LECTURE 1: INTRODUCTION AND FAULT MODELLING
Testing Digital Systems – Contents

• Why is testing a problem?
• Fault-Oriented Test Generation
• Untestable Faults
• Design for Test
• Built-In Self-Test
• Boundary Scan
The need for testing

- Real systems may have manufacturing defects. Short circuits, missing components, damaged components etc.

- Need to know if a system (board, IC, whole system) has a defect (and therefore doesn't work). Don't want to sell bad systems - don't want to reject good systems.

- ∴ Need for testing is economic.
Two approaches

- Functional testing - does system work correctly?
- Structural testing - does system contain a fault?
- Functional testing can imply a long and difficult task!
Fault Models

- Fault Modelling
  - What Defects occur?
  - How can they be modelled?
  - What do models imply?
PCB Defects

- Breaks in connections
  - bad etching
  - stress
- Short Circuits
  - solder flow
- Bad solder joints
IC Defects

- Open Circuits
  - electromigration
  - current overstress
  - corrosion
- Short Circuits
- Incorrect Transistor Action
  - silicon or oxide defects
  - mask misalignment
  - impurities
  - gamma radiation
- "Latch-up"
  - transient currents
- Data Corruption
  - alpha particles
  - EMI

---

**Electronics and Computer Science**

**UNIVERSITY OF Southampton**
How do defects manifest themselves?

- Static failures (50%)
  - shorts, breaks etc
- Dynamic failures (49%)
  - out of spec components
  - timing failures
- Intermittent failures (1%)
  - environmental
Fault Model

- Physical Defect manifests itself as a logical fault
  - Applies only to digital circuits
  - (No analogue fault models.)

- Stuck Fault Model
  - Many physical defects can be modelled as a circuit node being:
    - stuck at 1 (s-a-1)
    - stuck at 0 (s-a-0)
Single-Stuck Fault Model

- Assumptions:
  - The fault directly affects only one node
  - The node is stuck at either 0 or 1
- These assumptions make test pattern generation easier
Validity of Single-Stuck Fault Model?

- Multiple faults do occur
- Can multiple faults mask each other?
- Number of faults might rise with complexity
- Can all defects be modelled with stuck fault model?
- The model appears to be valid most of the time
- Almost all test pattern generation relies on this model
- Multiple faults are found with test patterns for single faults
LECTURES 2 & 3: TEST GENERATION
Fault-oriented Test Pattern Generation

- Prepare a fault list (e.g. all nodes stuck-at 0 & stuck-at 1)

  repeat
  
  write a test

  check fault cover (one test may cover > 1 fault)

  (delete covered faults from list)

  until fault cover target is reached
Test Generation

1. Test pattern generation (writing a test) may be random or optimised. Ideally we want a minimum number of tests - cheaper to apply!

2. One test may cover more than one fault, often faults are indistinguishable.

3. If we simply want a pass/fail test, we can remove faults from further consideration, once we have found a test for a fault. If we want to diagnose a fault (for subsequent repair) we probably want to find all tests for a fault to deduce where the fault occurs.

4. Fault cover target may be less than 100%. The higher the cover, the greater the number of tests and hence the cost of applying the test.
Testability

• How testable is a system?

a. Controllability - can we control all the nodes to establish if there is a fault?

b. Observability - can we observe and distinguish between a faulty node and a correct node?
Sensitive Path Algorithm

- 7 nodes, \( \therefore 14 \) stuck faults:
  - a/0, a/1, b/0, b/1, c/0, c/1, d/0, d/1, e/0, e/1, f/0, f/1, z/0, z/1
  - "a stuck-at-0" etc.
Sensitive Path Algorithm

• To test for a/0, need to set a to 1 (fault-free condition).
• Need to observe existence or otherwise of fault at z.
• If b is 0, e is 1 irrespective of a. \(\therefore b\) must be 1.

• Similarly if f is 1, z is 1, irrespective of e, \(\therefore f\) must be 0.
• Thus we are establishing a sensitive path from a to z.
• To force f to 0, either c or d or both must be 0.
• If the fault a/o exists, e is 1, z is 1. If the fault does not exist, e is 0, z is 0.

• We can conclude from this that a test for a/o is: a=1, b=1, c=0, d=1, for which the fault-free output is z=0. This can be expressed as 1101/0.

• Other tests are 1110/0 and 1100/0. Therefore, there is more than one test for the fault a/o.

• To test for e/1, requires that f=0 to make e visible at z. \( \therefore c \) or d or both must be 0. To make e=0 requires that a=b=1. So a test for e/1 is 1101/0.

• This is the same test as for a/o! So one test can cover more than one fault.
Algorithm

1. Select a fault;
2. Set up inputs to force the node to a fixed value;
3. Set up inputs to transmit node value to an output;
4. Check for consistency;
5. Check for coverage of other faults;
Fault Cover

- The aim is to find the minimum number of tests that cover all the possible faults. 100% fault cover may not be possible.

- Fan-out and reconvergence can cause difficulties for this algorithm. Improved algorithms (D-algorithm, PODEM) use similar techniques but overcome drawbacks.
Hazards

• Unexpected behaviour as a result of delays.
• Consider a function $Z = A \cdot C + B \cdot \overline{C}$
• $\overline{C}$ is generated from C by an inverter:

Let $A=1$, $B=1$.

• What happens if C changes from 1 to 0 and there is a 1 unit delay on each gate?

• NB $Z = 1.1 + 1.0 = 1$, $C = 1$;
• $Z = 1.0 + 1.1 = 1$, $C = 0$. 
Types of Hazard

Dynamic hazards do not occur in two-level circuits. Dynamic hazards require 3 or more unequal signal paths; often caused by poor factorisation in multi-level minimisation. Solution is to redesign.
Redundant Logic

To avoid hazards, the redundant term may be included:

\[ Z = A \cdot C + B \cdot \overline{C} + A \cdot B \]

F is independent of C. If A = B = 1, F = 1. F stays at 1 while C changes, therefore Z stays at 1.
Testing redundant logic

- To test for f/0, f=1, \( \therefore a=b=1 \)
- To transmit to z, d=e=0.
- For e=0, b=0 and/or c=1.
- For d=0, a=0 and/or c=0.
- Thus there is an inconsistency!
- Untestable faults are due to redundancy.
One test pattern can be used to find more than one potential fault.

For example, suppose we wish to detect if node $E$ is stuck at 0 ($E/0$).

$E/0$ cannot be distinguished from $G/0$ or $A/0$ or $B/0$. 
In all these cases, $G$ will be 1 normally and 0 in the presence of one of these faults.

Hence, the input pattern $A = 1$, $B = 1$, $C = 0$, $D = 0$ can be used to detect four possible faults.

As there are 7 nodes in the circuit, there are 14 possible stuck-at faults.

This pattern covers 4 faults and it can be shown that of the 16 possible input patterns, 6 are sufficient to detect all the possible stuck-at faults in the circuit.
Testing Sequential Circuits

• Testing combinational circuits is relatively easy, provided, there is no redundancy in the circuit. Number of test vectors \( << 2 \) (no. of inputs)

• Testing sequential circuits is difficult because circuits have states. May require long sequences of inputs to reach states. Some faults may be untestable, because certain states cannot be reached.

• N.B. All sequential circuits MUST have a set or reset to initialise all flip-flops or testing is nearly impossible.
Sequential ATPG

- Combinational ATPG uses sensitive path algorithm. i.e. sensitise a path from a potential fault site to a PO (or in the case of SISO, a pseudo PO – a scan register input).

- Can model a sequential circuit:

- as an iterative array, where each element represents a clock cycle
Sequential ATPG algorithm

For each SSF in C
    r=1; q=0;
    repeat
        build model with r+q time frames;
        ignore POs in 1st r+q-1 time frames;
        generate test (D algorithm) for fault as if all time frames
        represent a combinational circuit. PIs can be set up in all
        time frames; only POs in last time frame observed.
        if success then return success!;
        else increment r or q;
        until r+q = fmax;
    return failure!;
• Very computationally expensive. Need to balance cost of
  ATPG against cost of scan insertion.
LECTURE 4: DESIGN FOR TEST
Ad Hoc Design for Test Methods

• Testability expressed in terms of:
  • Controllability - ability to control logic value of an internal node from PI
  • Observability - ability to observe logic value of internal node at PO
Testability can be enhanced by:

- ad hoc design guidelines
- structured design methodology

Things to Watch:

- Redundant Logic – undetectable faults
- Asynchronous sequential systems
  - difficult to synchronize with tester
  - if necessary, confine to independent blocks
- Monostables
  - difficult to control
Improve Test Access

• Use test points to enhance controllability and observability
Initialisation

- Initialisation must be provided for all sequential elements

Synchronous
  - Reset at next clock

Asynchronous
  - Using Set/Clear pins

Any defined state will do - not necessarily all zeros
Multiple initial states can be useful
Multiplexers

- Use MUXs to share I/O pins for testing

- Circuit diagram

  - M1, M2 = 00: Operational Mode
  - M1, M2 = 01: Test L1 - observe N at PO
  - M1, M2 = 10: Test L2 - control N from PI
SISO

- Modify all flip-flops to include MUX
- Makes every state controllable and observable
- Sequential test problem reduced to combinational test
SISO Test Generation

- Generate tests for the combinational part using the sensitive path algorithm (or a variant) as before.

- For each test, need to distinguish between inputs that are applied through the scan path, and those that are applied through the primary inputs.

- Same with outputs.
Running a Test

1. Set M=1 and test the flip-flops as a shift register. If a sequence of 1s and 0s is fed into SDI, we would expect the same sequence to emerge from SDO delayed by the number of clock cycles equal to the length of the shift register (n). A useful test sequence would be 00110... which tests all transitions and whether the flip-flops are stable.

2. For each combinational test
   a) Set M=1 to set the state of the flip-flops after n clock cycles by shifting a pattern in through SDI.
   b) Set M=0. Set up the primary inputs. Collect the values of the primary outputs. Apply 1 clock cycle to load the state outputs into the flip-flops.
   c) Set M=1 to shift the flip-flop contents to SDO after n-1 clock cycles.
Costs of enhanced testability

- Extra I/O pins
- Includes interfaces etc
- Extra components (MUXs), extra wiring
- Degradation of performance because of extra gates in signal paths
- More things to go wrong!
- BUT the circuit will be easier to test!
- Inserting a scan path is (relatively) easy!
- Ad hoc methods are difficult to automate
- Note. Scan path can be inserted before or after layout. Ordering is therefore arbitrary. Test vectors need to be sorted – a job for a computer.
Reducing the cost

- Share SDI with a PI (MUX)
- Share SDO with a PO (is the last flip-flop already connected to a PO?)
- Do all flip-flops need to be in the scan path (partial scan – too much for now)?

- Build MUXes into next state logic, e.g.
  \[ S' = \overline{S.T} + S.T.A \]
  \[ T' = \overline{S.T}.A + \overline{S.T}.\overline{A} \]
- Becomes
  \[ S' = (\overline{S.T} + S.T.\overline{A}).\overline{M} + T.M \]
  \[ T' = (\overline{S.T}.A + \overline{S.T}.\overline{A}).\overline{M} + SDI.M \]
- Reduces delay.
Partial Scan

- Don't put every flip-flop in the scan path

- C1/R1/R2/C2 form a balanced structure. Similarly C2/R4/R1 are balanced.

- We can make R3 and R4 part of the scan path. (Or we could make R1, R2, R3 the scan path.)
Multiple Scan Paths

- One global scan path would take too long to load/unload
- Divide into multiple, equal length scan paths
LECTURE 5: BOUNDARY SCAN (JTAG)
Boundary Scan

- IEEE 1149.1 Also known as JTAG
- not possible to test mounted ICs (the pins may be connected together);
- PCBs now often have more than 20 layers of metal, so deep layers cannot be reached;
- Density of components on a PCB is increasing. Multi-chip modules (MCMs) take the chip/board concept further and have unpackaged integrated circuits mounted directly on a silicon substrate.
Backdriving problem
Off-chip faults

Short to ground (Stuck at 0)

Solder Bridge
Principle of JTAG
Test Architecture
BS Cell

Diagram:

- IN
- SCAN_IN
- ClockDR
- ShiftDR
- D Q
- MUX
- UpdateDR
- SCAN_OUT
- MUX
- OUT
- MODE_CONTROL
Modes of Operation.

1. Normal mode.
   - Normal system data flows from IN to OUT.

2. Scan mode.
   - ShiftDR selects the SCAN_IN input, ClockDR clocks the scan path.
   - ShiftDR is derived from the similarly named state in the TAP controller.
   - ClockDR is asserted when the TAP controller is in state Capture-DR or Shift-DR.
   - (Hence, of course, the Boundary Scan architecture is not truly synchronous!)
3. Capture mode.
   - ShiftDR selects the IN input, data is clocked into the scan path register with ClockDR to take a snapshot of the system.

4. Update mode.
   - After a capture or scan, data from the left flip-flop is sent to OUT by applying 1 clock edge to UpdateDR.
   - Again, this clock signal comes from the TAP controller when it is in state Update-DR. The TAP controller then enters the Run Test state and MODE_CONTROL is set as appropriate according to the instruction held in the instruction register.
Instructions

• EXTEST (Mandatory).

  – This instruction performs a test of the system, external to the core logic of particular devices.

  – Data is sent from the output boundary scan cells of one device, through the pads and pins of that device, along the interconnect wiring, through the pins and pads of a second device and into the input boundary scan cells of that second device.

  – Hence a complete test of the interconnect from one IC core to another is performed.
Instructions

• SAMPLE/PRELOAD (Mandatory).
  – This instruction is executed before and after the EXTEST and INTEST instructions to set up pin outputs and to capture pin inputs.

• BYPASS (Mandatory).
  – This instruction selects the Bypass register, to shorten the scan path.
Instructions

- **RUNBIST (Optional).**
  - Runs a built-in self-test on a component.

- **IDCODE, USERCODE (Optional).**
  - These instructions return the identification of the device (and the user identification for a programmable logic device). The code is put into the scan path.
Instructions

• INTEST (Optional).

  – This instruction uses the boundary scan register to test the internal circuitry of an IC.

  – Although such a test would normally be performed before a component is mounted on a PCB, it might be desirable to check that the process of soldering the component onto the board has not damaged it.

  – Note that the internal logic is disconnected from the pins, so if pins have been connected together on the board, that will have no effect on the standard test.
Instructions

• CONFIGURE (Optional).

  – An SRAM-based FPGA needs to be configured each time power is applied.
  – The configuration of the FPGA is held in registers. These registers can be linked to the TAP interface.
  – This clearly saves pins as the configuration and test interfaces are shared.

• etc
Tristates and Bidirectionals

Core Logic

Tristate Output

Bidirectional
LECTURE 6: BUILT-IN SELF-TEST
Why Built-In Test?

- Economic justification
  - Simplifies test equipment
  - Simplifies TPG
  - Allows easy field test
- Increases user confidence
Principle of BIST

- Test Vectors
- Circuit Under Test
- Check responses
  - Go/No Go (Perhaps diagnostics)
How to generate test vectors?

• Store pre-generated vectors in ROM
  – Possibly a very large number
  – Problem with sequential logic
• Exhaustive test
  – Use binary counter to generate all test vectors
  – Separate combinational logic from registers
Linear Feedback Shift Register (LFSR)

Generate test patterns on chip:

- An n-bit shift register

2^n-1 patterns, if feedback chosen correctly.

- Easy to build. 1-3 XOR gates.
- Pseudo-random sequence – can have implications for power dissipation.
- If n is large (>32), time taken for sequence may be unacceptable. All 0s state excluded.
LFSR – Feedback connections

- $n=2$, $D_1 = Q_1 \oplus Q_0$
- $n=4$, $D_3 = Q_1 \oplus Q_0$
- $n=8$, $D_7 = Q_3 \oplus Q_0$
- $n=16$, $D_{15} = Q_5 \oplus Q_4 \oplus Q_3 \oplus Q_0$
- $n=24$, $D_{23} = Q_7 \oplus Q_2 \oplus Q_1 \oplus Q_0$
- Not more than 4 feedback connections are needed to generate $2^{n-1}$ patterns
- For $n=24$, $>250,000$ possible feedback connections
- Circuit can be modified to generate all 0s state ($2^n$ patterns)
Check Responses

- Look-up table of correct responses
  - potentially very large!
- Signature Analysis
  - Data Compression
Multiple Input Signature Register

Compact responses on chip:

Probability of a fault in the CUT giving the fault-free signature \( -> 2^{-n} \) (aliasing)

Easy to build.
BIST Example

- This circuit consists of three parts:
  - a 3 stage LFSR,
  - a 3 stage MISR and
  - a circuit under test, with the following functions:
    \[ X = A \text{ XOR } B \text{ XOR } C \]
    \[ Y = A.B + A.C + B.C \]
    \[ Z = A'.B + A'.C + B.C \]
- The sum-of-product terms are implemented with two levels of NAND gates.
LFSR & MISR Equations

- **LFSR:**
  - $A^+ = B \text{ XOR } C$
  - $B^+ = A$
  - $C^+ = B$

- **MISR:**
  - $P^+ = X \text{ XOR } (Q \text{ XOR } R)$
  - $Q^+ = Y \text{ XOR } P$
  - $R^+ = Z \text{ XOR } Q$
Fault Free Simulation

- Both the LFSR and MISR are initialised to the 111 state. By simulation, the sequence of states is found to be:

<table>
<thead>
<tr>
<th>LFSR CUT MISR</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc xyz</td>
</tr>
<tr>
<td>111 111 111</td>
</tr>
<tr>
<td>011 011 100</td>
</tr>
<tr>
<td>001 101 001</td>
</tr>
<tr>
<td>100 100 001</td>
</tr>
<tr>
<td>010 101 000</td>
</tr>
<tr>
<td>101 010 101</td>
</tr>
<tr>
<td>110 010 100</td>
</tr>
<tr>
<td>111 111 000</td>
</tr>
</tbody>
</table>

<- signature of fault-free circuit
Simulation of a fault

• Suppose a is s-a-0, the sequence is now:

LFSR CUT MISR

\[
\begin{array}{ccc}
abc & xyz \\
111 & 011 & 111 \\
011 & 011 & 000 \\
001 & 101 & 011 \\
100 & 000 & 100 \\
010 & 101 & 010 \\
101 & 101 & 000 \\
110 & 101 & 101 \\
111 & 011 & 011 & \text{<- signature of faulty circuit}
\end{array}
\]

• Note that if a is s-a-1, the signature 000 is also generated. This is an example of aliasing.
- Combines LFSR and MISR into one. Typically 5 modes: normal, scan, synchronous reset, MISR, LFSR.
BILBO Example

- To test C1, R2 is LFSR, R1 is MISR
- To test C2, R1 is LFSR, R2 is MISR
- Scan signatures in R1, R2 to some comparator
- NB 2 test sessions, each of $2^n - 1$ cycles, + scan + decision
• A 32-bit LFSR would take about 2 seconds to go through the entire sequence at 2 GHz.

• STUMPS avoids this problem by using a single LFSR and MISR to generate patterns for all scan paths.

• Technique adopted by LogicVision.

• Disadvantage – same input to each scan path, delayed by a cycle each time.