UNIDAD 10: DETECCIÓN DE FALLAS EN CIRCUITOS LÓGICOS.

(86:44) Técnica Digital Avanzada- Unidad 9.

Profesor: Ing. Miguel Antonio Martínez.

Detección de fallas en circuitos lógicos.

El objetivo de hacer pruebas a nivel de compuertas es comprobar si cada una de ellas funciona correctamente y están bien las interconexiones entre las mismas. Si asumimos que en circuito hay una sola falla del tipo de bloqueo, el problema es construir un conjunto de vectores de prueba que detecte las fallas usando solo las entradas y las salidas del circuito.

Uno de los principales objetivos de las pruebas sobre el circuito es minimizar el número de patrones o vectores de test.

Si la función de un circuito con una falla es diferente al comportamiento de este sin falla (es decir un circuito no redundante), entonces un circuito combinacional de **n** entradas puede completamente testeado aplicando las $\mathbf{2^n}$ combinaciones posibles. Sin embargo, 2^n aumenta muy rápidamente a medida que crece n. Para un circuito secuencial de **n** entradas y **m** flip flop, el número total de pruebas para testear este circuito exhaustivamente es $\mathbf{2^n x\ 2^m = 2^{n+m}}$**.** Si, por ejemplo, n = 20 y m = 40, habría $\mathbf{2^{60}}$ pruebas distintas. A una velocidad de 10000 test por segundo, el tiempo total de prueba estaría cerca de 3,65 millones de años. Afortunadamente no es necesario usar todas las combinaciones de la tabla de verdad, si no es necesario las combinaciones de entrada que me permitan detectar la mayor cantidad posibles de fallas.

La eficiencia de un equipo de prueba se mide con una figura de mérito llamada **cobertura de fallas**. Este término se refiere al porcentaje de fallas del tipo bloqueo que sean únicas que este sistema detectará.

El tiempo computacional necesario para generar pruebas en un circuito combinacional es proporcional al cuadrado del número de compuertas que tenga el. mismo. Por ejemplo, el tiempo para testear un circuito con 100000 compuertas es 100 veces más grande que si el circuito tuviera 10000 compuertas.

La tarea es aún más complicada en circuitos secuenciales porque el número de estados internos es una función exponencial de los elementos de memoria que contenga el mismo. Por lo tanto, los circuitos secuenciales deben diseñarse de forma tal que la tarea de testing no sea muy engorrosa.

Generación de pruebas para circuitos combinacionales.

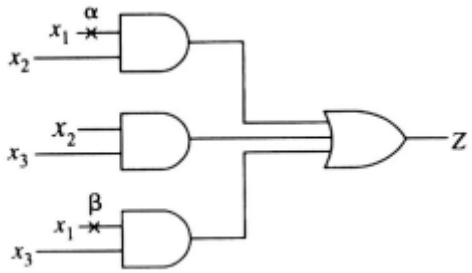
Se han desarrollado muchos métodos distintos de generación de pruebas a lo largo de los años. Todos se basan en la suposición de que el circuito es no redundante y se presenta una sola falla en el momento de la prueba.

El método más sencillo para generar pruebas es comparar la tabla de verdad del circuito sin fallas con la respuesta del circuito con falla para todas las combinaciones de entrada. Cualquier combinación de entradas en la cual las salidas no coinciden es una **prueba** para el circuito con falla.

Dado un circuito combinacional con entradas x_1, x_2,……., x_n y una salida Z. Llamaremos Z_α a la salida en presencia de una falla α. El método de generación de pruebas comienza con la construcción de una tabla de verdad para Z y Z_α. Se computa para cada combinación de entrada la función $Z \ominus Z_\alpha$, o sea una función exor entre la

salida sin falla y con falla. Si esta función da 1, esta combinación de las variables de entrada es un **vector de prueba** para esta falla.

Como ejemplo vemos el circuito de la siguiente figura:



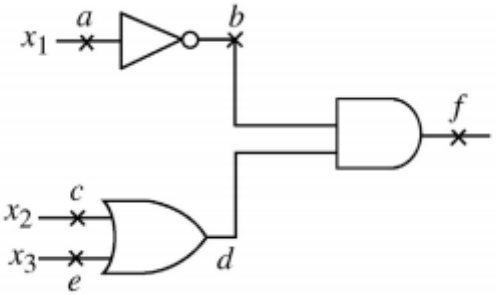
Asumimos el testeo para las fallas de α s-a-0 y de β s-a-1. La tabla se muestra en la siguiente figura:

x_1	x_2	x_3	z	z_α	z_β	$z + z_\alpha$	$z + z_\beta$
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	1
0	1	0	0	0	0	0	0
0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0
1	1	0	1	0	1	1	0
1	1	1	1	1	1	0	0

El vector para la falla es indicado con un 1 en las columnas Z ⊖ Z_α y Z ⊖ Z_β respectivamente. Por lo tanto, se ve que el vector de prueba para la falla α s-a-0 es $x_1x_2x_3$ = 110 y para β s-a-1 es $x_1x_2x_3$ = 001. Para todas las otras combinaciones de entrada no hay diferencia entre el circuito con falla y sin falla. Por lo tanto, no son vectores de prueba para α s-a-0 y para β s-a-1.

El número mínimo de vectores de prueba se pueden hallar con lo que se llama una **matriz de fallas**. La misma consiste en tener como columnas las fallas que se van a probar y como filas indican las pruebas.

Para ver esta matriz nos fijamos en el siguiente circuito:



La matriz de fallas para este circuito es la siguiente:

x_1	x_2	x_3	a s-a-0	b s-a-0	c s-a-0	d s-a-1	e s-a-0	f s-a-1
0	0	0				1		1
1	0	0						1
0	1	0		1	1			
1	1	0	1					1
0	0	1		1			1	
1	0	1	1					1
0	1	1		1				
1	1	1	1					1

Cada 1 que aparece en una columna corresponde a una falla y la fila correspondiente me da el vector de prueba para detectar esa falla.

Como vemos en figura anterior una matriz de fallas es idéntica a una Tabla de Implicantes Primos. Por lo tanto, ahora debemos encontrar una cobertura mínima de Implicantes Primos. Mirando la tabla vemos que las combinaciones de entrada 110, 101 y 111 son **equivalentes**. O sea, detectan la misma falla. Por lo tanto 101 y 111 pueden ser omitidas. Además, la fila 000 cubre la fila 100 y la fila 001 cubre la combinación 011. Por lo tanto, las filas 100 y 011 pueden ser omitidas. Eliminando las filas 100, 101, 011 y 111 nos queda un conjunto mínimo de vectores de test como se ve en la siguiente figura:

x_1	x_2	x_3	a s-a-0	b s-a-0	c s-a-0	d s-a-1	e s-a-0	f s-a-1
0	0	0				1		1
0	1	0		1	1			
1	1	0	1					1
0	0	1		1			1	

Estos cuatro vectores detectan las seis fallas consideradas. A partir de este ejemplo se ve que este método de matriz de fallas es impracticable cuando la cantidad de variables de entrada es grande.

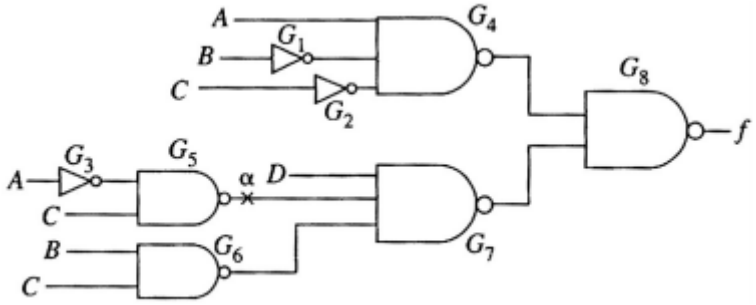
Ahora discutiremos algunas técnicas alternativas desarrolladas para resolver el problema de generación de pruebas.

Sensibilización de caminos.

El principio básico del método de **sensibilización de caminos** se basa en elegir una camino o ruta que me permita llegar desde el punto de la falla hasta la salida del circuito.

Se basa en el principio que el valor de la entrada de una compuerta, el efecto de la falla pueda ser propagado hasta la salida del circuito.

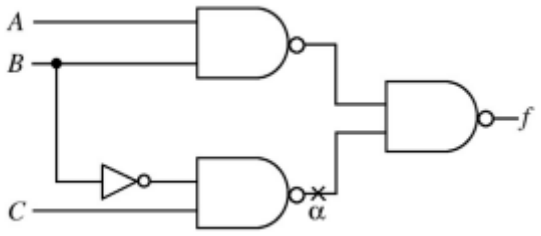
Para ilustrar el método vemos el circuito de la próxima figura:



Asumimos que el punto α es s-a-1. Para probar α, ambas G_3 y C deben estar seteadas en 1. Además, D y G_6 deben permanecer en 1 para que G_7 tome el valor 1 si la falla está ausente. Para propagar la falla desde G_7 hacia la salida del circuito vía G_8, requiere que la salida de G_4 esté en 1. Esto es porque si $G_4 = 0$, la salida f del circuito es forzada a 1, independientemente del valor a la salida de la compuerta G_7 . El proceso de propagación el efecto de una falla desde la locación original hacia la salida del circuito se conoce como seguimiento **directo.**

La próxima fase del método es lo que se llama **traza hacia atrás**. Esto consiste en fijar un valor de salida y buscar el camino hasta las entradas. Por ejemplo, si seteamos G_3 en 1, A debe estar en 0, lo que también establece que $G_4 = 1$. Siguiente el orden para que G_6 sea 1, B debe estar 0. Nótese que G_6 no puede ser seteada en 1 haciendo C = 0 porque esto sería inconsistente con la asignación de C en la fase de seguimiento directo. Por lo tanto, el vector ABCD = 0011 detecta la falla de α s-a-1, ya que la salida f será 0 para el circuito sin falla y será 1 en presencia de esta falla.

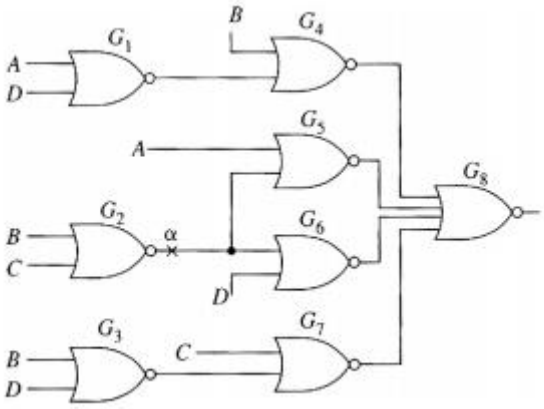
En general, un patrón de prueba obtenido por este método puede no ser único. Para lo cual vemos la siguiente figura:



En este circuito la falla α s-a-0 puede ser detectada con el patrón ABC = 01- o 0-0. En el primer patrón, C es inespecificada y el segundo lo es B. Un valor inespecífico indica que la prueba es independiente de la correspondiente entrada.

El mayor inconveniente de este método es que solo se sensibiliza una ruta por vez. Esto no garantiza que se pueda encontrar una falla, aunque exista.

Como ejemplo veamos la siguiente figura:



Suponemos una falla en el punto α del tipo s-a-0. Si propagamos la falla a través del camino G_2-G_6-G_8 requiere que B, C, y D permanezcan en 0. Para propagar la falla hasta G_8, es necesario que $G_4 = G_5 = G_7 = 0$. Dado que B y C se han establecido en 0, G_3 es 1, lo que hace que $G_7 = 0$. Para configurar $G_5 = 0$, A debe ser 1, como resultado $G_1 = 0$, que con B = 0 hará que $G_4 = 1$. Por lo tanto, no es posible propagar la falla a través de G_8. Del mismo modo no es posible sensibilizar el camino G_2-G_5-G_8. Sin embargo, A = 0 sensibiliza dos rutas simultáneamente y también hace $G_4 = 0$. Así dos entradas de G_8 cambian de 0 a 1 como resultado de una falla en α s-a-0, mientras que las dos otras entradas permanecen en 0. Consecuentemente, ABCD = 0000 causa que la salida del circuito cambie de 1 a 0 en presencia de la falla en α del tipo s-a-0, y es la prueba de la falla mencionada.

Esto demuestra que es necesario sensibilizar más de una ruta o camino al buscar pruebas para ciertas fallas. Esta es la principal idea que incorpora el próximo método llamado D-algoritmo.

D – Algoritmo.

Este método garantiza que encontrará un vector de prueba si existe una falla a detectar. Para lograr la cobertura de generación de test de prueba usa los conceptos de algebra booleana en forma cúbica. Considera tres tipos de cubos booleanos:

a) Los cubos singulares.
b) Los cubos de propagación D.
c) Los cubos primitivos de falla D.

Los **cubos singulares** corresponden a los **Implicantes Primos** de la función. En la siguiente figura vemos los cubos singulares para una compuerta NOR de dos entradas. Las posiciones nombradas con **x** corresponden a las redundancias, o sea que pueden ser 0 o 1.

a	b	c
0	0	1
x	1	0
1	x	0

Los cubos de **propagación D** representan el comportamiento de las entradas y las salidas funcionando bien y funcionando con falla. El símbolo **D** puede asumir valores 0 o 1

El cubo $\overline{D}$ representa el valor opuesto a D. Ejemplo si D=1, entonces $\overline{D}$=0 y viceversa. Las definiciones de D y de $\overline{D}$ pueden intercambiarse, pero tienen que se consistentes con la estructura del circuito en estudio. Entonces todas las D de un circuito indican que pueden tomar valores 0 o 1 y siempre $\overline{D}$ es el valor opuesto.

El cubo de propagación D de una compuerta causa que la salida de una compuerta dependa solo de una o varias entradas especificadas. Por lo tanto, una falla en una entrada especificada de la compuerta se propaga a la salida.

Como ejemplo vemos los cubos de propagación D para una compuerta NAND de dos entradas:

a	b	f
1	D	$\overline{D}$
D	1	$\overline{D}$
D	D	$\overline{D}$

Los cubos D de propagación 1D$\overline{D}$ y D1$\overline{D}$ indican que una de las entradas de la compuerta son 1. DD$\overline{D}$ propaga múltiples cambios de entrada a través de la compuerta NAND. Los cubos de propagación D se pueden construir haciendo intersecciones de los cubos singulares con los valores de salidas. Las reglas de intersección son las siguientes:

$$0 \cap 0 = 0 \cap x = x \cap 0 = 0$$
$$1 \cap 1 = 1 \cap \mathrm{x} = \mathrm{x} \cap 1 = 1$$
$$x \cap x = x$$
$$1 \cap 0 = D$$
$$0 \cap 1 = \overline{D}$$

Como ejemplo, los cubos de propagación D para una compuerta NOR de tres entradas se ilustra en la siguiente figura:

	a	b	c	f
c_1	0	0	0	1
c_2	x	x	1	0
c_3	x	1	x	0
c_4	1	x	x	0

(a) Singular covers of the NOR gate

	a	b	c	f
$c_2 \cap c_1$	0	0	D	$\overline{D}$
$c_3 \cap c_1$	0	D	0	$\overline{D}$
$c_4 \cap c_1$	D	0	0	$\overline{D}$

(b) Propagation D-cube of the NOR gate

El cubo primitivo D de una falla (pdcf) se usa para especificar la existencia de una determinada falla. Consiste en un patrón de entrada que muestra el efecto de una falla en la salida. Por ejemplo, si en la compuerta NOR que vimos de dos entradas falla es del tipo s-a-0, el correspondiente pdcf es:

a	b	f
0	0	D

Aquí, D se interpreta como 1 si el circuito está libre de falla y como 0 si hay presente una falla. El pdcf de la misma compuerta NOR con una falla de la salida del tipo s-a-1 es:

a	b	f
1	x	$\overline{D}$
x	1	$\overline{D}$

Los pdcf correspondientes a una falla de salida del tipo s-a-0 en una compuerta pueden obtenerse por la intersección de cada cubo singular con salida 1 con la compuerta sin fallas y cada cubo singular con salida 0 con la compuerta defectuosa. De forma similar, los pdcf de una falla del tipo s-a-1 en una salida puede obtenerse por la intersección de cada cubo singular con la salida en 0 en una compuerta sin fallas y lo mismo con cada cubo singular con salida en 1 con una compuerta con falla. Las reglas de intersección son similares a las usadas en los cubos de propagación D.

Como ejemplo, consideremos una compuerta NAND de tres entradas que llamamos **a, b y c**. Además, tiene una salida que denominamos **f**. Los cubos singulares para la compuerta libre de fallas son:

	a	b	c	f
c_1	0	x	x	1
c_2	x	0	x	1
c_3	x	x	0	1
c_4	1	1	1	0

Asumiendo que la entrada **b** tiene una falla del tipo s-a-1, los cubos singulares para este problema son:

	a	b	c	f
c_1'	0	x	x	1
c_2'	x	x	0	1
c_3'	1	x	1	0

De esta tabla se desprende que:

$$c_1 \cap c_3' = \overline{D}x1D \quad c_4 \cap c_1' = D11\overline{D}$$
$$c_2 \cap c_3' = 101D \quad c_4 \cap c_2' = 11D\overline{D}$$
$$c_3 \cap c_3' = 1x\overline{D}D$$

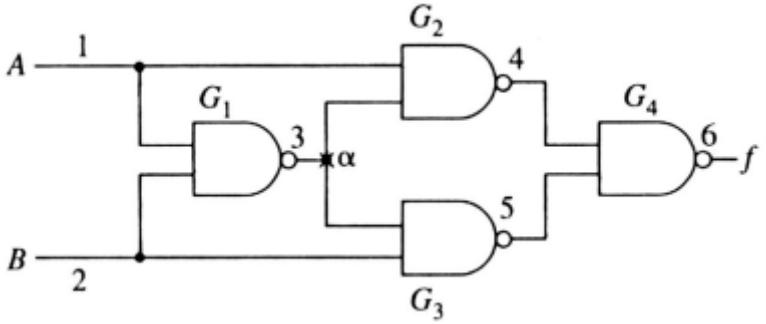
Por lo tanto, el cubo primitivo D para una falla en la entrada **b** del tipo s-a-1 es **101D**. Los pdcf para todas las fallas del tipo stuck at fault de una compuerta de tres entradas del tipo NAND son:

a	b	c	f	Fault
0	x	x	D	f s-a-0
x	0	x	D	f s-a-0
x	x	0	D	f s-a-0
1	1	1	$\overline{D}$	f s-a-1
0	1	1	D	a s-a-1
1	0	1	D	b s-a-1
1	1	0	D	c s-a-1

Consideremos a continuación como se utilizan los distintos cubos descritos en el método del algoritmo D para generar una prueba para una falla determinada. El proceso de generación de la prueba lleva tres pasos:

a) Seleccione un pdcf para la falla dada.
b) Conduzca el D (o $\overline{D}$) desde la salida de la compuerta bajo prueba a una salida del circuito interceptando el cubo actual bajo prueba con los cubos de propagación D de sucesivas compuertas. Un cubo de prueba representa los valores de la señal en varias líneas del circuito durante cada paso del proceso de generación de pruebas. La intersección de un cubo de prueba con el cubo de propagación D de una compuerta sucesora da como resultado el cubo de prueba.
c) Justifique los valores de una línea interna conduciendo hacia las entradas del circuito. Asignar valores a las variables de entrada de modo de obtener un conjunto coherente de valores de entrada del circuito obtenido.

Demostraremos la aplicación del **Algoritmo D** derivando para encontrar una falla en el punto α del tipo s-a-1 en el circuito de la siguiente figura:



La generación del proceso de testeo se ve en la siguiente figura:

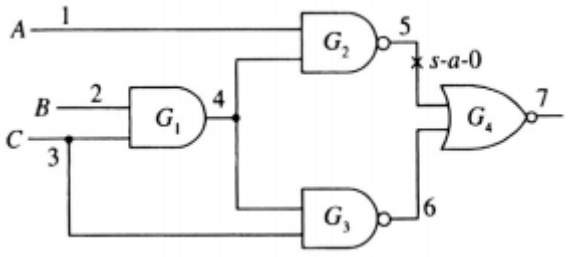
	1	2	3	4	5	6
Step 1: Select pdcf for α s-a-1.	1	1	$\overline{D}$	x	x	x
Step 2: Intersect the test cube with an appropriate propagation D-cube of G_2 (e.g., $1\overline{D}D$). (N.B., the parity of the D-cube is inverted.)	1	1	$\overline{D}$	D	x	x
Step 3: Intersect the test cube with the propagation D-cube $D1\overline{D}$ of G_4.	1	1	$\overline{D}$	D	1	$\overline{D}$
Step 4: Check that line 5 is at 1 from G_3 singular cubes.	1	#	$\overline{D}$	D	1	D

La operación de consistencia en el paso 4 termina sin éxito porque la salida de G_3 tiene que establecerse en 1. Esto solo puede realizarse si B = 0, sin embargo, a B se le ha asignado el valor 1 en el paso 1. Un problema similar tenemos si D es propagada a la salida vía G_3 en vez de G_2. La única forma que se puede resolver el problema de coherencia es si la salida D de G_1 se propaga a la salida del circuito vía G_2 y G_3 como muestra la siguiente tabla:

	1	2	3	4	5	6
Step 1: Select pdcf for α s-a-1.	1	1	$\overline{D}$	x	x	x
Step 2: Intersect the test cube with the propagation cube $1\overline{D}D$ of G_2.	1	1	$\overline{D}$	D	x	x
Step 3: Intersect the test cube with the propagation cube $\overline{D}1D$ of G_3.	1	1	$\overline{D}$	D	D	x
Step 4: Intersect the test cube with the propagation cube $DD\overline{D}$ of G_1.	1	1	$\overline{D}$	D	D	$\overline{D}$

No se necesita ningún análisis de consistencia en este caso, y el test para esta falla es AB = 11. Este mismo patrón de prueba sirve para detectar una falla en la salida de G_2 del tipo s-a-0, una falla en la salida de G_3 del tipo s-a-0 y, por último, una falla en la salida de G_4 del tipo s-a-1.

Como un ejemplo más de la aplicación de este algoritmo, derivemos un test para una falla del tipo s-a-0 en la salida de la compuerta G_2 del circuito de la siguiente figura:



El procedimiento para encontrar el vector de prueba se muestra en la siguiente tabla:

	1	2	3	4	5	6	7
Step 1: Select pdcf for line 5 s-a-0	0	x	x	1	D	x	x
Step 2: Intersect the test cube with the propagation D-cube $D\bar{0}D$ of G_4	0	x	x	1	$\overline{D}$	0	D
Step 3: Check line 6 is at 0 from the G_3 singular cubes; set input C at 1	0	x	1	1	$\overline{D}$	0	D
Step 4: Check line 4 is at 1 from the G_1 singular cubes; set input B at 1	0	1	1	1	$\overline{D}$	0	D

De esta tabla se desprende que el vector de prueba es ABC = 011.

PODEM.

Es un algoritmo de enunciación donde todas las combinaciones de entrada se examinan como pruebas para una determinada falla. La búsqueda de una prueba continúa hasta que se termine el espacio de búsqueda o se encuentre un vector de prueba. Si no se encuentra un vector de prueba, la falla se considera indetectable.

En el algoritmo D cuando reflejamos hacia atrás se le podía asignar valores a líneas internas para llegar a la entrada. En PODEM no se puede asignar valores a las líneas internas, solo a salidas y entradas. Esto reduce el número de retrocesos posibles. El método consta de 6 pasos:

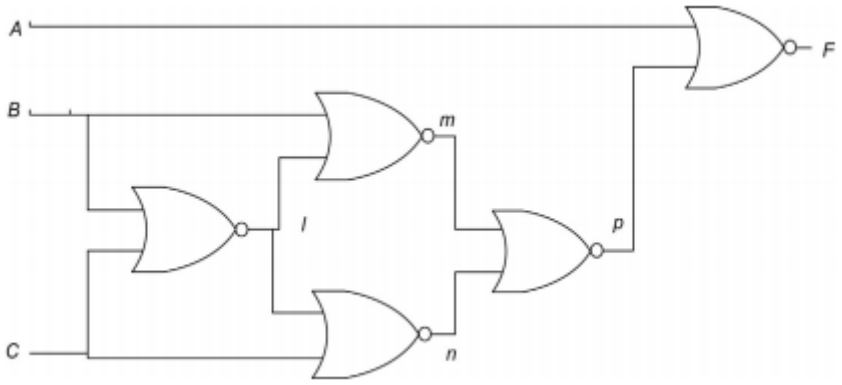
1) Suponga que todas las entradas primarias son **x**, o sea no están especificadas. Determine un objetivo inicial. Un objetivo se define como un valor lógico (0 o 1) denominado nivel lógico objetivo. El objetivo inicial es seleccionar un valor lógico que **sensibilice** la falla a detectar.
2) Seleccione una entrada primaria y asigne un valor lógico que tengas buenas probabilidades de satisfacer el objetivo inicial.
3) Propague el valor de la entrada seleccionada en conjunto con todas las **x** de las otras entradas hacia la salida usando cinco valores lógicos que son 0, 1, X, D y $\overline{D}$.
4) Si resulta un vector de prueba, se propaga D o $\overline{D}$ hacia la salida del circuito. Salga, de lo contrario propague el complemento del valor lógico asignado a la entrada seleccionada y determinar si es un vector de prueba.
5) Asigne un 0 o un 1 a otra entrada más y repita el paso 4 para chequear si esta nueva combinación es un vector de prueba.
6) continúe con los pasos 4 y 5 hasta que se encuentre un vector de prueba o se llegue a la conclusión que la falla es indetectable.

Las diferencias entre este método PODEM y el Algoritmo D son las siguientes:

1) En PODEM el retroceso solo se permite en las entradas primarias, no en **ninguna línea interna.**

2) PODEM no necesita la operación de verificación de inconsistencias.

Veremos un ejemplo suponiendo una falla del tipo s-a-1 en el punto **l** del siguiente circuito:



Dado que se va a establecer una prueba para la falla s.a.1 en L, el objetivo inicial es establecer l en 0. Se puede asignar 1 a B o C para satisfacer el objetivo. Asumiendo que se elige un 1 para B, el resultado de la propagación hacia adelante es:

A	B	C	l	m	n	p	F
X	1	1	$\overline{D}$	0	0	X	X

El siguiente objetivo es propagar D o $\overline{D}$ desde **n** hacia la salida **F.** Esto se puede hacer asignando un valor determinado a la entrada **C**. Suponemos que seteamos el valor de C en 1, por lo tanto, queda lo siguiente:

A	B	C	l	m	n	p	F
X	1	1	$\overline{D}$	0	0	X	X

Esto bloquea la propagación de D porque **n** se fuerza a 0. Sin embargo, si ha C se le asigna el valor 1, D es propagado a través de **n**.

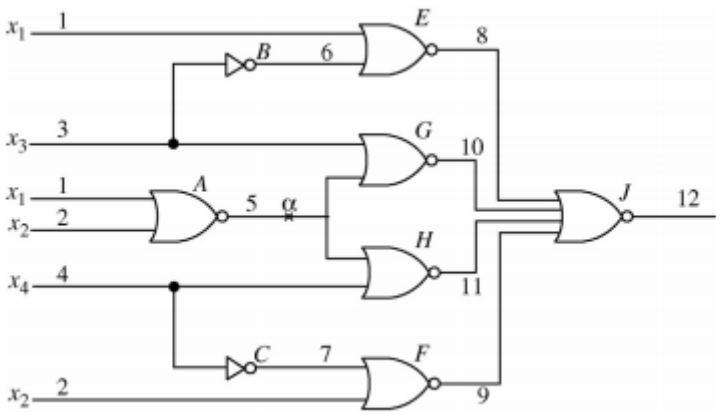
A	B	C	l	m	n	p	F
X	1	0	$\overline{D}$	0	D	$\overline{D}$	X

El objetivo final es propagar D o $\overline{D}$ hacia la salida F. Esto se puede hacer asignando valores lógicos en las entradas, por ejemplo, haciendo A = 0, lo que resulta:

A	B	C	l	m	n	p	F
0	1	0	$\overline{D}$	0	D	$\overline{D}$	D

Por lo tanto, el vector de prueba para una falla en el punto l del tipo s-a-1 es ABC = 010.

Como ejemplo adicional consideremos el circuito de la siguiente figura donde tenemos una falla en el punto α del tipo s-a-0.



El objetivo inicial es setear la salida de la compuerta A con un valor lógico 1. Las entradas primarias x_1 y x_2 excitan a la compuerta A. Asignemos primero, 0 a x_1 y nos queda los valores de la siguiente manera:

1	2	3	4	5	6	7	8	9	10	11	12
0	X	X	X	X	X	X	X	X	X	X	X

Dado que el vector $x_1x_2x_3x_4$ = 0XXX no es una prueba para esta falla, asignamos el valor 0 a x_2 (que es la otra entrada primaria de la compuerta A), lo que configura como D a la salida de la compuerta A. Por lo tanto, nos queda:

1	2	3	4	5	6	7	8	9	10	11	12
0	0	X	X	D	X	X	X	X	X	X	X

Debido a que la salida de la compuerta A, en la figura llamada nodo 5, no es X, es necesario buscar otra compuerta mas cercana a la salida primaria que tenga una D como entrada y una X como salida. Ambas compuertas G y H satisfacen este requerimiento.

La elección de la compuerta G y la subsecuente asignación de 0 en su entrada primaria X_3 resulta lo siguiente:

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	X	D	1	X	0	X	$\overline{D}$	X	X

Está claro que el vector $x_1x_2x_3x_4$ = 000X no es una prueba para esta falla porque una entrada primaria es X. La compuerta J tiene $\overline{D}$ como entrada desde el nodo 10 y X_s en las entradas provenientes de los nodos 9 y 11. El objetivo inicial es setear el nodo 12 con un valor igual a 1. La selección del nodo 9 como próximo objetivo resulta en asignar un 0 a la entrada primaria enumerada como x_4. Por lo tanto, todo queda así:

1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	D	1	1	0	0	$\overline{D}$	$\overline{D}$	D

Por lo tanto, se desprende, que el vector de prueba para la falla en el punto α del tipo s-a-0 es $x_1x_2x_3x_4$ = 0000.

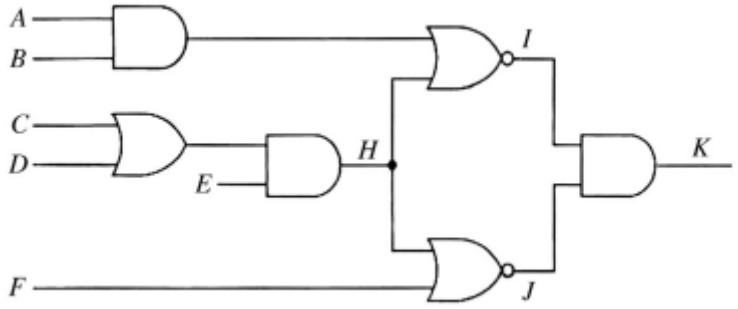
Se pudo encontrar la falla usando el método de Algoritmo-D, sin embargo, este requiere de más operaciones de ensayo y error. Esto se debe a las múltiples rutas de propagación y las pruebas de consistencia.

PODEM es más eficiente que el Algoritmo-D pues utiliza mucho menos tiempo de procesamiento en busca de las pruebas.

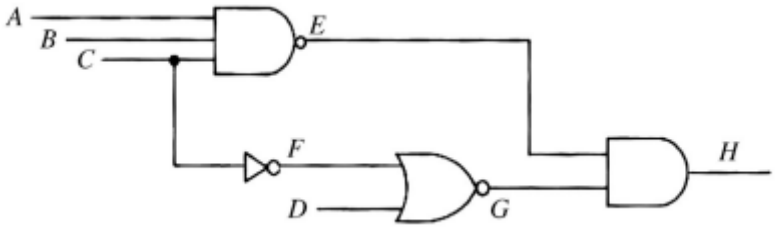
FAN.

El algoritmo FAN es en principio, similar al PODEM, pero es más eficiente porque reduce la cantidad de retrocesos. Se deben definir varios términos antes de explicar como es el mecanismo de generación de pruebas de este algoritmo nuevo. Una **línea acotada** es la salida de una compuerta que es parte de conexiones a otras puertas, o sea donde tenemos en cuenta el fan-out. Una línea que no sea **acotada** se conoce como **línea libre**. Una línea **titular** es una línea libre que excita una compuerta cuya salida está conectada a otras, o sea que es acotada. En la siguiente figura, los nodos H, I y J son líneas acotadas. De A hasta H son líneas libres y H y F son titulares. Dado que, por definición, las líneas titulares son líneas libres, se pueden considerar líneas primarias y, por ende, se pueden asignar valores arbitrarios. Por lo tanto, en el proceso de retroceso, si se llega una línea titular no es necesario seguir hasta una entrada primaria y el proceso se detiene.

FAN usa una técnica llamada **múltiples retrocesos** que reduce el numero de retrocesos durante el proceso de búsqueda.

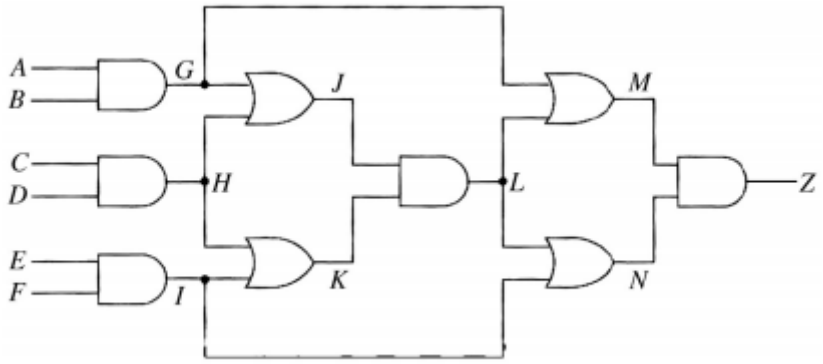


Como ejemplo veremos la siguiente figura:



El objetivo es setear el nodo H en 1. PODEM retrocederá por una ruta hasta las entradas primarias. Supongamos que el retroceso se hace a través de la ruta H-E-C que establece E en 1. Como E está en uno, C debe estar en 0. Sin embargo, un 0 en C, hace que F esté en 1, G en 0 y H en 0. Esta asignación no logra el objetivo deseado, por lo tanto, el retroceso debe hacerse por otra ruta, por ejemplo, H-G-F-C, y el objetivo planteado puede alcanzarse. Por lo tanto, en PODEM se realizan varios retrocesos hasta alcanzar el objetivo planteado. FAN supera este problema retrocediendo por varios caminos. Por ejemplo, si se realiza un retroceso múltiple a través de H-E-C y H-G-F-C, el valor de C se puede setear de tal forma de cumplir con el objetivo en H. En PODEM, un valor inicial en una entrada primaria puede no lograr el objetivo deseado y esto hace que tengamos que volver a empezar.

Aplicaremos el método FAN para el circuito de la siguiente figura:

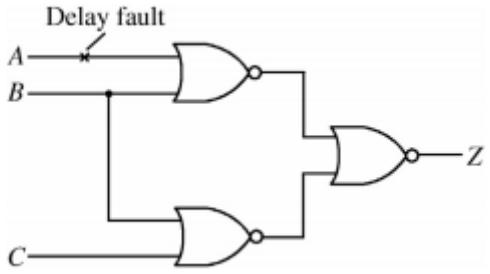


Consideremos una falla del tipo s-a-0 en el nodo Z. Primero se asigna el valor D al nodo Z y a cada una de las entradas M y N el valor 1. El objetivo inicial es setear M y N en 1. Mediante múltiples retrocesos, G y I son asignados en 1 (tener en cuenta que en lugar de G y I se podría haber seteado L en 1). Otra vez, por los múltiples retrocesos, tenemos el objetivo final A =1, B = 1, E = 1 y F = 1. La asignación A = 1, B = 1 hace que J = 1 y M = 1 y la asignación E = 1 y F = 1 hace que I = 1 y N = 1. Por lo tanto, la asignación A = B = E = F = 1 constituye un vector de prueba para la falla s-a-0 en el nodo Z. Es fácil ver que los primeros múltiples retrocesos se detenían en L y el segundo retroceso se detendría en H, la prueba sería C = D = 1.

Detección de Fallas de Retardo.

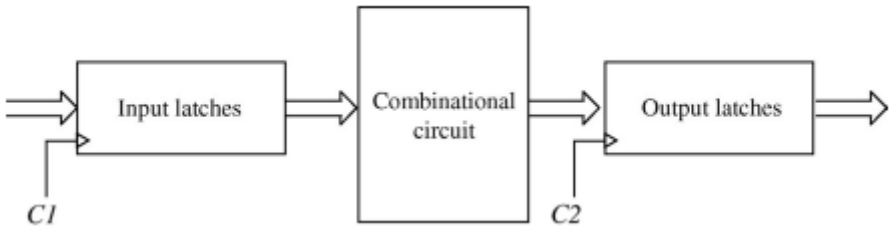
Una falla de retardo en un circuito combinacional solo puede ser detectada aplicando una secuencia de dos patrones de prueba. El primer patrón llamado **patrón de inicialización,** establece la condición inicial en un circuito de modo que la falla (que no es otra cosa que una señal de subida o de bajada lenta) en la entrada o en la salida de un circuito afecte la salida del circuito. El segundo patrón, conocido como **patrón de transición o de propagación,** propaga el efecto hacia una salida primaria del circuito.

Como ejemplo veremos el circuito de la siguiente figura:



En este circuito suponemos una falla de retardo del tipo subida lenta en la entrada A. El test para estudiar esta falla de subida lenta consiste en dos patrones como dijimos anteriormente. El padrón de inicialización seria ABC = 001 y el de propagación ABC = 101. De forma análoga si quisiéramos detectar una falla de bajada lenta los patrones a usar serían ABC = 101 y ABC = 001 respectivamente. Nótese que una falla de subida o de bajada lenta, no es otra cosa que fallas del tipo s-a-0 o s-a-1 **transitorias.**

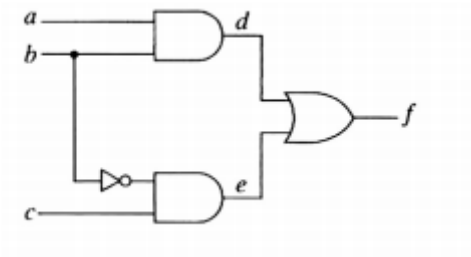
Para identificar fallas de retardo en circuitos combinacionales, el modelo de hardware que se utiliza es el mostrado en la siguiente figura:



El patrón de inicialización es cargado en el latch de entrada, una vez estabilizado el circuito se sincroniza el patrón de transición cargándolo en el latch de entrada usando la entrada de control C_1. El patrón de salida se carga en el latch de salida y se espera que se estabilice usando la entrada de control C_2. El periodo usado por C_2 para cargar la salida del circuito combinacional es el de un circuito sin falla. Por lo tanto, ante la presencia de una falla de retardo la salida final es distinta a la salida esperada.

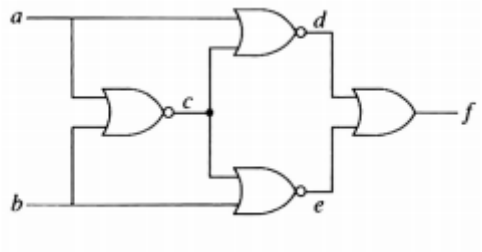
Los test para detectar fallas del tipo de retardo se dividen en dos grandes grupos: Los llamados test **Robustos** y los denominados **No robustos.** Por ejemplo, una falla no robusta es aquella que se puede detectar siempre y cuando no haya fallas en otros caminos.

Miramos la figura siguiente:



En este caso los vectores abc = 111 y abc = 101 sirven para detectar una falla de subida lenta en el punto **e** siempre y cuando que la ruta b-d-f no tenga una falla de retardo. Sin embargo, si hay una falla del tipo subida lenta en el punto **d**, la salida del circuito será correcta para el par de entradas dados, invalidando así la prueba para detectar una falla en **e**. Por lo tanto, la prueba (111 y 101) es **no robusta**.

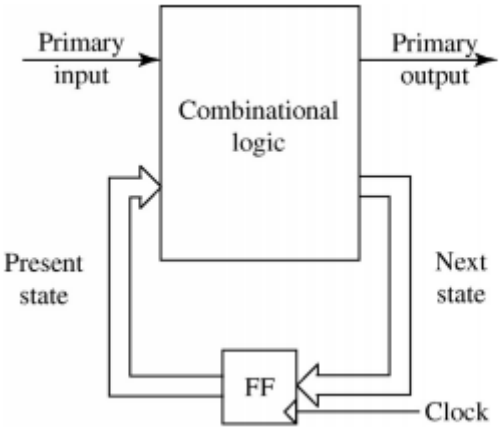
Un test de retardo de tiempo es considerado **robusto** si la detección de una falla en un punto es independiente de las fallas de retardo en otros caminos del circuito. Por ejemplo, miremos la siguiente figura:



Asumamos que hay una falla del tipo de bajada lenta en el punto **d** en el camino a-c-d-f. Los patrones de entrada 01 y 11 constituyen un **test robusto,** porque la salida de cualquier compuerta en cualquier otro camino no cambia cuando el segundo vector de la prueba se aplica al circuito. Por lo tanto, cualquier falla de retardo en otro camino no obstaculiza la detección de la falla considerada, No existen pruebas robustas para muchos caminos en grandes circuitos.

Pruebas en circuitos secuenciales.

La generación de test para circuitos secuenciales es extremadamente difícil ya que el comportamiento de este tipo de circuitos depende no solo de las entradas presentes si no también de los valores de entrada en el pasado. En la siguiente figura vemos un modelo general de un circuito secuencial sincrónico.



La parte combinacional del circuito recibe dos tipos de señales, las primarias que provienen del exterior y las secundarias que provienen del elemento de memoria (FF). La combinación de señales secundarias en un determinado tiempo recibe el nombre de **estado actual** del circuito, por lo tanto, las señales secundarias son llamadas **señales de estado**. Si el circuito posee **m** señales secundarias, puede poseer $\mathbf{2^n}$ posibles diferentes estados. La salida del bloque combinacional también tiene dos conjuntos de señales, Las salidas primarias son las que comandaran la acción hacia el exterior, las **salidas secundarias** se usan para especificar el **próximo estado** que asumirá el bloque de memoria. Se necesita una secuencia completa de entradas para poder detectar alguna de las posibles fallas de un circuito secuencial.

Los circuitos secuenciales se pueden probar verificando que los mismos funciones como lo indica su tabla de estadas. Este es un test exhaustivo, pero solo viable en pequeños circuitos secuenciales. El enfoque se puede resumir de la siguiente manera: **Encuentre un par de secuencia de entrada/salida (X,Z) tal que la respuesta del circuito a la entrada X sea la salida Z si y solo si el circuito está funcionando correctamente.** La aplicación de esa secuencia de entradas X y la verificación de la salida Z se llama **experimento de verificación,** y el par (X,Z) se denomina **secuencia de verificación.**

El origen de una secuencia de verificación para un circuito secuencial esta basada en asumir lo siguiente:

a) El circuito es **totalmente especificado o determinístico**. Este tipo de circuitos es en el cual el próximo estado está solo determinado por el estado actual y las entradas presentes.
b) El circuito está **fuertemente conectado**. Esto quiere decir que para cada par de estados $\mathbf{q_i}$ **y** $\mathbf{q_j}$**,** existe un para de entradas que hace evolucionar el circuito desde el estado $\mathbf{q_i}$ hacia el estado $\mathbf{q_j}$.

c) El circuito en presencia de fallas no tiene más estados que los enumerados en la especificación de diseño. En otras palabras, la presencia de una falla no incrementa el numero de estados del circuito.

Para diseñar experimentos de pruebas en este tipo de circuitos se deben conocer su **estado inicial** que está determinado por una **secuencia de referencia o distintiva.** Esto quiere decir que el circuito responde a esta secuencia de forma de saber unívocamente el estado final del mismo.

Por ejemplo, veamos la tabla de estados de la siguiente figura:

Present state	Input $x=0$	Input $x=1$
A	C,1	D,0
B	D,0	B,1
C	B,0	C,1
D	C,0	A,0

Tiene una secuencia de origen que es 101 como se indica en la siguiente tabla:

Initial state	Output sequence	Final state
A	0 0 1	C
B	1 0 0	A
C	1 0 1	B
D	0 1 1	C

Ya que todas las secuencias que se producen de aplicar la combinación 101 están asociadas con un solo estado final. Una secuencia inicial no necesariamente siempre tiene que referir al mismo estado final, solo es que el estado final pueda ser identificado a partir de una secuencia de salida.

Una secuencia distintiva es aquella que aplicada a la entrada de un circuito dará secuencias de salida distintas eligiendo distintos estados iniciales. Este tipo de secuencia también es una secuencia de referencia ya que conociendo el estado inicial y la secuencia de entrada se puede determinar unívocamente el estado final. El problema es que no todas las secuencias de búsqueda son distintivas. Por ejemplo, el circuito que responde a la siguiente tabla de estados tiene una secuencia de inicio igual a 010.

Present state	Input $x=0$	Input $x=1$
A	B,0	D,0
B	A,0	B,0
C	D,1	A,0
D	D,1	C,0

Present state	Response to 010	Final state
A	0 0 0	A
B	0 0 1	D
C	1 0 1	D
D	1 0 1	D

Como muestra la tabla de la derecha la salida en repuesta a la entrada 010 especifica unívocamente el estado final del mismo, pero no puede discernir el estado inicial entre C y D.

Cada circuito secuencial tiene una secuencia de referencia, pero solo un pequeño grupo tiene secuencias distintivas.

Al comienzo de un experimento, un circuito puede estar en cualquiera de los **n** estados del mismo. La incerteza inicial es el conjunto de todos los estados que pueden ser iniciales. Un subconjunto de estados que sabemos contiene el estado inicial se conoce con el nombre de **incerteza o incertidumbre.**

Por ejemplo, en la tabla inicial que vimos el circuito puede iniciarse en cualquiera de los cuatro estados que tiene. Por lo tanto, la incerteza inicial será (ABCD). Si aplica al circuito una entrada en 1, las siguientes incertezas son (AD) o (BC), dependiendo si la salida es 0 o 1 respectivamente. Las incertidumbres (C) y (DBC) son 0-sucesoras de (ABCD). Un **árbol sucesor**, que se define para un circuito específico y una incertidumbre inicial dada, es una estructura que muestra gráficamente las incertezas **x_i-sucesoras** para cada posible secuencia de entrada **x_i.**

Una colección de incertidumbres lleva el nombre de **vector de incertidumbres,** cada incerteza individual es una componente de ese vector. Si cada componente del vector contiene un solo estado, a este vector se lo llama **vector de incertidumbre trivial.** Un vector cuyos componentes refieren a estados individuales o estados repetidos idénticos se lo llama **vector de incertidumbre homogéneo.** En el ejemplo anterior, los vectores (AA), (B), (C) y (A), (B), (A), (C) son vectores homogéneos y triviales respectivamente.

Se obtiene una secuencia distintiva o de referencia a partir de un árbol de referencia, un árbol de este tipo es aquel que un nodo se convierte en terminal si ocurre alguna de las siguientes condiciones:

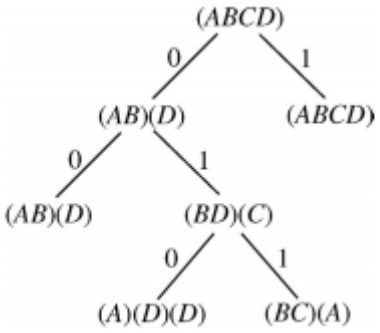
1) El nodo está asociado con un vector de incertidumbre, los componentes no homogéneos de este están asociados con el mismo nodo en un nivel anterior.
2) El nodo está asociado con un vector trivial o homogéneo.

El camino que conduce desde la incertidumbre inicial hasta el nodo en que vector es trivial o homogéneo define una **secuencia de inicio.**

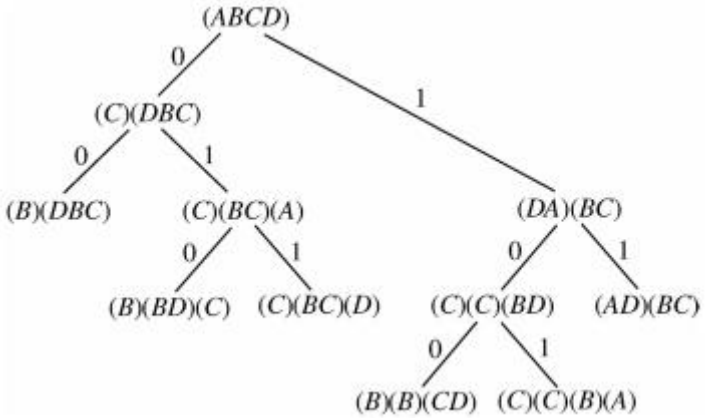
Un árbol es **distintivo** si es un árbol sucesor si un nodo se convierte en terminal si ocurre alguna de las siguientes condiciones:

1) El nodo está asociado con un vector de incertidumbre, los componentes no homogéneos de este están asociados con el mismo nodo en un nivel anterior.
2) El nodo está asociado con un vector de incertidumbre que contiene un componente trivial ni homogéneo.
3) El nodo está asociado con un vector trivial o homogéneo.

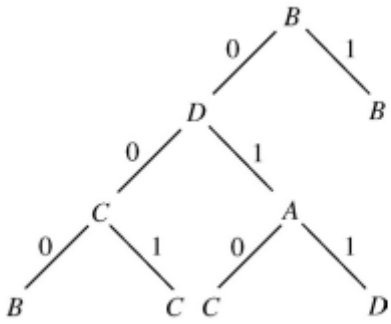
Como ejemplo obtenemos la secuencia de inicio 010 en la siguiente figura, aplicando las reglas vistas a la tabla de estados presentada anteriormente.



Si seguimos la secuencia 101 a la tabla de estados mencionada resulta en el siguiente gráfico:



Durante el proceso de los experimentos de verificación, a menudo es necesario llevar al circuito a un estado predeterminado, después que se haya aplicado la secuencia de inicio. Esto se hace con la ayuda de una **secuencia de transferencia**, que es la secuencia mas corta que lleva un circuito desde el estado S_i al estado S_j. El procedimiento es adaptativo pues la secuencia de transferencia está determinada por la respuesta a la secuencia de inicio. Como ejemplo tomemos la tabla de estados vista anteriormente, haremos una secuencia de transferencia desde el estado B al estado C. Para hacer esto se supone que el circuito está en el estado B. Se forma el árbol de transferencia siguiente:



Se puede ver que la secuencia de trasferencia estimativa que llevará del estado B al C es 00.

Diseño de experimentos de comprobación.

Básicamente el propósito de un experimento de verificación es verificar la tabla de estados del mismo describe con precisión su comportamiento. Si durante la ejecución del experimento, el circuito produce una respuesta que es diferente a la correspondiente con el funcionamiento correcto del mismo, el circuito se define como **defectuoso.**

Estos experimentos detectan si el circuito es defectuoso o no, no determina que componente es el que falla.

Para un circuito secuencial **fuertemente conectado** que tiene al menos **una secuencia distintiva,** el diseño de un test de prueba se puede dividir en tres partes:

1) **Fase de Inicialización.** Durante esta fase el circuito es llevado de un estado desconocido a un estado fijo. Un circuito reducido y fuertemente conectado se puede manipular en estados fijos por el siguiente método:
   a) Aplique una secuencia de inicio e identifique el estado actual del circuito.
   b) Si el estado actual no es **s**, aplique una secuencia de trasferencia para llevar el circuito al estado **s**.
2) **Fase de Identificación de Estados.** En esta etapa se aplica al circuito una secuencia de entrada para lograr que el mismo pase por todos los estados y ver la respuesta a la secuencia distintiva.
3) **Fase de verificación de la transición.** Durante esta fase se hace que el circuito pase por todos los estados, cada transición de estados se verifica usando la secuencia distintiva.

Aunque estas tres fases son distintas, muchas veces se solapan para lograr un Experimento y hacerlo más corto.

Generación de pruebas usando la estructura del circuito y la tabla de estados.

Este método se basa en la sensibilización de caminos vistos para circuitos combinacionales. Esta técnica toma en cuenta tanto la estructura del circuito como su tabla de estados. Se supone que el circuito bajo prueba tiene un estado de reinicio. Se aplica una secuencia de prueba tomando como estado de comienzo este estado de reinicio.

El método de generación de pruebas consta de tres pasos:

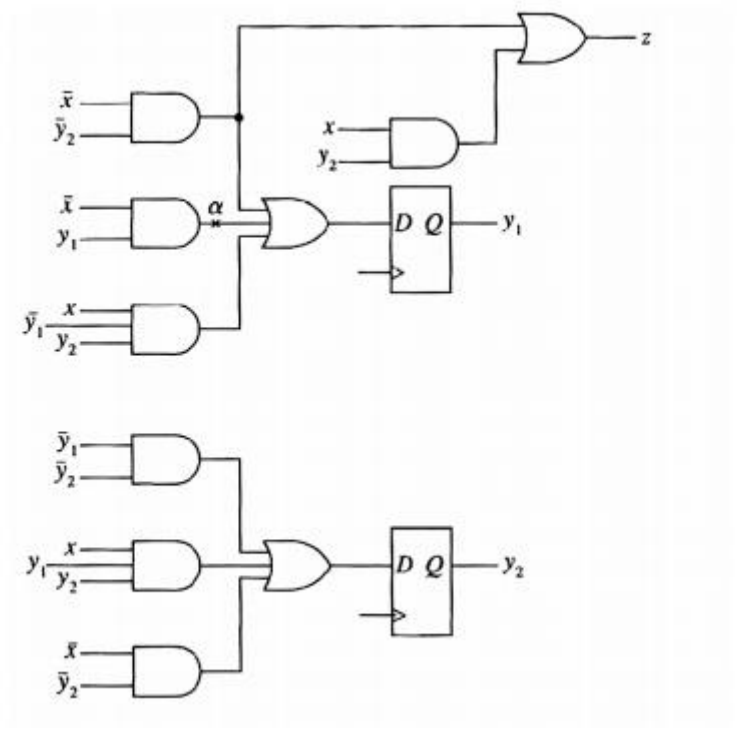
1) Genere un vector de prueba para una supuesta falla del tipo stuck at fault de modo que el efecto de la falla se propague a las salidas primarias o secundarias, es decir a las salidas de los flip flops. Cada salida primaria o secundaria se considera una salida independiente de un circuito combinacional. Un vector de prueba para una falla se le da el nombre de **vector de excitación**, y la parte del estado actual contenida en el vector de excitación se conoce como **estado de excitación.**
2) Aplicar una secuencia de entrada de forma tal de llevar el circuito de estado de reinicio al estado de excitación, a esta secuencia se la llama **secuencia de**

justificación. Obviamente esta secuencia no es necesaria si el estado de reinicio es parte del vector de excitación.

3) Aplicar una secuencia de entrada de tal forma que el ultimo bit de la secuencia produzca una salida diferente para el circuito libre de fallas y para el circuito defectuoso. Tal secuencia de entrada se llama **secuencia diferenciadora.**

Se obtiene una secuencia de prueba concatenando la secuencia de justificación, El vector de inicialización y la secuencia diferenciadora. Esta secuencia se simula con la presencia de una falla y se ve si esta es detectada. Si no se detecta falla, la secuencia no es válida.

Aplicaremos lo visto en un ejemplo para el siguiente circuito siguiente suponiendo una falla del tipo s-a-0 en el punto α:



La tabla de estados de este circuito libre de fallas es la siguiente:

	$x=0$	$x=1$
A	C,1	B,0
B	A,0	D,1
C	D,0	B,1
D	C,1	A,0

Los estados están codificados de la siguiente manera:

	y_1	y_2
A	0	0
B	0	1
C	1	1
D	1	0

Primero se aplica un vector de prueba parcial para la falla:

x	y_1	y_2
0	1	–

El valor de y_2 se elige de tal forma que la falla pueda se propagada a una salida primaria. Si $y_2 = 0$, la falla no se propaga a la salida ni afecta al siguiente estado. Por otro lado, si $y_2 = 1$, esto afecta el próximo estado de la variable y_1. Por lo tanto, el vector de excitación para la falla mencionada es:

x	y_1	y_2
0	1	1

Quedando así el estado de excitación como el 11.

A continuación, aplicamos la secuencia de justificación que puede sacar la maquina libre de fallas del estado de reinicio A al estado de excitación C. Se puede ver en la tabla de estados que la secuencia de justificación debe contener un solo bit, que es 0. Por lo tanto, la secuencia de prueba derivada hasta ahora es:

0 0

Y la secuencia de salida libre de fallas y el próximo estado es la siguiente:

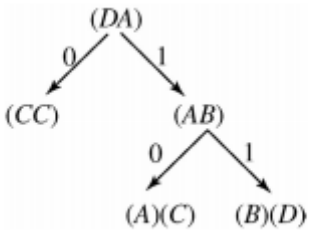
$$\begin{matrix} 0 & 0 \\ A \rightarrow & C \rightarrow D \\ 1 & 0 \end{matrix}$$

En presencia de la supuesta falla del tipo s-a-0 en el punto α, la secuencia vista anteriormente se transforma en:

$$\begin{matrix} 0 & 0 \\ A \rightarrow & C \rightarrow A \\ 1 & 0 \end{matrix}$$

Debido a que la secuencia de salida es la misma que en el caso libre de fallas, la falla **no es detectable**. Sin embargo, el estado final en presencia de la falla es A, en contraposición con el estado esperado libre de fallas D. El efecto de la falla solo se propaga a la salida de los flip flops. Por lo tanto, esta diferencia del estado de salida con falla y sin falla debe concatenarse con la secuencia de prueba derivada anteriormente.

La secuencia de diferenciación se obtiene de la siguiente manera:



Se puede usar 10 o 11 como secuencia de diferenciación. La elección de 11 da como resultado la siguiente secuencia de prueba para la falla supuesta:

0 0 1 1

Y la siguiente secuencia de estados para el circuito libre de fallas:

$$A \xrightarrow[1]{0} C \xrightarrow[0]{0} D \xrightarrow[0]{1} A \xrightarrow[0]{1} B$$

Y para el circuito con falla:

$$A \xrightarrow[1]{0} C \xrightarrow[0]{0} A \xrightarrow[0]{1} B \xrightarrow[1]{1} D$$

Por lo tanto, la falla es detectada por la secuencia de prueba aplicada.