( M_1, M_2 = 10 \): Signature generation mode
  - \( M_1, M_2 = 01 \): Reset mode

Bilbo Architecture

- Scan initial pattern in Bilbo1, reset FFs in Bilbo2
- Use Bilbo1 as PRBS generator for given number of clock cycles and use Bilbo2 to produce signature
- Scan out Bilbo2 and compare signature; Scan in initial test pattern for CN2; Reset the FFs in Bilbo1
- Use Bilbo2 as PRBS generator for a given number of clock cycles and use Bilbo1 to produce signature
- Scan out Bilbo1 and compare signature;
Summary

- Fault models
  - Approach for determining how to develop a test pattern sequence
  - Weakness is the single fault assumption

- Scan Path
  - Technique for applying test inputs deep within the system, usually for asserting state
  - Technique for getting internal state to edges of circuit for observation

- Built-in Test
  - Founded on the approach of random testing
  - Generate pseudo random sequences; compute signature; determine if signature generated is same as signature of a correctly working circuitry

Midterm II Results

- Mean: 31/Standard Deviation: 7
- Min: 10/Max: 42.5