Outline

- Sequential circuits without feedback
  - BIST
- Sequential circuits with feedback
  - DFT: scan
  - Extended D-algorithm
  - Functional testing of FSMs
- Testing semiconductor RAMs
  - Organization, fault model, testing
- Incremental testing for complex systems
Testing Sequential Circuits

- **Test for**
  - Functionality
  - Timing (components too slow, too fast, not synchronized)

- **Parts:**
  - Combinational logic: faults: stuck 0/1, delay
  - Flip-flops: faults: input, output stuck 0/1, delay no latch 0/1 capability

- **Cases:**
  - Without feedback
  - With feedback

---

### Testing Sequential circuits without feedback

1. Generate tet vectors for the combinational circuit.
2. Place a vector in A
3. Examine response in B
4. 2,3 can be done using
   a. Machine instructions
   b. Microinstructions
   c. Specialized hardware

---

BIST (Built-in self-test)

- Generator generates pseudorandom vectors. Often an ALFSR.
- Signature analyzer compresses all successive responses into a signature. Usually an LFSR.
- Compared with known good signature.
- Aliasing probability: prob. that a bad circuit can result in good signature. Very small.
- BILBO: PIPO, SA functions

ALFSR: autonomous linear feedback shift register. Better generators include our antirandom test generator.

Sequential Circuits with feedback

- Example: processor control unit
  - Input: status, opcode
  - Output: control lines
- Structural testing:
  - How to obtain a desired (x,y) vector
  - How to propagate error
- Functional testing:
  - Prove presence of all states
  - Prove all transitions/outputs correct
Sequential Circuits with feedback: Approaches

- Consider the sequential behavior:
- Design for testability: feedback-less during testing
- Scan Design: modes
  - Normal: parallel in/out
  - Test: serial in/out
- Sequence
  - Scan a vector: test mode
  - Latch response: normal mode
  - Scan response out: test mode
- Chain too long
  - Multiple chain
  - Partial scan

D-Algo: Sequential Circuits

Equivalent iterative array model:

Psuedo-flipflop is a combinational block.
D-Algo.: Seq Circuit Extension

- One time-frame for each clock-period. Converts problem to combinational. Only $4^n$ distinct states possible (boolean values 0,1, D, $\overline{D}$).
- Assume faults in the combinational logic only.
- Assume initial state given.
- Procedure:
  - Get iterative array model, initial state
  - D-algo. to drive D or $\overline{D}$ towards an output $z(k)$
  - Terminate when D or $\overline{D}$ at an output.
    - a state is repeated.
    - when $k>4^n$.

Seq D-Algo: Example

- Start: fault $\alpha$ s-a-1, initial state
  - $y_1$, $y_2=$(0,0)
- Time frame 0: Since $y_2$ already 0, line $\alpha = \overline{D}$
  - For 4= $\overline{D}$ need $x(0)=1$
  - Since 3=0 already, $D_2= \overline{D}$.
  - Also $D_1=0$
  - Next state $Y(1)=$(0, $\overline{D}$), $z(0)=1$
Seq D-Algo: Ex (pt2)

- Time frame 0:
  - Next state $Y(1) = (0, \overline{D})$, $z(0) = 1$
- Time frame 1: error can not yet be propagated to $z$.
  - For $D_1 = \overline{D}$ need $x(1) = 1$. Gives $D_2 = \overline{D}$. $Y(2) = (\overline{D}, \overline{D})$,
- Time frame 2: Since $\overline{y}_1 = D$, choosing $x(2) = 1$ gives $z = D$.
- Answer: Test sequence $(1, 1, 1)$ for initial state $(0, 0)$.

Initialization Problem

- Power-up: state undefined
- How to get FSM in a known state?
  1. Applying an input sequence
  2. Resetting to an initial state
- Resetting always used in practice
  - using a reset/clear line
  - Computers: power-on sequence to initialize PC, SP, interface registers etc.
- Problem with reset mechanism: highly testable
  - Hence we can neglect it
Functional Testing of FSMs

- **Given:** state table  **Objective:** confirm it is obeyed
- **Approaches:**
  1. Only outputs observable: “Checking sequence”
     a. Prove all states exist
     b. For all inputs for each state, these are correct
        - Next state
        - Outputs
          - Complex and lengthy (we’ll skip it for now)
  2. State observable:
     1. Much easier
     2. Euler path, if it exists, can minimize testing

Semiconductor RAMs

- **Types:** Static (cell is flip-flop), dynamic (charged capacitor)
- **Organization:** assume m address lines, assume 1 bit wide chip
- **External:** \( \text{words} \times \text{bits/word} = 2^m \times 1 \)
- **Internal:** \( m = r + q \)
  - r lines for word-line selection
  - q lines for bit-line selection
RAM Chip Organization

- Read/write
- Data bit in
- Dec
- Bit-line
- Coincident selection
- Data bit out
- Word-line
- Cell
- Selector
- r
- q

RAM Fault Types

- **Soft errors**: temporary bit errors:
  - require concurrent detection
  - Corrective action: fault tolerance (retry or encoding)
- **Hard defects**
  - Logical ✓
  - Parametric: access-time, leakage current etc.
  - Tested during
    - Manufacturing
    - Power-up etc
- **Highly regular design**: allows systematic optimization
Functional Fault Model By Abraham, Reddy etc

- One or more cells stuck-at 0 or 1.
  - Defects in cells
  - Read/write electronics stuck-at 0/1
  - Decoding faults
- One or more cells fail to do 0→1 or 1→0.
  - Defects in cells
- One or more pair of cells “coupled”
  - i,j coupled= 0→1 or 1→0 transition of i changes the state of j
    (not necessarily vice versa)
  - Decoding faults, read/write electronics coupling
- More than one cells accessed during a read or write
  - Decoding faults

RAM Testing: “March”

- Step 1: Write all 0's.
- Step 2: For i=0,...,n-1 do
  Read \( c_i (=0) \)
  \( c_i \leftarrow 1 \)
  Read \( c_i (=1) \)
- Step 3: For i=n-1,...,0 do
  Read \( c_i (=1) \)
  \( c_i \leftarrow 0 \)
  Read \( c_i (=0) \)
- Step 4: Repeat steps 1-3 for complementary pattern.

Complexity: 12n
Testing for Coupling

- To test for coupling between all cell pairs \((i,j)\)
  - Test bit \(i\)
  - Test bit \(j\)
  - Do some sequence of read, write operations for \(i\) & \(j\)
  - Repeat for all \(i\)
  - Repeat for all \(j\)

- Complexity:
  - All \(i\) and \(j\): \(\text{const} \times n^2\)
  - Same row & column only: \(k \times n \times (2\sqrt{n}) = k \times n^{1.5}\)

- Careful sequencing can reduce complexity
  - Suk & Reddy’s test satisfies fault model with \(14n\).

Testing Complex Systems

- Approach:
  - Assume a fault model for each component
  - Assume a system model that takes into account interaction of components

- Device a testing strategy such that these are adequately tested
  - each individual component
  - Interaction of components

- A component may be a
  - Physical component
  - Segment of the functionality
Incremental Testing

- Partition into layers such that layer i can be exercised using only layers 0, .. i-1.
- Test components in each layer in the sequence L0, L1,..Ln.
- Layering may require
  - Assumptions
  - Disabling feedback during testing
- Proofs of complete coverage can be constructed.
- Fault isolation can be done.

Incremental testing: Example: Processor Based systems

- Self-test processor:
  - Basic instructions
  - Addressing modes
  - Complex instructions
- Test Memory system and buses using processor
- Test I/O devices/ports
- Test peripheral devices
- Test software integrity