UNIDAD 12: AUTOPRUEBA INCORPORADA.

(86:44) Técnica Digital Avanzada- Unidad 12.
Profesor: Ing. Miguel Antonio Martínez.

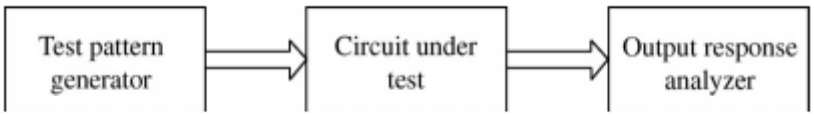
Autoprueba incorporada.

Este tema es la traducción de **Built in Self Test (BIST)**. La tarea de probar un chip para garantizar su funcionalidad es extremadamente compleja, y a veces con mucha pérdida de tiempo. Además, al incorporar chips a sistemas más complejos ha provocado que el costo de generación de pruebas crezca exponencialmente.

Un enfoque ampliamente aceptado para tratar el tema de las pruebas es incorporar una autoprueba integrada (BIST) dentro del chip. Esto facilita la generación de pruebas y la detección de fallas.

En los métodos de prueba tradicionales, los patrones de prueba son generados exteriormente usando herramientas de diseño asistido (CAD). El patrón de prueba y el resultado esperado del circuito bajo prueba son utilizados por un equipo de prueba automático (ATE) para determinar si las respuestas reales coinciden con las esperadas. En BIST la generación de pruebas y la evaluación de resultados se realizan en el mismo chip, por lo tanto, nos ahorramos el uso de costosas maquinas ATE para su testeo.

Una configuración básica de esta técnica se ve en la siguiente figura:



El bloque generador es un productor de vectores de prueba que alimentan el circuito a testear, suponemos que este circuito es un combinacional con múltiples salidas. Las respuestas del circuito son transferidas al analizador de respuestas de salida. Inicialmente, un esquema BIST debe ser de fácil implementación y cubrir un amplio espectro de fallas.

Generación de patrones de prueba para BIST.

El enfoque para la generación de pruebas en el esquema BIST se pueden dividir en cuatro grandes grupos:

1) Pruebas exhaustivas.
2) Prueba seudoexhaustivas.
3) Pruebas pseudoaleatorias.
4) Pruebas determinísticas.

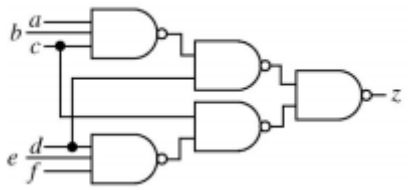
1-Pruebas Exhaustivas.

En este enfoque todos los posibles patrones de entrada se aplican al circuito bajo prueba. Por ejemplo, en un circuito combinacional de **n** entradas se necesitan $\mathbf{2^n}$ patrones de prueba. La ventaja de esta opción es que se pueden detectar todas las fallas no redundantes, sin embargo, si se produce una falla que transforme el circuito combinacional en uno secuencial no es detectada, por ejemplo, una falla tipo puente.

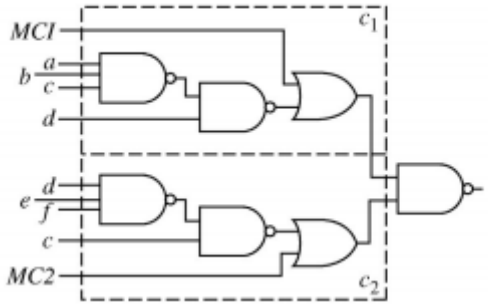
El inconveniente de este enfoque es que cuando la cantidad de entradas n es grande, se torna prohibitivo, incluso con altas velocidades de reloj. Por lo tanto, este enfoque se usa cuando la cantidad de entradas son relativamente pocas.

2-Pruebas seudoexhaustivas.

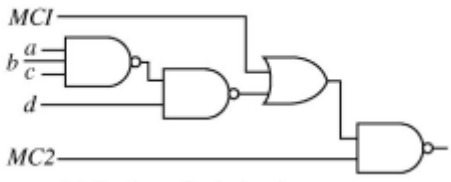
Es una modificación del enfoque anterior. Conserva las ventajas de este enfoque, pero reduce la cantidad de pruebas aplicadas al circuito. La idea básica es particionar el circuito bajo prueba en subcircuitos, estos tienen pocos recursos por lo cual se puede aplicar a cada uno el método exhaustivo. Para ilustrar el método mostramos el siguiente circuito.



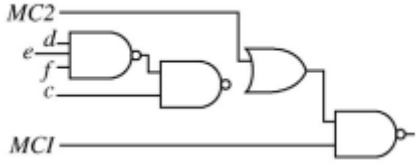
A este circuito lo subdividimos en dos subcircuitos denominados **C_1** y **C_2** como se muestra a continuación:



Las funciones de las dos entradas de control agregadas **MC_1** y **MC_2** son para lo siguiente, cuando MC1 = 0 y MC2 = 1, el subcircuito C_2 es deshabilitado y el subcircuito C_1 se puede probar de todas las formas posibles aplicando valores en las entradas **a, b, c y d.** Esto se muestra en la siguiente figura:

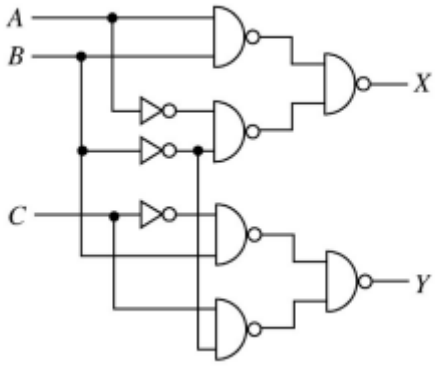


Similarmente, cuando MC1 = 1 y MC2 = 0, el subcircuito C_1 está deshabilitado y se puede probar el C_2 aplicando todas las combinaciones posibles en las entradas **c, d, e y f**, como se ve en la siguiente figura:



Cuando MC1 = MC2 = 0, el circuito funciona como si no hubiera sido modificado salvo por el incremento de tiempo de retardo de la compuerta agregada. La ventaja de este método es que cualquier falla en el circuito mismo o en el circuito de prueba puede ser detectada.

Una prueba mejorada de la prueba autónoma es la llamada prueba de verificación. Este método es aplicable a circuitos combinacionales de múltiples salidas, siempre y cuando cada salida dependa de un subconjunto de entradas. Este método de prueba se deriva de la **matriz de dependencia.** Esta matriz tiene **m** filas y **n** columnas, cada fila representa una de las salidas y cada columna una de las entradas. Una entrada [(i, j) : i = 1,……..m, j = 1,…., n] en la matriz es 1 si la salida depende de la entrada **j**, de lo contrarios la entrada es 0. Para ilustrar esto veremos el circuito del siguiente ejemplo:



Su matriz de dependencia es la siguiente:

	A	B	C
X	1	1	0
Y	0	1	1

Una matriz dependiente de la partición se desprende de la matriz de dependencia. Está formada por la división de las columnas de esta última matriz en un número mínimo de subconjuntos, teniendo cada fila de un conjunto a lo máximo una sola entrada en 1, puede haber más de una matriz dependiente de la partición por cada matriz de dependencia. Volviendo al ejemplo anterior una matriz dependiente de la partición para la matriz de dependencia dada es:

	A	C	B
X	1	0	1
Y	0	1	1

Un conjunto de pruebas de verificación se obtiene asignando los mismos valores a todas las entradas que pertenecen a la misma partición dentro de la matriz dependiente de la partición, dos entradas de diferentes particiones reciben valores distintos.

En la siguiente figura vemos el conjunto de pruebas de verificación para el ejemplo que venimos trabajando:

A	B	C
0	0	0
0	1	0
1	0	1
1	1	1

Se puede buscar un conjunto de pruebas de verificación reducidas, esto se obtiene eliminando todas las repeticiones de columnas idénticas. Esta técnica aplicada al ejemplo nuestro queda:

A / C	B
0	0
0	1
1	0
1	1

compara circuitos que puedan ser transformados en combinacionales durante la prueba, como ser los del tipo LSSD que vimos anteriormente. Sin embargo, la generación de la matriz de dependencia, que es la parte más importante de esta estrategia de pruebas, es una tarea bastante compleja para circuitos del tipo VLSI.

Generación de Patrones Pseudoexhaustivos.

Un circuito combinacional con **n** entradas puede ser probado pseudoexhaustivamente con $\mathbf{2^n}$ o menos de patrones binarios. Por ejemplo, los siguientes seis patrones de entrada se pueden usar para probar pseudoexhaustivamente un circuito de seis entradas siempre que las salidas no dependan de más de una variable de entrada.

1	1	1	1	1
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Estos patrones de prueba pueden ser generados por un polinomio no primitivo, por ejemplo, $x^6 + x^4 + x^3 + x^2 + x + 1$. Sin embargo, si un circuito de múltiples salidas, una salida depende de más de dos variables de entrada, la generación de patrones de prueba a partir de un polinomio no primitivo puede no ser factible.

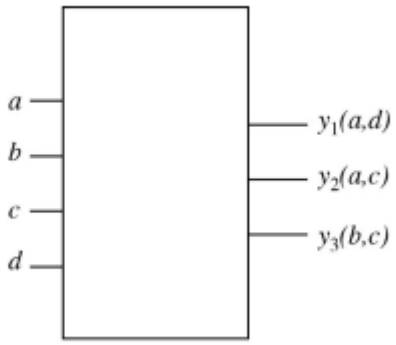
En general, los patrones pseudoexhaustivos para un circuito de **n** entradas y **m** salidas se generan a partir de algunos de los siguientes métodos:

1) Síndrome driver counter.
2) Contador de peso constante.
3) Registro de desplazamiento lineal LFSR/SR
4) Compuertas LFSR/EXOR gates.

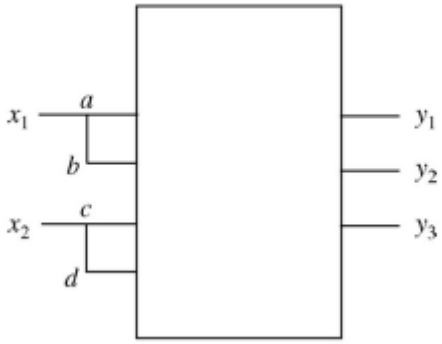
El primer método chequea si **p** entradas de un circuito (menor a **n**) pueden

compartir los mismos patrones de test con las **n – p** entradas restantes. Entonces el circuito se puede probar con una cantidad de $\mathbf{2^n}$ patrones.

Para ver esto observemos el circuito de la siguiente figura:

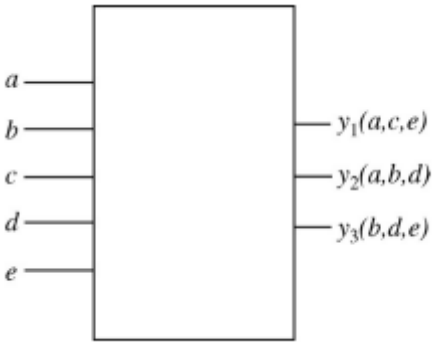


Ninguna salida es función de ambas entradas **c y d** por lo tanto pueden compartir la misma señal de entrada. Además, ninguna salida es función de **a y b** simultáneamente, por lo tanto, pueden compartir la misma señal de entrada. Por lo tanto, el circuito puede ser testeado con 2^2 (p=2) combinaciones como se muestra en la siguiente figura:



Este método se puede ser un contador binario. El inconveniente es que, si **p** es cercano a **n**, requiere tantos patrones de prueba como el testeo exhaustivo.

El método de **contador de peso constante** usa un código llamado “x de y” para probar exhaustivamente un circuito de **m** entradas, donde **x** es el número máximo de variables de entrada en las que cualquier salida bajo prueba depende. El valor de **y** se elige de modo que estén disponibles las $\mathbf{2^x}$ combinaciones de cualquier columna **m** de las palabras del código. Por ejemplo, para el circuito de la siguiente figura:



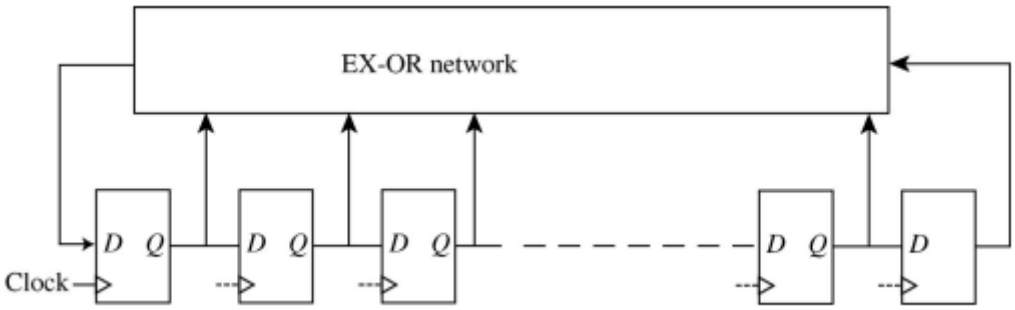
Este circuito tiene 5 entradas y 3 salidas, pero ninguna de las salidas depende de más de tres entradas, entonces nos queda, **m = 5** y **x = 3.** Si elegimos **y = 6**, el contador de peso constante generará las siguientes 20 palabras de código:

000111	100011
001011	100110
001101	100101
001110	101100
010011	101010
010101	101001
010110	110001
011100	110010
011010	110100
011001	111000

La elección de 5 columnas cualesquiera del código garantizará una prueba exhaustiva asociado con cada salida. En general, este tipo de pruebas es de longitud mínima. Sin embargo, la complejidad del contador aumenta considerablemente para códigos “x de y” más grandes.

Un método alternativo para generar pruebas pseudoexhaustivas es usar un LFSR (Linear Feedback Shift Register). En un LFSR un número determinado de salidas se retroalimentan a la entrada a través de una red de compuertas EXOR. Un LSFR de n bits se puede representar por un polinomio irreductible y primitivo. Si el polinomio es de grado n, entonces el LFSR generará los 2^{n-1} patrones distintos de cero en secuencia. Dicha secuencia se denomina **Longitud máxima de secuencia del LFSR**.

La siguiente figura muestra un LFSR basado en un polinomio primitivo:



El polinomio que representa la figura es el siguiente:

$$P(x) = x^n + p_{n-1}x^{n-1} + \ldots + p_2x^2 + p_1x + p_0$$

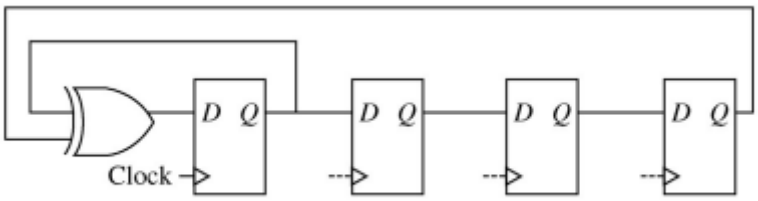
Las conexiones de realimentación necesarias para implementar un LFSR se pueden derivar directamente del polinomio primitivo. Para ilustrar esto supongamos un polinomio de grado 4:

$$x^4 + x + 1.$$

Esta expresión puede ser reescrita como sigue:

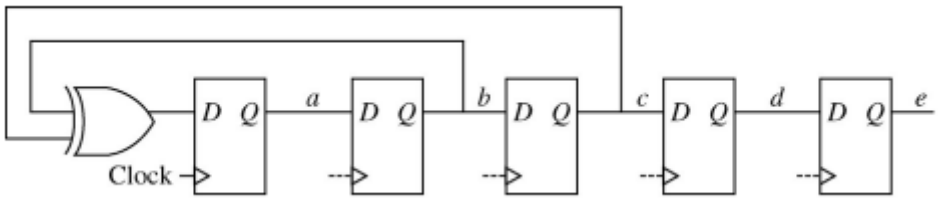
$$P(x) = 1 \cdot x^4 + 0 \cdot x^2 + 1 \cdot x + 1 \cdot x^0$$

La siguiente figura muestra un LFSR de cuatro etapas construido usando este polinomio. A continuación, se muestra la secuencia de longitud máxima correspondiente a este diseño:



1	1	1	1
0	1	1	1
1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	1
0	1	1	0
0	0	1	1
1	0	0	1
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0
1	1	0	0
1	1	1	0

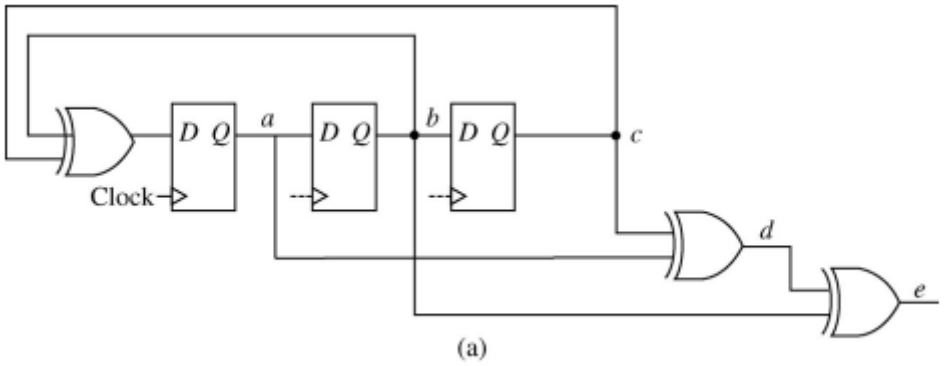
La siguiente figura muestra la combinación de un LFSR de 3 bits con un registro de desplazamiento (SR) de 2 bits como así también la secuencia de salida del conjunto:



a	*b*	*c*	*d*	*e*
1	1	1	0	1
0	1	1	1	0
0	0	1	1	1
1	0	0	1	1
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Se necesitan dos patrones de inicio separados, uno para el LFSR y otro para el SR. La combinación LFSR/SR se puede usar para probar cualquier circuito de cinco entradas en el cual ninguna salida es función de más de dos variables de entradas. Este enfoque garantiza un mínimo de patrones de prueba cuando el número de variables de entrada de la que dependa una salida del circuito es menos de la mitad del número total de variables de entrada.

Una variación a la implementación LFSR/SR es cambiar el registro de desplazamiento SR por una red de compuertas EXOR. Por ejemplo, en la figura anterior se hacer que la variable **d** sea la suma lineal de **a** y **c**, y la variable **e** sea la suma linear de **b** y **d**. El circuito resultante y el patrón generado se puede visualizar en las siguientes figuras:



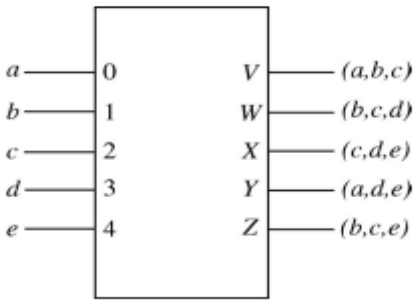
(a)

a	*b*	*c*	*d*	*e*
0	0	1	1	1
1	0	0	1	1
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	0	1
0	1	1	1	0

En general el enfoque LFSR/EXOR produce patrones de prueba muy cercanos al esquema LFSR/SR.

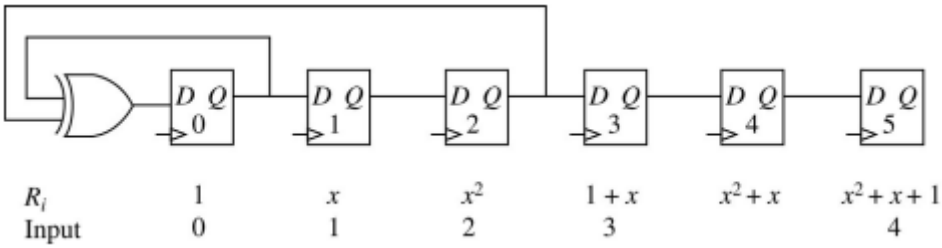
Una técnica alternativa basada en el enforque LFSR/SR es la llamada **LFSR/SR convolucionada.** Este método usa un registro SR de **n** etapas y un LFSR de grado **w** para generar patrones de prueba pseudoexhaustivos para un circuito con **n** entradas y **m** salidas, con ninguna salida que sea función de más de **w** variables de entrada.

El primer paso de este método es asignar residuos $\mathbf{R_0}$ a $\mathbf{R_i}$ a las entradas numeradas desde **0** a **i** del circuito bajo prueba. El residuo de la etapa **i** es $\mathbf{x^i \bmod P_{(x)}}$ donde $P_{(x)}$ es el polinomio primitivo usado para implementar el LFSR. Para ilustrar el cálculo de residuos veremos un circuito de 5 entradas y 5 salidas como el siguiente:



El circuito tiene 5 entradas por lo tanto se necesita un LFSR/SR convolucionado de 5 etapas. Como w = 3, las primeras tres etapas del SR se utilizan para generar un polinomio primitivo de grado 3, por ejemplo: $x^3 + x + 1$.

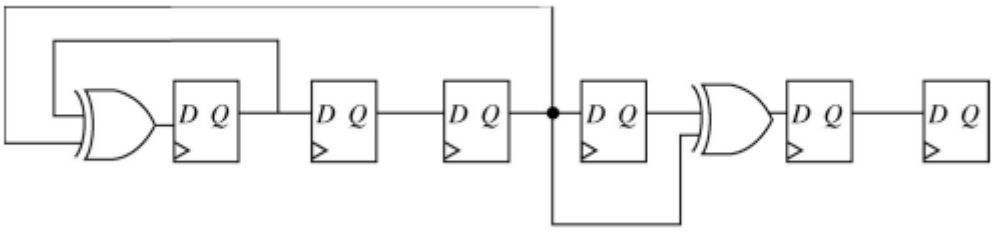
La siguiente figura muestra el LFSR/SR convolucionado resultante:



Los residuos de cada etapa son computados como **x^i (i=0,….,4) mod (x^3 + x + 1)**. Por ejemplo, el residuo de la etapa 3 es x^3 mod (x^3 + x + 1), es decir x + 1.

El siguiente paso en la generación de proceso de prueba, es asignar residuos a las entradas del circuito bajo prueba de modo que ningún valor de salida sea dependiente linealmente de valores ya asignados. Para el circuito dado, a las entradas **a, b, c y d** se le pueden asignar residuos **1, x, x^2** y **(x+1)** respectivamente. Sin embargo, a la entrada **e** no se le puede asignar el residuo **x^2 + x** porque esto resultaría un residuo de un conjunto de salidas **Z,** llamados **(R_1, R_2, R_4)** que forma un conjunto linealmente dependiente. Esto puede evitarse asignando el residuo **R_4** a la entrada **e**.

Si el residuo R_{i+j} es seleccionado como asignación de la entrada i +1, porque los residuos R_{i+1}, R_{i+2}, ……, R_{i+j-1}, no se pueden asignar a esta entrada debido a la dependencia lineal, entonces la etapa i + 1 se puede usar para generar el residuo R_{i+j}, encontrando la suma lineal de uno o más residuos previamente asignados a la etapa i+1. Para el circuito considerado, la etapa 4 genera el residuo x^2 + x + 1 alimentando la suma lineal de los residuos R_2 y R_3 a la etapa 4 como muestra la siguiente figura:



Asumiendo que la secuencia inicial del LFSR es 110 y la del registro de corrimiento 011, los siguientes patrones pseudoaleatorios son generados por el LFSR/SR convolucionado.

1	1	0	0	1	1
1	1	1	0	0	1
0	1	1	1	1	0
1	0	1	1	0	1
0	1	0	1	0	0
0	0	1	0	1	0
1	0	0	1	1	1

Generación de Patrones Pseudoaleatorios.

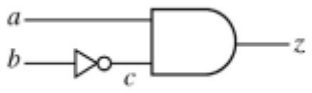
Los patrones pseudoaleatorios son de naturaleza suficientemente aleatorias para reemplazar a las secuencias aleatorias. El método LFSR es ampliamente utilizado para chequear circuitos combinacionales porque son fácil de implementar. Sin embargo, es necesario considerar tres cuestiones relacionadas para medir la eficacia de las pruebas pseudoaleatorias. Ellas son:

1) Determinación del número de patrones de prueba.
2) Evaluación de la cobertura de fallas.
3) Detección de fallas resistentes a patrones aleatorios.

La cobertura de pruebas se puede evaluar usando una simulación de fallas exhaustiva. Sin embargo, los patrones pseudoaleatorios necesarios para chequear un circuito suelen ser grandes, por lo tanto, la simulación es costosa. Una relación entre una secuencia de test pseudoaleatorios de longitud **L** y la cobertura de fallas esperada **E(c)** está dada por la siguiente ecuación:

$$E(c) = 1 - \sum_{k=1}^{2^n-1} \left(1 - \frac{L}{2^n}\right)^k \frac{b_k}{M}$$

Donde **n** es el número de entradas del circuito, $\mathbf{b_k}$ **(k= 1, 2, 3,……, 2^n)** es el número de fallas en el circuito que pueden ser detectadas por **k** vectores de entrada y **M** es el número total de fallas en el circuito bajo prueba. Queda claro viendo la ecuación que $\mathbf{b_k}$ debe conocerse a priori para evaluar **E(c).** Los $\mathbf{b_k}$ para el circuito de la siguiente figura:



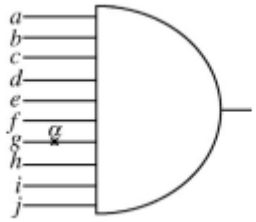
son (b_1, b_2, b_3, b_4) = (7, 0, 1, 0) y se muestran en la siguiente tabla:

TEST	FAULT							
ab	*a* s-a-0	*a* s-a-1	*b* s-a-0	*b* s-a-1	*c* s-a-0	*c* s-a-1	*z* s-a-0	*z* s-a-1
00		×						×
01								×
10	×			×	×		×	
11			×			×		×
k	1	1	1	1	1	1	1	3
		$b_1 = 7,$		$b_2 = 0,$		$b_3 = 1,$	$b_4 = 0$	

Para circuitos complejos, los $\mathbf{b_k}$ son difícil de hallar y solo pueden aproximarse usando análisis probabilístico.

Un problema importante asociado con las pruebas pseudoaleatorias es el gran número de patrones de pruebas para detectar una falla resistente a patrones aleatorios.

Por ejemplo, consideremos una falla del tipo stuck and fault en 1 en el punto α del circuito de una compuerta AND de diez entradas como se ve en la siguiente figura:

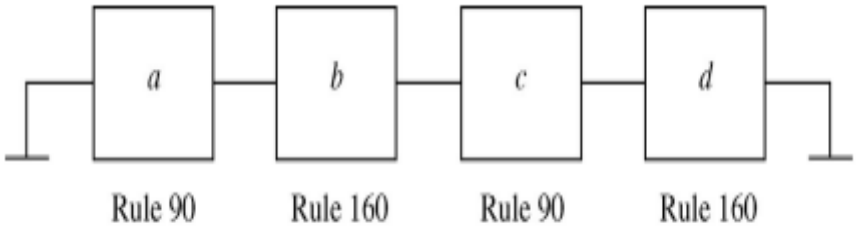


Está claro que solo el vector de prueba abcdefghij = 1111110111 puede detectar la falla. La probabilidad de un LDSR que genere este vector de prueba es 2^{-10} . Por tanto, una gran cantidad de patrones pseudoaleatorios necesita ser aplicada a un circuito bajo prueba que contiene fallas resistentes a pruebas aleatorias para garantizar una alta cobertura de fallas. La debilidad inherente a estas pruebas pseudoaleatorias para encontrar fallas resistentes a patrones aleatorios es que cada bit tiene una probabilidad de 0.5 de ser un 0 lógico o un 1 lógico. Si en lugar de generar patrones con una distribución de probabilidad uniforme, generarnos patrones con una distribución sesgada hay una mayor probabilidad de encontrar vectores de prueba en circuitos con fallas resistentes a patrones aleatorios.

Una alternativa a generar patrones pseudoaleatorios es usar una **automatización celular (CA)**. Una **CA** consiste en un número determinado de celdas idénticas interconectadas espacialmente de una manera regular. Cada celda consiste en un Flip Flop tipo D y una lógica combinacional que genera el próximo estado de la celda. Varias reglas pueden ser usadas para computar el próximo estado de la celda. Por ejemplo, algunas reglas son como las siguientes:

$$y_i(t+1) = y_{i-1}(t) \oplus y_{i+1}(t),$$
$$y_i(t+1) = y_{i-1}(t) \oplus y_i(t) \oplus y_{i+1}(t),$$

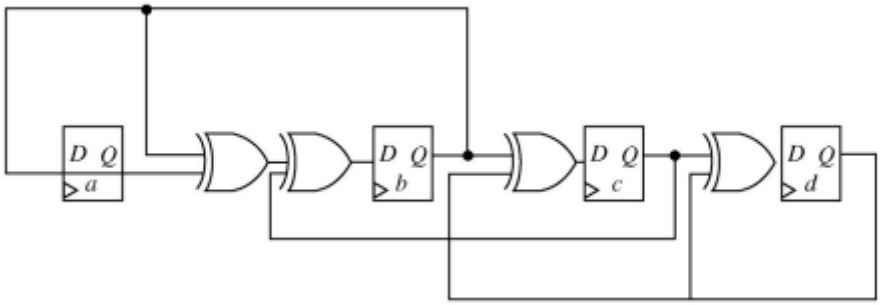
Donde $y_i(t)$ denota el estado de la celda **i** en el tiempo **t**. Se puede demostrar que combinando ambas ecuaciones es posible generar una secuencia de longitud máxima igual a $\mathbf{2^s - 1}$, donde **s** es el número de celdas del automatismo **CA**. Para ilustrar esto vemos un automatismo formado por cuatro celdas. En dicha figura donde dice regla 90 se refiere a la primera ecuación y donde dice regla 160, a la segunda,



Asumiendo que el primer estado de este automatismo es abcd = 0101, la longitud máxima de la secuencia para este CA se muestra en la siguiente tabla:

a	*b*	*c*	*d*
0	1	0	1
1	1	0	1
1	0	0	1
0	1	1	1
1	0	0	0
0	1	0	0
1	1	1	0
1	1	1	1
1	1	0	0
1	0	1	0
0	0	0	1
0	0	1	1
0	1	1	0
1	0	1	1
0	0	1	0

La implementación de este automatismo CA es mostrada en la siguiente figura. Tenga en cuenta que un LFSR de 4 bits que implementa un polinomio primitivo de grado 3 también generará una secuencia de longitud 16. Una CA basada en las reglas 90 y 160, puede generar todos los polinomios primitivos e irreductibles de un grado dado. Además, las CA no necesita largas realimentaciones, con lo cual sus retardos son pequeños y tiene diseños eficientes.



Pruebas determinísticas.

Las técnicas tradicionales de generación de pruebas también se pueden utilizar para crear patrones de prueba en circuitos que trabajan en el modo BIST. Los patrones de prueba y las respuestas del circuito generalmente se almacenan en una memoria ROM. Si las respuestas del circuito bajo prueba no coinciden con las respuestas esperadas se asume la presencia de una **falla**. Aunque en principio este es un esquema satisfactorio, generalmente no se utiliza por el espacio de memoria necesario para almacenar las respuestas del circuito y los patrones de prueba.

Análisis de la respuesta de salida.

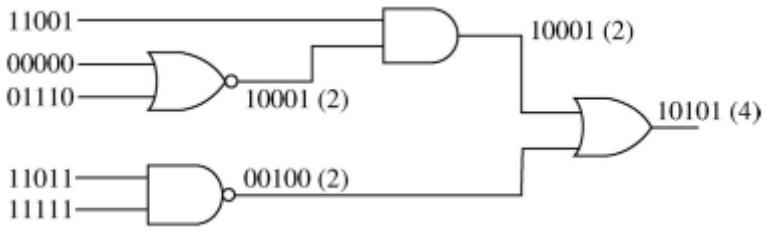
Como se indicó anteriormente, las técnicas BIST combinan un generador de pruebas binario incorporado con circuitos para comprimir los datos de respuesta del circuito bajo prueba. La respuesta de los datos comprimidos se compara con una respuesta sin fallas conocida. Durante años se han propuesto distintas técnicas de compresión que se utilizan en un entorno BIST. Entre ellas están las siguientes:

1) Contador de transiciones.
2) Comprobación de síndrome.
3) Análisis de firma.

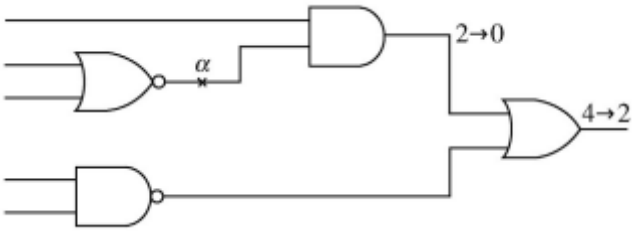
Contador de transiciones.

El recuento de transiciones se refiere a la cantidad de transiciones de 0 → 1 y 1 → 0 que se producen en la salida de un circuito debido a una secuencia de entrada dada. Por ejemplo, para una secuencia de salida igual a Z = 10011010, la cantidad de transiciones es c(Z) = 4. Por lo tanto, en vez de registrar toda la respuesta de salida de un circuito, se computa la cantidad de transiciones y se comporta con la cantidad de transiciones esperadas. Si estos dos valores difieren, el circuito bajo prueba se considera **defectuoso.**

La siguiente figura muestra un circuito con la respuesta de salida a una secuencia de entrada de longitud igual a 4.



Supongamos que en un punto del circuito denominado nodo α existe una falla del tipo s-a-0 como se muestra en la siguiente figura:



La presencia de la falla cambia la cantidad de transiciones en ciertos nodos del circuito (mostrados con flechas). Como puede verse en la figura un nodo de salida cambia de 4 transiciones a 2, esto hace que detectemos la falla en el punto α del tipo s-a-0.

La principal ventaja de este método es que no necesitamos almacenar la secuencia de respuesta correcta del circuito y tampoco la respuesta actual. Solo la cuenta de las transiciones se debe computar. Claramente esto reduce los requerimientos de almacenamiento de datos, sin embargo, esta ventaja puede traer aparejado el problema de **enmascaramiento de fallas.** Esto se debe a que la mayoría de los recuentos de transiciones corresponden a más de una secuencia. Por ejemplo, la

cuenta de transición 2 es generada por cada una de las siguientes secuencias de 5 bits: 01110, 01100, 01000, 00110, 11011 y 10001. Por lo tanto, existe la posibilidad que una secuencia errónea produzca el mismo número de recuentos de transiciones que una secuencia sin fallas, con el resultado de no ser detectada. Sin embargo, a medida que se incremente la longitud de las secuencias, disminuye el riesgo de enmascaramiento de fallas.

Comprobación de síndrome.

El síndrome de una función lógica se define con la expresión **$S = K / 2^n$**, donde **K** es el número de minitérminos de la función y **n** es la cantidad de variables independientes de la función. Por ejemplo, el síndrome de una compuerta AND de tres entradas es igual a 1/8 y el de una compuerta OR de dos entradas es igual a 1/4. Dado que el síndrome es una función propia, varias realizaciones de la misma función tienen el mismo síndrome. La relación de los síndromes para un circuito que contiene varios bloques interconectados depende de si las entradas a los bloques son disjuntas o no, así como es la compuerta donde los bloques finalizan. Para un circuito con dos bloques con entradas no compartidas, si **S_1 y S_2** denotan los síndromes de cada bloque. La relación con el síndrome de entrada salida es:

TERMINATING GATE	SYNDROME RELATION *S*
OR	$S_1 + S_2 - S_1S_2$
AND	S_1S_2
EX-OR	$S_1 + S_2 - 2S_1S_2$
NAND	$1 - S_1S_2$
NOR	$1 - (S_1 + S_2 - S_1S_2)$

Si los bloques 1 y 2 tienen entradas compartidas y realizan las funciones F y G respectivamente, se comprueban las siguientes relaciones:

$$S(F+G) = S(F) + S(G) - S(FG),$$
$$S(FG) = S(F) + S(G) - S(\overline{FG}) - 1,$$
$$S(F \oplus G) = S(\overline{F}G) + S(F\overline{G}).$$

Como ejemplo hallemos el síndrome y el número de minitérminos del siguiente circuito. En el mismo se comprueba que:

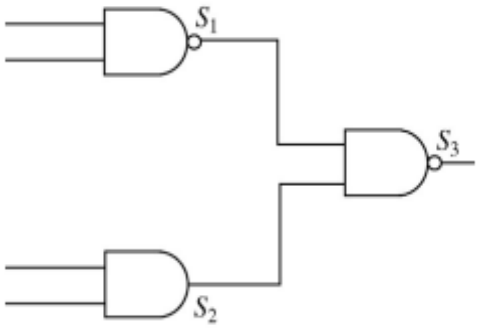
$S_1 = ¾$ y $S_2 = ¼$.

Entonces:

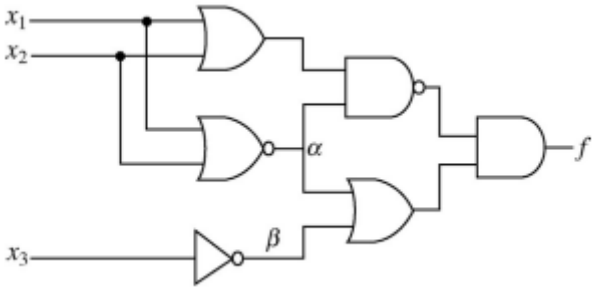
$S_3 = 1 - S_1.S_2 = 13/16$

Y

$K = S.2^n = 13$



Si ahora observamos el siguiente circuito:



La siguiente figura muestra una tabla donde se ve el circuito libre de fallas, y con dos fallas en los puntos α y β.

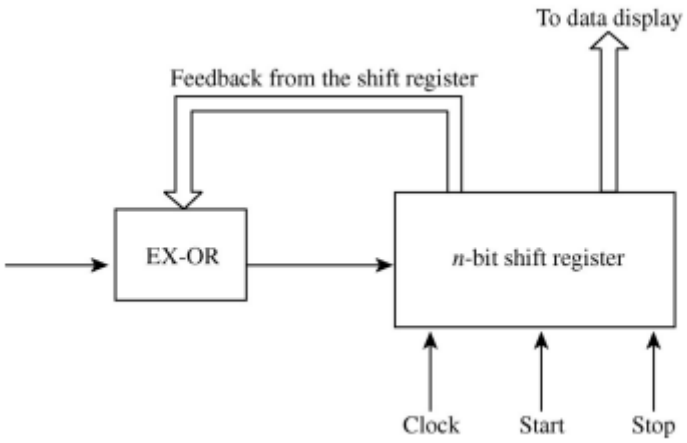
TABLE 4.2: Fault-free and faulty syndromes

	OUTPUT RESPONSE		
$X_1X_2X_3$	*FAULT-FREE*	α s-a-1	β s-a-0
000	1	1	1
001	1	1	1
010	1	0	0
011	0	0	0
100	1	0	0
101	0	0	0
110	1	0	0
111	0	0	0
Syndrome	5/8	2/8	2/8

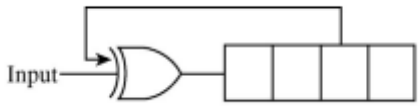
Análisis de Firmas.

La técnica de análisis de firmas es pionera de Hewlett-Packard que detecta errores en el flujo de datos debido a fallas del hardware. Se utiliza una compactación de los datos para reducir los mismos y se trasmite a un código llamado **firma**. Las firmas pueden crearse a partir de un flujo de datos que alimente un registro de desplazamiento

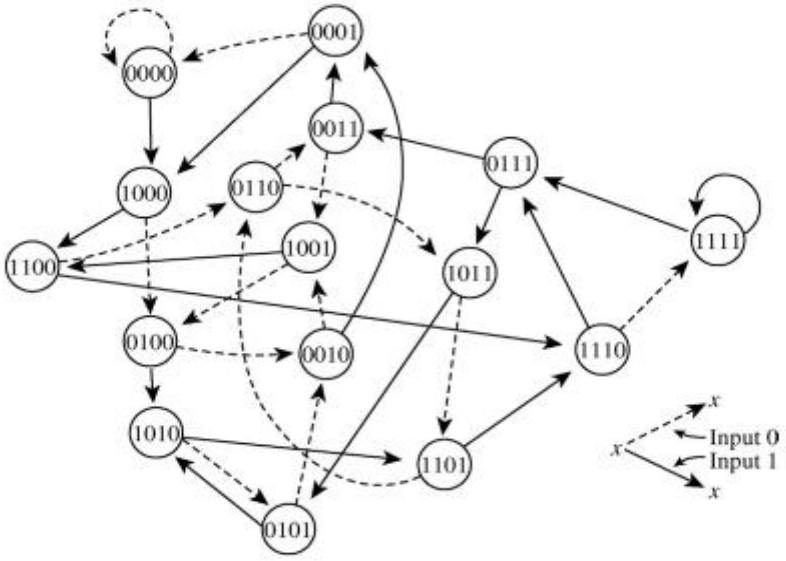
de n bits del tipo LFSR. El mecanismo de realimentación consiste en un bloque de compuertas EXOR seleccionado por una señal de entrada de datos en forma serial como muestra la siguiente figura:



Una vez que el flujo de datos es almacenado en registro de desplazamiento se deja un residuo de estos en el registro. Este residuo es único y representativo del flujo de datos y se llama **firma.** Otro flujo de datos puede diferir en 1 bit con el flujo original y su firma ser totalmente distinta. Para generar la firma de un flujo de datos, se inicializa en registro de desplazamiento en un estado conocido, generalmente todo en 0. Después se produce el desplazamiento con el flujo de datos almacenado. La siguiente figura muestra un generador de firma de 4 bits:



Asumiendo que todo el registro está en 0, se aplica un 1 a la entrada del circuito, esto hace que la salida de la compuerta EXOR valga 1. El segundo pulso de entrada hará la salida de la compuerta se desplace hacia el primer bit del registro y los demás bits quedaran en cero, lo que deja el registro con el valor 1000, o sea el 8 en decimal. En la siguiente figura se puede ver el diagrama de estados se puede visualizar la generación de firmas para cualquier flujo de entrada de datos:



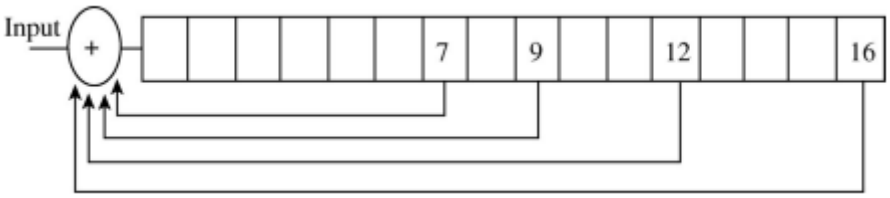
Un generador de firmas de n estados puede generar 2^n firmas, sin embargo, muchas secuencias de entrada pueden mapear en una misma firma. En general, si la longitud de los datos de entradas es **m**, el generador de firmas consta de **n** etapas, luego las 2^m posibles secuencias de entrada se mapean dentro de las 2^n firmas. En otras palabras, 2^{m-n} secuencias de entrada se mapean dentro de cada firma. Solo una de las 2^m secuencias de entrada está libre de errores y produce la firma correcta. Sin embargo, también las 2^{m-n-1} secuencias restantes pueden mapearse en una firma correcta. Este fenómeno lleva el nombre de **aliasing**, es decir la firma generada a partir de un flujo de salida defectuosa de un circuito puede generar una firma idéntica a la producida por un circuito libre de fallas. En otras palabras, se enmascara la presencia de una falla en el circuito. La probabilidad **P** de que una secuencia de entrada se ha deteriorada en otra que tengo la misma firma que ella se calcula en base a suponer que cualquier secuencia de entrada puede ser defectuosa o no. La expresión de esta probabilidad es:

$$P = \frac{2^{m-n} - 1}{2^m - 1}$$

Si se comprueba que m >>> n, la expresión vista se reduce a:

$$P = \frac{1}{2^n}$$

Esto indica que la probabilidad que se produzca aliasing se reduce si la cantidad de etapas aumenta, además con muchas etapas puede generar mucho caudal de firmas. Por ejemplo, un generador de firmas de 16 bits como se muestra en la siguiente figura:



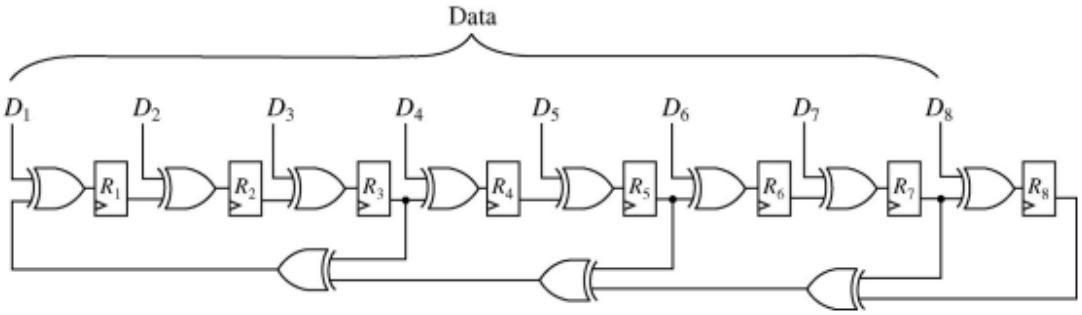
Este generador puede producir 66636 firmas, por lo tanto, aplicando la expresión anterior, la probabilidad que dos secuencias de entrada puedan generar la misma es firma es igual a 0.002 %. Las propiedades de detección de errores de este método de análisis de firmas son las siguientes:

1) La probabilidad que dos secuencias de entrada idénticas produzcan las mismas firmas es igual a 1.
2) La probabilidad que dos secuencias de entrada distintas pero que difieran en un solo bit produzcan la misma firma es igual a 0.

Por ejemplo, supongamos dos secuencias de entrada largas que difieran en un solo bit alimentando el registro de 16 bits visto en la figura anterior. A medida que el bit de error se desplaza dentro del registro tiene 4 posibilidades de cambiar el registro de entrada de firmas antes que desborde y se pierda (después de 16 ciclos de reloj). El error de realimentación sigue propagándose, cambiando las firmas correspondientes. Por ser un error de un solo bit, no existe otro error que compense la realimentación de este error.

En este caso el análisis de firma está supeditado da encontrar errores de un solo bits. Este tipo de datos son los típicos que se generan en un circuito VLSI.

Otro diseño alternativo para generar análisis de firmas es un circuito que llamado **registro de firmas de entrada múltiple (MIRS).** Un MIRS de **k bits** puede compactar una secuencia de salida de **m bits** empleando **m/k ciclos**, siendo m >>> k. Por lo tanto, un MIRS puede ser considerado como un analizador de firmas paralelo. En la siguiente figura se muestra un MIRS de 8 bits:

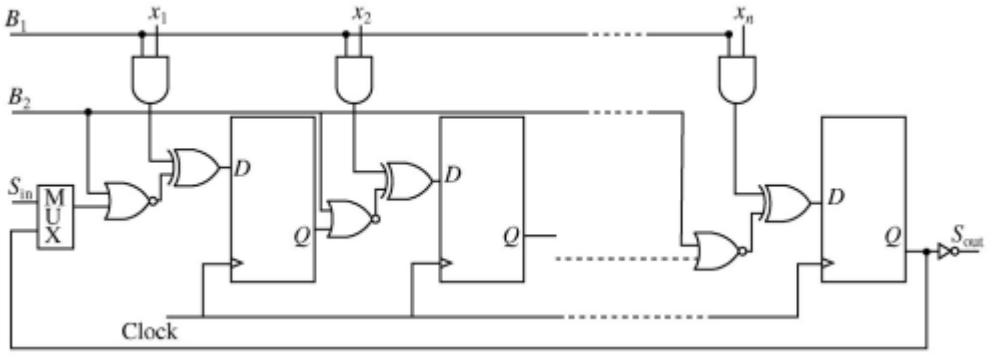


Arquitecturas BIST.

A lo largo de los años, investigadores de la industria y de las universidades han propuesto varias arquitecturas de tipo BIST. Discutiremos algunas de ellas.

Observador con bloque lógico incorporado.

Esta estructura originalmente llamada **In built-on logic block observer (BILBO)** utiliza tanto la ruta de escaneo como el generador de firmas incorporados en un solo bloque, llamado BILBO. Este bloque puede ser configurado para que funciones como un generador de pruebas de entrada como así también como generador de firmas de salida. Está formado por una fila de flip flop y una estructura adicional de compuertas para funcionar como un registro de desplazamiento y también operaciones de realimentación. En la siguiente figura se muestra el diagrama lógico de un bloque BILBO:



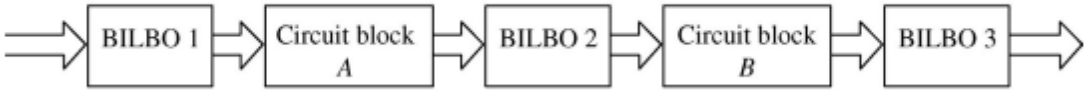
re

Las entradas de control B_1 y B_2 son usadas para seleccionar uno de los siguientes cuatro modos de operación:

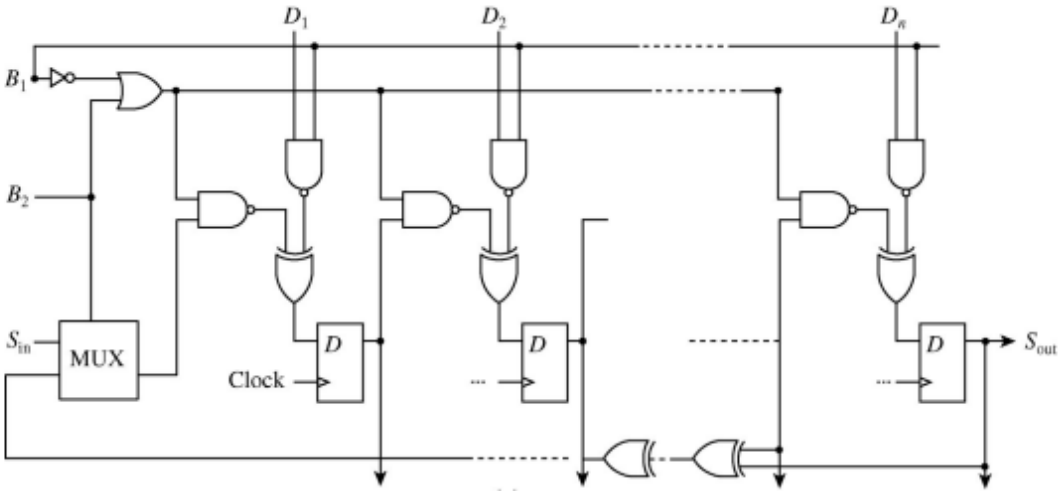
1) Modo 1. $B_1 = 0$ y $B_2 = 1$. Todos los flip flop son reseteados.

2) Modo 2. $B_1 = 1$ y $B_2 = 1$. El BILBO se comporta como un latch. Las entradas de datos x_1, x_2, ……., x_n son introducidas simultáneamente en los flip flop y pueden ser leídas en las salidas Q de los flip flops.
3) Modo 3. $B_1 = 0$ y $B_2 = 0$. El bloque BILBO funciona como un registro de desplazamiento serie. Los datos entran en el mismo a través de la señal S_{in}, mientras que el contenido del registro se puede observar en las salidas Q de los flip flops o ser sincronizadas y vistas en la salida S_{out}.
4) Modo 4. $B_1 = 1$ y $B_2 = 0$. El bloque BILBO se convierte en un MISR. En este modo, se puede usar para realizar análisis de firmas en paralelo o para generar secuencias pseudoaleatorias. La última aplicación se logra manteniendo las entradas x_1, x_2, ……., x_n con valores fijos.

La siguiente figura se ve una arquitectura BIST basada en BILBO para dos bloques de circuitos A y B conectados en cascada. BILBO 1 está configurado como un generador de patrones pseudoaleatorios, las salidas de este bloque se usan para excitar las entradas de datos del circuito A. BILBO 2 está configurado como un registro paralelo de firmas y recibe las entradas desde el circuito A. Similarmente, los bloques BILBO 2 Y BILBO 3 deben ser configurados como generadores de prueba pseudoaleatorios y como registro de firmas respectivamente. Cabe aclarar que los circuitos A y B no pueden ser testeados en paralelo, ya que BILBO 2 debe ser modificado para cambiar su función durante las pruebas de los dos bloques.



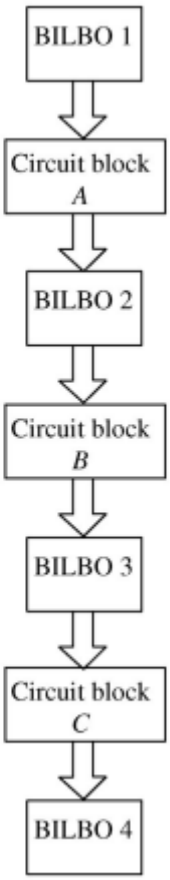
Una modificación de la estructura convencional BILBO se muestra en la siguiente figura:



Además de la función normal, el escaneo en serie y el MIRS, esta modificación del BILBO puede funcionar como un LFSR, generando patrones pseudoaleatorios de acuerdo con la siguiente tabla:

B_1	B_2	Mode
0	0	Serial Scan
0	1	LFSR
1	0	Normal
1	1	MISR

El BILBO modificado puede utilizarse para hacer pruebas simultáneas, en una estructura del tipo **pipeline**. Viendo la siguiente figura:

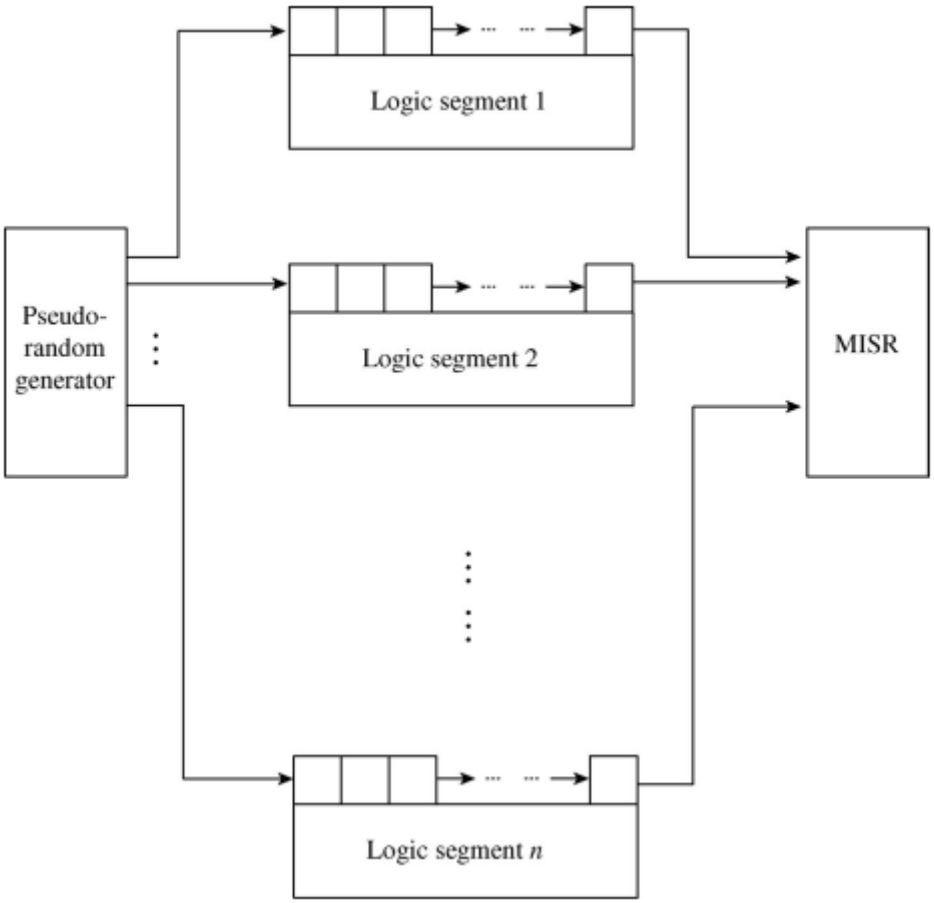


Los circuitos A y C pueden ser simultáneamente testeados a través de los BILBOs 1 y 3 en el modo LFSR y los BILBOs 2 y 4 en el modo MISR. El circuito B puede ser testeado individualmente haciendo que los BILBOs 2 y 3 operen en los modos LFSR y MISR, respectivamente.

Autocomprobación usando un MISR y un registro de desplazamiento paralelo. Generador de secuencia.

Está técnica llamada **STUMPS** usa múltiples rutas de escaneo en serie que son alimentadas por un generador de números pseudoaleatorios como lo muestra la siguiente figura. Debido a que las rutas de escaneo pueden no tener todas la misma longitud, el generador pseudoaleatorio se ejecuta hasta que se cargue la ruta de exploración más larga. Una vez que los datos se han cargado en las rutas de escaneo, el

reloj está activado. Los resultados de las pruebas se cargan en las rutas de exploración y de ahí se transfieren al MIRS.

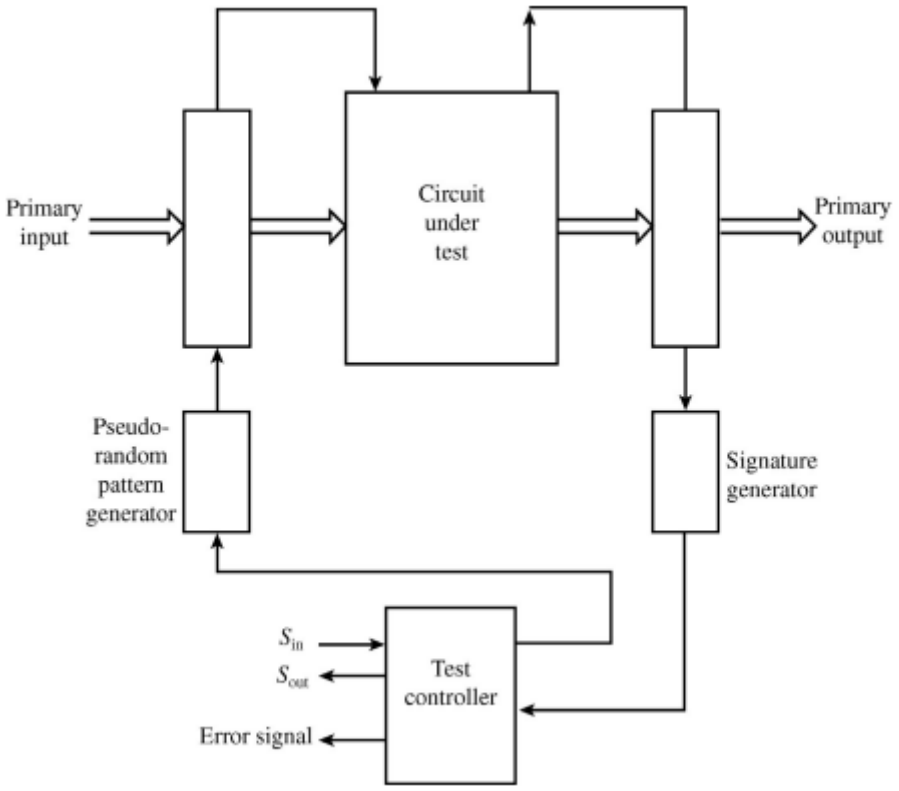


Autoprueba sobre el chip-LSSD.

La autoprueba LSSD sobre el chip llamada **LOCST** combina la prueba pseudoaleatoria con la estructura de circuito basado en LSSD. Las entradas se aplican vía celdas de escaneo de límites. Además, las salidas también se obtienen de estas celdas de escaneo. Las entradas y salidas de estas celdas de escaneo de límites más los elementos de memoria en el circuito bajo prueba forman la ruta de exploración. Algunos de los elementos de memoria al comienzo de la ruta se configuran dentro de un LFSR para generar números pseudoaleatorios. Otros elementos de memoria, más cercanos a la salida se configuran dentro de otro LFSR que funciona como un generador de firmas.

El proceso de prueba comienza cargando datos en serie en la ruta de escaneo que consiste en celdas de escaneo de límites, con patrones pseudoaleatorios generados por el LFSR. Estos patrones se aplican a la parte combinacional del circuito bajo prueba. Y los patrones de salida se cargan en forma paralelo en la ruta de escaneo consistente en las celdas de escaneo de límites de salida. Luego estos bits de salida se desplazan dentro del registro de firmas. La firma resultante se compara con la firma de referencia para proporcionar la verificación del circuito bajo prueba.

En la siguiente figura se ve el diagrama correspondiente a este método:


Primary input
Circuit under test
Primary output
Pseudo-random pattern generator
Signature generator
S_{in}
S_{out}
Test controller
Error signal