**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
  - Wire stuck-at-0: \( w/0 \)
  - Wire 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
  - Obviously a very simplistic model!

**Fault Model**

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

**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:

  ![Diagram](image1)

  To activate the path, set inputs so that \( w_0 \) can influence \( f \):
  
  \[ w_0 = 1, w_1 = 0, w_2 = 1 \]

  AND gate: one input at 1 passes the other input

  NOR gate: one input at 0 inverts the other input

  To test: \( w_0 \) set to 1 should generate \( f = 0 \) if path ok

  Faults \( a/1, b/1, c/0 \) cause \( f = 0 \)

  Faults \( a/0, b/0, c/1 \) cause \( f = 1 \)

  One test can capture several faults at once!

**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

  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

**Fault Propagation**

![Diagram](image2)

**Fault Propagation**

![Diagram](image3)

**Tree Structured Circuits**

- To test inputs stuck-at-0 at given AND gate

  - Set input at either gate to generate AND output of zero

  - Force inputs at selected gate to generate a one

  - If \( f = 1 \) then circuit ok, else fault

- To test inputs stuck-at-1 at given AND gate

  - Drive input to test to 0, rest of inputs driven to 1

  - Other gate driven with inputs that force \( f \) to 0

  - If \( f = 0 \) then fault, else OK
Random Testing

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

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
- Scan in pattern of 0s and 1s
- Non-state inputs can also be on the scan path
- Run system for one clock cycle in normal mode—next state captured in scan path
- Return to shift register mode and shift out the captured state and outputs

Scan Path Example

wyly2 test vector 001
- Scan 01 into y1, y2 FFs

wyly2 test vector 001
- Scan 01 into y1, y2 FFs
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**

- Random Test Pattern
- Starting with the pattern 1000, generates 15 different patterns in sequence and then repeats

**Complete Self-Test System**

- Combinational Circuit
- FFs and Muxes
- PRBS Generator
- Normal/Scan Inputs
- Random Test Sequences

**Built-in Logic Block Observer (Bilbo)**

- Test generation and compression in a single circuit!
- \[ M_1, M_2 = 11: \text{Regular mode} \]
- \[ M_1, M_2 = 00: \text{Shift register mode} \]
- \[ M_1, M_2 = 01: \text{Signature generation mode} \]
- \[ M_1, M_2 = 00: \text{Reset mode} \]

**Bilbo Architecture**

- Scan initial pattern in Bilbo 1, reset FFs in Bilbo 2
- Use Bilbo 1 as PRBS generator for given number of clock cycles and use Bilbo 2 to produce signature
- Scan out Bilbo 2 and compare signature; scan in initial test pattern for CN2; reset the FFs in Bilbo 1
- Use Bilbo 2 as PRBS generator for given number of clock cycles and use Bilbo 1 to produce signature
- Scan out Bilbo 1 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 circuit