Outline

• Introduction and Terminology
• Test Generation for Single Stuck-At Faults
• Functional Testing
  • Software Testing
• Design For Test (DFT)
• Built-In Self-Test (BIST)

Software Testing

• Goal: stress-test the software in order to find bugs more quickly than would occur for typical inputs
• This is a job that many entry-level software engineers start with
  – Trying to break software forces engineer to better understand it
• Software testing tends to be almost entirely functional testing

When Do We Stop Testing?

• Comprehensive software testing is impossible problem for almost all real software
  – Must try to find as many bugs as possible, while realizing that it’s impossible to find all of them
• So when do we stop testing?
  – When rate of bug detection decreases below threshold
• Threshold depends on software
  – People rely more on Microsoft Windows than they do on a small utility program → Microsoft should test more thoroughly
• Key idea: if you’re still finding lots of bugs, it’s likely that there are still a bunch more that you haven’t found yet

Software Testing Techniques

• Test software in parallel
  – Get many people to test it for you → release a beta test version of cool software and people will test it for free
  – This is how I helped to debug a simulator called Simics
• Provide software with worst-case or unusual inputs
  – Input a number when prompted for a string
  – Input a character when asked for a floating point number
  – When asked how many entries you’ll need in a database, give it a huge number and see if that breaks it (e.g., because the database stupidity tries to allocate all that memory immediately)
  – When configuring a simulator, give it unusual parameters
  – Etc.
Testing Non-deterministic Software

- **Non-determinism**: If a program won't behave the same exact way each time it is run with the same inputs.
- With non-determinism, it is very hard to reproduce a bug, so a test that finds it once may not be able to find it again.
  - You may not be able to tell the programmers what to fix.

Outline

- Introduction and Terminology
- Test Generation for Single Stuck-At Faults
- Functional Testing
- Software Testing
- Design For Test (DFT)
- Built-In Self-Test (BIST)

Design for Test (DFT)

- How do we design a system so that it is has good testability?
- We must improve the system's:
  - **Controllability**: Ability to control the value of lines in the circuit
    - 1-controllability: Ability to set line to 1
    - 0-controllability: Ability to set line to 0
  - **Observability**: Ability to observe the values of lines in the circuit
- We're going to cover two classes of approaches
  - Ad hoc schemes
  - Systematic schemes that use scan registers

Ad Hoc Scheme 1: Test Points

- We can add test points to the circuit
  - Control points
  - Observation points
- Control points are added inputs that we use to improve controllability
- Observation points are added outputs that we use to improve observability
Control Point Example

- Adding new input C enables us to better control line X
  - Can easily set X to 1 (by setting C to 0)

Observation Point Example

- Adding new output D enables us to easily observe line Y

Ad Hoc Scheme 2: Initialization

- For sequential circuits, it is important to be able to get the circuit into a known state
  - It’s tough to test something that’s in an unknown state

Ad Hoc Scheme 3: Partitioning

- For large circuits or circuits with many states, it is easier to test them if we break them up into smaller sub-circuits
  - As functions of circuit size, testing time and complexity are greater than linear
- In general, we can partition circuits by adding multiplexor logic and new controllability inputs:
Ad Hoc Scheme 4: Avoiding Redundancy

- Recall that redundancy in circuits can lead to undetectable faults
- Two types of redundancy
  - Intentional: used for fault tolerance (good!)
  - Unintentional: most circuits have redundancy unless they've been optimized specifically to eliminate it (not good)
- We'd like to get rid of unintentional redundancy
- We'd like to keep intentional redundancy, but this means that we'll need to add test points to make sure that we can detect faults in the redundant logic

Systematic DFT: Scan Registers

- Test points are great, but they add lots of pins to a chip (both for inputs and outputs)
  - Pins are an expensive, scarce resource

- One way to add test points without adding so many pins is to use scan registers (shift registers)
  - Sequentially scan test vector into scan register via a single pin, and then apply shift register contents to control points
  - Similarly, scan observation points into scan register and read out sequentially via a single pin

- Advantage
  - Fewer pins!
- Disadvantages
  - Need scan hardware
  - Sequential \(\rightarrow\) slower!!

- Commercial chips have scan registers
  - JTAG and IEEE standards for scan register design

- There are many, many different types of scan registers and scan design methodologies, but we will not cover these in this course
Goals of BIST

- Want system to be able to test itself
- Why?
  - Generation of test vectors within chip eliminates pins used for testing \( \rightarrow \) cheaper and higher bandwidth
  - Enables in-field testing (after deployment)
  - If hard fault detected, can notify user and/or reconfigure system to handle the fault

Off-line vs. Online BIST

- Off-line
  - Testing while system is not performing its normal functions
- On-line or concurrent
  - Testing during operation
  - Also known as dynamic verification

A Paradox?

- If the system uses its own BIST hardware to test itself, how does it test this BIST hardware?
- Must be able to boot-strap based on assumption that a certain amount of the system (called the hardcore, for reasons I can’t explain) is operational
  - Remember this concept from the Teramac paper??
- Examples of hardcore circuitry
  - Power and ground
  - Clock distribution
  - Test generation logic
- Probably need to do external testing to determine if hardcore is faulty
Hardware for Generating Pseudo-Random Tests

- **Linear feedback shift register (LFSR)**
  - Shift register with XOR-based feedback loop
  - \( C_i \) are feedback coefficients
  - \( C_i = 1 \) implies that a connection exists
- Based on initial state and choice of \( C_i \), LFSR can generate long pseudo-random sequences (up to \( 2^n - 1 \), where \( n \) is number of stages in LFSR)

---

**Outline**

- Introduction and Terminology
- Test Generation for Single Stuck-At Faults
- Functional Testing
- Software Testing
- Design For Test (DFT)
- Built-In Self-Test (BIST)

---

**Flavors of BIST**

- **Exhaustive**
  - Generate and apply all \( 2^n \) possible input combinations
- **Pseudo-random**
  - Similar to random ATG, but tests generated with LFSRs
- **Pseudo-exhaustive**
  - Partitions system and exhaustively tests these smaller sub-circuits