25 TEST DE CIRCUITOS INTEGRADOS

El test es el procedimiento que se aplica a cada circuito integrado, al culminar el proceso de fabricación, para comprobar que dicho ejemplar (cada chip individual) se ha fabricado bien, es decir, para verificar que el resultado de la fabricación (en ese integrado particular) es correcto; el resultado del test es binario:

Si : el ejemplar es correcto, se encapsula y da lugar a un circuito integrado disponible para ser utilizado.

No : tal ejemplar ha resultado «mal fabricado» y se desecha.

En el caso de un circuito integrado muy pequeño, el test puede hacerse probando todas las situaciones del mismo: todos los vectores de entrada posibles, junto con todos los estados si el circuito es secuencial. Pero en circuitos de tipo medio o grandes este tipo de comprobación no es factible pues se necesitarían años para ella (un simple sumador de números de 26 dígitos precisa de más de un siglo, a una velocidad de 10^6 vectores/seg).

Y, sin embargo, es necesario comprobar que cada una de las celdas (cada puerta, cada biestable) se ha fabricado bien. Para ello se han desarrollado «modelos de test» que permiten identificar los fallos posibles, encontrar los vectores que detectan dichos fallos y conocer qué fallos detecta cada vector de test.

En el caso de circuitos secuenciales, la extraordinaria complejidad que implica controlar y observar el estado del circuito se resuelve añadiendo (en cada biestable) circuitería específica para el test; creando un «camino de exploración» (scan path) que permita establecer, directamente sobre los biestables, el estado que se desee y observar directamente el estado anterior.

Esta técnica de «scan path» ha resultado tan simple y útil que ha sido extendida al test de placas circuitales (test de conjuntos de circuitos integrados y de las pistas de interconexión de los mismos) y al test de sistemas digitales completos.

Asimismo, la incorporación de circuitería añadida para el test permite la realización del mismo en el interior del propio circuito integrado: test interno. Sencillos algoritmos de test de memorias (BIST), registros realimentados para generación aleatoria de vectores de test (LFSR), compactación en una sola palabra de los resultados del test (FIRMA) y aprovechamiento conjunto de LFSR y FIRMA para ejecución del test, son mecanismos de test interno que requieren añadidos circuitales relativamente pequeños. Ello simplifica el proceso de comprobación de la fabricación y, además, ofrece la posibilidad de verificar el integrado posteriormente, en el sistema digital del que forma parte.

Tomás Pollán Santamaría. Tecnología Electrónica. E.U.I.T.I.Z. Universidad de Zaragoza. tpollan@posta.unizar.es

25.1. El test como última etapa del proceso de fabricación

El test de un circuito integrado es la comprobación, posterior a la fabricación del mismo, que garantiza que el circuito se ha fabricado bien; tiene que garantizar que todas las celdas que configuran el circuito han sido integradas correctamente de forma que el resultado físico coincide con el esquema circuital resultante del diseño.

El test no puede plantearse desde la funcionalidad del circuito sino desde su configuración física; no es una simulación de su comportamiento respecto a las especificaciones que se pretenden, sino una verificación de su realidad física interna.

Téngase en cuenta que el resultado de la fabricación para cada circuito integrado individual (para cada uno de los muchos *chips* que se fabrican en una sola oblea) es incierto; basta una microscópica partícula de polvo, una irregularidad en la superficie de la red cristalina de la oblea, un fallo en el proceso litográfico de reproducción de las máscaras sobre la oblea, etc.,... para que un transistor no se construya y el circuito resultante sea erróneo.

El test se realiza sobre la propia oblea, previamente a la etapa de corte y separación de los numerosos *chips* que contiene, mediante costosas máquinas de test que activan los terminales (*PADs*) de entrada del circuito integrado, con un conjunto de vectores de entrada sucesivos (secuencia de vectores de test), y observan los terminales de salida, comparando los vectores de salida con la secuencia de resultados correctos. De forma que cada vector de test ha de incluir ambas partes: entradas y salidas:

Vector de test = { vector de entrada, vector de salida }

Los vectores de test han de permitir «controlar», desde las entradas del circuito integrado, cada uno de los «nudos booleanos» internos del circuito (para situarlos a valor **0** y a valor **1**) y «observar», desde las salidas del integrado, el estado booleano de cada uno de dichos nudos; de esta forma, podremos comprobar que cada celda básica está en su sitio y actúa correctamente.

El test se ejecuta posicionando automáticamente un conjunto de «puntas» o sondas de la máquina de test sobre los «puntos de soldadura» (*PADs*) terminales del *chip* que, posteriormente, se conectarán a los terminales (*pines*) del encapsulado. Se dispone para el test solamente de los terminales (entrada/salida) del circuito integrado, sin poder actuar sobre nudos internos.

El test se aplica «globalmente» sobre el circuito integrado completo; aun más, sobre el circuito «encapsulado», es decir, siendo accesibles solamente los terminales (*PADs*) del circuito: se dispone únicamente de las entradas físicas del circuito (*PADs* de entrada), para actuar sobre el mismo (vectores de entrada), y de sus salidas físicas (*PADs* de salida), para observarlo.

Además, es preciso minimizar el tiempo de utilización de la máquina de test, debido al elevado coste de tales equipos.

Por ello, el test presenta los siguientes inconvenientes:

1. Hay que comprobar todos y cada uno de los circuitos integrados: mientras que el proceso de fabricación opera sobre múltiples obleas a la vez y cada oblea contiene un amplio número de circuitos (es decir, centenares o miles de *chips* en cada «hornada»), el test ha de aplicarse individualmente a cada uno de ellos.
2. Un circuito integrado tiene muchos componentes (transistores) configurando una red con muchos nudos: todos ellos han de ser comprobados, pero el número de nudos accesible para el test es muy reducido, se dispone solamente de las entradas y salidas globales del circuito integrado.
3. La secuencia de vectores de test no puede ser excesivamente larga porque ello supondría un tiempo de máquina de test muy prolongado y repercutiría gravemente en el coste del circuito.

En suma, muchos circuitos integrados y muchos nudos (en cada uno de ellos) a comprobar y, en cambio, pocos nudos accesibles y relativamente pocos vectores de test a utilizar. Como respuesta a estos inconvenientes, hay que desarrollar metodologías de test y de diseño orientado al test que permitan superarlos.

Cuantifiquemos un poco la magnitud y complejidad del test:

¿Cómo estar seguros de que un circuito integrado se ha fabricado bien? Una primera respuesta podría ser: probando todas y cada una de sus posibilidades.

En el caso de un circuito combinacional, el número de posibilidades es sencillo: si cuenta con **m** entradas, las situaciones posibles son $\mathbf{2^m}$ (que conforman su «tabla de verdad»). Por ejemplo, un sumador de números de 16 bits cuenta con 32 entradas que suponen 2^{32} situaciones posibles, es decir, unos 4.300 millones de vectores de test que necesitan más de 1 hora, a una velocidad de un millón de vectores por segundo. Realmente se consiguen mayores velocidades de test, ~ 10^7 vectores/s, que suponen unos 7 minutos de test, que sigue siendo un tiempo alto para un simple sumador de 16 dígitos; un sumador de 32 bits, 2^{64} posibilidades, necesita cerca de 600 siglos de test (a 10^7 v/s).

En el caso de circuitos secuenciales, la magnitud del problema crece aún mucho más, por dos razones que se suman recíprocramente:

- los estados añaden posibilidades en progresión geométrica:

m entradas **n** biestables = $\mathbf{2^n}$ estados $\mathbf{2^m \times 2^n}$ posibilidades = $\mathbf{2^{m+n}}$

Por ejemplo: con 16 entradas y 32 biestables, 2^{48} posibilidades, unos 100 billones, que, a 10^7 vectores/s, suponen más de 4 meses dedicados al test.

- el orden de los vectores de entrada influye decisivamente:

Para conseguir cada estado en el conjunto de biestables, es necesaria una secuencia de vectores de entrada adecuada. Para verificar cada posibilidad no basta un solo vector de entrada sino que hay que conseguir llevar a los biestables al estado deseado, mediante una secuencia de vectores de entrada; esto no sólo aumenta el número de vectores de test, sino que requiere encontrar tal secuencia, lo cual no es obvio.

La magnitud del problema del test (puesta de manifiesto en los ejemplos anteriores, que, también, ponen en evidencia el mayor grado de complejidad que presentan los sistemas secuenciales), hace necesaria la utilización de *modelos de fallos* que permitan:

a) determinar con precisión los fallos posibles y contar su número,

b) encontrar vectores de test para cada fallo,

c) conocer qué fallos detecta cada vector de test,

y d) calcular la cobertura de fallos, es decir, el conjunto de fallos que pueden ser detectados por un conjunto dado de vectores de test.

En la práctica, se utiliza el modelo de fallos por bloqueo *stuck-at* cuya justificación y descripción se detallan en el apartado siguiente.

Ahora bien, en los razonamientos anteriores nos hemos referido solamente a uno de los aspectos del test de circuitos integrados digitales: el aspecto *funcional-booleano*: comprobar que cada celda básica está en su sitio (es decir, en sus salidas realiza la función booleana correspondiente) y adecuadamente conectada (ya que el resultado de dicha función se detecta en el nudo adecuado, tras haber actuado a través de los nudos que corresponden a sus entradas).

Existe un segundo aspecto que también debería ser objeto de test: el comportamiento *temporal*, referido a los tiempos de propagación (y de anticipación y mantenimiento), comprobando que tales tiempos coinciden con (o no son peores que) los estimados.

En la práctica, el test temporal no se realiza en forma individualizada sobre cada *chip* sino que se supone que si un transistor ha sido fabricado en su sitio sus características coinciden con las previstas.

Para tener una cierta garantía de ello suele realizarse un test global de fabricación: en la oblea se incluyen, aparte de los múltiples circuitos integrados iguales (*chips*), una serie de componentes electrónicos de tipo simple (resistencias, capacidades, uniones, PN, transistores individuales,...) cuyas características se miden (generalmente en unas pocas obleas de cada «hornada») para comprobar que el proceso de fabricación ha sido correcto, es decir, ha construido componentes de las características eléctricas previstas. Es pues un «test de calidad» del proceso de fabricación, análogo al que se realiza en otros tipos de procesos de fabricación, tomando varias muestras al azar y comprobando en detalle sus características.

25.2. Modelo de fallos para el test

Definir los fallos posibles en un circuito integrado es tarea ardua y compleja; las posibilidades son enormes:

- en principio los circuitos integrados digitales en tecnologías MOS están compuestos por transistores MOS conectados entre sí;
- en cuanto a los transistores puede haber fallos por quedar cortocircuitadas dos de sus cuatro regiones constitutivas: fuente, drenaje, puerta y substrato, o por quedar en circuito abierto (o no haberse formado, lo cual es lo mismo) la fuente, el drenaje o la puerta;
- respecto a las conexiones puede darse el caso de pistas rotas, pistas interconectadas entre sí (puenteadas), etc;
- ...

Para poder manejar tan gran número de posibilidades es precisa una fuerte simplificación, se requiere un modelo simplificado de fallos.

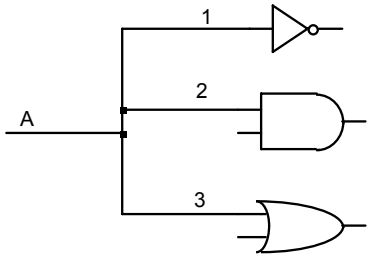
Un modelo de fallos muy simple pero que ha dado excelentes resultados y es el que se utiliza actualmente en la fabricación industrial es el modelo de «fallos por bloqueo», el cual admite solamente dos posibilidades de fallo: *stuck-at-0* (bloqueo a **0**) y *stuck-at-1* (bloqueo a **1**) y supone las siguientes hipótesis:

1. Se consideran como elementos componentes del circuito las celdas básicas («celdas estándar» con las que se efectúa el diseño: puertas, biestables,...) y se utiliza la *net-list* (lista de conexiones del circuito) configurada por dichas celdas básicas.

2. Se consideran como «nudos» todas y cada una de las entradas a las celdas básicas y, además, las salidas del circuito.

Por ello, estos «nudos a efectos de test» no coinciden exactamente con los «nudos topológicos» del circuito, sino que cada nudo eléctrico ha de ser «desplegado» en varios, tantos como entradas a las que está conectado. De forma que el número de nudos a efectos de test es muy superior al número de nudos de la *net-list* SPICE.

Por ejemplo, el nudo A de la figura siguiente ha de ser contabilizado como tres nudos (1, 2 y 3), ya que conecta con 3 entradas.



3. Para cada nudo se consideran como fallos posibles solamente dos:

- que el nudo esté bloqueado a **0** *STUCK-AT-0*
- que el nudo esté bloqueado a **1** *STUCK-AT-1*

(*"stuck" es el participio del verbo "to stick": clavar, pegar, adherir ...*).

4. Cada nudo ha de ser comprobado dos veces, para asegurar que no se encuentra bloqueado a **0** ni a **1**.

5. El no-bloqueo (la puesta a **1** y a **0**) de cada entrada (nudo interior) ha de ser detectado pasando a través de la celda básica a la que corresponde dicha entrada, es decir, para cada celda básica ha de verificarse el paso de un **1** y el de un **0** desde cada una de sus entradas; además, ha de verificarse la posibilidad de **1** y de **0** para cada salida del circuito.

6. Se supone que en el circuito no se presentan múltiples fallos que se compensen entre sí: *hipótesis de fallo único en el circuito*; en realidad lo que se supone es que si un vector de test es capaz de detectar un fallo individual, dicho vector detecta siempre dicho fallo, es decir, no se da el caso de que el fallo quede enmascarado (compensado) por otros fallos que tengan lugar a la vez.

Las simplificaciones que introduce este modelo (en particular, las hipótesis de «solo dos tipos de fallos» y de «fallo único en el circuito») son muy fuertes y es obvio que no asegura, en principio, la detección de todas las posibilidades de fallo. Sin embargo, el modelo se ha mostrado muy eficaz en su utilización práctica y es, hoy por hoy, el único que se emplea a nivel industrial; ha demostrado que detecta un porcentaje muy alto (aunque no todos) de los fallos posibles y se ha comprobado que, efectivamente, sirve para asegurar que un circuito integrado ha sido fabricado bien.

Habida cuenta que cada nudo se comprueba dos veces (bloqueos a **0** y a **1**), que se consideran como nudos independientes cada una de las entradas a las celdas básicas y por cada una de dichas celdas se hace pasar un **0** y un **1** desde cada una de sus entradas (detectando específicamente dicho paso),… es sumamente probable que un test completo conforme al modelo *stuck-at* detecte cualquier fallo resultante del proceso de fabricación.

El modelo *stuck-at* es capaz de detectar los siguientes fallos:

- un cortocircuito entre dos terminales de un transistor MOS generalmente equivale a un *stuck-at;*
- un transistor MOS en circuito abierto en muchas ocasiones será detectado como *stuck-at*; en este caso, el orden de los vectores de test influye en la detección del fallo pues generalmente la salida de la correspondiente puerta lógica depende del anterior valor de salida (que se conserva capacitivamente) [Se ha desarrollado un modelo de fallos *stuck-open* para tratar los correspondientes a circuitos abiertos, basado en detectar las transiciones (de **0** a **1** y de **1** a **0**) pero requiere un número de vectores mucho mayor que el *stuck-at* porque es preciso detectar cada nudo en valor **0** y de detectar su paso a **1** y viceversa.];
- lo mismo podría decirse de pistas cortadas;

- la interconexión entre pistas (*bridging*: puentes) no es detectada directamente, pero habida cuenta que ha de verificarse el no-bloqueo a **0** y a **1** de cada una de ellas (al menos 4 pruebas) y que dicha verificación se hace varias veces en el caso de estar unidas a más de una entrada es muy probable que tal fallo se detecte (existen también modelos de fallos por puentes);
- algo parecido puede decirse de los fallos múltiples: en principio se encuentran excluidos del modelo pero es sumamente probable que alguno de ellos sea detectado, en alguna de las múltiples veces que los vectores de test observan los «nudos» afectados por tales fallos (téngase en cuenta que carece de interés conocer si hay más de un fallo: basta uno de ellos para que el circuito sea erróneo y deba ser desechado).

La detección de un fallo presenta dos aspectos complementarios:

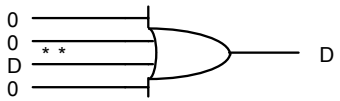
a) el CONTROL del correspondiente nudo desde las entradas del circuito integrado para situarlo en el valor booleano adecuado (el que demuestra que el nudo no está bloqueado, es decir, el contrario al bloqueo que se investiga)

b) la OBSERVACIÓN del valor booleano presente en dicho nudo desde las salidas del circuito, pasando a través de la celda básica a la que pertenece el nudo como entrada de la misma.

En la observación hay que pasar a través de las celdas básicas «hacia adelante» hasta llegar a las salidas; en el control hay que pasar también a través de las celdas básicas pero «hacia atrás» (de las salidas hacia las entradas) hasta llegar a las entradas del circuito.

Para realizar este paso «a través» de las celdas básicas se utiliza un «modelo de fallos» de cada celda que permite razonar cuál debe ser el valor booleano de todas sus entradas para atravesar la celda y, también, qué bloqueos detecta en cada caso:

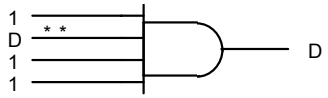
- denotemos con **D** el valor booleano que debe atravesar la celda (**D** será el valor presente en el nudo que estamos considerando: **D = 1** para investigar bloqueos a **0** y **D = 0** para tratar los bloqueos a **1**);
- una puerta "**o**", para que pase **D** desde una de sus entradas, ha de tener el resto de ellas a **0** y la salida será **D**; cuando **D** sea **0**, la salida **y = 0** detecta «no-bloqueo a **1**» en todas las entradas, cuando **D** sea **1**, la salida **y = 1** detecta solamente «no-bloqueo a **0**» en dicha entrada **D**:



D = 0 **y = 0** $\Rightarrow$ no-bloqueo a 1 de ninguna entrada

D = 1 **y = 1** $\Rightarrow$ no-bloqueo a 0 de la entrada **.

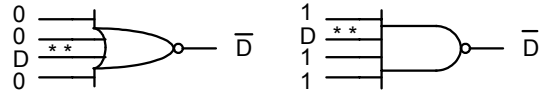
- una puerta "**y**", para dejar pasar **D** desde una entrada, requiere que el resto se encuentre a **1** y la salida será **D**; cuando **D** sea **0**, **y = 0** detecta solamente «no-bloqueo a **1**» de dicha entrada, pero cuando **D** sea **1**, **y = 1** detecta «no-bloqueo a **0**» de todas las entradas:



D = 0 **y = 0** $\Rightarrow$ no-bloqueo a 1 de la entrada **

D = 1 **y = 1** $\Rightarrow$ no-bloqueo a 0 de ninguna entrada.

- el modelo para puertas "**o-negada**" *Nor* e "**y-negada**" *Nand* es el siguiente:



- la puerta "**o-exclusiva**" presenta dos posibilidades:

25.3. Análisis de fallos y simulación de fallos

25.3.1. Generación de vectores de test: análisis de fallos

Para detectar un fallo es preciso encontrar un vector de test capaz de:

a) *controlar* el nudo correspondiente, es decir, actuar sobre dicho nudo desde las entradas de forma que adquiera el valor booleano deseado;

b) *observar* dicho nudo, es decir, establecer a partir del mismo un conjunto de valores sobre los nudos que interaccionan posteriormente con él, de forma que el valor booleano de una salida sea diferente en el circuito con fallo en relación con el correcto.

La parte **a** («control»), se consigue llevando «hacia atrás» los valores booleanos desde el nudo considerado hacia las entradas, lo cual, en principio, siempre es posible (en un circuito combinacional). La parte **b** («observación») trata de llevar «hacia adelante» el valor booleano del nudo hacia las salidas, imponiendo en el resto de los nudos que influyen sobre este camino «hacia adelante» el valor booleano que permita distinguir el valor que está pasando; ello no siempre es posible pues puede haber problemas de compatibilidad entre los valores booleanos de unos nudos y otros, habida cuenta de que todos ellos han de ser impuestos desde las entradas del circuito.

La parte **b** («observación») ha de aplicarse antes que la **a** ya que determina valores booleanos en otros nudos internos del circuito que, posteriormente (parte **a**: «control»), han de ser llevados (todos ellos, además del correspondiente al nudo objeto de investigación) hacia las entradas.

Se demuestra que, en el modelo de fallos por bloqueo STUCK-AT, si el circuito es combinacional y no contiene redundancias todos los fallos son «testeables»: para cada fallo existe un vector de test que, actuando desde la entradas del circuito, diferencia el comportamiento del circuito con y sin fallo, produciendo salidas distintas en ambos casos.

El empleo de la notación **D** (**D = 1** para investigar bloqueos a **0**, **D = 0** para los bloqueos a **1**) permite encontrar, a la vez, los vectores de test para ambos bloqueos de un nudo dado. El proceso de obtención de los vectores de test para un determinado nudo se realiza mediante el siguiente esquema procedimental, denominado ***D-algoritmo***:

1. se asigna el valor **D** al nudo a «testear»;

2. a partir de dicho nudo y pasando por la correspondiente celda básica se lleva **D** hacia una salida asignando valores booleanos adecuados a los nudos de las puertas por las que se pasa;

3. el valor **D** y el resto de los valores booleanos asignados se llevan hacia las entradas.

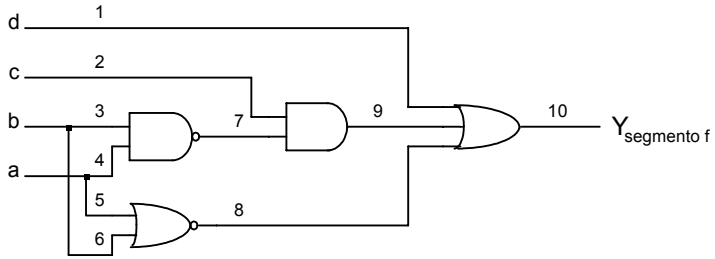
Para ejecutar los pasos 2 y 3 (consistentes en atravesar «hacia adelante» y «hacia atrás», respectivamente, las celdas básicas) se utilizan los «modelos de fallos» de las correspondientes celdas. En todo caso, es preciso tener bien presente la «hipótesis de fallo único en el circuito»: si en el circuito existe un fallo, éste es único (se desecha la posibilidad de que se produzca un segundo fallo que compense el primero).

El *D-algoritmo* resulta fácilmente programable (de hecho, existen varios algoritmos equivalentes desarrollados en detalle para su programación), lo cual permite la generación automática de vectores de test (**ATPG**).

Ejemplo de aplicación manual del D-algoritmo

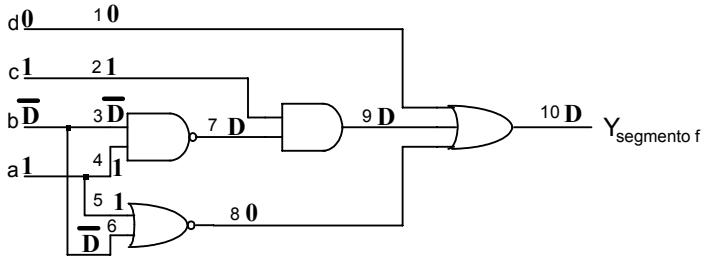
Consideremos el caso de una simple función booleana, la correspondiente a la activación del segmento **f** de un conversor BCD a 7 segmentos, expresada en la siguiente forma algebraica:

$$Y_{segmento\ f} = d + c.\overline{b} + c.\overline{a} + \overline{b}.\overline{a} = d + c.(b * a) + (b \Delta a)$$



«A efectos de test» el circuito presenta 10 nudos, tal como han quedado numerados en la figura; por tanto, el número de fallos posibles es de 20: 10 *bloqueos a* ***0*** y 10 *bloqueos a* ***1***.

Efectuemos el análisis de fallos del circuito anterior, respecto al nudo 7:



1 asignamos **D** a 7;

2 observación: nudo 2 a **1**; nudo 9 a **D**, nudos 1 y 8 a **0**, 10 a **D**;

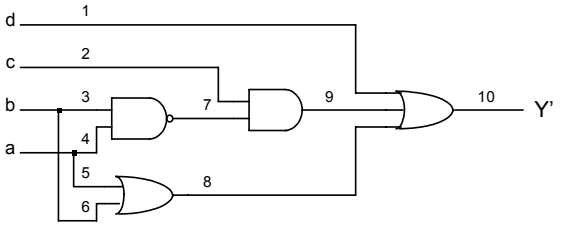
3 control: nudo 1 a **0** (por tanto, entrada D = **0**); nudo 2 a **1** (entrada C = **1**); nudo 7 a **D** y, para ello, nudo 3 a $\overline{D}$ y nudo 4 a **1** (entradas B = $\overline{D}$ y A = **1**); nudo 8 a **0**, lo cual es correcto ya que «entrada A» = nudo 6 = **1**.

vectores de test: **01**$\overline{D}$**1** **0101** $\Rightarrow$ **D = 1** : detecta *bloqueo a* ***0*** en el nudo 7

0111 $\Rightarrow$ **D = 0** : detecta *bloqueo a* ***1*** en dicho nudo.

Cuando en el circuito hay redundancia, o sea, cuando es posible simplificar la función y eliminar algunas de las puertas que configuran el circuito los nudos afectados por la redundancia no son «testeables», es decir, no es posible encontrar un vector de test que diferencie el circuito con y sin fallo. Lo cual es consecuencia de que el valor booleano de tales nudos no «determina» por sí mismo el valor de las salidas, al existir caminos lógicos redundantes.

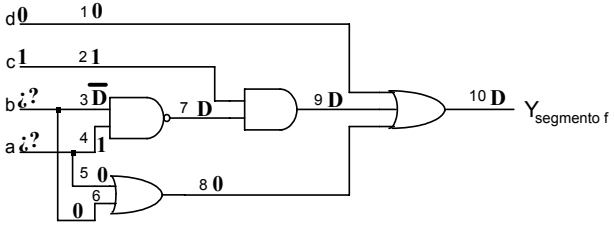
Por ejemplo, si en el circuito anterior cambiamos la puerta "**o-negada**" por una "**o**":



1 nudo 7 a **D**;

2 observación: nudo 2 a **1**; nudo 9 a **D**, nudos 1 y 8 a **0**, nudo 10 a **D**;

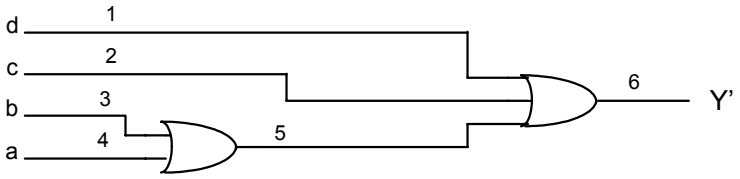
3 control: para que el nudo 8 se sitúe a **0** se requiere que los nudos 5 y 6 estén a **0**; en cambio para que el nudo 7 se sitúe a **D** es preciso o bien que el nudo 3 esté a $\overline{D}$ y el 4 a **1** o sino que el nudo 3 esté a **1** y el nudo 4 a $\overline{D}$ o que ambos nudos 3 y 4 estén a $\overline{D}$;



Como los nudos 3 y 6 han de tener el mismo valor booleano y lo mismo sucede con los nudos 4 y 5, para **D = 0** tales condiciones (nudos 3 y 4 a **1** y nudos 5 y 6 a **0**) son incompatibles entre sí; para **D = 0**, si existe vector de test, ya que corresponde a los 4 nudos (3, 4, 5 y 6) a **0**:

- el vector **0100** detecta *el bloqueo a **0*** del nudo 7;
- pero, en cambio, ningún vector es capaz de detectar el *bloqueo a **1*** del nudo 7.

Ello es debido a que la función contiene redundancia, como se demuestra al simplificarla: $Y' = d + c.(b * a) + (b + a) = d + c + b + a$.



Las puertas "**y-negada**" e "**y**" son redundantes y pueden eliminarse.

Por tanto, en relación con el test, interesa evitar toda redundancia circuital: en ocasiones se ha recomendado introducir redundancia en el diseño de circuitos combinacionales para eliminar «espurios» (*glitches*) debidos a la conmutación de las puertas (a sus tiempos de propagación); esto resulta nefasto para el test: los *glitches* deben ser evitados a través de un diseño adecuadamente síncrono.

25.3.2. Cobertura de test: simulación de fallos

No es preciso utilizar un vector de test para cada uno de los fallos posibles, pues un mismo vector de test sirve para detectar varios fallos a la vez.

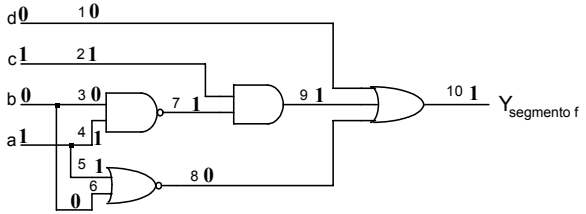
Por eso, una vez obtenido un vector de test conviene averiguar qué fallos es capaz de detectar. Esta operación se denomina *simulación de fallos* y se realiza yendo desde las salidas del circuito hacia las entradas y deduciendo, para cada celda básica a partir de su «modelo de fallos», los fallos que son detectables en las entradas de dicha celda.

Por ejemplo, recordando el modelo de fallos de las puertas "**o**" e "**y**":

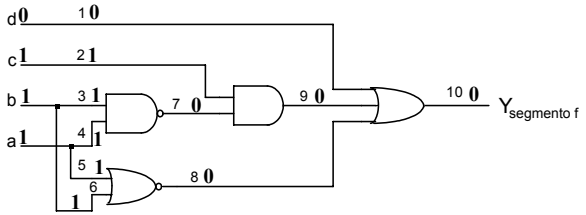
- una puerta "**o**" con salida **0** detecta cualquier *bloqueo a **1*** en cualquiera de sus entradas; dicha puerta "**o**" con salida **1** solamente es capaz de detectar el *bloqueo a **0*** de una de sus entradas y ello con la condición de que el resto de ellas esté a **0**;
- una puerta "**y**" con salida **1** detecta *bloqueo a **0*** en cualquier entrada y con salida **0** sólo es capaz de detectar el *bloqueo a **1*** de una de sus entradas y con la condición de que el resto de ellas se encuentre a **1**;

En el ejemplo anterior (segmento f de un conversor BCD a 7 segmentos):

- el vector de entrada **0101** es capaz de detectar *bloqueo a **0*** en los nudos 10, 9, 2, 7 y *bloqueo a **1*** en el nudo 3;



- el vector de entrada **0111** es capaz de detectar *bloqueo a **1*** en los nudos 10, 1, 9, 8, 7 y *bloqueo a **0*** en los nudos 3 y 4.



Bastan los dos vectores obtenidos para testear el nudo 7 (**01$\overline{D}$1**) para conseguir una cobertura de test del 60 % (12 fallos/20 posibles).

Siempre sucede que los primeros vectores de test detectan muchos fallos pero, a medida que se añaden nuevos vectores para aumentar la cobertura de test, éstos detectan muy pocos fallos nuevos.

En este ejemplo (segmento f), para una cobertura de fallos completa (100 %) son necesarios los siguientes vectores (en la tabla que sigue se indican, para cada vector de entrada, los fallos que son detectados por dicho vector; se señalan con # los «nuevos» fallos y con ∗ los fallos que ya han sido detectados por vectores anteriores):

	*bloqueos a **0***										*bloqueos a **1***										
nudos :	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
vector **0101**		#					#		#	#			#								25 %
vector **0111**			#	#							#						#	#	#	#	60 %
vector **0000**								#		*					#	#					75 %
vector **0001**					#						*	#						*	*	*	85 %
vector **0010**						#					*	*						*	*	*	90 %
vector **0110**		*					*		*	*				#							95 %
vector **1010**	#									*											100 %

en total se necesitan 7 vectores de test para un test completo del circuito.

Éste es el procedimiento común para generar los vectores de test:

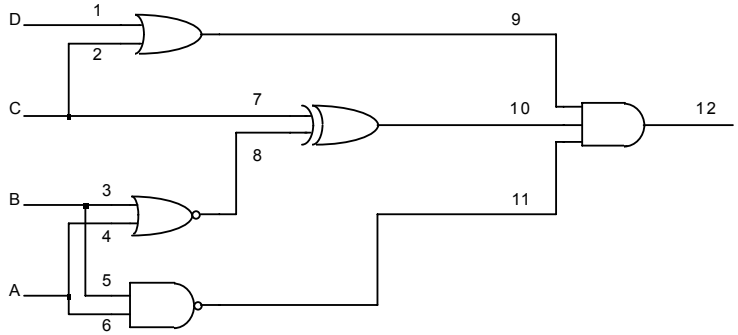
- se escoge un *subconjunto inicial de vectores de test*: tal subconjunto puede ser el de los vectores correspondientes a una simulación funcional del circuito (conjunto de vectores que habremos utilizado en el proceso de diseño para verificar que el circuito diseñado es correcto) o bien, sencillamente, un conjunto de vectores generados al azar (método que suele dar muy buenos resultados para el test);
- mediante *simulación de fallos* se averigua qué fallos son detectados por dichos vectores y cuáles no;
- mediante *análisis de fallos* se obtienen nuevos vectores para fallos aún no cubiertos;
- se reitera la *simulación de fallos* y se repite el proceso hasta que se consigue un grado de cobertura adecuado (a ser posible el 100 %).

En los ASICs digitales, los vectores de test son la referencia «legal» entre fabricante y cliente: el fabricante se compromete a que cada uno de los circuitos integrados que suministra verifica los vectores de test (la secuencia de vectores de test) aceptados por ambos (son objeto de firma específica en el contrato) y el cliente no puede exigirle legalmente otra cosa sino el cumplimiento de los vectores de test.

Los fabricantes de ASICs suelen exigir, como condición previa, una cobertura de test cercana al 100 % (en muchos casos, cobertura total), a fin de poder comprobar con adecuadas garantías los circuitos integrados resultantes de la fabricación.

Además, el fabricante señala un límite en el número de vectores test, a fin de que el tiempo de utilización de la máquina de test para cada *chip* no sea muy alto (y, en consecuencia, costoso). Generalmente el número máximo de vectores de test suele ser del orden de 64K o de 256K; el cliente puede solicitar la utilización de mayor número de vectores de test, pero ello repercutirá en el precio por unidad del circuito integrado.

Otro ejemplo de búsqueda de vectores de test



El circuito tiene 12 nudos «a efectos de test»: son posibles 24 fallos: 12 *bloqueos a **0*** y otros 12 *bloqueos a **1***.

1. Comenzamos por el nudo 1 y buscamos su vector de test

vector de test para el nudo 1: `D000`

a) el vector **`0000`** detecta *bloqueo a **1*** en los nudos: **1, 2, 9 y 12**

b) el vector **`1000`** detecta *bloqueo a **0*** en los nudos: **1, 8, 9, 10, 11 y 12**
y detecta *bloqueo a **1*** en los nudos: **3, 4 y 7**

La **cobertura de test** conseguida con ambos vectores superior al **50 %**: 13 / 24

2. Tomando un segundo nudo, no cubierto por los anteriores vectores, por ejemplo el nudo 5, buscamos su vector de test

vector de test para el nudo 5: `01D1`

a) el vector **`0101`** detecta *bloqueo a **0*** en los nudos: **2, 4 y 7**
y detecta *bloqueo a **1*** en los nudos: **5 y 8**

b) el vector **`0111`** detecta *bloqueo a **0*** en los nudos: **5 y 6**
y detecta *bloqueo a **1*** en los nudos: **11**

La **cobertura de test** conseguida es cercana al **90 %**: 21 / 24. Solamente quedan por detectar el *bloqueo a **0*** del nudo 3 y los *bloqueos a **1*** de los nudos 6 y 25.

3. Buscamos el vector de test que detecta el *bloqueo a **0*** del nudo 3: **`1010`**

dicho vector **`1010`** detecta *bloqueo a **0*** en el nudo: **3**
y detecta *bloqueo a **1*** en el nudo: **10**

Con ello, la **cobertura de test** alcanzada es superior al **95 %**: 23 / 24

4. Buscamos el vector de test que detecta el *bloqueo a **1*** del nudo 6: **`1110`**

dicho vector **`1110`** completa el **100 %** de **cobertura de test** del circuito.

Los vectores utilizados en la simulación funcional, a lo largo del proceso de diseño, suelen proporcionar una cobertura de test entre el 60 % y el 80 %, según los casos; además, gran parte de ellos son redundantes respecto al test, no aumentan la cobertura que ya realizan los demás: el problema del test no se reduce al de la simulación funcional.

A lo largo de este apartado (y también en el anterior), al hablar de vectores de test puede parecer que solamente se hace referencia a «vectores de entrada» para el test; debe tenerse siempre presente que un vector de test ha de incluir ambas partes: *{vector de entrada, vector de salida}*; no sólo es preciso activar adecuadamente las entradas del circuito, sino, también, comparar los valores de las salidas con los que corresponden a un circuito correcto (bien fabricado).

25.3.3. Diseño orientado al test

Aunque la aplicación efectiva de los vectores de test la realiza el fabricante (la ejecuta la máquina de test sobre cada uno de los *chips* obtenidos en el proceso de integración), la obtención de los vectores de test es responsabilidad del diseñador y debe constituir un punto de referencia de todo el proceso de diseño: es necesario diseñar para el test.

El diseño condiciona el test y, sobre todo, puede facilitarlo en gran manera. No basta un diseño funcional sino que hay que efectuar, a la vez, un diseño para el test y, en la práctica, los aspectos relativos al test pueden resultar más difíciles y complejos que el propio diseño funcional. Ya hemos comentado que la redundancia da lugar a nudos no testeables: ha de evitarse toda redundancia, lo cual queda asegurado si los procesos de simplificación se realizan bien.

Por otra parte, comprender bien la dinámica del test permite definir estrategias que facilitan el test: el test se basa en *controlar* desde las entradas y en *observar* desde las salidas los nudos internos del circuito. Por ello interesa aumentar la observabilidad y la controlabilidad del circuito, facilitar el control y la observación de los nudos del circuito:

- el aumento del número de salidas: conectar nudos internos del circuito a salidas adicionales del mismo; lo cual se puede conseguir físicamente, aumentando en igual medida el número de terminales (*pines*), o por medio de multiplexado, utilizando los mismos terminales para varios nudos;
- el control de entradas intermedias: hacer que nudos internos adquieran el valor booleano directamente; lo cual requiere distinguir la situación de test de la correspondiente al funcionamiento normal, a través de un multiplexado apropiado;
- la división del circuito en bloques separables: configurar particiones, de forma que cada bloque sea controlable y observable desde el exterior, bien por multiplexado, bien mediante un bus interno, etc.,...;
- en general, cualquier medida que permita «acercar» nudos internos a las entradas (para controlarlos) o «aproximarlos» a las salidas (para observarlos) facilitará en gran manera el test (la obtención de vectores de test y la reducción del número de ellos necesario para una cobertura total).

25.4. Test de circuitos secuenciales

Los circuitos combinacionales presentan, en relación con el test, la ventaja de que el orden de los vectores no afecta a la comprobación de los fallos; en cambio, en los circuitos secuenciales el orden de los vectores de test es transcendental: los valores booleanos de los nudos del circuito no dependen solamente del vector de entrada, sino también del estado (interno) del circuito, es decir, del conjunto de valores almacenados en los biestables contenidos en él.

En un circuito secuencial, para imponer un determinado valor en un nudo, no basta con un vector de entrada, sino que es necesaria una secuencia adecuada de tales vectores. Ello aumenta, en gran medida, el número de vectores de entrada necesarios para el test y, lo que es mucho peor, aumenta extraordinariamente la complejidad de la obtención de dichos vectores de test; no hay una solución general que permita obtener la *secuencia de vectores de test* necesaria para detectar el bloqueo de un determinado nudo.

Por ejemplo, en un contador compuesto por 20 biestables, para analizar el bloqueo a **0** del más significativo de ellos es preciso realizar el contaje de 2^{19} unidades, es decir, más de medio millón de ciclos de reloj (el doble en cuanto a vectores de entrada) para llegar a establecer un **1** en dicho biestable.

Imaginemos la complejidad que presenta un circuito de control con varios contadores, diversos registros y algunos grafos de estado entrelazados mutuamente: intentar imponer valores booleanos determinados sobre un conjunto de nudos internos que dependan de tales bloques (y, además, hacerlo desde las entradas del circuito que, en principio, no conectarán directamente con ellos) es, sin duda, una labor ardua y, en muchos casos, inabordable.

En los circuitos secuenciales el problema del test no puede ser abordado a posteriori, sino que ha de ser un condicionante básico del diseño, desde el inicio del mismo: es indispensable un diseño orientado al test y desde tal perspectiva (diseñar para el test) no resulta difícil llevar a cabo un diseño bien estructurado y obtener la correspondiente secuencia de vectores de test para el mismo.

En general, resulta sumamente conveniente:

1 que el circuito sea completamente síncrono y esté bien diseñado como tal (y, como es obvio, con adecuada distribución del reloj que asegure la simultaneidad del mismo, y con ausencia de violaciones de los tiempos de *set-up* y de *hold*);

2 que el estado inicial de los biestables sea conocido, es decir, que exista un procedimiento directo de inicialización (*reset*) que permita llevar a los biestables a un estado determinado (**0** ó **1**).

La condición de «sincronismo» suele ser una exigencia de partida en el diseño de ASICs con celdas estándar pero, además, es una situación deseable desde el punto de vista de la seguridad funcional. El sincronismo resulta sumamente conveniente, tanto desde el punto de vista del test, como desde la propia seguridad funcional del circuito: un buen diseño secuencial es, hoy día, un diseño síncrono.

La posibilidad de inicialización viene exigida no sólo por el test sino también por la propia simulación funcional: no es simulable un circuito cuyo «estado inicial» no sea conocido; todo proceso de simulación comenzará efectivamente a partir de un estado inicial, que se establece (se impone) previamente sobre los biestables. Por ello, todo diseño secuencial debe contar con su entrada de inicialización (*reset*) que actúe, directa o indirectamente, sobre todos los biestables forzándoles a adoptar un estado conocido (por ejemplo, su borrado: puesta a **0**).

25.4.1. Reducir el test secuencial al combinacional: SCAN PATH

Supuesto un circuito síncrono, la idea es poder *controlar* y *observar* cada uno de sus biestables, de forma que el test se reduzca al caso combinacional; es decir, manejar «desde fuera» el vector de estado del circuito (el estado de todos los biestables del mismo), convirtiéndolo en un vector de entrada (*control* directo sobre el estado anterior) y en un vector de salida (*observación* del estado siguiente), de manera que el circuito «deja de ser secuencial»: si el estado anterior es controlable como entrada y el estado siguiente es observable como salida, la función de memoria que da carácter a los circuitos secuenciales desaparece.

Supongamos que podemos fijar el valor booleano en la salida de cada biestable y, después de un solo pulso de reloj, podemos observar las salidas de todos ellos, el test a formular se referirá únicamente a la parte combinacional del circuito. De esta forma y en lo que se refiere al test, se prescinde de los biestables como tales (como elementos de memoria interna) y se produce una «duplicación» de ellos: por un lado, se convierten en «entradas» del circuito (sobre ellos se fija, desde fuera, el estado anterior) y por otro actúan como «salidas» (se observa el nuevo estado tras el pulso de reloj).

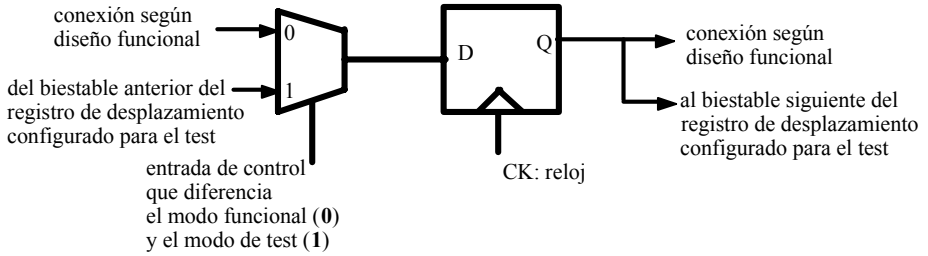
¿Cómo puede controlarse y observarse, en forma directa, desde el exterior del circuito integrado, el estado de todos los biestables del mismo? Una forma relativamente simple de hacerlo es conectando, a efectos de test, todos los biestables en largos registros de desplazamiento con entrada y salida en terminales específicos del circuito integrado.

Haciendo que, a los efectos de control y de observación para el test, los biestables se encuentren, todos ellos, conectados en serie, formando un registro de desplazamiento, con una entrada (serie) exterior y una salida externa (la del último biestable). Para dar valores a los biestables bastará almacenarlos sucesivamente, a través de la entrada, utilizando **n** pulsos de reloj (tantos como biestables contengan el registro de desplazamiento); pero, a la vez, durante tales **n** pulsos de reloj, la salida del registro presentará sucesivamente los valores contenidos en los biestables. De forma que, la *observación del estado* de los biestables (adquirido tras el pulso de reloj) y la *fijación de un nuevo estado anterior* en ellos se realiza simultáneamente.

Esta forma de organizar el test de los circuitos secuenciales, mediante el control y la observación «en serie» de todos sus biestables, recibe el nombre de *scan path* (camino de exploración): conexión de todos los biestables en cadena, permitiendo acceder a ellos por desplazamiento con doble finalidad, como entrada para fijar su valor y como salida para observar su estado.

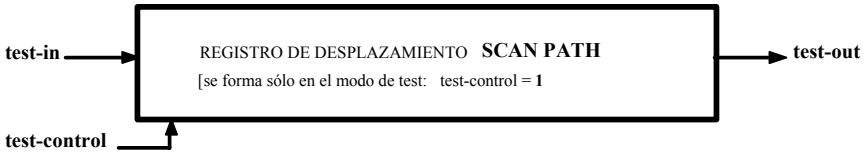
Cada biestable ha de tener dos modos de actuación:

1. el modo funcional, que corresponde al diseño propio del circuito integrado;
2. el modo test, de conexión al registro de desplazamiento, para efectuar el control (del estado anterior) y la observación (del nuevo estado) durante el test.



El circuito integrado ha de disponer de 3 terminales adicionales:

- la entrada serie al registro de desplazamiento *test-in*
- la salida de dicho registro (del último biestable) *test-out*
- y una entrada de control del modo de operación *test-control.*



Obviamente, si el número de biestables (del *scan path*) es muy alto, pueden organizarse en varios registros de desplazamiento (cada uno de ellos precisará dos nuevos terminales: su entrada y su salida).

En el test con *scan path*, el primer paso será la comprobación de «no bloqueos» en el propio «camino de observación», en los propios biestables del circuito; para ello, se introduce en el registro de desplazamiento una secuencia de «ceros» y «unos» alternados y se comprueba la salida de dicha secuencia, después del retraso correspondiente a los **n** biestables del registro de desplazamiento.

Una vez comprobado el *no bloqueo* de los biestables, el *análisis de fallos* y la *simulación de fallos* se efectúan como si el circuito fuera combinacional, suponiendo que el estado anterior son entradas externas del circuito y el nuevo estado (tras un pulso de reloj) se observa en salidas externas del mismo. De esta forma, por métodos puramente combinacionales, se generan los vectores de test y se estudia la cobertura de test que proporcionan, teniendo en cuenta que cada vector de test incluye cuatro componentes:

vector de test = {vector de entrada, estado anterior, vector de salida, estado siguiente}

La aplicación del test, al final del proceso de fabricación, ha de efectuarse según la siguiente secuencia:

1. modo de test: se carga «en serie» el estado anterior sobre los biestables, lo cual requiere tantos pulsos de reloj y tantos «vectores de entrada» como sea el número de biestables **n**,

2. modo de funcionamiento: se fija el vector de entrada, se aplica un pulso de reloj y se observa el vector de salida,

3. modo test: se observa el estado siguiente, extrayéndolo «en serie» de la cadena de biestables (**n** pulsos de reloj).

Obviamente, a la vez que se observa el estado siguiente (etapa 3) se efectúa simultáneamente la carga del «nuevo estado anterior» (etapa 1), correspondiente al siguiente vector de test; de manera que el número efectivo de pulsos de reloj que requiere cada vector de test es **n + 1**, siendo **n** el número total de biestables que conforman el registro de desplazamiento (o cada uno de los registros de desplazamiento, caso de configurarse varios de ellos).

25.5. Test de placas y de sistemas completos

La extraordinaria complejidad que alcanzan hoy día las placas de circuitos digitales y los sistemas digitales completos repercute fuertemente sobre su comprobación (test), que resulta sumamente difícil.

La filosofía desarrollada en el método *scan path* para el test de circuitos secuenciales es aplicable al test de placas de circuitos integrados digitales (y al test de sistemas digitales completos) en la siguiente forma:

- ha de incluirse un biestable en cada terminal (entrada o salida) de los circuitos integrados; dicho biestable actuará en «modo de test» formando parte (junto con los biestables internos) del registro de desplazamiento que conforma el camino de exploración (*scan path*) para el test del circuito integrado;
- los caminos de exploración (registros de desplazamiento) de los diversos circuitos integrados serán conectados en serie para formar un largo camino de exploración global para el test de la placa;
- este gran *scan path* permite controlar en serie todas las entradas de los circuitos integrados y observar en serie todas las salidas (y, también, el estado interno) de los mismos, de forma que podemos «testear» cada uno de dichos circuitos utilizando sus propios vectores de test;
- asimismo este *scan path* permite fijar valores en las salidas de los circuitos integrados y observar, luego, las entradas de todos los circuitos, lo cual hace posible comprobar las pistas de conexión (el circuito impreso), verificando si los valores fijados en las salidas de los circuitos son recibidos correctamente en las entradas de los circuitos conectados a tales salidas.

Esta forma de abordar el test de placas digitales, mediante la inclusión de «biestables en el contorno» de todos los circuitos integrados, para formar con ellos (junto con los biestables internos propios) un camino de exploración (*scan path*) de la placa, constituye un método de comprobación que recibe el nombre de *boundary scan* (exploración de contorno). Se encuentra en proceso de estandarización a través de un grupo de IEEE, el JTAG (*Joing Test Action Group*) encargado de redactar la correspondiente normativa.

El *boundary scan* requiere disponer de circuitos integrados especiales, preparados para dicho test de contorno; tales circuitos incorporan «biestables periféricos» en todos sus terminales (*pines*), es decir, en todas sus entradas y salidas, y cuatro terminales adicionales dedicados específicamente al «test»:

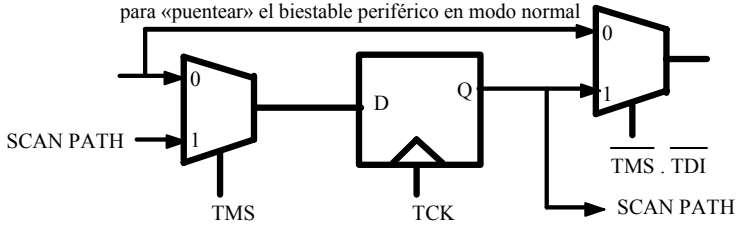
- dos para la entrada **TDI** y salida **TD0** del registro de desplazamiento (*scan path*),
- uno como entrada de control para la ejecución del test **TMS** (*test mode switch*)
- y otro como entrada de reloj para el test **TCK**.

Si el circuito integrado es secuencial, los «biestables propios» del circuito son conectados, asimismo, al registro de desplazamiento (*scan path*) junto con los «biestables periféricos», para poder efectuar el test completo de los circuitos.

El circuito funciona en tres modos, completamente diferentes:

- el «modo normal» en el cual no actúan los biestables periféricos (se encuentran «puenteados») y «no existe» el *scan path*: **TMS = TDI = 0**;
- el «modo de desplazamiento» **TMS = 1, TCK = pulsos de reloj**, en que se produce el desplazamiento del registro que configura el *scan path* y, por tanto, la entrada y la salida de los vectores de test;
- el «modo de ejecución de test»: **TMS = 0, TDI = 1, TCK = un solo pulso de reloj** en que los «biestables periféricos» actúan como entradas de los circuitos integrados (para realizar su test interno) y como salidas de los mismos (para realizar el test de las conexiones entre ellos).

Estos tres modos de funcionamiento se diferencian, en los biestables periféricos, mediante dos multiplexores**:**



El test de cada circuito integrado se efectúa en la forma siguiente:

① vector de entrada para dicho test:
- entradas al circuito integrado: se cargan sobre los biestables periféricos de las entradas del mismo;
- estado anterior: se carga sobre los biestables internos propios del circuito.

② un pulso de reloj en modo de ejecución de test;

③ vector resultante «de salida» del test:
- salidas del circuito integrado: se encontrarán en los biestables periféricos de las salidas del mismo;
- nuevo estado: en los mismos biestables internos propios del circuito.

El test de las pistas de la placa de circuito impreso se efectúa así:

① valores a comunicar a través de las pistas: se cargan sobre los biestables periféricos de las salidas de los diversos circuitos integrados;

② un pulso de reloj en modo de ejecución de test;

③ valores transmitidos a través de las pistas: se encontrarán en los biestables periféricos de las entradas de los circuitos integrados.

El test se realiza mediante la ejecución de las siguientes fases:

- fase de desplazamiento: **TMS = 1, TCK = n** pulsos de reloj (siendo **n** el número de biestales, la longitud del *scan path*); se produce el desplazamiento del registro que configura el camino de exploración y, por tanto, la entrada del vector, bit a bit, de test a dicho registro;
- fase de ejecución: **TMS = 0, TDI = 1, TCK =** un solo pulso de reloj; dicho pulso de reloj ejecuta el test y los biestables recogen el resultado del mismo: los biestables propios y los biestables periféricos de las salidas reciben el resultado del test interno de cada circuito integrado; los biestables periféricos de las entradas reciben el test de conexiones entre circuitos (pistas del circuito impreso);
- de nuevo una fase de desplazamiento: **TMS = 1, TCK = n** pulsos de reloj; permite observar el resultado del test y, a la vez, introducir el vector para el siguiente test.

Al igual que cada circuito integrado, la placa circuital dispondrá de cuatro terminales específicos para el test:

- **TDI** (*Test Data In*): entrada de vectores de test
- **TDO** (*Test Data Out*): salida de resultados del test
- **TCK** (*Test Clock*): reloj para la ejecución del test
- **TMS** (*Test Modo Switch*): control «modo test»

Dentro de la placa, las entradas **TDI** y las salidas **TDO** de test de los circuitos integrados que contiene han de estar unidas en cascada (**TDO → TDI'**), formando una larga cadena de registros de desplazamiento (un *scan path* global). Siendo **p** el número total de biestables presentes en la placa, el test «global» de la misma requiere **p** pulsos de reloj (con **TMS = 1**) para la «fase de desplazamiento» (en la cual, se observan los resultados del test anterior y se introducen, a la vez, los vectores de entrada para el test siguiente) y **1** pulso de reloj (con **TMS = 0** y **TDI = 1**) para la «la fase de ejecución».

Los «vectores de entrada» necesarios son los correspondientes al test de los diversos circuitos integrados contenidos en la placa, adecuadamente encadenados entre sí (según el *scan path* que tales circuitos configuran), intercalados (conforme a los «biestables periféricos» de las salidas de los integrados) con los vectores para el test de conexiones entre circuitos (test de las pistas del circuito impreso), que serán conjuntos aleatorios de ceros y unos.

Del mismo modo (con la misma filosofía y con los mismos recursos) que se ha planteado el test de una placa de circuitos digitales, puede desarrollarse el test de un conjunto de placas o, lo que es lo mismo, el test de un sistema digital completo.

25.6. Test interno

En circuitos integrados de gran complejidad (en particular, en bloques correspondientes a estructuras matriciales tales como RAM, ROM, PLA, etc.,) conviene incluir, en el propio circuito integrado módulos que permitan realizar «internamente» el test; es decir, circuitos complementarios para:

- generar por sí mismos un conjunto apropiado de vectores de test (vectores de entrada)
- y analizar los resultados (vectores de salida) del test.

Téngase en cuenta que un *scan path*, para ejecutar un sólo test, requiere **n + 1** ciclos (siendo **n** el número de biestables presentes en dicho camino de exploración), ya que es preciso almacenar, bit a bit, el vector de entrada (y también extraer, bit a bit, el resultado del test); el número de ciclos puede reducirse por un factor **n + 1** si los vectores de entrada se generan internamente y los resultados se analizan (o se almacenan en forma compacta) en el interior del circuito integrado. Tal es el objetivo del *test interno*.

Consideremos una memoria RAM de gran capacidad: su test implica escribir y leer en cada uno de sus registros varias veces, lo cual requiere un alto número de ciclos. Sin embargo, la generación y el análisis de vectores de test es muy sencilla (como veremos a continuación); resulta conveniente incluir tales funciones (generación y análisis) dentro del circuito integrado, lo cual permite prescindir (en buena medida) de las costosas máquinas de test y realizar el test con un equipo de observación mucho más simple.

El test interno permite comprobar los circuitos posteriormente, una vez instalados en el sistema electrónico concreto en que vayan a ser utilizados, y verificar (en cualquier momento y de forma sencilla) que dichos circuitos integrados siguen funcionando bien. Es decir, el test interno no es sólo una facilidad para la necesaria verificación del circuito en el proceso de fabricación, sino que configura un recurso permanente (una prestación de la cual se dispone siempre) que permite efectuar nuevas verificaciones «in situ», a lo largo del tiempo de utilización del circuito integrado; en ambos casos, la ejecución del test es muy simple ya que, básicamente, se realiza en el interior del circuito.

El *test interno* plantea la realización del test, por medio de un circuito auxiliar (por *hardware*), dentro del propio circuito integrado; es decir, incluir en su diseño (y, posteriormente, en su fabricación) un añadido circuital específico para la ejecución del test. Dicho circuito auxiliar ha de contener, obviamente, dos partes (entradas / salidas):

- un generador de los vectores de test (vectores de entrada)

- y un analizador de los resultados del mismo (vectores de salida).

Una forma sencilla de «generar vectores» de **p** bits la constituye un contador de **p** biestables, que producirá todos los vectores de **p** bits en orden numérico binario; otra posibilidad, tan sencilla o más que la anterior, es generar vectores «aleatoriamente» utilizando un registro de desplazamiento. La segunda opción tiene la ventaja de que los vectores contiguos presentan menor correlación (son más independientes entre sí) que los que proceden de un contador.

En ocasiones (por ejemplo, en el test BIST para memorias RAM que se describe a continuación), los resultados del test presentan una gran uniformidad y basta un pequeño circuito combinacional para detectar si son correctos; una salida del circuito integrado servirá para informar de que los sucesivos resultados del test son válidos (dicha salida se mantendrá constantemente a **0** durante el test: un simple pulso a **1** indica mal funcionamiento del circuito).

Este tipo de solución combinacional para analizar el test no siempre es posible y en muchas ocasiones hemos de limitarnos a almacenar, en forma compactada, los sucesivos vectores de salida de test; la *firma sobre **p** bits* (una extensión de la paridad a **p** bits) permite compactar, sobre una longitud **p**, cualquier serie de bits por larga que sea, detectando error en la misma siempre que el número de errores no sea superior a **p**.

En los siguientes subapartados se detallan diversas configuraciones relativas al test interno de circuitos integrados:

- el test interno de memorias, **BIST**,
- la generación aleatoria de vectores de test, **LFSR**,
- el almacenamiento compacto de vectores resultantes del test, **FIRMA**
- y el aprovechamiento de un registro como LFSR y como FIRMA, **BILBO**.

25.6.1. BIST (Built In Self Test): test interno de memorias

Genéricamente el término **BIST** (autotest construido internamente) alude a cualquier procedimiento de *test interno*, pero suele emplearse más específicamente para el test de memorias, en el cual las direcciones se generan mediante un contador y los contenidos de los registros corresponden a *todo unos*, a *todo ceros* y a un «ajedrezado» de *ceros y unos* alternativos.

El test de las memorias RAM suele realizarse mediante el siguiente algoritmo (supuesta una memoria de N registros, cuya numeración irá de 0 a N-1):

1	de 0 a N-1	escribir todos los registros con **111111 ...**
2	de 0 a N-1	leer y comprobar cada registro y escribir en el mismo **000000 ...**
3	de 0 a N-1	leer y comprobar cada registro y escribir **111111 ...**
4	de N-1 a 0	leer y comprobar cada registro y escribir en el mismo **000000 ...**
5	de N-1 a 0	leer y comprobar cada registro y escribir **111111 ...**
6	de 0 a N-1	escribir sucesivamente registros impares con **101010 ...** y registros pares con **010101 ...**
7	de 0 a N-1	leer y comprobar cada registro
8	de 0 a N-1	escribir sucesivamente registros impares con **101010 ...** y registros pares con **010101 ...**
9	de 0 a N-1	leer y comprobar cada registro
10	test completo, memoria disponible para ser utilizada.	

De esta forma, se observa no sólo la escritura y lectura de un **0** y de un **1** sobre cada biestable de la memoria, sino también la influencia de los biestables contiguos: de ahí que, a la vez que progresa la lectura de «unos», se escriban «ceros» antes de leer el registro contiguo y viceversa y de ahí, también, la utilización de ajedrezados con «ceros» y «unos» intercalados en las dos direcciones y en las dos posibilidades.

Puede comprobarse que, con el algoritmo anterior, cada biestable (i , j, siendo i la fila del biestable y j la columna del mismo) se prueba en sus dos valores (**0** / **1**) respecto a todas las posibilidades de los ocho biestables «vecinos» que le rodean: los dos biestables contiguos de la misma fila (i , j-1 ; i , j+1), los tres biestables análogos de la fila anterior (i-1 , j-1; i-1 , j ; i-1 , j+1) y los tres de la siguiente (i+1 , j-1; i+1 , j ; i+1 , j+1). Asimismo, se comprueban las diversas posibilidades de interacción entre biestables del mismo registro (que se encontrarán en «hojas» diferenciadas, véase la estructura de las memorias RAM en 19.2., segundo volumen).

Este test requiere un contador para generar las direcciones de los registros; un segundo contador de cuatro bits y un decodificador para distinguir las diez fases de ejecución del test; un circuito combinacional muy simple que genera, a la vez, los vectores de entrada y de salida del test y un comparador entre la lectura de los registros y los citados vectores de salida.

Los vectores a generar son muy simples:

- dos componentes: $\mathbf{D_{impar}}$ biestables impares y $\mathbf{D_{par}}$ biestables pares de los registros;
- en la fase 1, $\mathbf{D_{impar},D_{par} = 11}$; en la fase 2 y en la fase 4 **11**(lectura) y **00** (escritura) alternativos; en la fase 3 y en la 5 **00** (lectura) y **11** (escritura) uno tras otro;
- en la fase 6 y en la 7, $\mathbf{D_{impar},D_{par} = 10}$ (registros pares) y **01** (impares) sucesivamente y en las fases 8 y 9 **01** (registros pares) y **10** (impares).

Basta un pequeño circuito combinacional, a partir del decodificador de fases citado, de un biestable que distinga entre lectura y escritura (en las fases 2, 3, 4 y 5) y de la línea de direccionamiento **A0** (registros pares e impares), para generar los citados vectores de test que cubren tanto la entrada (escritura) como la salida (lectura) del mismo.

El comparador entre la lectura de los registros y el vector de referencia generado por el circuito combinacional ha de habilitarse solamente en los momentos de lectura; de forma que su salida se encontrará normalmente a **0** y solamente pasa a **1** si la lectura del registro no coincide con el valor esperado.

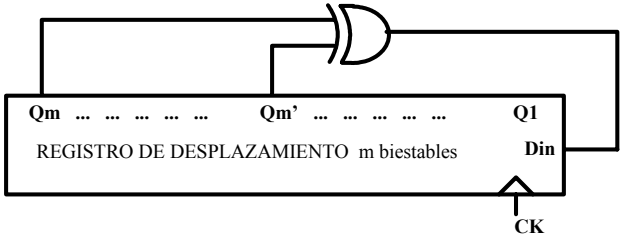
Este tipo de test interno (**BIST**) requiere solamente tres terminales: **TMS** de control del «modo de test», **TCK** para enviar un reloj que haga avanzar a los contadores y **TDO** el terminal de salida del comparador; el test se realiza haciendo **TMS = 1**, enviando una alta frecuencia de reloj por **TCK** y observando la ausencia de **1** en **TDO**. No es necesario contar los pulsos enviados a **TCK**: basta dedicar suficiente tiempo al test para asegurar que se ha llegado a la fase 10.

25.6.2. LFSR (Linear Feedback Shift Register): generación «aleatoria» de vectores

La ejecución del test (sobre un bloque combinacional) requiere «vectores de entrada» con **p** componentes (**p** entradas); tales vectores pueden generarse mediante un contador de **p** biestables (módulo 2^p) y, en tal caso, aparecerán según el orden numérico binario.

Otra posibilidad consiste en generar vectores de forma «aleatoria», que asegura menor correlación (nula si realmente fueran aleatorios) entre vectores sucesivos, por lo cual resultan mucho más adecuados para el test; en particular, cuando no se utiliza un conjunto completo (2^p) de «vectores de entrada» (se ha comentado antes que los vectores de entrada aleatorios consiguen «coberturas de fallos» relativamente altas).

Los registros de desplazamiento (SR: *Shift Register*) con realimentación lineal (LF: *Linear Feedback*) constituyen sencillos generadores de vectores «seudoaleatorios». Para determinados valores de longitud de un registro de desplazamiento (**m** biestables, siendo **m** el más significativo), se verifica que, realimentando hacia la entrada, a través de una puerta "**o-exclusiva**", la salida del biestable **m** y otra de las salidas **m'**, adecuadamente escogida, el registro genera sucesivamente y en forma «seudoaleatoria» todos los vectores de **m** bits menos el nulo **0000...**: 2^m -1 vectores.



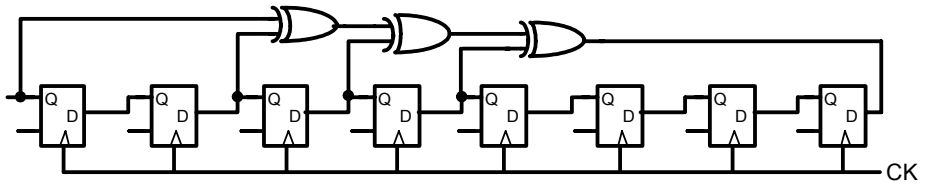
El número **m'** del segundo biestable a realimentar, para conseguir dicha generación «máxima» de vectores seudoaleatorios (2^m -1), se calcula por técnicas matemáticas de «polinomios generadores»; por ejemplo:

m = 10	m' = 7	1.023 vectores;
m = 15	m' = 14	32.767 vectores;
m = 20	m' = 17	1.048.575 vectores;
m = 25	m' = 22	33.554.431;
m = 31	m' = 28	2 mil millones de vectores;
m = 39	m' = 35	0,5 billones.

Si se añade un inversor detrás de la puerta "**o-exclusiva**" se obtienen todos los vectores de **m** bits menos el «todo unos» **1111...**, lo cual permite utilizar el vector nulo **0000…** para iniciar la generación de vectores seudoaleatorios.

Para longitudes habituales (8, 16, 24, 32,...) es preciso realimentar, a través de puertas "**o-exclusiva**" (paridad), más de 2 biestables hacia la entrada:

para m = 8 hay que realimentar los biestables 8°, 6°, 5° y 4° (255 vectores),



para m = 16: se realimentan 16°, 15°, 13° y 14° y produce 65.535 vectores distintos,

para m = 24: se realimentan 24°, 23°, 22° y 17° y genera 16.777.215 vectores.

Los registros de desplazamiento con realimentación lineal **LFSR** son los módulos utilizados normalmente como generadores internos de vectores de test. Respecto a contadores de igual longitud tiene la ventaja de que su circuitería es más simple (**m** biestables y una o varias puertas "**o-exclusiva**", sin tener que constituir las complejas funciones que corresponden al contaje) y que generan los vectores en forma «cuasi aleatoria» y no en orden numérico binario.

25.6.3. FIRMA: almacenamiento compacto de vectores

Al realizar un test con **n** vectores de entrada obtenemos **n** vectores de salida, es decir, una secuencia de **n** palabras de **p** bits: **n . p** bits; la idea es «comprimir» dichos resultados de forma que la cantidad de memoria necesaria para almacenarlos sea pequeña (en la práctica una sola palabra de igual longitud que cada vector salida: **p** bits) y de forma, también, que la obtención del resultado global del test sea sencilla: bien en cuanto a número de ciclos para leer el registro de resultados, bien en cuanto al «circuito de comparación» que evalúa si el resultado es o no correcto.

Entendemos por «firma sobre **p** bits» (del inglés *firm*) la reducción a una sola palabra de **p** dígitos de **n** palabras binarias, cada una de ellas de **p** bits, con las siguientes propiedades:

- la firma es una aplicación unívoca de **n** palabras de **p** dígitos sobre una palabra de **p** bits,
- dos conjuntos distintos, de **n** palabras de **p** dígitos cada uno, que no se diferencien en más de **p** bits, tendrán firmas diferentes entre sí,
- es decir, si en un conjunto de **n** palabras de **p** dígitos se modifican (se inviertan) **p** de los **n.p** bits (o menos), la firma del conjunto modificado será diferente de la firma del conjunto inicial.

De manera que la *firma sobre **p** bits* permite detectar errores que afectan a **p** bits o a menos de **p** bits, pues el «análisis de firma» de dos conjuntos de palabras diferentes conduce a dos firmas distintas, salvo que dichos conjuntos difieran en más de **p** bits (en cuyo caso sus firmas pueden ser diferentes o iguales).

Sea **S** el conjunto de **n . p** dígitos, su «firma» es una aplicación sobre **p** bits

$$\mathbf{S}\ (n\ .\ p\ bits) \longrightarrow \mathbf{F(S)}\ (p\ bits)$$

tal que, si dos conjuntos **S1** y **S2**

$$\mathbf{S1} \longrightarrow \mathbf{F(S1) = F1}$$

$$\mathbf{S2} \longrightarrow \mathbf{F(S2) = F2}$$

tienen la misma firma **F1 = F2**, puede afirmarse que:

- o bien ambos conjuntos son iguales (**S1 = S2**)
- o bien difieren en más de **p** bits (**S1** $\oplus$ **S2** contiene más de **p** «unos»).

Por ello, la firma detecta errores cuando su número no es superior a **p**, ya que la firma de un conjunto **S'** erróneo será distinta de la firma del conjunto correcto **S**:

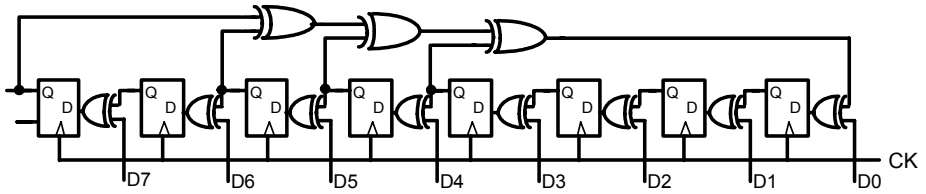
F' $\neq$ **F** si **S** $\neq$ **S'** y **S** $\oplus$ **S'** no contiene más de **p** «unos».

Obviamente, la bien conocida «paridad» corresponde a la «firma sobre 1 bit».

Para calcular la «firma sobre **p** bits» basta modificar un registro de desplazamiento realimentado linealmente **LFSR**, añadiendo puertas "**o-exclusiva**" en las entradas de sus biestables en la forma que sigue:

- cada palabra de **p** dígitos es recogida por el registro de desplazamiento a través de **p** «entradas paralelo especiales» y es «mezclada» con el estado del registro;
- cada biestable recibe, en su entrada D, la operación "**o-exclusiva**" entre la salida del biestable anterior (conexión serie propia del registro de desplazamiento) y el bit correspondiente de la palabra recibida (entradas paralelo especiales);
- el registro se encuentra, además, realimentado linealmente en la forma propia de los registros de desplazamiento con realimentación lineal.

Tal registro de desplazamiento, con entrada paralelo en forma de "**o-exclusiva**" y realimentación lineal, realiza la «firma sobre **p** bits» siendo **p** el número de sus biestables y el número de dígitos de las palabras que recibe:

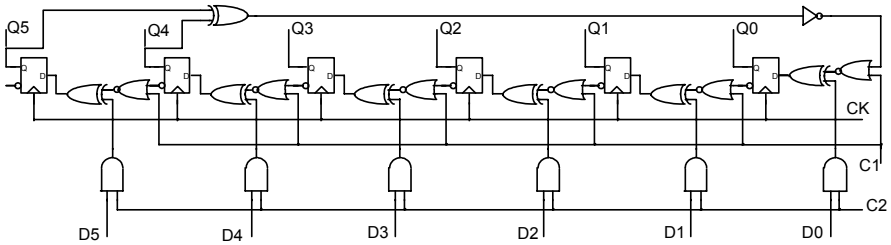


En el «registro de desplazamiento» anterior, al recibir uno tras otro los **n** «vectores de salida» de test (supuesto que sean de 8 bits), se realiza la firma del conjunto de dichos n vectores (**n**.8 bits) sobre una palabra de 8 dígitos. Puede demostrarse que dos conjuntos diferentes (de **n** palabras de 8 dígitos cada uno), cuya firma sea la misma, diferirán en más de 8 bits; es decir, permite detectar errores que afecten a 8 bits o menos.

Los circuitos que realizan la firma sobre **p** bits reciben el nombre de analizadores de firma.

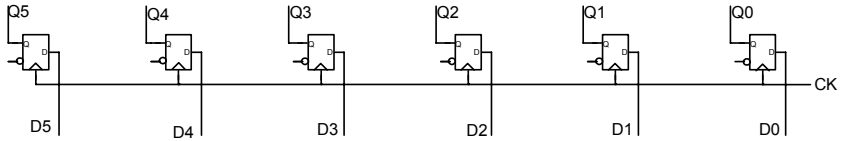
25.6.4. BILBO (Built In Block Observer): test interno mediante LFSR y FIRMA

Un registro síncrono, de entradas paralelo (o, lo que es lo mismo, un conjunto de biestables de tipo D), puede ser «reconfigurado» de forma que, además de actuar como registro retención, pueda hacerlo como generador de vectores (LFSR) y como analizador de firma o, también, pueda borrarse en forma síncrona. Basta, para ello, añadir al registro la lógica correspondiente al analizador de firmas, junto con dos entradas de control **C1** y **C2** que actúan a través de sendas puertas "**y**" y "**o-negada**", según el esquema de la figura.



Cuando ambas entradas tienen valor **1**, **C1** = **C2** = **1**, el bloque actúa como un simple registro de entradas paralelo:

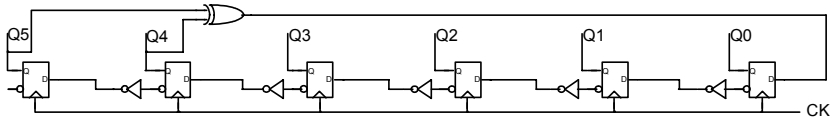
C1 = C2 = 1



Registro de entradas paralelo

Para ambas entradas a valor **0**, **C1** = **C2** = **0**, el bloque queda en forma de registro de desplazamiento realimentado linealmente **LFSR**, o sea, un generador de vectores de test:

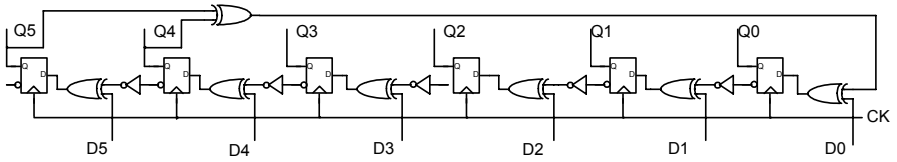
C1 = C2 = 0



Generador de vectores de test

Para **C1** = **0** y **C2** = **1**, el bloque se configura como analizador de firma:

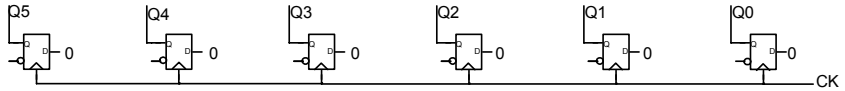
C1 = 0 C2 = 1



Analizador de firma

Y con **C1** = **1** y **C2** = **0**, se produce el borrado síncrono del registro:

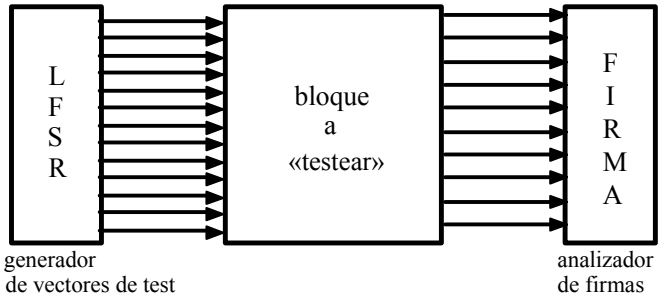
C1 = 1 C2 = 0



Borrado síncrono

Este tipo de registro, que puede ser empleado bien como registro de retención (o biestables individuales) con borrado síncrono, bien como generador de vectores o como analizador de firmas, recibe el nombre de registro **BILBO** (*Built In Block Observer*). Con esta estructura cualquier registro interno, o cualquier conjunto de biestables, de un circuito integrado puede ser utilizado, a efectos de test, como generador de vectores de test y como analizador de firma.

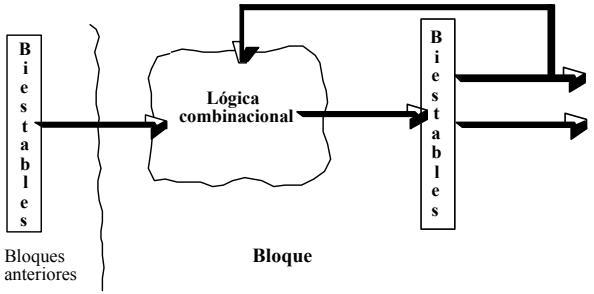
Los registros **BILBO** dan lugar a una configuración de test interno que consiste en un generador de vectores de test (**LFSR**) y un analizador de firmas; el primero genera vectores de test sobre las entradas del bloque a testear y el segundo almacena, en forma comprimida, los vectores de salida resultantes del test.



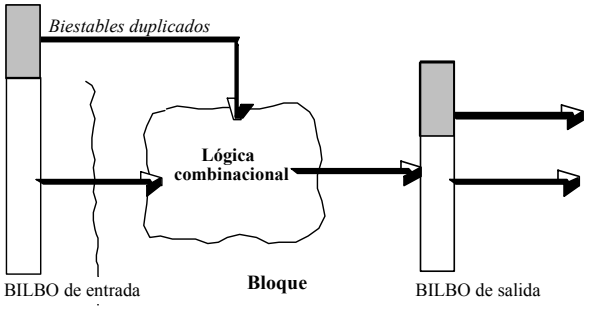
Para efectuar el test, esta configuración requiere tres terminales: **TMS** para controlar el modo de test, **TCK** como reloj y **TDI** para la salida del analizador de firmas; dicha salida puede obtenerse en forma de firma global (**p** bits) que se extrae «por desplazamiento» (mediante *scan path*) bit a bit o en forma unitaria (1 bit), añadiendo un comparador que confronte la firma obtenida con la esperada en el caso correcto.

Los registros de «entrada» (generador de vectores) y «salida» (analizador de firma) pueden ser registros añadidos a los solos efectos del test, pero también pueden ser registros o conjuntos de biestables propios del diseño, configurados en forma de **BILBO**.

Según se comentó en el capítulo 15 (volumen 2), un circuito secuencial síncrono puede «segmentarse» en bloques o módulos conformados por una parte combinacional y un registro o conjunto de biestables que recogen las salidas de ella y conectan con el bloque o bloques siguientes; algunas salidas del propio bloque se realimentarán sobre el mismo. De manera que cada «parte combinacional» se encuentra entre dos registros o conjuntos de biestables, uno de ellos correspondiente al bloque o bloques anteriores (entradas) y el otro, el propio del bloque (salidas).

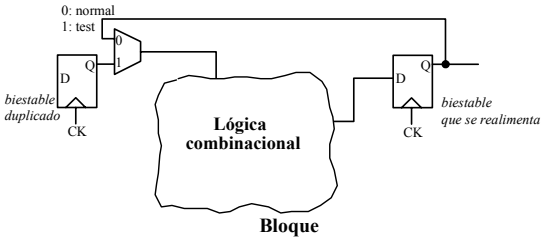


A partir de la estructura representada en esta figura, para obtener una configuración BILBO es necesario «duplicar» (en el modo de test) los biestables que se realimentan sobre el propio bloque, de forma que una mitad siga siendo biestables de salida del bloque para configurar, con el resto de los biestables de salida, un registro BILBO que actúe como analizador de firma y la otra mitad se agregue a los biestables de entrada, formando otro registro BILBO como generador de vectores de test.



De manera que un circuito integrado complejo puede dividirse (a efectos de test) en partes, formadas (cada una de ellas) por un núcleo combinacional entre dos registros **BILBO** y, de esta forma, puede realizarse el test interno de cada parte utilizando el registro anterior, como generador de vectores de test, y el registro siguiente, como analizador de firmas; para cada una de las firmas, ha de comprobarse su coincidencia (o diferencia) con las que corresponden al circuito sin fallos.

La configuración necesaria para «duplicar» los registros que se realimentan sobre el bloque puede ser la siguiente:



25.6.5. Autotestabilidad, seguridad y tolerancia frente a fallos

La inclusión en el propio circuito integrado de módulos que permiten realizar su test internamente introduce un nuevo concepto, la autotestabilidad: la capacidad de verificar «in situ» y en cualquier momento la configuración funcional del circuito integrado. De esta forma, el circuito puede autocomprobarse repetidamente y avisar cuando tal comprobación detecte fallos.

Cada vez son más los sistemas digitales que se «autochequean», antes de comenzar su actuación (o, también, periódicamente a lo largo de la misma, aprovechando pausas funcionales), para asegurar que todas sus partes se encuentran en perfecto estado de funcionamiento.

Un paso más allá en la detección de errores consiste en realizar (al menos parcialmente) el test del circuito en forma simultánea a su funcionamiento como tal, de manera que el circuito integrado supervise su actuación continuamente y en paralelo con la misma y avise de la existencia de fallos lo antes posibles y sin necesidad de efectuar una solicitud exterior de test. En este caso el circuito presentará una gran seguridad frente a fallos (permitiendo «desechar» con inmediatez cualquier funcionamiento erróneo).

Una solución que ya se utiliza es la de duplicar determinados sistemas o partes digitales y verificar la coincidencia de los cálculos realizados por ambos.

Lo óptimo sería que el circuito, además de avisar en cuanto existe un fallo (seguridad frente a fallos), fuera capaz de corregir su propio funcionamiento y su respuesta fuera correcta, aun en presencia de fallos: circuito tolerante a fallos. Al igual que existen códigos autocorrectores de errores (que permiten tal corrección cuando el número de errores es limitado) es posible diseñar circuitos que, para un número limitado de fallos, la presencia de ellos no introdujera errores en la respuesta del circuito.

Al igual que la duplicación de partes digitales permite que se «vigilen» mutuamente, su «triplicación», añadiendo un circuito de «decisión mayoritaria», da lugar a que, si una de las tres partes iguales que funcionan en paralelo «falla» (si proporciona una respuesta diferente a la de las otras dos), sus resultados sean desechados y se adopte como respuesta correcta la común de las otras dos partes.