Design for Test of Digital Systems
TDDC33

Erik Larsson
Department of Computer Science

Course Outline

- Introduction; Manufacturing, Wafer sort, Final test, Board and System Test, Defects, and Faults
- Test generation; combinational and sequential test generation
- Design-for-Test techniques; test point insertion, scan, enhanced scan
- Test data compression, Built-In Self-Test; Logic BIST and memory BIST
- System Chip Test; test architectures, test planning, test scheduling, and power constraints
- System Test and Boundary Scan

Problems

- Large ICs -> large test data volumes ->
  - Need of large tester memories
  - Long test times
  - Long test times because of long scan-chain. More scan-chains would reduce test time but requires more ATE channels
- Can we reduce cost of test?
  - Lower test time while keeping test quality
  - Reduce need of ATE requirement (memory and channels), which means we can increase the lifetime of a tester

Test Application

- Automatic Test Equipment (ATE)
  - Test stimuli (TS)
    - 0010100
    - 0110000
  - Expected responses (ER)
    - 1011001
    - 1101010
    - 0111011
    - 0100101
  - Produced responses (PR)
    - 0111011
    - 0100101

- Few scan-chains, test data in long bit strings. Many scan-chains, the reverse.
ATE-based testing

- Advantages with ATE-based testing
  - High quality test
  - Diagnosis possible
- Disadvantages with ATE-based testing
  - ATEs are expensive
  - Interface between ATE and device-under-test
  - Low throughput due to:
    - Long scan-chains
    - Low test shift speed
    - Limited number of channels
    - Limited tester memory
    - Only possible to test at manufacturing

Alternatives

- Test data compression
- Built-In Self-Test

Important aspects:
- Test stimuli
- Test responses

Test generation

Example: Create test for output connected to \( V_{dd} \)

Fault-free: \( X \)

Faulty: \( V_{dd} \)

Don't care bit

Analysis of Scan Test

- ATPG first creates test cubes
  - Merged for several faults; many unspecified don't care values
Analysis of Scan Test

- Then, convert cubes to vectors by filling don’t cares
  - Fault simulate vectors to mark off additional faults

<table>
<thead>
<tr>
<th>Scan chain</th>
<th>Logic Under Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 0 1 1</td>
<td>fault 1</td>
</tr>
<tr>
<td>1 1 1 0 0 0</td>
<td>fault 2</td>
</tr>
<tr>
<td>0 0 0 1 1 1</td>
<td>fault 3</td>
</tr>
</tbody>
</table>

= fill values  = care

Care-Bit Example

- Care-bit density varies from test to test
- Example: 15,624 scan cells, 1,764 tests

Source: O. Farnsworth et al., IBM

Test data compression

For all modules
- Fill test stimuli don’t care bits
- Simulate to find test responses
- Compress test stimuli
- Design decompression logic
- Design compactor

<table>
<thead>
<tr>
<th>Vector</th>
<th>Stimuli</th>
<th>Expected response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1xx xx0 xxx} {x0x xxx x11}</td>
<td>{x0x xxx x11}</td>
</tr>
<tr>
<td>2</td>
<td>{xx1 xx1 xxx} {1xx 0xx xxx}</td>
<td>{1xx 0xx xxx}</td>
</tr>
<tr>
<td>3</td>
<td>{0x0 xxx xxx} {xxx 1xx xxx}</td>
<td>{xxx 1xx xxx}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vector</th>
<th>Stimuli</th>
<th>Expected response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{111 110 000} {x0x xxx x11}</td>
<td>{101 100 011}</td>
</tr>
<tr>
<td>2</td>
<td>{001 111 111} {1xx 0xx xxx}</td>
<td>{100 010 010}</td>
</tr>
<tr>
<td>3</td>
<td>{000 000 000} {xxx 1xx xxx}</td>
<td>{010 101 100}</td>
</tr>
</tbody>
</table>
Test data compression

For all modules
- Fill test stimuli don’t care bits
- Simulate to find test responses
- Compress test stimuli
- Design decompression logic
- Design compactor

\[
\begin{array}{c|c|c}
\text{vector (stimuli)} & \text{vector (expected response)} & \text{vector (compressed stimuli)} \\
\hline
1 & \{111\ 110\ 000\} & \{101\ 100\ 011\} \\
2 & \{001\ 111\ 111\} & \{100\ 010\ 010\} \\
3 & \{000\ 000\ 000\} & \{010\ 101\ 100\} \\
\end{array}
\]

Test data compression

For all modules
- Fill test stimuli don’t care bits
- Simulate to find test responses
- Compress test stimuli
- Design decompression logic
- Design compactor

\[
\begin{array}{c|c|c}
\text{vector (stimuli)} & \text{vector (expected response)} & \text{vector (compressed stimuli)} \\
\hline
1 & \{111\ 110\ 000\} & \{101\ 100\ 011\} \\
2 & \{001\ 111\ 111\} & \{100\ 010\ 010\} \\
3 & \{000\ 000\ 000\} & \{010\ 101\ 100\} \\
\end{array}
\]

Test data compression

Few ATE channels and many (short) scan-chains lead to shorter test times

Faulty module!
Test Compression Categories

- Test Stimulus Compression
  - Code-based schemes
  - Linear decompression
  - Broadcast scan
- Test Response Compaction
  - Space compaction
  - Time compaction
  - Space/time compaction
  - Handling of Xs

Code-based Schemes

- **Fixed-to-Fixed**
  - The original test stimuli are partitioned into \(n\)-bit blocks to form symbols. The symbols are then encoded with code words. Each code word have \(b\) bits. (\(b < n\) to get compression). Dictionary codes.
- **Fixed-to-Variable**
  - The original test stimuli are partitioned into \(n\)-bit blocks to form symbols. These symbols are encoded using variable-length code words. Statistical coding such as Huffman code.
- **Variable-to-Fixed**
  - The original test stimuli are partitioned into variable length symbols, and the codewords are each \(b\)-bits long. Ex. Run-Length Code
- **Variable-to-Variable**
  - The test stimuli are partitioned into variable length symbols, and the symbols are partitioned into variable length code words. Ex. Golomb Code.

Dictionary Code (Fixed-to-Fixed)

```
Tester  Dictionary  n
  |       |       |   |
  |   b   | Scan chain 0 |   |
  |   | Scan chain 1 |   |
  |   | Scan chain 2 |   |
  |   | Scan chain n |
```

Driving Multiple Scan Chains

- \(TD = \{t_1, t_2, ..., t_n\}\)
- \(l\) : length of a scan chain (bits)
- \(m\) : number of scan chains
- Data word : \(m\)-bit long

```

```

ATE  Decoder

0111000
0101000
111001

MISR

```
Dictionary Generation

- Goal: include as many words as possible in the dictionary
- Limit the size of the dictionary
  - Selective entries
  - Find compatible words and properly map the Xs in the test set
- Compatibility
  - If any pair of words in a set are compatible, this set can be mapped to a single word

```
0 1 0 0
1 1 1 1
0 1 0 0
0 0 0 0
```

- Problem: find \(|D|\) disjoint subsets containing pair-wise compatible words, such that the total number of the words in these subsets is maximized
- Clique partitioning problem
  - NP-complete
  - Heuristic solution

Test Response Analysis: Test Data Compression and BIST

- How store/analyze test responses on-chip?
- Compression – does not lose information
- Compaction – does lose information
- Compaction alternatives:
  - Parity check
  - One counting
  - Transition counting
  - Signature analysis

Parity check is simple but lossy (in respect to information).
One counting

- Example of "one counting". The result from the fault free circuit is "5" and the result from a faulty circuit is 8.

Signature analysis

- Signature analysis is the most commonly used test response compaction technique. It was proposed by N. Benowitz et al., "An advanced fault isolation system for digital logic", IEEE Transactions on Computers, Vol. C-24, No. 5, pp. 489-497.
- Serial-input signature register (SISR) and parallel signature analysis using multiple-input signature register (MISR).

Transition counting

- Count the number of transitions and compare at the end of the testing with the correct number of transitions for a fault-free circuit.

Serial signature analysis

- A single-input signature register (SISR) takes one input and compacts it to an L-bit sequence.
- Example, L=4, XOR-network gives \( f(x)=1+x+x^4 \).
- General form: \( f(x)=a_0x^0+a_1x^1+a_2x^2+a_3x^3+a_4x^4 \). And with \( a_0=a_1=a_3=1 \) and \( a_2=a_4=0 \), \( f(x)=1+x+x^4 \).
Parallel signature analysis

Compaction Options

- MISR (Multiple-Input Signature Register)
  - Very high compression ratio
  - BUT: does not tolerate x-states in test responses
    - May require product logic re-design OR
    - X-state gating logic
  - Error information loss can impact diagnostics
  - Combinational space compaction
- XOR-network
  - Can be x-state tolerant (e.g., X-Compact from Intel)
  - Less compression
  - Possibly better for diagnostics

Response Compaction: Motivation

- Compaction of test responses necessary for verifying the test response
- Store compacted response called signature and compare to known fault-free signature

\[ N >> W \]

Space Compaction

- Compress test response in the spatial dimension
- Compress k-bit-wide response stream to q-bit signature stream (\( k >> q \))
- Space compactor is a combinational circuit
Time Compaction

- Compress test responses in the temporal dimension
- Compress m-bit (or word) test response stream to q-bit (or word) signature stream
- Time compactor is a sequential circuit (finite-state machine)

\[ b_1 b_2 \ldots b_m \rightarrow s_1 s_2 \ldots s_q \]

Aliasing Problem

- Loss of fault information is inevitable in response compaction
- Aliasing occurs when fault-free response and faulty response map to the same signature, i.e. fault is not detected (masked) by the compaction process
- Aliasing reduces fault coverage (even when complete test sets are used) and makes fault coverage hard to determine

Serial signature analysis

- Assume: \( f(x) = 1 + x + x^4 \). Start pattern (seed): \{0000\}. Fault-free test response sequence \( M = \{10011011\} \), gives \( R = \{1011\} \)
- Faulty test response: \( M' = \{10001011\} \) gives \( R' = \{1110\} \), and \( M'' = \{10011011\} \) gives \( R'' = \{1011\} \)
- The faulty response \( M' \) results in \( R' \) which is different from \( R \) while the response \( R'' \) from \( M'' \) is not different from \( R \).
- The fault detection problem (\( M \) and \( M' \)) is called aliasing.
Unknows (X)

- Output from analog blocks
- Memories and non-scan storage elements
- Combinational feed-back loops
- Asynchronous set/reset
- Tristate buses
- False paths (not normal functional paths)
- Critical paths
- Multi-cycle paths
- Floating ports
- Bidirectional I/O

X-handling

- X-Blocking; block X’s at source (where it is generated)

- X-Masking; block X’s in front of compactor

MISR Gate

- Requires masking data
  - msk=’0’ blocks x-states
  - msk=’1’ propagates test responses
- Trade-offs required
  - One mask value for all chains (simple)
  - Vector with unique mask value for each chain (complex)

X-State Gating
Built-In Self-Test

- Key component to discuss:
  - Test pattern storage/generation
  - Test stimuli storage/generation
  - Test response analysis
  - Test control

- In a non-BIST environment:
  - Test generation is performed by ATPG; a tool such as FlexTest can generate deterministic test patterns.
  - Test stimuli and expected test responses are stored in the ATE, and the ATE controls the testing and performs test evaluation.

On-chip/off-chip

Test Application

Let ATE start tests and analyze final results

Test Application
BIST Concept

- Integrate Test Stimulus Generation and Test Response Processing Into Each Chip
- Use simple interface to access on-chip BIST Hardware
- Simple test protocol: load initialization data, start BIST, apply clocks, unload and compare final result

Test Pattern Generation

- How store/generate test patterns on-chip?
- Deterministic test patterns
- Exhaustive test patterns
- Pseudo-exhaustive/random test patterns
- Random test patterns

Commercial tools usually make use of a random test generation for 60-80% of the faults (easy to detect) and deterministic test generation for the remaining part (hard to detect)

Test Pattern Generation

- Exhaustive test generation; simple hardware (a counter), 100% fault coverage but too time consuming
- Deterministic test generation; high fault coverage but requires ATE for test pattern storage
- Pseudo-exhaustive test generation using Linear-Feedback Shift-Registers (LFSR)
Random Pattern Resistant Faults

- The effectiveness of a test is given based on the test’s fault coverage, length, and hardware/data storage requirement.
- Probability to create a 1 at the output: $1/2^n$ where $n$ is the number of inputs. $n=2; P=0.25, n=4; P=0.0625$

Random pattern resistance

- Some logic takes too long to test with pseudo-random patterns
  - Too many specific input bit values are required
  - Too many pseudo-random trials needed to achieve the required value combination

STUMPS: Self-testing using MISR and parallel shift register sequence generator

Built-In Self-Test (BIST)
Test-per-clock/scan (cycle)

- Test generation:

![Test generation diagram]

- Response compaction:

![Response compaction diagram]

- Make register test generator/compactor -> test-per-clock

Test-per-clock

- Change registers so they become test sources and test sinks. Some registers can act as test source and test sink (but not at the same time).
- Test controller determines when a register should be test source or test sink.
- The test controller determines the test schedule; when each logic block should be tested.
- Optimization problem to find lowest test time without violating constraints.

Testing Memories

- Memory testing is an increasingly important issue
- RAMs are key components for electronic systems
- Memories represent about 30% of the semiconductor market
- Embedded memories are dominating the chip yield
- Memory testing is increasingly difficult
- Growing density, capacity, and speed
- Emerging new architectures and technologies
- Embedded memories: access, diagnostics & repair, heterogeneity, custom design, power & noise, scheduling, compression, etc.
- Cost drives the need for more efficient test methodologies and automation
- Failure analysis, fault simulation, ATG, diagnostics BIST/BIRA (Built-In Redundancy Analyzer)/BISR (built-in self repair)
Testing Memories

- Data Inputs
  - Write drivers
  - Memory array
  - Decoders
  - Address Inputs
  - Sense Amplifiers
  - Data Outputs

Fault Models for Memory Test

- Problems
  - Classical fault models are not sufficient to represent all important failure modes in RAM
  - Sequential ATPG is not possible for RAM
  - Scan is too expensive
  - But,
  - Memories are, different to logic, composed of regular structures (memory cells)
  - Functional fault models are commonly used for memories

Fault Models for Memories

- Stuck-At Fault (SAF)
  - Cell (line) SA0 or SA1
  - A stuck-at fault (SAF) occurs when the value of a cell or line is always 0 (a stuck-at-0 fault) or always 1 (a stuck-at-1 fault)
  - A test that detects all SAFs guarantees that from each cell, a 0 and a 1 must be read.

- Transition Fault (TF)
  - Cell fails to transit from 0 to 1 or 1 to 0 in specified time period
  - A cell has a transition fault (TF) if it fails to transit from 0 to 1 or from 1 to 0.

- Bridging Fault (BF)
  - A bridging fault (BF) occurs when there is a short between two cells
    - AND-type BF
    - OR-type BF

- Stuck-Open Fault (SOF)
  - A stuck-open fault (SOF) occurs when the cell cannot be accessed due to, e.g., a broken word line
  - A read to this cell will produce the previously read value

- Coupling Fault (CF)
  - A coupling fault (CF) between two cells occurs when the logic value of a cell is influenced by the content of, or operation on, another cell
Fault Models for Memories

- Address-Decoder Fault (AF)
  - An address decoder fault (AF) is a functional fault in the address decoder that results in one of four kinds of abnormal behavior:
    - Given a certain address, no cell will be accessed
    - A certain cell is never accessed by any address
    - Given a certain address, multiple cells are accessed
    - A certain cell can be accessed by multiple addresses

State Diagram Representation

- Fault-free memory cell
- SA0
- SA1
- Rising TF
- Falling TF

Memory Test

- A memory test is a finite sequence of test elements
  - A test element contains a number of memory operations (access commands)
    - Data pattern (background) specified for the Read and Write operation
    - Address (sequence) specified for the Read and Write operations
  - A march test algorithm is a finite sequence of march elements
    - A march element is specified by an address order and a finite number of Read/Write operations

March Test Notation

- \(\uparrow\): address sequence is in the ascending order
- \(\downarrow\): address changes in the descending order
- \(\uparrow\uparrow\): address sequence is either \(\uparrow\) or \(\downarrow\)
- \(r\): the Read operation
  - Reading an expected 0 from a cell (r0); reading an expected 1 from a cell (r1)
- \(w\): the Write operation
  - Writing a 0 into a cell (w0); writing a 1 into a cell (w1)
- Example (MATS+): \(\{w0;\uparrow(r0,w1);\downarrow(r1,w0)\}\)
Zero-One Algorithm (MSCAN)

- Zero-One Algorithm [Breuer & Friedman 1976]
- Also known as MSCAN
- For SAF
- Solid background (pattern)
- Complexity is 4N

\{ \text{w(0)}; \text{r(0)}; \text{w(1)}; \text{r(1)} \}

Galloping Pattern (GALPAT)

- Complexity is 4N**2
- All AFs, TFs, CFs, and SAFs are located

1. Write background 0;
2. For BC = 0 to N-1
   \{ Complement BC;
   For OC = 0 to N-1, OC != BC;
      \{ Read BC; Read OC; \}
      Complement BC;
   \} 
3. Write background 1;
4. Repeat Step 2;

Classic Test Algorithms

- MSCAN, complexity 4N: \{ \text{w(0)}; \text{r(0)}; \text{w(1)}; \text{r(1)} \}
- Checkerboard Algorithm, complexity 4N
- Galloping Pattern (GALPAT), complexity 4N2
- Sliding (Galloping) Row/Column/Diagonal (Based on GALPAT, but instead of a bit, a complete row, column, or diagonal is shifted), complexity 4N1.5
- Butterfly Algorithm, complexity 5NlogN
- Moving Inversion (MOVI) Algorithm, for functional and AC parametric test, Functional (13N) and Parametric (12NlogN)
- Surround Disturb Algorithm

N=number of bits

Classic Test Algorithms

- Zero-one and checkerboard algorithms do not have sufficient coverage
- Other algorithms are too time-consuming for large RAM
- Test time is the key factor of test cost
- Complexity ranges from N2 to NlogN
- Need linear-time test algorithms with small constants
- March test algorithms
### Test Time

<table>
<thead>
<tr>
<th>Memory size</th>
<th>Complexity of test algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
</tr>
<tr>
<td>1M</td>
<td>0.01s</td>
</tr>
<tr>
<td>16M</td>
<td>0.16s</td>
</tr>
<tr>
<td>64M</td>
<td>0.66s</td>
</tr>
<tr>
<td>256M</td>
<td>2.62s</td>
</tr>
<tr>
<td>1G</td>
<td>10.5s</td>
</tr>
<tr>
<td>4G</td>
<td>42s</td>
</tr>
<tr>
<td>16G</td>
<td>2.8m</td>
</tr>
</tbody>
</table>

### March Test

- **Zero-One (MSCAN) [Breuer & Friedman 1976]**
  \[\{\downarrow (w_0); \downarrow (r_0); \downarrow (w_1); \downarrow (r_1)\}\]
- **Modified Algorithmic Test Sequence (MATS) [Nair, Thatte & Abraham 1979]**
  \[\{\downarrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1)\}\]
- **MATS+ [Abadir & Reghbati 1983]**
  \[\{\downarrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0)\}\]
- **Marching 1/0 [Breuer & Friedman 1976]**
  \[\{\uparrow (w_0); \downarrow (r_0, w_1, r_1); \downarrow (r_1, w_0, r_0); \downarrow (r_1, w_0, r_1); \downarrow (r_0, w_1, r_1)\}\]
- **MATS++ [Goor 1991]**
  \[\{\downarrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0, r_0)\}\]
- **March X**
  \[\{\downarrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, r_0); \downarrow (r_0)\}\]
- **March C [Marinescu 1982]**
  \[\{\downarrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0)\}\]
- **March C+ [Goor 1991]**
  \[\{\downarrow (w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0); \downarrow (r_0, w_1); \downarrow (r_1, w_0)\}\]

### Memory BIST

- **Memories are one of the most universal cores**
- **In Alpha 21264, cache RAMs represent 2/3 transistors and 1/3 area; in StrongArm SA110, the embedded RAMs occupy 90% area [Bhavsar, ITC-99]**
- **In average SOC, memory cores represent more than 90% of the chip area by 2010 [ITRS 2000]**
- **Embedded memory testing is increasingly difficult**
  - High bandwidth (speed and I/O data width)
  - Heterogeneity and plurality
  - Isolation (accessibility)
  - AC test, diagnostics, and repair (The primary difference between DC stuck-at scan and AC scan is the clocking during the sample cycle. For AC scan, only the launch and capture of a signal need to be at-speed. In DC scan, all are slow)
- **BIST is considered the best solution**

### Memory BIST (MBIST)

- **Methodology**
  - Processor-based BIST
    - Programmable
  - Hardwired BIST
    - Fast
    - Compact
- **Interface**
  - Serial (scan, 1149.1)
  - Parallel (embedded controller; hierarchical)
- **Patterns (address sequence)**
  - March
  - Pseudorandom
Controller and Sequencer

- Controller
  - Microprogram
  - Hardwired
  - Shared CPU core
  - IEEE 1149.1 TAP
- Pattern Generator
  - Counter
  - LFSR
  - LUT

Summary

- Test data compression
  - Explore high number of don’t care bits to compress test data
  - Add decompressor logic to system
  - Analyze response in system
- Built-In Self-Test (BIST)
  - Store/generate test vectors in system
  - Analyze response in system
  - Logic BIST and Memory BIST
  - Testing can be performed in-field