Chapter 10
Logic Built-in Self test
Outline

- Introduction
- Test Pattern Generation
- Response Analyzers
- BIST Examples
Definition & Advantages of BIST

- Built-In Self-Test (BIST) is a design-for-testability (DFT) technique in which testing (test generation, test application) is accomplished through built-in hardware features.
  - [V.D. Agrawal, C.R. Kime, and K.K. Saluja]
- Can lead to significant test complexity reduction
- Especially attractive for embedded cores
Benefits of BIST

- **Complexity**
  - Testing is a global problem.
  - Partitioning, which reduces design complexity, can also reduce test complexity if blocks can test themselves.
  - With BIST, tests can be applied systematically
    - Component test; Interconnect test; PCB test; System test

![Diagram showing test sets and designers](image_url)
Quality
- For some large systems (Motorola six sigma project), keeping the defect level to 10 ppm or better is only possible through BIST.

Test generation
- It’s difficult to deliver patterns through layers of design hierarchy.
- Local test pattern generation (BIST) reduces the test generation and delivery efforts.

Test application
- BIST allows “electronic” rather than “physical” probes, and thus reduces test application cost.

Test development time
- BIST circuitry should be automatically generated by CAD tools.

In-filed test & maintenance test
- BIST can test circuit once a while to ensure system reliability
BIST Associated Costs

- Area overhead
  - Additional gate counts and routing resources
- Pin overhead
  - Usually 1 pin at least is needed to invoke the BIST operation.
  - Integrate with 1149.1 is desirable
- Performance overhead
  - Due to delays added to signal paths
- Yield loss and reliability reduction
  - Due to increased die size
- Increased design effort and time
  - Shift test efforts into design domains
- Testability of the BIST hardware
  - Self-testability of added hardware
## BIST Costs Distributed Among Components

<table>
<thead>
<tr>
<th></th>
<th>Design &amp; test</th>
<th>Fabrication</th>
<th>Production test</th>
<th>Maintenance test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chips</td>
<td>+/-</td>
<td>+</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Boards</td>
<td>+/-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Systems</td>
<td>+/-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Extra design time is compensated by reduced test development cost.
- Can the benefits in other factors balance the increased fabrication cost?
  - To evaluate test strategies under economic models
Reality of Logic BIST

- BIST is NOT a replacement for scan
  - it can be built on top of full-scan
- BIST does NOT result in fewer patterns
  - it usually uses many more patterns than ATPG patterns
- BIST does NOT remove the need for testers
  - tester still required to
    - initiate test
    - read response
    - apply ATPG patterns to other part of IC
- With proper designed BIST, the added cost will be more than balanced by the benefits in terms of reliability and reduced maintenance cost
General Organization of BIST

Simple on-chip pattern generation

To avoid expensive bit-to-bit comparison

Test Generator

Circuit Under Test (CUT)

Response Compressor

off-line pre-computed fault-free signature

signature

Pass-or-fail
Outline

- Basics
- Test Pattern Generation
  - How to generate patterns on chip using minimum hardware, while achieving high fault coverage
- Response Analyzers
- BIST Examples
# On-Chip Pattern Generation

<table>
<thead>
<tr>
<th>Hardware Types</th>
<th>Pattern Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Stored Patterns</td>
<td>• Deterministic</td>
</tr>
<tr>
<td>• Counter Based</td>
<td>• Pseudo-Exhaustive</td>
</tr>
<tr>
<td>• LFSR Based</td>
<td>• Pseudo-Random</td>
</tr>
<tr>
<td>• Cellular Automata</td>
<td></td>
</tr>
</tbody>
</table>
Pseudo-Random Pattern Generation

- **Pseudo-Random**
  - Patterns generated in a pre-determined order but appeared to be random (low correlation between adjacent bits).

- Implemented by Linear Feedback Shift Register (LFSR) or Cellular Automata (CA)
  - Most popular BIST circuits
  - Easy implementation
  - Low hardware costs
Linear Feedback Shift Register (LFSR)

- Linear: superposition principle applicable
  - Response to linear combination of stimuli is the linear combination of the responses to the individual stimuli.
- Simple and regular structure
- No input except clocks is supplied: autonomous machine
- Components:
  - Shift registers: one cycle delay implemented by flip-flops
  - XOR gate: modulo-2 addition

**Type 1: external-XOR LFSR**

\[
z = y_4 + y_1 = D^4(z) + D(z)
\]

**Type 2: internal-XOR LFSR**

\[
z = y_4 = D(y_3 + y_4) = D(D^3(z) + z) = D^4(z) + D(z)
\]
Type 1: external-XOR LFSR

\[
\begin{align*}
X_0 (t+1) \\
X_1 (t+1) \\
\vdots \\
X_{n-3} (t+1) \\
X_{n-2} (t+1) \\
X_{n-1} (t+1)
\end{align*}
\]

\[
\begin{bmatrix}
0 & 1 & 0 & \cdots & 0 & 0 \\
0 & 0 & 1 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 1 & 0 \\
0 & 0 & 0 & \cdots & 0 & 1 \\
1 & h_1 & h_2 & \cdots & h_{n-2} & h_{n-1}
\end{bmatrix}
\]

\[
X (t+1) = T_S X (t)
\]

characteristic function: \( f (x) = 1 + h_1 x + h_2 x^2 + \cdots + h_{n-1} x^{n-1} + x^n \)
Type 2: internal-XOR LFSR

\[
\begin{bmatrix}
X_0(t+1) \\
X_1(t+1) \\
X_2(t+1) \\
\vdots \\
X_{n-3}(t+1) \\
X_{n-2}(t+1) \\
X_{n-1}(t+1)
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 0 & \ldots & 0 & 0 & 1 \\
1 & 0 & 0 & \ldots & 0 & 0 & h_1 \\
0 & 1 & 0 & \ldots & 0 & 0 & h_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \ldots & 0 & 0 & h_{n-3} \\
0 & 0 & 0 & \ldots & 1 & 0 & h_{n-2} \\
0 & 0 & 0 & \ldots & 0 & 1 & h_{n-1}
\end{bmatrix}
\begin{bmatrix}
X_0(t) \\
X_1(t) \\
X_2(t) \\
\vdots \\
X_{n-3}(t) \\
X_{n-2}(t) \\
X_{n-1}(t)
\end{bmatrix}
\]

\[X(t+1) = T_m X(t)\]

characteristic function: \( f(x) = 1 + h_1 x + h_2 x^2 + \ldots + h_{n-1} x^{n-1} + x^n \)
The register cycles through $2^4-1$ states if the seed is not all-0.
LFSR Example

\[
\begin{bmatrix}
y_0(t+1) \\
y_1(t+1) \\
y_2(t+1) \\
y_3(t+1)
\end{bmatrix} =
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
y_0(t) \\
y_1(t) \\
y_2(t) \\
y_3(t)
\end{bmatrix}
\]

Characteristic polynomial

\[ g(x) = x^4 + x^3 + 1 \]
LFSR Characteristics

- **Maximum-length sequence**
  - A sequence generated by an \textit{n-stage LFSR} is called a maximum-length sequence if it has a period of \(2^{n-1}\).
  - A LFSR can generate maximum-length sequence if its characteristic polynomial is primitive.

- **Primitive polynomial**
  - The characteristic polynomial associated with a maximum-length sequence is called a primitive polynomial.
  - Primitive (irreducible) polynomials
    - cannot be factorized into two (or more) parts, i.e., it is not divisible by any polynomial other than 1 and itself.
LFSR Properties As a Pattern Generator

- Pseudo-random sequence
  - The sequence generated by an LFSR is called a pseudo-random sequence

- No. of 1s and 0s
  - The number of 1s in a maximum-length sequence differs from the number of 0s by one

- Transition Count
  - The number of transitions between 1 and 0 that an \( m \)-sequence makes in one period is \( (m+1)/2 \).

- The correlation
  - Between any two output bits is very close to zero
When only a fraction of the $2^n-1$ can be applied, LFSR is better than counters.

<table>
<thead>
<tr>
<th>Counter</th>
<th>LFSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 0 1</td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 0 1</td>
</tr>
</tbody>
</table>

Sequences of LFSR is more random

Every bit is random
Autocorrelation between different bits:

\[
C(\tau) = \frac{1}{2^m - 1} \sum_{n=1}^{2^m-1} b_n b_{n-\tau}, \quad \text{where } \begin{cases} 
    b_n = 1 & \text{when } a_n = 0 \\
    b_n = -1 & \text{when } a_n = 1 
\end{cases}
\]

The autocorrelation function of every maximum-length LFSR of period \( p=2^m-1 \) is:

- \( C(0)=1 \)
- \( C(\tau)=-1/p \quad i \neq j \)

Ex:

<table>
<thead>
<tr>
<th>( D_1 )</th>
<th>( D_2 )</th>
<th>( D_3 )</th>
<th>( k=3 )</th>
<th>( C(1,2)=-1/7 )</th>
<th>( C(1,3)=-1/7 )</th>
<th>( C(2,3)=-1/7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cellular Automaton (CA)

- An one-dimensional array of cells
- Each cell contains a storage device and next state logic
- **Next state** is a function of current state of the cell and its neighboring cells

![Diagram of a cellular automaton with three-cell neighbors](image)
Rule Number

- Rule Number is the name of CA functions
- Is determined by its truth table

<table>
<thead>
<tr>
<th>State</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
<th>A7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ci+1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Ci</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Ci-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Example:

\[
F_{CA} = C_{i-1} \oplus C_i
\]

Rule = 64+32+4+2 = 102

\[\text{Name} = \sum_{i=0}^{7} A_i 2^i\] (defined by Wolfram)
Cellular Automata – Hardware

CA with Null Boundary Condition

Standard – All the CAs are of the same type
Hybrid – The CAs are of different type
Cellular Automata – Hardware

CA with cyclic Boundary Condition

\[ \text{Fca} \]

\[ \text{D} \]

\[ \text{Q} \]
Cellular Automata Example: Null-Boundary Hybrid Using Rule 90 and 150

Rule 90: $C_i(t+1) = C_{i-1}(t) \oplus C_{i+1}(t)$

Rule 150: $C_i(t+1) = C_{i-1}(t) \oplus C_i(t) \oplus C_{i+1}(t)$

The construction of pseudo-random pattern generator using CA has no definite rule like searching for primitive polynomials.
Example of Last Page’s Cellular Automata: State Transitions

- CAs have no shift-induced bit value correlation as LFSR.
- A linear phase shifter can make LFSR patterns more random

<table>
<thead>
<tr>
<th>$D_4$</th>
<th>$D_3$</th>
<th>$D_2$</th>
<th>$D_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Fault coverage of Pseudo-Random Patterns

- Is a function of the test length and the random testability of the circuits
  - Certain faults have very few patterns to cover them
  - These faults have very low probability to be detected by random patterns
- Certain circuits are more resistant to random patterns than others
  - If it has more such faults
Random Pattern Resistant Faults

- An RPRF cannot be detected by random patterns, which is a major cause of low fault coverage in BIST
- **Inadequate coverage can be boosted by**
  - Test point insertion
  - Stored ATPG patterns (reseeding)
  - Weighted random patterns
Example: Hard-To-Detect Fault

- Hard-to-detect faults
  - Faults that are not covered by random testing
  - E.g., an output signal of an 18-input AND gate
Weighted Pseudo Random Testing

It was observed that weighted random patterns could achieve higher fault coverage in most cases!

LFSR Based

Weighted Cellular Automaton
Signal of An Arbitrary Weight

- To implement a signal
  - with a signal-1 probability (weight) of \( \frac{5}{32} \)

- Procedure
  1. **Decompose** into a sum of basic weights
     \[
     \frac{5}{32} = \frac{4}{32} + \frac{1}{32} = \frac{1}{8} + \frac{1}{32}
     \]
  2. Use AND and OR gates to realize the weight

\[
z = y_1 y_2 y_3 + \overline{y_1} y_2 y_3 y_4 y_5
\]

a signal with a weight of \( \frac{5}{32} \)
### Programmable Weighted PRPG

**Diagram**

- **Central Register**
  - Numbers: 1, 2, 3, 4
- **Control Register**
  - States: 1000, 0100, 0010, 0101, 0011, 1100, 1010, 0110
- **Approximate Signal Probability**
  - Values: .5, .25, .125, .75, .875, .625, .5625, .3437

**Table**

<table>
<thead>
<tr>
<th>Control Register State</th>
<th>Approximate Signal Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>.5</td>
</tr>
<tr>
<td>0100</td>
<td>.25</td>
</tr>
<tr>
<td>0010</td>
<td>.125</td>
</tr>
<tr>
<td>0101</td>
<td>.75</td>
</tr>
<tr>
<td>0011</td>
<td>.875</td>
</tr>
<tr>
<td>1100</td>
<td>.625</td>
</tr>
<tr>
<td>1010</td>
<td>.5625</td>
</tr>
<tr>
<td>0110</td>
<td>.3437</td>
</tr>
</tbody>
</table>

**Output**
Optimal Weights to Inputs of AND/OR Gates

- Theorem: The optimal signal probability assignment to the inputs of an AND or NAND gate with n inputs is \((n-1)/n\).

- Theorem: The optimal signal probability assignment to the inputs of an OR or NOR gate with n inputs is \(1/n\).
Heuristic for Signal Probability Assignment

For each gate in the circuit do:

Begin

- Step 1: Assign to the gate inputs the asymptotic optimal assignment according to the theorems in last slide
- Step 2: Moving backward from the lines being assigned in Step 1, compute the signal probability assignment at the primary inputs by propagating signal probability values from outputs toward inputs according to the formulas shown in the next slide
- Step 3: Record the signal probability assignments computed for the primary inputs

End
Step 4: To each primary input, assign a weight which equals the average of all signal probability assignments recorded in Step 3. Use this weight as the signal probability of the primary input in a weighted random test.

Rules used in Step 2:

For primitive gates: (the input and output signal probabilities are denoted as $p_i$ and $p_0$):

- A k-input AND gate: $p_i = (p_0)^{1/k}$
- A k-input OR gate: $p_i = 1 - (1 - p_0)^{1/k}$
- An INVERTER: $p_i = 1 - p_0$
- A k-input NAND gate: $p_i = (1 - p_0)^{1/k}$
- A k-input NOR gate: $p_i = 1 - (p_0)^{1/k}$

For a fan-out with branch signal probabilities $p_i$, $i = 1, 2, \ldots, k$, the stem signal probability $p_s$ is

$$p_s = \frac{1}{k} \sum_{i=1}^{k} p_i$$
An Example

Backtracing the local requirement for gate $G_4$.

The computed weights for the primary inputs.

(a) $X_1$ (.5, .293) $G_2$ $X_1$ (.293)

(b) $X_1$ (.5, .293) $G_2$ $X_2 X_3 X_4$

(.333, .206) $G_3$ $X_5 X_6$

(.63) $X_2 X_3 X_4$ $G_1$ $X_1$ (.5, .293) $G_2$ $X_1$ (.293) $G_4$ $.5$ $G_4$ $.5$
An Example (Cont’d)

The nearly optimal signal probability assignment.
X will be assigned with \((W_1 + W_2)/2\). However, \(w_1\) and \(W_2\) are two extreme values.
On-Chip Exhaustive Testing

- Exhaustive testing
  - Apply all possible input combinations to CUD
  - A complete functional testing
  - 100% coverage on all possible faults

- Limitation
  - Only applicable for circuits with medium number of inputs
Pseudo Exhaustive Testing (PET)

- Apply all possible input combinations to every partitioned sub-circuits
- 100% fault coverage on single faults and multiple faults within the sub-circuits
- **Test time** is determined by the number of sub-circuits and the number of inputs to the sub-circuit
- **Partitioning** is a difficult task
  - Cone Segmentation
  - Sensitized-Path Segmentation
  - Physical Segmentation
    - Use DFT circuits to aid partitioning
Example for Cone Segmentation

$2 \times 2^5 = 64$ vectors are enough to pseudo-exhaustively test this circuit, Compared to $2^8 = 256$ vectors for naive exhaustive testing
Example for Sensitized-Path Segmentation

- Testing procedure
  - Apply input pattern X2 such that D=0
  - Apply exhaustive patterns to C1
  - Apply input pattern X1 such that C=0
  - Apply exhaustive patterns to C2
Outline

- Basics
- Test Pattern Generation
- Response Analyzers
  - How to compress the output response without losing too much accuracy
- BIST Examples
Why Compaction?

- **Motivation**
  - Bit-to-bit comparison is infeasible for BIST

- **Compaction**
  - Compress a very long output sequence into a single signature
  - Lossy compression
  - **Signature analysis**: Compare the compressed word with the pre-stored gold signature to determine the correctness of the circuit

- **Problems**
  - Many output sequences may have the same signature after the compression leading to the aliasing problem
Lossy Compression V.S. Coding

- We want to sort inputs into different groups
  - Inputs of our problem: 1 million 32 bit responses from circuits
  - Different inputs are put into the same code as fault free one
The signature & its collection algorithm should meet the following guideline:

1. The algorithm must be **simple** enough
   - To be implemented as part of the built-in test circuitry.
2. The implementation must be **fast** enough
   - To remove it as a limiting factor in test time.
3. The compression method should **minimize the loss of information**.
   - Minimize the aliasing, i.e., loss of evidence of a fault indicated by a wrong response from the circuit under test.
Signature Analysis by LFSR

- Procedure
  - Apply predetermined patterns
  - **Divide** the output sequence by LFSR
LFSR for Polynomial Division

- Suppose we are interested in modulo 2 division.
  \[ P(x)/G(x) = (x^7 + x^3 + x)/(x^5 + x^3 + x + 1) \]
- The longhand division can be conducted in terms of the detached coefficients only:

  $\begin{array}{c|c}
  101011 & 101 & Q(x) = x^2 + 1 \\
  \hline
  10001010 & 101011 & R(x) = x^3 + x^2 + 1 \\
  00100110 & \hline
  001101 = R & \\
  \end{array}$

- This division process can be mechanized using a LFSR.
Galois Field GF(2)

- **Operation**
  - Modulo-2 addition, subtraction, multiplication, and division of binary data

- **Properties**
  - Modulo-2 addition and subtraction are identical
  - $0+0=0$, $0+1=1$, $1+0=1$, $1+1=0$

Bit-stream multiplication

\[(x^3 + x^2 + x + 1) \times (x^2 + x + 1)\] is given by:

\[
\begin{align*}
x^3 + x^2 + x + 1 \\
x^2 + x + 1 \\
x^3 + x^2 + x + 1 \\
x^4 + x^3 + x^2 + x \\
x^5 + x^4 + x^3 + x^2 \\
x^5 + 0 + x^3 + x^2 + 0 + 1
\end{align*}
\]

Bit-stream division

\[
\begin{align*}
x^2 + x^1 + 1 \bigg| x^5 + 0 + x^3 + x^2 + 0 + 1 \\
x^5 + x^4 + x^3 \\
x^4 + 0 + x^2 \\
x^4 + x^3 + x^2 \\
x^3 + 0 + 0 \\
x^3 + x^2 + x \\
x^2 + x + 1 \\
x^2 + x + 1 \\
0
\end{align*}
\]
When a shift occurs, $x^5$ is replaced by $x^3 + x^1 + x^0$.

Thus, whenever a quotient coefficient (the $x^5$ term) is shifted off the right-most stage, $x^3 + x^1 + x^0$ is added to the register (or subtracted from the register since addition is the same as subtraction modulo 2). Effectively, the dividend has been divided by $x^5 + x^3 + x + 1$. 

Ex: LFSR implementing division by $f(X) = x^5 + x^3 + x + 1$
If the LFSR is initialized to zero & the message word (or dividend) $P(x)$ is serially streamed to the LFSR input, high-order coefficient first, the content of the LFSR after the last message bit is the remainder from the division of the message polynomial by the divisor $G(x)$. 
Example: \( P(x) = x^7 + x^3 + x \)  input=10001010

\[
\begin{array}{ccccccc}
\text{input} & \rightarrow & + & \rightarrow & + & \rightarrow & + & \rightarrow & + & \rightarrow & + & \rightarrow & + & \rightarrow & x^5 \\
\text{time} & \rightarrow & x^0 & \rightarrow & x^1 & \rightarrow & x^2 & \rightarrow & x^3 & \rightarrow & x^4 & \rightarrow & x^5 & \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
\end{array}
\]

Remainder \( R = x^3 + x^2 + 1 \)  
Quotient \( Q = x^2 + 1 \)
Any data, such as the test response results from a circuit, can be compressed into a “code word” by an LFSR.

This code word, the remainder from the division process, is called the “signature” of the input data stream.

The LFSR itself is called a “signature analyzer”.

\[ P(x) = Q(x) G(x) + R(x) \]

- **divisor**
- **signature**

(LFSR polynomial)
If \( P(x) \) is the polynomial of the correct data
\[ \Rightarrow P'(x) = P(x) + M(x)G(x) \]
will have the same signature as \( P(x) \) for any \( M(x) \).

**Example:** \( P(x) = x^7 + x^3 + x \)

\[ G(x) = x^5 + x^3 + x + 1 \]

signature \( R(x) = x^3 + x^2 + 1 \)

\[ P'(x) = P(x) + G(x) = x^7 + x^5 + 1 \]

\[ P''(x) = P(x) + x \cdot G(x) = x^7 + x^6 + x^4 + x^3 + x^2 \]

\( P'(x) \) and \( P''(x) \) have same signature \( x^3 + x^2 + 1 \).

**Aliasing:** condition in which a faulty circuit with erroneous response produces the same signature as a good circuit.

**Aliasing probability** is usually used to measure the quality of a data compressor.
What is the aliasing probability of using LFSR as a data compressor?

- Suppose the input string is m-bit long. There are $2^m - 1$ possible bit streams.
- If the divisor polynomial $G(x)$ is of degree r, then it has $2^{m-r} - 1$ nonzero multiples that result in a polynomial of degree less than m.

  => There are $2^{m-r} - 1$ wrong m-bit streams that can map into the same signature as the correct bit stream.

  => Aliasing prob. $P(M) = (2^{m-r} - 1)/(2^m - 1)$

  => For large m, $P(M) \approx 1/2^r$. 
Comparison with other compression methods

- **One count compression**
  - The signature is the sum of the number of 1’s appearing on the circuit output during the test.

- **Suppose that the data to be compressed is**\(N\)-bit long and \(r\) is the expected ones count, \(0 \leq r \leq N\).

\[
P(M/r) = \frac{\binom{N}{r}}{2^N} - 1
\]

<table>
<thead>
<tr>
<th>(r)</th>
<th>(P(M/r))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.047</td>
</tr>
<tr>
<td>2</td>
<td>0.157</td>
</tr>
<tr>
<td>3</td>
<td>0.268</td>
</tr>
<tr>
<td>4</td>
<td>0.268</td>
</tr>
<tr>
<td>5</td>
<td>0.157</td>
</tr>
<tr>
<td>6</td>
<td>0.047</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Ex: \(N=7\)
Transition-Counting Signature

- **Procedure**
  - Apply predetermined patterns
  - Count the number of $0\rightarrow 1$ and $1\rightarrow 0$ transitions
Aliasing of Transition-Counting

- Consider a sub-sequence of bits

  \( \ldots r_{j-1} \, r_j \, r_{j+1} \ldots \)

  If \( r_{j-1} \) is not equal to \( r_{j+1} \), then an error occurring at \( r_j \) will not be detected by transition counting.

- Example

  1. \((0, 1, 1) \rightarrow (0, 0, 1)\)
  2. \((0, 0, 1) \rightarrow (0, 1, 1)\)
  3. \((1, 1, 0) \rightarrow (1, 0, 0)\)
  4. \((1, 0, 0) \rightarrow (1, 1, 0)\)
- Transition count compression
  - The signature in transition count testing is the number of logical transitions (0->1 & 1->0) in the data stream.

- Assume the data stream’s length is $N$ and a transition count $= i$

\[
P(M/i) = \frac{2 \left( \binom{N-1}{i} \right) - 1}{2^N - 1}
\]

<table>
<thead>
<tr>
<th>$i$</th>
<th>$P(M/i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00787</td>
</tr>
<tr>
<td>1</td>
<td>0.0866</td>
</tr>
<tr>
<td>2</td>
<td>0.228</td>
</tr>
<tr>
<td>3</td>
<td>0.307</td>
</tr>
<tr>
<td>4</td>
<td>0.228</td>
</tr>
<tr>
<td>5</td>
<td>0.0866</td>
</tr>
<tr>
<td>6</td>
<td>0.00787</td>
</tr>
</tbody>
</table>

Ex: $N=7$

Low aliasing probability when $i$ is small or large
LFSR: 

\[ P(M) = \frac{2^{N-m} - 1}{2^N - 1} \]

Ex:  
- \( m=3 \) \( P(M) \approx 0.125 \)
- \( m=4 \) \( P(M) \approx 0.0625 \)
- \( m=5 \) \( P(M) \approx 0.03125 \)
Multiple-Input Signature Register (MISR)

- For built-in testing of multiple-output circuits, the overhead of a single-input signature analyzer on every output would be high.
- A multiple-input signature register (MISR) is used for multiple-output circuits.
It can be proved that the aliasing probability of a MISR is:
\[
\frac{2^{n-1}-1}{2^{m+n-1}-1} \approx \frac{1}{2^m}
\]
m: the number of stages in MISR
n: length of data to be compressed.

Improving Signature Analysis

- Increase the length of LFSR
- Repeat test using different feedback structure of LFSR
- Repeat test using different orders of vectors
- Multiple observation on signatures
Outline

- Basics
- Test Pattern Generation
- Response Analyzers
- BIST Examples
Pseudo Random Testing Hardware

Combinational

- LFSR
- Combinational circuit
- SA

Sequential

- LFSR
- Combinational circuit
- SA

Fault coverage could be low
BIST – Pseudo Random Testing Hardware

10-stage LFSR → Shift register

Circuit Under Test

SA

(CEBT)

test-per-clock configuration

LFSR

SR

CUT

SR

CUT

SR

SA

(STUMPS)

test-per-scan configuration
Built-In Logic Block Observation (BILBO)

<table>
<thead>
<tr>
<th>B1</th>
<th>B2</th>
<th>operation mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>shift register</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>LFSR pattern generation</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>MISR response compressor</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>parallel load (normal operation)</td>
</tr>
</tbody>
</table>
Test procedure

- Each logic block C1, C2, C3 are tested in a serial manner.
- BIST controller needs to configure each BILBO registers properly during self-testing.

When testing C1
- BILBO1 is a PRPG
- BILBO2 is a MISR
Concurrent BILBO

Logic with self-loop

BILBO

C1

concurrent BILBO

Operate as PRPG and MISR simultaneously

<table>
<thead>
<tr>
<th>B1</th>
<th>B2</th>
<th>Operation mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>Normal</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Scan</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>PRPG/MISR</td>
</tr>
</tbody>
</table>

top-row of D-FFs $\rightarrow$ MISR
bottom-row of D-FFs $\rightarrow$ PRPG
STUMPS Architecture

Self-Test Using a MISR and Parallel Shift register sequence generator

- Saving some hardware by reducing LFSR/MISR, but increase test time
- Require PIs and POs to be caught by some scan chain
## Comparison of BILBO, STUMPS, and External ATE

<table>
<thead>
<tr>
<th>Test time (CP=1/325MHz)</th>
<th>BILBO</th>
<th>STUMPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PxCP</td>
<td>0.00615s</td>
<td>0.615s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test time (CP=1/1GHz)</th>
<th>BILBO</th>
<th>STUMPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PxCP</td>
<td>0.002s</td>
<td>0.2s</td>
</tr>
</tbody>
</table>

- **P=2000,000**
- ATE operate at 325MHz
- L=scan chain length=100bit
- CP=clock period
- K=ATE speed/clock period

STUMPS is still limited by scan clocks, unless scan enable is locally controlled per chain.

P can be much smaller than BIST

Scan clock do not scale well as K
## Comparison of BILBO, STUMPS, and External ATE (II)

<table>
<thead>
<tr>
<th></th>
<th>BILBO</th>
<th>STUMPS</th>
<th>ATE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CP=1/325MHz)</td>
<td>PxCP</td>
<td>PxLxCP</td>
<td>P1xLxCP1</td>
</tr>
<tr>
<td></td>
<td>0.00615s</td>
<td>0.615s</td>
<td>0.2s</td>
</tr>
<tr>
<td><strong>Test time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CP=1/1GHz)</td>
<td>PxCP</td>
<td>PxLxCP</td>
<td>P1xLxCP1</td>
</tr>
<tr>
<td></td>
<td>0.002s</td>
<td>0.2s</td>
<td>0.2s</td>
</tr>
</tbody>
</table>

- P=2000,000; P1=20000
- ATE operate at 325MHz
- L=scan chain length=100bit
- CP=clock period; CP1=1/(10M)
- K=ATE speed/clock period
LBIST with SGL Architecture
SGL Logics

LFSR Test Control

BFF

Scan chain #1
Scan chain #2
...

L F S R
Source: Application of Deterministic Logic BIST on Industrial Circuits