Hardware testing and design for testability
Introduction

A Digital System requires testing before and after it is manufactured

- **Level 1**: behavioral modeling and test benches
  - Check for correct design and algorithms
- **Level 2**: logic level testing
  - Check for correct logic and whether the design meets specifications
- **Level 3**: circuit level testing
  - Check for correct implementation and whether the timing is correct
- **Level 4**: post-manufacture testing
  - Check for defects
Introduction

- Digital systems should be designed so that they are easy to test

- Important to develop efficient testing methods
  - Design for testability (DFT)
  - Automatic test pattern generators (ATPG)
  - Built in Self Test (BST)

- Testing Combinational Logic
- Testing Sequential Logic
- Scan Testing
- Boundary Scan
Testing Combinational Logic
Testing Combinational Logic

- **Stuck-at-1 (s-a-1) and Stuck-at-0 (s-a-0)**
  - if the input to a gate is *shorted to ground*, it is **s-a-0**
  - if the input to a gate is *shorted to positive power supply*, it is **s-a-1**
  - if the input to a gate is an *open circuit*, it may act as **s-a-0** or **s-a-1**

- **For s-a-0, provide '1' as input**
  - To check if it will flip ‘1’ to ‘0’

- **For s-a-1, provide '0' as input**
  - To check if it will flip ‘0’ to ‘1’
Testing Combinational Logic: stuck-at faults

- Finding AND and OR Gate Stuck-at-faults

If $s\cdot a\cdot 0$ AND gate will produce 0

\[ \begin{array}{c}
\text{If } s\cdot a\cdot 0 \text{ AND gate will produce 0} \\
\text{(a)} \\
\text{(b)} \\
\text{(c)} \\
\text{(d)} 
\end{array} \]
We have no clue about the internal faults, we can provide inputs and look at outputs: the fault will propagate by setting specific pins to 1 or 0.
Testing Combinational Logic: stuck-at faults

- Multi-level circuit

![Multi-level circuit diagram]

<table>
<thead>
<tr>
<th>Test Vectors</th>
<th>Normal Gate Inputs</th>
<th>Faults Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>a b p c q r d s t u v w F</td>
<td>a1 p1 c1 v1 f1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>0 1 0 0 0 1 1 1 0 1 1 0 1 0</td>
<td>a0 b0 p0 q1 r0 d0 u0 v0 w0 f0</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>1 1 1 0 0 1 1 1 0 1 1 1 1 1</td>
<td>b1 c0 s1 t0 v0 w0 f0</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1 0 0 1 1 0 1 0 1 0 1 1 1 1</td>
<td>a0 b0 d1 s0 t1 u1 w1 f1</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>1 1 1 0 0 1 0 1 0 0 1 0 0</td>
<td>a0 b0 q0 r1 s0 t1 u1 w1 f1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1 1 1 1 1 0 1 1 0 0 1 0 0</td>
<td>a0 b0 q0 r1 s0 t1 u1 w1 f1</td>
</tr>
</tbody>
</table>
Testing Combinational Logic: bridging faults

- **Bridging fault occurs when two unconnected signal lines are shorted**

- Finding a minimum set of test vectors that will test all possible faults is very difficult
  
  - Use small set of testing vectors that can test most faults
  
  - Use algorithms and programs that will generate set of test vectors
Testing Sequential Logic
Testing Sequential Logic

- Testing Sequential Logic is more difficult than combinational logic

- Sequence of inputs and resulting outputs

- No access to state of flipflops

- Very large number of tests are required

- Use a small set of test sequences that can be adequate
Testing Sequential Logic

- Look for all possible state transitions and outputs

\[ X=010110011 \]

\[ Z=001111100 \]

(both solid and dashed)
Testing Sequential Logic

- **Find the distinguishing sequence**
  - Two states are distinguishable if an input sequence produces different output sequences
  - In this example, **11** is the distinguishing sequence

- **How to find distinguishing sequence?**
  1) Apply 1 to all states and see which states produce identical outputs
     
     1 → \{S₀, S₃\} → 0 and 1 → \{S₁, S₂\} → 1
  2) Apply 1 again to find distinct states within each group
     
     11 → S₀ → 01, 11 → S₃ → 00, 11 → S₁ → 11, 11 → S₂ → 10
Testing Sequential Logic

<table>
<thead>
<tr>
<th>Q1Q2</th>
<th>State</th>
<th>Next State $X = 0$</th>
<th>Next State $X = 1$</th>
<th>Output $X = 0$</th>
<th>Output $X = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>$S_0$</td>
<td>$S_0$</td>
<td>$S_1$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>$S_1$</td>
<td>$S_0$</td>
<td>$S_2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_3$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>$S_3$</td>
<td>$S_2$</td>
<td>$S_0$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Transition Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>001</td>
<td>S0 to S2</td>
</tr>
<tr>
<td>111</td>
<td>011</td>
<td>S0 to S3</td>
</tr>
<tr>
<td>1011</td>
<td>0101</td>
<td>S1 to S0</td>
</tr>
<tr>
<td>1111</td>
<td>0110</td>
<td>S1 to S2</td>
</tr>
<tr>
<td>11011</td>
<td>01100</td>
<td>S2 to S3</td>
</tr>
<tr>
<td>11111</td>
<td>01100</td>
<td>S2 to S3</td>
</tr>
<tr>
<td>110011</td>
<td>011110</td>
<td>S3 to S2</td>
</tr>
<tr>
<td>110111</td>
<td>011001</td>
<td>S3 to S0</td>
</tr>
</tbody>
</table>
Scan Testing
Scan Testing

- Instead of observing outputs, observe the state of flip-flops
- How can we observe the state of all flip-flops without using up a large number of pins on the IC?
  - Create parallel to serial shift-register out of the flipflops and use a single serial output pin
Scan Testing

- **Scan Path Test Circuit (Normal Operation)**

![Diagram of a scan path test circuit](image)
Scan Testing

- Scan Path Test Circuit (Test)

Diagram:
- SDI is scan data input
- SCK is system clock
- SDO is scan data output
- TCK is test clock

Circuit Components:
- \( X_1, X_2, \ldots, X_n \)
- \( Z_1, Z_2, \ldots, Z_m \)
- \( Q_1^+, Q_2^+, \ldots, Q_k^+ \)
- FF\(_1\), FF\(_2\), FF\(_k\)
- D1, D2
- C1, C2

Logic Functions:
- Combinational logic

Test Path:
- SDI → SDI is scan data input
- C1, C2
- D1, D2
- SCK → SCK is system clock
- TCK → TCK is test clock
- SDO → SDO is scan data output
Scan Testing

- Scan Path Test Circuit (Test)

- Scan in the test vector $Q_i$ values via SDI using test clock TCK
- Apply the corresponding test values to the $X_i$ inputs
- Verify output $Z_i$ values
- Apply one clock pulse SCK to store new values of $Q_i^+$ into the FFs
- Scan and verify Qi values by pulsing test clock TCK
- Repeat the above for each test vector
Scan Testing: Example

<table>
<thead>
<tr>
<th>$Q_1^+Q_2^+Q_3^+$</th>
<th>$Z_1Z_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>X$_1$X$_2$</td>
<td>00 01 11 10</td>
</tr>
<tr>
<td>101</td>
<td>010 110 011 111</td>
</tr>
</tbody>
</table>

- **Shift 101 using TCK via SDI:** $Q_3$ (LSB) First, $Q_1$ (MSB) last

Note that 101 is a unique pattern which does not rely on $X_1$ and $X_2$
### Scan Testing: Example

**Table:**

<table>
<thead>
<tr>
<th>(Q_1Q_2Q_3)</th>
<th>(X_1X_2=)</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>010</td>
<td>110</td>
<td>011</td>
<td>111</td>
<td>10</td>
<td>11</td>
<td>00</td>
<td>01</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- **TCK:**
- **SCK:**
- **X1:**
- **X2:**
- **SDI:**
- **SDO:**
- **Z1:**
- **Z2:**

*Read output (output at other times not shown)*

- **Apply input** \(X_1X_2=00\), **verify that** \(Z_1Z_2=10\)
Scan Testing: Example

<table>
<thead>
<tr>
<th></th>
<th>$Q_1^+Q_2^+Q_3^+$</th>
<th></th>
<th></th>
<th></th>
<th>$Z_1Z_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1Q_2Q_3$</td>
<td>$X_1X_2=$</td>
<td>00</td>
<td>01</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>101</td>
<td>010</td>
<td>110</td>
<td>011</td>
<td>111</td>
<td>10</td>
</tr>
</tbody>
</table>

*Read output (output at other times not shown)*

- Single pulse on **SCK** to advance circuit to state **010**
Scan Testing: Example

<table>
<thead>
<tr>
<th>( Q_1Q_2Q_3 )</th>
<th>( X_1X_2= )</th>
<th>( Q_1^+Q_2^+Q_3^+ )</th>
<th>( Z_1Z_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

TCK | SCK | X1 | X2 | SDI | SDO | Z1 | Z2 |

*Read output (output at other times not shown)*

- Verify that **SDO** shows the previous state **101**
### Scan Testing: Example

<table>
<thead>
<tr>
<th>$Q_1 Q_2 Q_3$</th>
<th>$X_1 X_2$</th>
<th>$Q_1^+ Q_2^+ Q_3^+$</th>
<th>$Z_1 Z_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>010</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>011</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCK</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SCK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDI</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SDO</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Z1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Read output (output at other times not shown)*

- Shift **101** using **TCK** via **SDI**: $Q_3$ (LSB) First, $Q_1$ (MSB) last

**Note that 101 is a unique pattern which does not rely on $X_1$ and $X_2$**
Scan Testing: Example

<table>
<thead>
<tr>
<th>Q₁Q₂Q₃</th>
<th>X₁X₂=</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td></td>
<td>010</td>
<td>110</td>
<td>011</td>
<td>111</td>
<td>10</td>
<td>11</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

*Read output (output at other times not shown)*

- **Apply input** \( X₁X₂=01 \), **verify that** \( Z₁Z₂=11 \)
### Scan Testing: Example

**Q1+Q2+Q3+** | **Z1Z2**
--- | ---
| **Q1Q2Q3** | **X1X2** | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10
101 | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01

- **TCK**
- **SCK**
- **X1**
- **X2**
- **SDI**
- **SDO**
- **Z1**
- **Z2**

*Read output (output at other times not shown)*

- **Single pulse on SCK to advance circuit to state 110**
### Scan Testing: Example

**Verify that SDO shows the previous state 010**

*Continue to the next state...*
Scan Testing for an IC

- Replace flip-flops with two-port flip-flops
- Connect them together in a chain and create Scan chain (shift register)
Scan Testing for Multiple ICs on single PCB

- Chain scan registers together
- The whole PCB can be scanned using single serial port
Boundary Scan
Boundary Scan

- Boundary Scan was developed to test a complex PCB with many ICs

- JTAG (Joint Test Action Group) developed the standard for boundary scan
  - Standard Test Access Port and Boundary-Scan Architecture
  - Multiple ICs can be linked together on PCB and tested using few pins on the edge
Boundary Scan

One cell of boundary scan shift register (BSR) is placed between each I/O and the core logic.

**TDI:** Test data input (serial input into the shift register)
**TCK:** Test Clock
**TMS:** Test Mode Select
**TDO:** Test Data Output (serial output from the shift register)
**TRST:** Test Reset (resets TAP controller and test logic)
Boundary Scan Cell

Diagram showing the Boundary Scan Cell with inputs and outputs:
- **Serial out** (to next cell or TDO)
- **Parallel in** (from core logic or input pin)
- **Serial in** (from TDI or previous cell’s TDO)
- **Parallel out** (to core logic or output pin)

Diagram details:
- TDO serial out
- Parallel in (from core logic or input pin)
- TDI serial in
- Load/shift
- Capture or shift
- Capture reg.
- Update reg.
- Update
- Normal/test
PCB with Boundary Scan ICs
Boundary Scan Architecture

IEEE Standard 1149.1 Compliant Device

TAP State Machine

TMS → Select Next State

TCK → Shift-IR/Shift-DR

TDI → TDO

Instruction Register

Instruction Decoder

Bypass[1] Register

IDCODE[32] Register

Boundary-Scan[n] Register

I/O I/O I/O I/O
TAP Controller State Machine

STATE MACHINEM

TEST-LOGIC-RESET

RUN-TEST/IDLE

SELECT-DR-SCAN

SELECT-IR-SCAN

CAPTURE-DR

SHIFT-DR

EXIT1-DR

PAUSE-DR

EXIT2-DR

UPDATE-DR

CAPTURE-IR

SHIFT-IR

EXIT1-IR

PAUSE-IR

EXIT2-IR

UPDATE-IR

UG332_C9_03_060906
Boundary Scan in Xilinx Spartan 3E PCB

Figure 12-15: Attaching a JTAG Parallel Programming Cable to the Board

Table 12-2: Cable Connections to J12 Header

<table>
<thead>
<tr>
<th>Cable and Labels</th>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>J12 Header Label</td>
<td>SEL</td>
</tr>
<tr>
<td></td>
<td>SDI</td>
</tr>
<tr>
<td></td>
<td>SDO</td>
</tr>
<tr>
<td></td>
<td>SCK</td>
</tr>
<tr>
<td></td>
<td>GND</td>
</tr>
<tr>
<td></td>
<td>VCC</td>
</tr>
<tr>
<td>JTAG3 Cable Label</td>
<td>TMS</td>
</tr>
<tr>
<td></td>
<td>TDI</td>
</tr>
<tr>
<td></td>
<td>TDO</td>
</tr>
<tr>
<td></td>
<td>TCK</td>
</tr>
<tr>
<td></td>
<td>GND</td>
</tr>
<tr>
<td></td>
<td>VCC</td>
</tr>
<tr>
<td>Flying Leads Label</td>
<td>TMS/PROG</td>
</tr>
<tr>
<td></td>
<td>TDI/DIN</td>
</tr>
<tr>
<td></td>
<td>TDO/DONE</td>
</tr>
<tr>
<td></td>
<td>TCK/CCLK</td>
</tr>
<tr>
<td></td>
<td>GND/GND</td>
</tr>
<tr>
<td></td>
<td>VREF/VREF</td>
</tr>
</tbody>
</table>
Boundary Scan in Xilinx Spartan 3E PCB

Does this look familiar?