Design for Test

- Definition:

  Design for test (DFT) refers to those design techniques that make test generation and test application cost-effective.

- Types:
  - Design for Testability
    - Enhanced access
  - Built-In Self-Test
    - Internal test generation and response evaluation
  - Self-Checking Circuits
    - Error detection and correction codes
Increasing Test Costs

• Increasing chip logic-to-pin ratio – harder observability

• Increasingly dense devices and faster clocks

• Increasing test generation and application times

• Increasing size of test vectors stored in ATE

• Hard testability insertion – designers unfamiliar with gate-level logic, since they design at behavioral level

• *In-circuit testing* no longer technically feasible

• Circuit testing cannot be easily partitioned
Increasing Quality Requirements

- 98% single stuck-at fault coverage
- 100% interconnect fault coverage
- Reject ratio – 1 in 100,000
Built-In Self-Test Motivation

Requirements for the ATE:
- high frequencies
- accuracy
- memory
- noise immunity

High Cost

tester functions moved (integrated) into the chip or board.
Define Built-In Self-Test

- Implement the function of automatic test equipment (ATE) on circuit under test (CUT).

- Hardware added to CUT:
  - Pattern generation (PG)
  - Response analysis (RA)
  - Test controller

![Diagram of BIST system]

ATE

- Stored Test Patterns
- Pin Electronics
- Test control HW/SW
- Comparator hardware

CUT

- Stored responses

Test control logic

- CK
- PG
- RA

Go/No-go signature

BIST Enable
BIST Benefits

• Faults tested:
  ▪ Single combinational / sequential stuck-at faults
  ▪ Delay faults
  ▪ Single stuck-at faults in BIST hardware

• BIST benefits
  ▪ Reduced testing and maintenance cost
  ▪ Lower test generation cost
  ▪ Reduced storage / maintenance of test patterns
  ▪ Simpler and less expensive ATE
  ▪ Can test many units in parallel
  ▪ Shorter test application times
  ▪ Can test at functional system speed
Additional BIST Motivation

• Useful for field test and diagnosis (less expensive than a local automatic test equipment)

• Hardware BIST benefits:
  ▪ Lower system test effort
  ▪ Improved system maintenance and repair
  ▪ Improved component repair
  ▪ Better diagnosis
Economics – BIST Costs

- Chip area overhead for:
  - Test controller
  - Hardware pattern generator
  - Hardware response compacter
  - Testing of BIST hardware
- Pin overhead -- At least 1 pin needed to activate BIST operation
- Performance overhead – extra path delays due to BIST
- Yield loss – due to increased chip area or more chips in system because of BIST
- Reliability reduction – due to increased area
- Increased BIST hardware complexity – happens when BIST hardware is made testable
BIST Process

- **Test controller** – Hardware that activates self-test simultaneously on all PCBs
- Each board controller activates parallel chip BIST
  Diagnosis effective only if very high fault coverage
Diagnosis

• BIST must be implemented at all (or most) levels
  – The more the better

• Test is performed in one level at a time and reported to the higher level
• Note: BIST cannot test wires and transistors:
  - From PI pins to Input MUX
  - From POs to output pins
Some Definitions

- **CUT** – *Circuit-under-test*
- **TPG** – Hardware *test pattern generator*
- **ORA** – Output Response Analyser
- **Signature** – Any statistical circuit property distinguishing between bad and good circuits
Pattern Generation

• Store in ROM – too expensive
• Exhaustive – impractical for $n > 15$
• Pseudo-exhaustive
• Pseudo-random (LFSR) – Preferred method
• Test pattern augmentation
  ▪ LFSR combined with a few patterns in ROM
Pseudo-Exhaustive Pattern Generation
Random Pattern Testing

Bottom: Random Pattern Resistant circuit

(a) Top curve -- random pattern testing with acceptable fault coverage.
(b) Bottom curve -- unacceptable random pattern testing.
Random Patterns Generation

• Just guess bits values
  – Not repeatable!

• Algorithmically
  – Define a seed
  – Operate over the seed
  – ULA

• Cheaper hardware?
Pseudorandom Integers

\[ X_k = X_{k-1} + 3 \pmod{8} \]

Sequence: 2, 5, 0, 3, 6, 1, 4, 7, 2 \ldots

\[ X_k = X_{k-1} + 2 \pmod{8} \]

Sequence: 2, 4, 6, 0, 2 \ldots

Maximum length sequence: 3 and 8 are relative primes.
Polynomials

• Vector = binary number

• Binary number can be read as a polynomial module 2

• $101001 = 1 \times x^5 + 0 \times x^4 + 1 \times x^3 + 0 \times x^2 + 0 \times x^1 + 1 \times x^0$

• Module 2 sum = XOR!

• So, the binary number can be written as:

$$1 \times x^5 \oplus 0 \times x^4 \oplus 1 \times x^3 \oplus 0 \times x^2 \oplus 0 \times x^1 \oplus 1 \times x^0$$
Polynomials

- How can we represent or implement such a sum in HW?
- Coefficients must be stored: FFs
- Sums must be performed: XOR gates!
- So, a LFSR can represent a polynomial and vice-versa
• **Standard Linear Feedback Shift Register (LFSR) (Type 1)**

• The location of the XOR gates represent the non-zero coefficients of the associated polynomial (**characteristic polynomial**)

• Ex: $x^3 + x^2 + x + 1$ or $x^3 + x + 1$
Matrix Equation for Standard LFSR

\[
\begin{bmatrix}
X_0(t+1) \\
X_1(t+1) \\
\vdots \\
X_{n-3}(t+1) \\
X_{n-2}(t+1) \\
X_{n-1}(t+1)
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1 & 0 & \ldots & 0 & 0 \\
0 & 0 & 1 & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 1 & 0 \\
0 & 0 & 0 & \ldots & 0 & 1 \\
h_1 & h_2 & \ldots & h_{n-2} & h_{n-1} & 1
\end{bmatrix}
\begin{bmatrix}
X_0(t) \\
X_1(t) \\
\vdots \\
X_{n-3}(t) \\
X_{n-2}(t) \\
X_{n-1}(t)
\end{bmatrix}
\]

\[X(t+1) = T_s X(t) \quad (T_s \text{ is companion matrix})\]
LFSR Properties

- Must not initialize to all 0’s – hangs
- If $X$ is initial state, LFSR progresses through states
  \[ X, T_S X, T_S^2 X, T_S^3 X, \ldots \]

- \textit{Matrix period}:
  Smallest $k$ such that $T_S^k = I$
  - $k$ = LFSR cycle length
  - Maximum length $k = 2^n - 1$, when feedback (characteristic) polynomial is primitive
  - Example: $1 + X + X^3$

- \textit{Characteristic polynomial}:
  \[ 1 + h_1 X + h_2 X^2 + \ldots + h_{n-1} X^{n-1} + X^n \]
LFSR as TPG

- LFSR has cyclic and repetitive behavior
- Ex: characteristic polynomial $= x^3 + x + 1$
LFSR as TPG

- LFSR has cyclic and repetitive behavior
- Ex: characteristic polynomial = $x^3 + x + 1$
- Does it remind you of something?
  - Maybe patterns...
- What if initial state is 000????
- LFSR can be used as a pseudo-random TPG!
LFSR: $1 + X + X^3$

Test of primitiveness: Characteristic polynomial of degree $n$ must divide $1 + X^q$ for $q = n$, but not for $q < n$
Still LFSRs

- All LFSR is cyclic but the number of distinct patterns generated varies...
- Ex: For the same 3 bits, let’s try the characteristic polynomial: $x^3 + x^2 + x + 1$
- Start with 010
- Then with 001
- What do we want for testing?
Primitive Polynomial

- The more distinct patterns the better
- A LFSR can generate all $2^N - 1$ distinct values when the characteristic polynomial is a primitive one.
- Primitive polynomial: it divides the polynomial $1 + x^k$, for $k = 2^N - 1$
  - $x^3 + x^2 + x + 1$ ?
  - $x^3 + x + 1$ ?
Primitive Polynomials

- To find a primitive polynomial is NP-complete problem!
- 8-bit LFSR has 16 primitive polyn.
- 16-bit LFSR has 2048
- 32-bit LFSR has 67108864
- Which polynomial must we use??????
- Look up for the tables!!!
Notation

• 3: 1 0
  – \( x^3 + x + 1 \)

• 12:7 4 3 0
  – \( x^{12} + x^7 + x^4 + x^3 + 1 \)

• First and last coefficients are ALWAYS there!

• Now, define the LFSR representing the following polynomial:
  – 5: 2 0

• What if we read the FFs from left to right?!?!?!?
Generic Modular LFSR (type 2)
Example Modular LFSR

- $f(x) = 1 + x^2 + x^7 + x^8$
- Read LFSR tap coefficients from left to right
- Same theory applies!
Test Pattern Augmentation

- Secondary ROM – to get LFSR to 100% SAF coverage
  - Add a small ROM with missing test patterns
  - Add extra circuit mode to *Input MUX* – shift to ROM patterns after LFSR done
  - Important to compact extra test patterns