Introducción al Diseño de CIs

Universitat Autònoma de Barcelona

Curso académico 2009-10

Elena Valderrama

Carles Ferrer

Capítulo 10 : Diseño para la Testabilidad y BIST

Capítulo 10: Diseño para la Testabilidad y BIST

Elena Valderrama; Carles Ferrer

Capítulos

Introducción

En el capítulo anterior hemos insistido mucho en la idea de que, para que un CI de complejidad media/alta pueda testearse con garantías, la estrategia de test debe ir pensándose desde las primeras etapas del diseño. Por otro lado, hay varios problemas del test de CIs que hemos "ocultado" hasta ahora; por ejemplo,

1. ¿Cómo se generan los vectores de test para las partes secuenciales del circuito?. El método de sensibilización de caminos, tal como lo hemos presentado, no es aplicable al caso secuencial.
2. ¿Cómo se implementan las técnicas de test concurrente y no-concurrente que permiten comprobar el circuito a lo largo de toda su vida activa?.
3. ¿Con qué máquina de test se comprueban los circuitos que utilizan las tecnologías "punteras" (las más rápidas existentes, por ejemplo)?.

Estas preguntas merecen un curso completo, pero aquí vamos a tratar de esbozar las dos o tres ideas más importantes que nos permitirán intuir que éstas preguntas tienen respuesta y por donde va ésta.

En primer lugar estudiaremos el concepto de **Diseño para la Testabilidad** y veremos cómo insertando lógica adicional en nuestro circuito podremos vislumbrar soluciones para los problemas planteados. Comenzaremos viendo una serie de técnicas simples que nos permitirán (1) aumentar la testabilidad del circuito y (2) solventar el problema de las partes secuenciales.

Presentaremos más tarde (como una técnica específica de diseño para la testabilidad) el concepto de **BIST** (*Built In Selft Test*) y veremos cómo introduciendo de nuevo lógica adicional podemos construir circuitos autocomprobables (con diferentes grados de seguridad) que (1) hacen posible el test periódico del chip dentro del propio sistema y (2) solventan la dificultad conceptual de cómo testear las tecnologías "punteras" puesto que, si la lógica de test está dentro del propio circuito esta es, por definición, tan rápida como éste.

Introducción

Un punto que hay que tener muy claro es que la mejora de la testabilidad del circuito tiene un precio, y este precio es el aumento de área debida a la inclusión de lógica extra. *Esta lógica extra (1) incrementa el coste del circuito (más área, más pines) y (2) degrada las prestaciones del mismo en términos de velocidad y de consumo*. A pesar de estas afirmaciones, es indispensable en los grandes CIs incorporar circuitería de test no sólo para aumentar su testabilidad, sino incluso para simplemente hacerla posible.

Diseño para la Testabilidad

El concepto de diseño para la testabilidad abarca un gran espectro de posibilidades. Se puede considerar que, por definición, cualquier técnica o metodología que se aplica a un circuito determinado para mejorar su comprobabilidad (controlabilidad y observabilidad). Evidentemente, hay muchas formas de realizar el diseño para test, ya sea a partir de métodos simples y poco costosos, o con métodos más complejos, los cuales también suelen ser mucho más caros.

Williams y Parker dieron en 1983 una primera clasificación de técnicas de diseño para test. Clasificaban estas técnicas en técnicas ád-hoc y en técnicas estructuradas.

Las **técnicas ád-hoc** son técnicas que resuelven los problemas de testabilidad de un determinado diseño y, por lo tanto, son difícilmente generalizables. Suelen ser técnicas totalmente heurísticas, aunque en ciertos casos pueden llegar a sistematizarse. En términos generales ofrecen soluciones poco costosas.

Las **técnicas estructuradas** intentan resolver el problema general con una metodología de diseño que utiliza un conjunto de reglas que garantizan que el circuito resultante es fácilmente testable. Son técnicas que suelen dar un rendimiento mucho mejor en cuanto a la testabilidad de los circuitos que las técnicas àd-hoc aunque, como contrapartida, suelen ser más costosas que éstas. Dentro de las técnicas estructuradas, se pueden distinguir dos grupos diferenciados según cual sea la metodología usada para conseguir la testabilidad deseada. Los dos principales tipos de técnicas estructuradas son las siguientes:

- Las técnicas que denominaremos, en general, de tipo **scan-path**, utilizan como método operativo la combinación de unas normas de diseño para conseguir la modificación correcta (evitando introducir azares y carreras) de los latches que constituyen el sistema, con el re-conexionado de los mismos en una serie de registros de desplazamiento que el sistema activa en modo test. Estos registros de desplazamiento permiten controlar y observar los nodos internos, facilitando la tarea de comprobación del circuito

Diseño para la Testabilidad

- Las técnicas denominadas **built-in-selftest (BIST),** o de *generación y compresión pseudoaleatoria de vectores de test,* que se basan en la construcción de un registro especial que actúa como generador y/o compresor de los patrones de test que se aplican al circuito. A diferencia de las técnicas de scan-path, estas técnicas no necesitan introducir ni observar los vectores de test al/del circuito ya que este registro especial, tal y como indica su nombre, actúa como generador y/o compresor. Comparando la signatura o vector final guardado en el registro, se sabe entonces si el sistema ha operado o no correctamente.

La figura 1 muestra cómo, aunque que se haya hablado de tres técnicas diferenciadas por las características de sistematicidad y manera de aplicación, los límites existentes entre ellas no siempre están claramente definidos. De hecho, son muchas las técnicas que no pertenecen puramente, a uno de los métodos presentados anteriormente y, conforme la evolución de los circuitos integrados alcanza mayores cotas de complejidad, mayores son las interacciones que se establecen entre ellas.

ver figura >> 01

En la práctica los circuitos tienen tal complejidad que no es suficiente con aplicar una única técnica de las mencionadas hasta ahora. En las librerías de celdas, los generadores de macroceldas del tipo PLAs, memorias u otras, suelen incorporar circuitería BIST para que el módulo en particular se pueda autotestear. Para poder aislar la lógica combinacional de la secuencial y poder atacar el test del circuito “a trozos”, el diseñador suele incluir uno o varios registros de scan, a la vez que tiene cuidado en no incluir estructuras poco testables y seguir las reglas de diseño para test que se verán en el apartado de técnicas ád-hoc. Como se ve, ***no hay una técnica única que resuelva todos los problemas***; el diseñador debe conocer todas las técnicas que tiene a su disposición y aplicarlas de la mejor manera posible para alcanzar el objetivo de hacer un circuito fácilmente testable.

Técnicas àd-hoc

Hasta mediados de la década de los ochenta, todas las modificaciones para test que se realizaban en un circuito se dejaban en segundo término (generalmente se tenían en cuenta solamente cuando el circuito no funcionaba correctamente) y servían para localizar y determinar donde y porqué se producía el error en el circuito integrado. Los principales métodos que se utilizaban para localizar los fallos se basaban en normas de partición de módulos que permitían un mayor control y observación de los nodos internos (los métodos de partición de módulos en circuitos integrados suelen ser consecuencia de particiones establecidas a nivel de placas).

Las técnicas ád-hoc son simplemente un conjunto de normas a tener presentes a la hora de diseñar un circuito integrado que mejoran el acceso y control de nodos interiores. Estas normas hacen especial referencia a nodos conflictivos, difíciles de *controlar* (*controlar* = forzar valores) y de *observar* (*observar* = conocer el valor lógico en el que se encuentra el nodo en cada momento).

Ciertos autores consideran que este conjunto de normas no pueden considerarse propiamente como una metodología de diseño para el test; sin embargo, se ha creído conveniente incluirlas como normas de diseño que, de tenerse presentes a la hora de diseñar un circuito integrado, pueden facilitar el test del mismo a un coste reducido.

Normas prácticas de diseño para test

Muchas de las normas que aquí se comentan son utilizados por técnicas posteriores para conseguir unos mayores rendimientos de aplicabilidad en el momento de utilizar una técnica concreta de diseño para test, aunque muchas de ellas son de sentido común. La mayoría de ellas se basan, fundamentalmente, en la mejora de la controlabilidad y de la observabilidad de los nodos internos.

Técnicas àd-hoc

Regla 1: Facilitar el acceso y la observación de nodos internos.

Esta regla hace especial referencia a señales internas del circuito que suelen ser señales clave de control de nodos difícilmente accesibles o observables. Típicamente las señales reloj, *presets* y *clears*, señales de selección de datos, de tri-states, señales de habilitación (*enable*), etc.. son señales que conviene que sean muy controlables, mientras que las señales de control del circuito, los valores de contadores y registros de desplazamiento, las señales que viajan por líneas de realimentación y las salidas de los elementos de memoria en general son señales que conviene que tengan una alta observabilidad.

¿Cómo se puede aumentar la controlabilidad y la observabilidad de tales señales?. Pues por ejemplo ...

- Añadiendo pines de entrada/salida adicionales y convirtiendo las señales más conflictivas en señales de E/S.
- Utilizando multiplexores y demultiplexores para facilitar el acceso/observación a/de nodos internos.
- Usando registros de desplazamiento (entrada serie, salida paralelo) para mejorar la controlabilidad y registros de desplazamiento (entrada paralelo, salida serie) para mejorar la observabilidad.

Regla 2: Asegurar la inicialización de todo elemento de memoria interna.

Es fundamental para el test del circuito saber en todo momento que todas las señales internas hayan estado inicializadas y, por lo tanto, que en caso de funcionar correctamente el circuito, han de conducir a estados perfectamente predefinidos. Así por ejemplo, todos los elementos de memoria del sistema conviene que tengan señales de*reset* y/o *preset* que, en el momento de arrancar el sistema conduzcan a estados conocidos.

Regla 3: Evitar la lógica redundante.

Los estados y la lógica redundante pueden (1) introducir nuevos fallos producidos precisamente por la lógica redundante y (2) pueden enmascarar la detección de fallos.

Técnicas àd-hoc

Regla 4: Evitar la generación de señales de reloj internas así como de asincronismos.

Todas las señales de reloj que llegan a los diferentes elementos de memoria del circuito deberían provenir directamente de señales externas que no pasen por más puertas lógicas que las del árbol de inversores (*clock buffering*). Recordemos que la lógica combinacional genera *glitches* que pueden interpretarse como flancos de reloj.

Regla 5: Partición del circuito.

En caso de circuitos muy grandes, conviene establecer algún mecanismo que permita dividirlos en varios bloques accesibles directamente desde el exterior.

Regla 6: Dar un buen informe del circuito diseñado.

Es importante a la hora de realizar la comprobación del circuito una vez integrado, tener toda la información necesaria para comprobar que el circuito cumple realmente con las especificaciones para las que fue diseñado. Hay que tener en cuenta que es de la máxima importancia conocer:

- El funcionamiento lógico del circuito, especificando los distintos modos de funcionamiento.
- Las señales de reloj y de control.
- La sincronización de todas las señales externas, tanto de las señales de entrada como de las de salida, así como sus diagramas de tiempo y la correspondiente secuencia lógica de la salida de datos.
- En el caso de incluirse lógica adicional para el test, cabe dejar bien claro cuales son las partes que se comprueban y, si es necesario, especificar los vectores de test que deben aplicarse.

Técnicas de scan-path

Las técnicas de *scan-path* se basan fundamentalmente en el hecho de que si todos los *latches* internos pueden ser controlados y observados de manera directa, entonces la generación de vectores de test así como la simulación de fallos puede llegar a reducirse a la aplicación de vectores de test y de simulación ***sólo*** a las partes combinacionales del circuito.

Imaginemos un circuito cualquiera, con una parte combinacional y una parte secuencial como la representada en la figura 2 por un serie de *flip-flops D* (la argumentación puede extenderse a cualquier tipo de *flip-flop* o *latch*).

ver figura>>02

El *scan-path* propone, en principio, añadir multiplexores [1] a la entrada de los elementos de memoria tal como muestra la figura 3, de manera que a través de una señal de *Scan-Select* podamos "re-conexionar" todos los *flip-flops* del circuito formando un registro de desplazamiento. Esto nos permitiría separar el test de la parte secuencial, que se limitaría ahora a testear un registro de desplazamiento, del test de la parte combinacional para la cual ya conocemos técnicas de generación de los vectores de test.

ver figura>>03

El circuito así diseñado puede estar en dos situaciones bien diferentes :

1. Cuando *Scan-Select*=0, los elementos de memoria reciben las entradas de la lógica combinacional, tal como debe ser en el circuito que estamos diseñando. Se dice entonces que el circuito (o sistema) estáen **modo sistema**.

2. Por el contrario, cuando *Scan-Select*=1, la entrada de cada elemento de memoria viene del elemento de memoria anterior o de una entrada externa a la que llamaremos *Scan-Data-In* (*SDI*), formando el conjunto de ellos un registro de desplazamiento con entrada serie (*SDI*) y salida serie (*Scan-Data-Out: SDO*). Este es el modo de funcionamiento que nos interesará para realizar el test del circuito y, consecuentemente, diremos que cuando *Scan-Select*=1 el circuito está en **modo test.**

[1] Las librerías de celdas ofrecen flip-flops específicos que contienen en su interior el multiplexor o otros sistemas de conmutación de sus entradas.

Técnicas de scan-path

Para testear el sistema procederemos de la siguiente manera:

1. Generaremos el conjunto de vectores de test que comprueben el funcionamiento de la parte combinacional con una cobertura de fallos adecuada, tal como vimos en el capítulo anterior. Cada vector de test especificará (1) el valor de las entradas a la lógica combinacional (nótese que parte de ellas serán entradas externas - llamémoslas x_1, x_2, ... x_{k-y} - parte serán salidas de los elementos de memoria - llamémoslas y_1, y_2,....y_m -) y (2) los valores esperados en las salidas que, de una manera análoga, parte de ellas serán salidas externas (z_1, z_2, ... z_r) y parte serán entradas a los elementos de memorias (s_1, s_2, ... s_t).

2. Test de la parte secuencial :

   Pondremos el circuito en modo test (*Scan-Select*=1) y testearemos el registro de desplazamiento formado a través de su entrada SDI y su salida SDO. Existen diversos algoritmos diseñado específicamente para comprobar el funcionamiento de registros de desplazamiento en la literatura que no nos pararemos a explicar. Baste decir, sin embargo, que una forma de testear el registro de desplazamiento con una cobertura de fallos adecuada es (1) hacer un *reset* del registro, (2) a continuación pasar un 1 a lo largo de todo el registro, comprobando la salida, y finalmente (3) pasar la secuencia 001100110011.... etc.

3. Una vez hemos comprobado el buen funcionamiento de los elementos de memoria, atacaremos el test de la parte combinacional. Para cada vector de test:

   a) Pondremos el circuito en modo test (*Scan-Select*=1) y entraremos el vector de test (1) forzando los valores de (x_1, x_2, ... x_k) directamente en los pines de entrada y (2) entrando secuencialmente los valores de (y_1, y_2,....y_m) a través de *SDI*. Si el circuito tiene *n* elementos de memoria, esta operación nos llevará *2.n* ciclos de test.

Técnicas de scan-path

b) Pondremos el circuito en modo sistema (*Scan-Select*=0) y esperaremos a que, en el siguiente ciclo de test, las salidas de la parte combinacional se almacenen en los *flip-flops*. Las señales de error (1/0 o 0/1) estarán, o bien en una de las salidas externas (z_1, z_2, ... z_r), directamente detectable por la máquina de test, o bien en una de las salidas de la parte combinacional (s_1, s_2, ... s_t) y en tal caso se habrán almacenado en alguno de los elementos de memoria. En este último caso la señal de error todavía no es accesible a la máquina de test, así que necesitaremos un tercer paso en el que ...

c) Pondremos de nuevo el circuito en modo test (*Scan-Select*=1) y "vaciaremos" el registro de desplazamiento, haciendo visibles a la máquina de test las señales (s_1, s_2, ... s_t). Aquí de nuevo necesitaremos *2.n* ciclos de test. Si el conjunto de vectores de test consta de *v* vectores, el test completo de la parte combinacional requerirá :

$$v.(2.n+1) \quad ciclos\ de\ test$$

Como se puede ver, el principal inconveniente de estas técnicas es el elevado número de ciclos de test necesarios, que convierten el test en un proceso realmente largo a poco complejo que sea el circuito. Como ventaja primordial fijémonos que hemos reducido el test de la parte secuencial al test de un registro de desplazamiento.

Técnicas de scan-path

Ventajas del *scan-path*:

1. Disociación lógica combinacional-secuencial. Sólo es necesario generar vectores de test para la parte combinacional.
2. Permite la partición del circuito utilizando varios *scan-paths.*
3. Una forma sencilla de reducir el número de ciclos de test necesarios para comprobar la parte combinacional es distribuir los elementos de memoria en dos o más *scan-paths*. Si por ejemplo utilizamos 4 *scan-paths* de forma que distribuimos equitativamente el número de flip-flops por scan (*n/4* en cada uno de ellos), habremos reducido el número de ciclos necesarios para comprobar la parte combinacional de :

$$m.(2.n+1) \approx 2.m.n \qquad a \qquad m.(2.\frac{n}{4}+1) = m.(\frac{n}{2}+1) \approx \frac{m.n}{2}$$

como contrapartida, habremos pasado de necesitar 3 pines extras (*Scan-Select*, *Scan-Data-In*, *Scan-Data-Out*) a 9 (4 pines de *Scan-Data-In,* 4 pines de *Scan-Data-Outy* 1 pinde *Scan-Select*común a los 4 *scan-paths).*

4. Optimiza el uso de los simuladores de fallos y ATPGs. Tanto los APTGs como los simuladores pueden aplicarse por separado a las partes de lógica combinacional "aisladas" por los distintos *scan-paths* que utilicemos, aumentando su velocidad.
5. Cierta capacidad de diagnóstico.

Hasta ahora hemos hablado de la capacidad de testear el funcionamiento del circuito, pero una vez identificado la presencia de un fallo, ¿sabemos localizarlo (=diagnosticar donde está)?. La respuesta en general es NO, pero el *scan-path* nos

Técnicas de scan-path

permite al menos tener una cierta idea de por donde está el fallo. Al ir extrayendo los valores del registro de desplazamiento por *Scan-Data-Out* la máquina de test puede conocer en qué *flip-flop* estaba la señal de error, lo que nos proporciona pistas sobre la zona de lógica combinacional donde está el fallo.

Desventajas del scan-path

1. Necesita lógica adicional (elementos de memoria que ocupan más área por contener el multiplexor o circuito equivalente, más conexionado extra). En general se evalúa en un 5-15% el "*overhead*"de área debida a la inclusión de *scan-paths*.
2. Necesidad de pines adicional, el *Scan-Select* y tantos pines de *Scan-Data-In* y *Scan-Data-Out* como registros de desplazamiento hayamos incluido.
3. Se requiere un elevado número de ciclos de test (test lento).
4. El circuito no se teste a en sus condiciones de funcionamiento reales; fijémonos que (1) se carga el vector en el registro de desplazamiento, (2) se pone a funcionar el circuito en modo sistema durante un ciclo y (3) se vuelve a pasar a modo test para extraer el resultado.
5. En general, se degradan las prestaciones del circuito por la inclusión de lógica que (1) aumenta los retardos, (2) aumenta el consumo, y (3) aumenta el riesgo de malfuncionamiento.

Apesar de estas desventajas aparentemente tan serias, todos los ASICs de tamaño medio/elevado utilizan en algún momento *scan-paths.*

Bajo la idea del *scan-path* han aparecido en el mercado toda una serie de técnicas, con diferentes nombres, aunque basadas fundamentalmente en la construcción de registros de desplazamiento que permitan acceder a partes internas del circuito observando los resultados a posteriori.

Técnicas de scan-path

Ampliación

Así pues, encontramos como técnicas *scan-path*, (1) el ***scan-path*** propiamente dicho, establecido por NEC en 1975, que se basa en el biestable con multiplexación de entradas de Kobayashi y que es una variación de la metodología que establecieron Angell y Williams en 1973. En 1977, bajo el nombre de **Level Sensitive Scan Design (LSSD),** Eichelberger y Williams establecieron unas condiciones que deben seguirse y que aseguran la implementación del *scan-path* con elementos de memoria libres de azares y carreras.

Posteriormente aparecieron otras técnicas similares como el **Random Access Scan (RAS),** método que utiliza un direccionamiento de los biestables del circuito parecido al de las memorias RAM. De esta manera el RAS establece una forma de acceder y controlar los diferentes elementos de memoria interna que constituyen el circuito. El **scan/set** es otra técnica que significa un paso intermedio entre la los métodos ád-hoc y el scan-path propiamente dicho.

Random Access Scan (RAS)

El RAS es una herramienta ideada por Ando que, como el LSSD, quiere controlar y observar todos los elementos de memoria internos. Sin embargo, el método utilizado para conseguirlo varia sustancialmente respecto al LSSD. Así como el LSSD se basa en la construcción de registros de desplazamiento internos para conseguir llegar a cada uno de los puntos de memoria internos, Ando utiliza un direccionamiento similar al utilizado en memorias RAM para llegar a cada *latch* o *flip-flop* interno. Para esto utiliza el PHL direccionable (PHLD) que se muestra en la figura 4.

ver figura>>04

Técnicas de scan-path

En modo normal de operación el dato de entrada se almacena por un pulso negativo de la señal *CK*, pudiendo ser utilizada a través de la salida Q por el sistema. En modo test, el *latch* almacena el dato de *SDI* mediante el reloj *SCK*. Se puede observar que el reloj *SCK* solo afecta al *latch* direccionado a través de las líneas selectores de dirección *x* e *y*. De forma similar a cómo se realiza la entrada de datos en modo test, la salida en modo test selecciona un único *latch* que puede ser observado desde el exterior (direccionando convenientemente las líneas *x* e *y*). Esto permite que todas las salidas de test de los *latches* puedan ser conectadas a través de una única puerta AND para proporcionar una única salida de test. Cada *latch* direccionable puede ser seleccionado por medio de las líneas de direccionamiento *x* e *y* mediante decodificadores de dirección, de forma similar a cómo se realiza el direccionamiento en las memorias RAM. Los pines adicionales que se necesitan en el RAS son:

- Tres fijos: el *scan data in (SDI),* el *scan data out (SDO)* y el *scan clock (SCK).*
- Un número indeterminado de líneas variable según el número de *latches* a direccionar: corresponde a las líneas de direccionamiento *x* e *y*.

La lógica scan-set (scan-set logic)

El *scan/set* fue descrito por Stewart como una herramienta parcialmente estructurada, no tan rigurosa como las anteriores que sirven de ayuda en el test de circuitos secuenciales. Una clara diferencia respeto los técnicas anteriores es que el circuito original permanece prácticamente sin tocar y que se añade el registro *scan/set* "externo" para realizar las operaciones de ayuda el test.Tal como se ve en la figura 5 se puede decir que el *scan/set* permite introducir *n* datos serie hacia el registro que pueden controlar *n* puntos de control interno, o bien puede observar *n* nodos internos, que después de cargarlos en el registro scan/setson desplazados hacia el exterior.

ver figura>>05

Fin de la ampliación

BIST : *Built In Self Test*

En todas las técnicas de ayuda al test vistas hasta ahora es necesario aplicar los vectores de test desde una ATE ext~~erna~~. Las técnicas de *Built-In-Self-Test (BIST)* son unas técnicas de test no-concurrente que añaden una lógica extra al circuito capaz de (1) generar internamente los vectores de test, forzándolos sobre el circuito y (2) comprobar los valores obtenidos con los esperados, todo ello sin necesidad de equipo externo. Por lo tanto, las ventajas frente de las técnicas vistas previamente son que:

1. Se eliminan, o al menos se reducen, los costes debidos a la generación de vectores de test y la simulación de fallos,
2. El test de los circuitos se puede realizar a las velocidad de operación normal del circuito, y
3. Se simplifica el equipo externo necesario para comprobar los circuitos.

Como contrapartida,

1. Se necesita un espacio propio para la lógica adicional que ha de realizar la comprobación del circuito, siendo el incremento de área mayor que en el caso de las técnicas de *scan-path*.
2. No se puede acceder ni observar nodos internos determinados comoen el caso de las técnicas de *scan-path*.

Las técnicas BIST se basan en el concepto de **compactación del test**.

La figura 6 muestra un esquema que nos servirá para comprender este concepto: El ASIC final se compone de (1) el circuito propio cuya funcionalidad se desea conseguir (representado por el bloque “circuito” en la figura), (2) un generador de vectores de test que se encargará de ir aplicando uno a uno los vectores de test sobre el circuito y (3) un compactador de la secuencia de salida (representado por el bloque “Analizador de signatura”en la figura). Dependiendo de una señal de control, el “circuito”recibirá las entradas/salidas del exterior, funcionando tal como se desea (modo sistema), o recibirá las entradas desde el generador de vectores de test, lanzando las salidas hacia el módulo “analizador de signaturas” (funcionamiento en modo test).

BIST : *Built In Self Test*

ver figura>>06

El principal problema reside en que, si deseamos comprobar una a una las respuestas de nuestro circuito a cada uno de los vectores de test necesitaríamos una gran cantidad de memoria para almacenar las respuestas esperadas que nos ocuparía una gran parte del área total del circuito. La solución a este problema consiste en comprobar la respuesta del circuito a *todo* el conjunto de vectores de test en vez de comprobar las respuestas una a una. ¿Cómo?: las salidas del circuito (cuando está funcionando en modo test) se hacen entrar a una máquina de estados finitos que tiene un estado determinado que etiquetaremos como "*correcto"* (se puede definir más de un estado *correcto*), y el resto de estados que etiquetaremos como "*incorrectos*". Esta máquina de estados finitos, que no es más que el bloque "Analizador de signatura" se diseña de manera que, si el "circuito"está libre de fallos, al acabar de pasar por el circuito todos los vectores de test la máquina de estados finitos acaba en el estado *correcto* y, si el circuito presenta algún fallo, la máquina de estados finitos acaban en uno de los estados *incorrectos.* El estado (o estados) *correcto* recibe el nombre de **signatura del circuito** y la máquina de estados finitos, como ya hemos dicho, recibe el nombre de **analizador de signaturas**.

Generación interna de los vectores de test

La generación de los vectores de test se puede realizar de muchas maneras. De entre todas ellas, podemos distinguir tres como las más utilizadas:

Pre-almacenando los vectores.

Los vectores de test se generan tal como hemos explicado hasta ahora, y se almacenan previamente en un área local destinada a este fin. Generalmente, se suele utilizar como sistema de almacenaje una memoria ROM. En un primer paso, esta metodología necesita realizar la generación de los vectores de test que se han de aplicar al circuito, y posteriormente es necesario almacenar el conjunto de vectores en una ROM y disponer de un controlador (puede ser simplemente un contador) que lo aplique, vector a vector, al circuito a comprobar.

BIST : *Built In Self Test*

Test exhaustivo.

El test exhaustivo aplica los 2^N vectores de test posibles al circuito (donde N es el número de entradas del circuito). El generador puede ser, por lo tanto, cualquier módulo capaz de generar las 2^N combinaciones: un contador, un generador de código, ... El test exhaustivo, como dijimos en el capítulo anterior, es imposible a menos que se trate de un circuito trivial, pero lo que si es posible es aplicar el BIST a *partes* del circuito que si puedan testearse exhaustivamente.

Test pseudoaleatorio.

Se basa en la aplicación de un subconjunto de las 2^N combinaciones de entradas escogidos de una manera *pseudoaleatoria*. Por p*seudoaleatoria* se entiende una secuencia de combinaciones de los valores de entrada que, si bien una vez determinada se aplica siempre la misma secuencia, en su selección se han tenido en cuenta los criterios matemáticos de aleatoriedad.

Un generador de secuencias pseudoaleatorias es el **linear feedback shift register (LFSR).** No vamos a entrar en detalles por falta de espacio y tiempo, pero baste por el momento con saber que un LFSR (1) es un tipo de registro de desplazamiento que puede tener sólo puertas tipo XOR e inversores en las entradas de sus elementos de memoria, y (2) que conectando de una determinada manera estas puertas puede conseguirse que el LFSR actúe como generador de secuencias pseudoaleatorias o como analizador de signaturas [1].

Analizadores de respuestas

Los analizadores de respuesta (o analizadores de *signatura*) son sistemas que realizan una compresión de los resultados de la aplicación del patrón, obteniendo finalmente un residuo o "*signatura*" que se compara con el resultado del sistema libre de fallos. Algunos de los sistemas más utilizados como analizadores de respuesta son:

[1] El tema del uso de los LFSH queda fuera del alcance de este curso. Un libro muy adecuado (por lo básico) para consultar es el siguiente: Bardell PH, William H., McAnney, Savir J. "Built-in test for VLSI pseudorandom techniques". John Wiley 1987.

BIST : *Built In Self Test*

El comprobador de paridad.

La signatura de la secuencia de salida se define como la paridad de la misma; esto es, la signatura es 0 si el número de unos de la secuencia de salida es par, y 1 si dicho número es impar. Es un método poco eficiente desde el punto de vista de la detección de errores, pero muy fácil de implementar.

Analizador por conteo de 1s.

La signatura de la secuencia de salida se define como el número de unos de la misma; esto es, la signatura es *k* cuando el número de unos de la secuencia de salida es *k*. Es un sistema algo más eficiente que el primero que se utiliza en algunas ocasiones.

Compresión por transición de estados.

Se cuentan el número de transiciones de 0 a 1, o de 1 a 0 que aparecen en una secuencia de N bits. Un circuito compresor muy simple es un contador de al menos $log_2 N$ estados.

Ejemplo de un BIST

La figura 7 muestra un ejemplo de un ASIC autotestable. Supongamos que se ha diseñado un circuito que implementa unas ciertas funciones y al que llamaremos “Circuito F”. Dicho circuito tiene una serie de entradas y salidas externas que están convenientemente representadas en la figura.

ver figura>>07

Para hacer autotestable este circuito se ha añadido un módulo generador de vectores de test y un analizador de signaturas. El primero genera una secuencia pseudo-aleatoria de *n* vectores de test y los aplica directamente al circuito F. Como resultado, el circuito F genera una secuencia de *n* bits por cada una de sus salidas (en el ejemplo suponemos que hay 4 salidas que irían (1) al exterior y (2) a 4 analizadores de paridad). Un pin específico del ASIC (no representado en el dibujo) indicaría cuando éste está funcionando en modo sistema (es decir, implementando la funcionalidad para el que fue diseñado) o en modo test.

BIST : *Built In Self Test*

Cuando el ASIC está funcionando en modo sistema, el circuito F tiene en cuenta las entradas y salidas externas; cuando está funcionando en modo test, el circuito F sólo “hace caso” de las entradas que le llegan del generador de vectores y lanza sus salidas hacia los 4 analizadores de signatura.

Una vez pasada la secuencia de vectores de test completa se mira la signatura para ver si coincide con la esperada (cosa que se interpreta como que el circuito funciona libre de fallos) o no (cosa que se interpreta como presencia de fallos).

BILBO

Uno de los sistemas de test interno no-concurrente más populares que aplican las técnicas de generación-compresión de vectores de test es la técnica **BILBO (*Built-in Logic Block Observer*)**, de Köenemanny *et alt*.

En el BILBO se integran los conceptos del *scan-design* y de análisis de signatura utilizando un LFSR interno al circuito integrado que puede funcionar como analizador de signaturas y puede generar un conjunto de vectores de test de forma pseudoaleatoria. En la figura 8 se muestra la estructura de un registro BILBO de 4 bits.

ver figura>>08

El registro BILBO se comunica con el resto del circuito mediante las siguientes señales:

- S_{in} y S_{out}, que constituyen los pines de *scan-in* y *scan-out* del registro de 4 bits.
- B1 y B2, que permiten seleccionar 4 modos de funcionamiento del LFSR:
  - ✓ B1·B2=11 → Los elementos de memoria actúan como un conjunto de *latches*normales. Es el modo de funcionamiento normal.
  - ✓ B1·B2=00 → Funciona como registro de desplazamiento. Los datos entran en serie por S_{in} y salen por S_{out}. En este modo de funcionamiento el registro de desplazamiento actúa como un *scan-path*.
  - ✓ B1·B2=10 → El BILBO funciona como un LFSR de entrada paralela. Este LFSR de entrada paralela se puede utilizar como un analizador paralelo de signatura.

    Si se fijan los valores de $Z_1,Z_2,...,Z_8$ (por ejemplo todos a 0) el registro se convierte en un LFSR autónomo que genera una secuencia pseudoaleatoriade longitud máxima. La longitud del periodo depende del número de biestables y del polinomio característico.
  - ✓ B1·B2=01→ Realiza el *reset* del registro.

BILBO

La técnica BILBO funciona muy bien en el caso de circuitos muy modulares o con una fuerte estructura de bus. Normalmente se utilizan *latches* asíncronos que actúan como interfaz entre los módulos y los buses, substituyendo en estos casos los registros por registros BILBO de forma similar a como se muestra en la figura 9.

ver figura>>09

Cada módulo tiene dos registros BILBO. Uno se utiliza como generador del patrón y el otro como un analizador de signaturas en paralelo. Este último comprime la respuesta en una signatura que será comprobada fuera del circuito.

Los registros BLBO están conectados en forma de registro de desplazamiento que se pueden estructurar, ya sea como un generador de secuencias pseudoaleatoria o ya sea como un compresor. Seguidamente, el BILBO pasa a modo test (B1B2=10) donde cada circuito bajo test recibe una secuencia pseudoaleatoria de vectores de entrada y sus salidas permanecen almacenadas en el analizador de signaturas. Después de pasar un cierto número de vectores de test, el sistema vuelve a ponerse en modo registro de desplazamiento y las signaturas que quedan en el BILBO son extraídas en serie y comparadas con los valores correctos. Cada módulo del circuito se comprueba independientemente de los otros.

Como que el LFSR crea secuencias pseudoaleatorias, el sistema BILBO se adapta bien a todos aquellos circuitos susceptibles de aceptar vectores generados pseudoaleatoriamente. En general circuitos con fan-in grande, por ejemplo PLAs, no se suelen adaptar demasiado bien a sistemas BILBO ya que, por ejemplo, para detectar un s-a-0 de una AND de n entradas es necesario generar un 11...1. La probabilidad de detectar este error es de $1/2^n$.

Resumen : ¿Cómo aplicar las técnicas de DFT?

Resolver el problema de la testabilidad de un circuito VLSI es uno de los factores más difíciles con que se encuentran los diseñadores de hoy en día, debido principalmente al gran número de módulos y de funciones internas que componen el circuito.

Hoy por hoy, la única forma de conseguir mejorar la testabilidad de circuitos determinados es mediante el uso de las técnicas de DFT vistas. Sin embargo, no toda técnica es igualmente buena para aplicar a un circuito o en otro. Hay que tener presente que la mejor solución suele obtenerse mediante la aplicación de técnicas diferentes a cada uno de los diferentes módulos que componen el circuito. Así, por ejemplo cabe distinguir dos tipos importantes de circuitos:

- Circuitos o módulos compuestos básicamente por circuitería secuencial. Su construcción típica viene dada por la inclusión de biestables dentro de un montón de lógica aleatoria. Son módulos generalmente difíciles de comprobar, principalmente cuando forman parte de bloques internos del circuito solo observable a través del correcto funcionamiento de una decodificación de instrucciones del módulo. Para este tipo de módulos se suele usar una aproximación de *scan-path* conectado a todos o una parte de los elementos de memoria internos. De esta manera es posible acceder y observar cada uno de los módulos internos.
- Sistemas modulares con mucha entrada/salida, tales como ROMs, RAMs, PLAs,.... Suelen ser estructuras muy regulares y generalmente están construidas por repetición de unas pocas celdas unitarias básicas. En estos casos no resulta práctico utilizar una técnica de *scan-path* por el hecho de que necesita muchos vectores (y por lo tanto mucho tiempo) para conseguir aplicar un patrón determinado al módulo. Tampoco es práctico, en general, una técnica BIST, ya que probablemente la cobertura que se obtendrá será inferior a la deseada. Sin embargo, sí suelen aplicarse técnicas BIST “dedicadas”, las cuales cubren un espectro determinado de fallos (por ejemplo los fallos de *cross-point*).

Resumen : ¿Cómo aplicar las técnicas de DFT?

Hoy en día, son múltiples las referencias de la literatura que proponen la utilización de estas metodologías en muchos aspectos del diseño de circuitos integrados. Se ha pasado de intentar aplicar las técnicas de diseño para test de forma general a particularizar el problema para cada caso concreto y actuar en consecuencia.

Debido al hecho de el ASIC se compone habitualmente de múltiples módulos diferentes, es normal ver que para cada parte concreta del circuito se aplica una metodología concreta de DFT. Esto permite minimizar enormemente los costes, tanto en área de silicio como en tiempo de test. De esta manera, encontramos que las técnicas BIST son muy utilizadas en estructuras modulares altamente repetitivas y con mucha entrada/salida como es el caso de las PLAs, de las RAMs, etc., mientras que se considera fundamental el uso del *scan-path* a la hora de comprobar componentes altamente secuenciales así como en aquellos circuitos que tienen mucha lógica aleatoria interna.

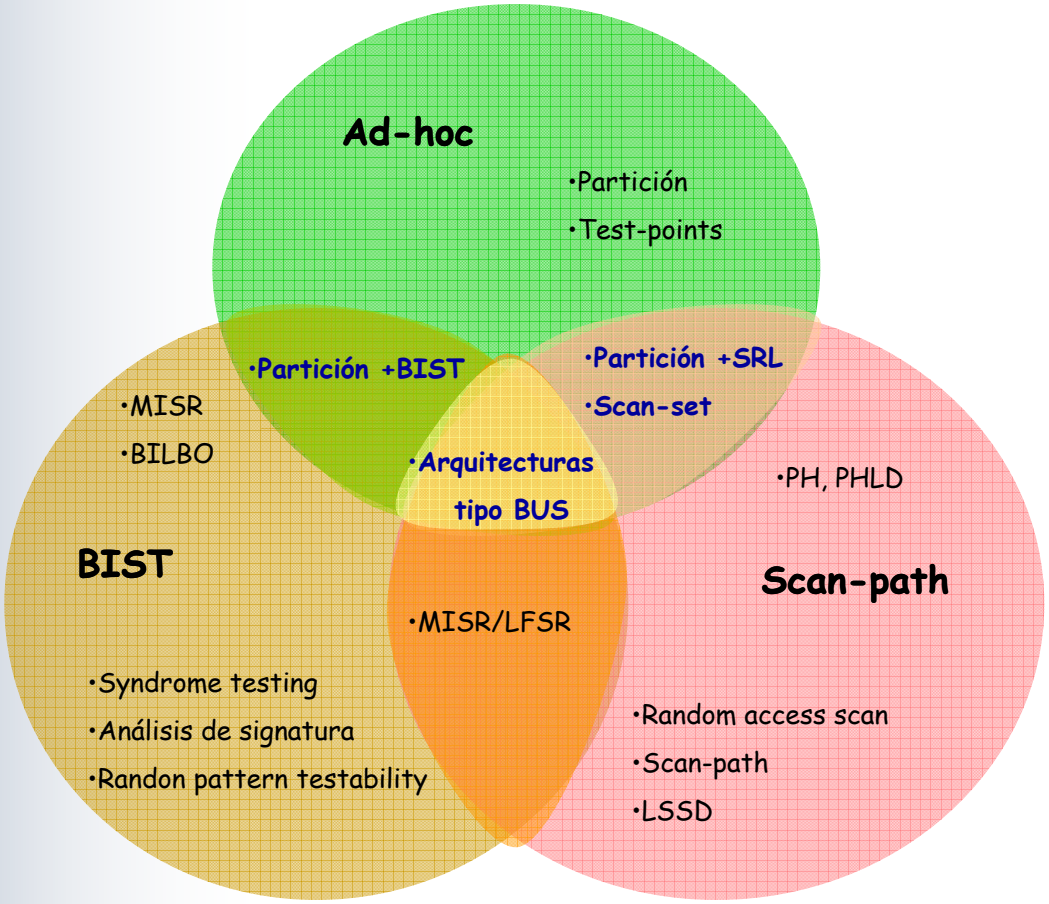
Sin embargo, no existen un límite claramente definido de cada una de estas técnicas. Ninguna de ellas, ni el *scan-path* ni el BIST dominan la una sobre la otra. En sistemas grandes, como puede ser el caso del diseño de microprocesadores, la estrategia de test se suele basar en la aplicación de una técnica DFT (la más conveniente en cada caso) para cada tipo de módulo que incluye el circuito.

Resumen

1. Vector o patrón de test
2. Fallo detectable y fallo indetectable
3. Testabilidad de un circuito
4. Fallos equivalente y dominantes (reducción de fallos)
5. Modelos de fallos
6. Cobertura de un conjunto de vectores (patrones) de test
7. Método de sensibilización de caminos
8. Generadores automáticos de vectores de test (ATPGs)
9. Simuladores de fallos.

Fin del capítulo 10

Figura 1



Técnicas de test clasificadas por los tipos ádhoc, scan-path (estructuradas) y BIST

Figura 2

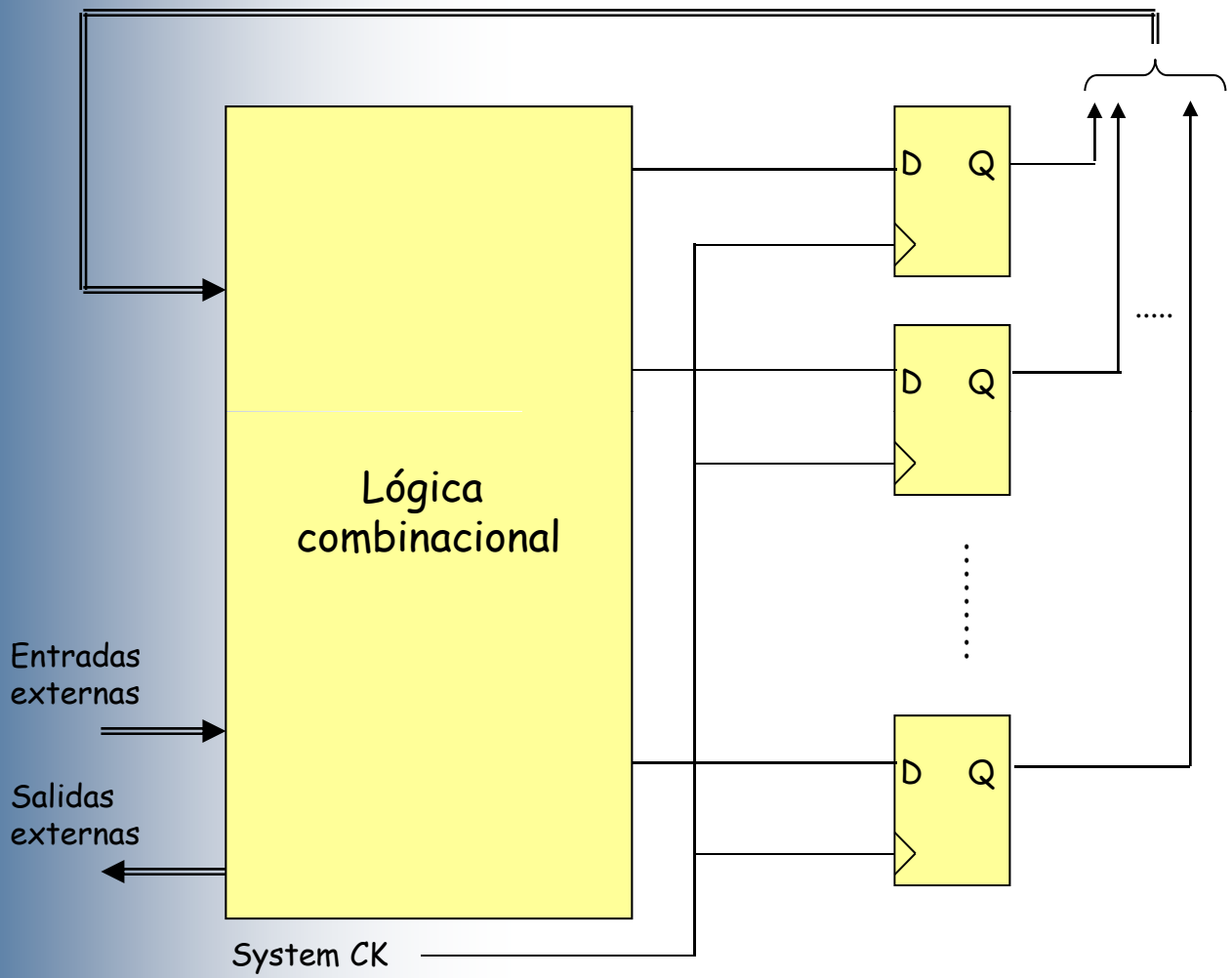

D
Q
D
Q
D
Q
Lógica
combinacional
Entradas
externas
Salidas
externas
System CK

Figura 3

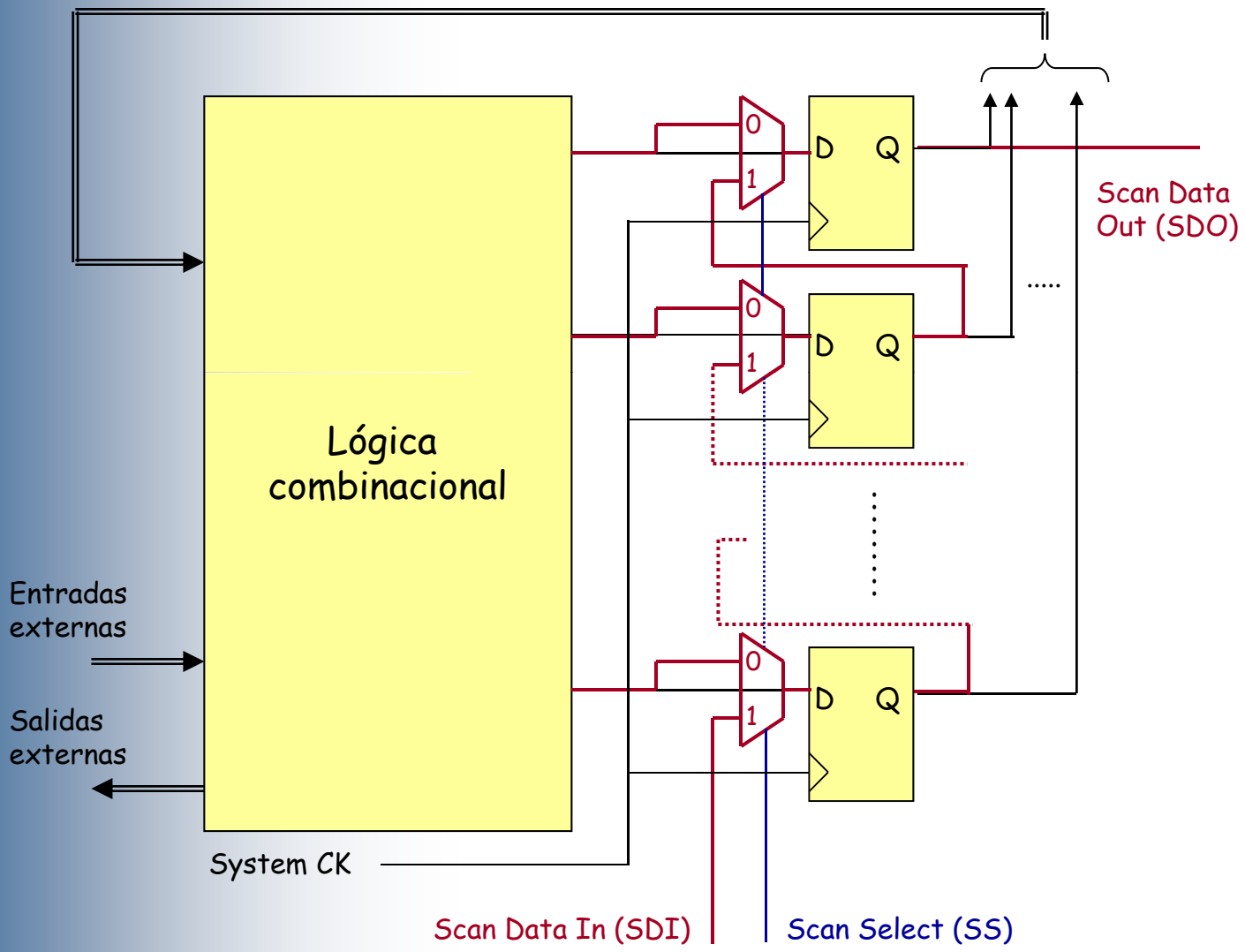
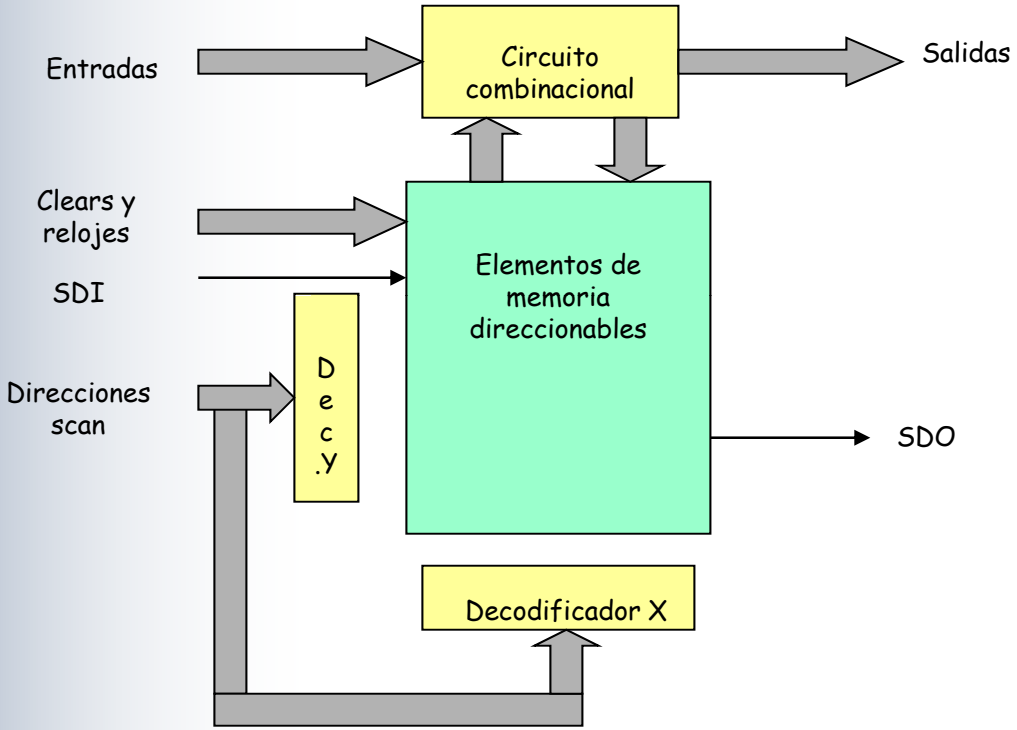
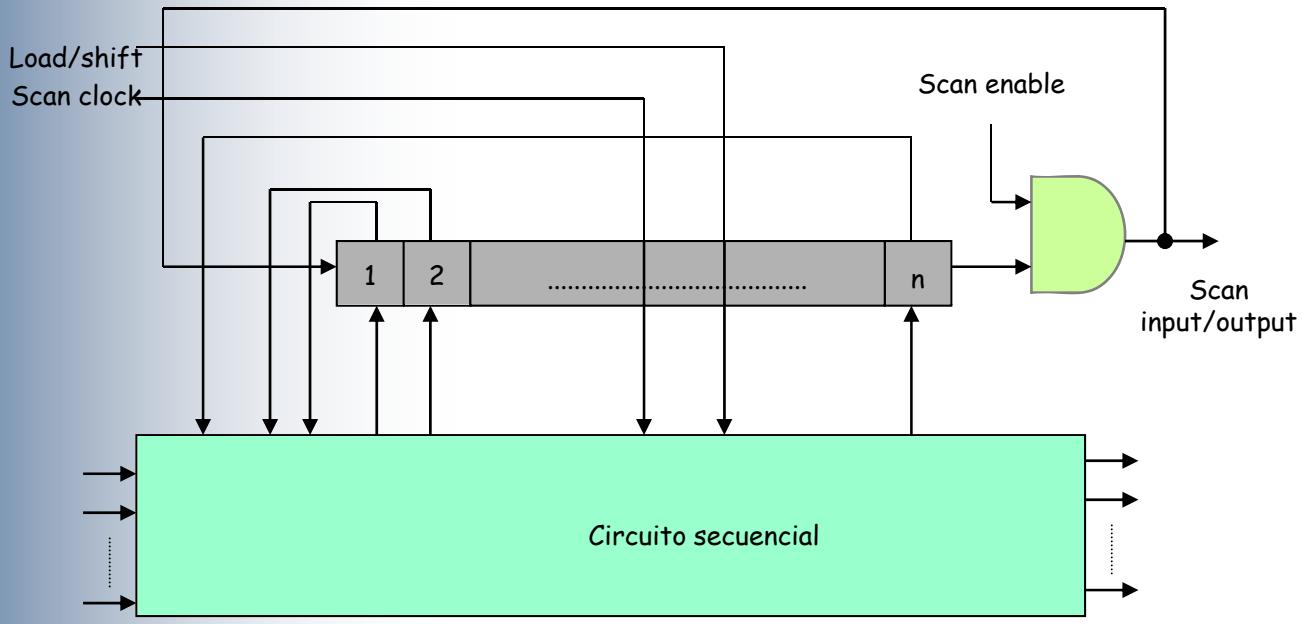

Lógica combinacional
Entradas externas
Salidas externas
System CK
0
1
D
Q
Scan Data Out (SDO)
Scan Data In (SDI)
Scan Select (SS)

Figura 4



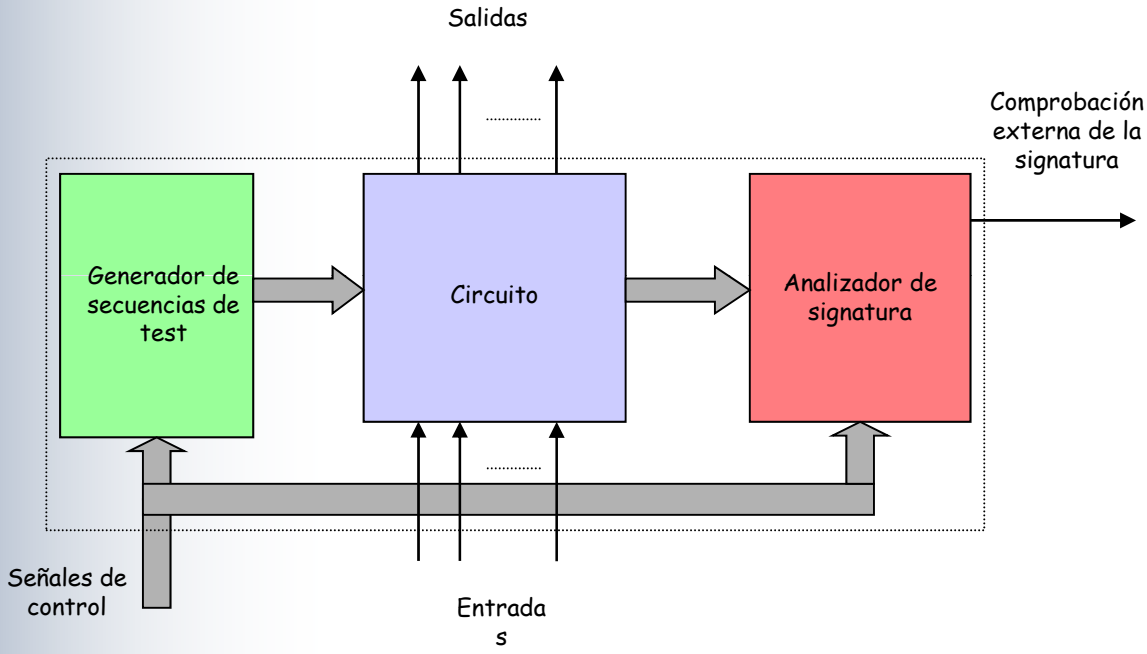
Random Access Scan (RAS)

Figura 5



Scan-Set Logic

Figura 6



Esquema de los elementos que se añaden para la comprobación interna del circuito bajo test.

Figura 7

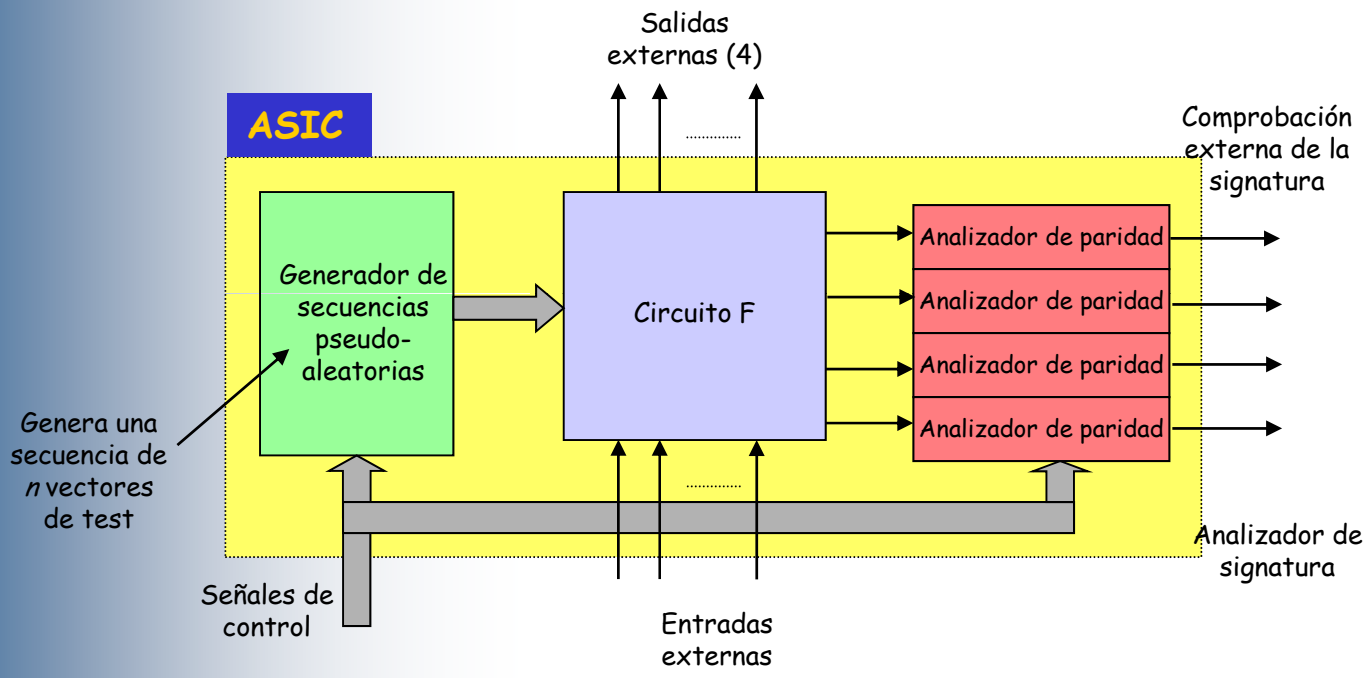

Salidas externas (4)
ASIC
Comprobación externa de la signatura
Generador de secuencias pseudo-aleatorias
Circuito F
Analizador de paridad
Analizador de paridad
Analizador de paridad
Analizador de paridad
Genera una secuencia de n vectores de test
Señales de control
Entradas externas
Analizador de signatura

Figura 8

Registro BILBO

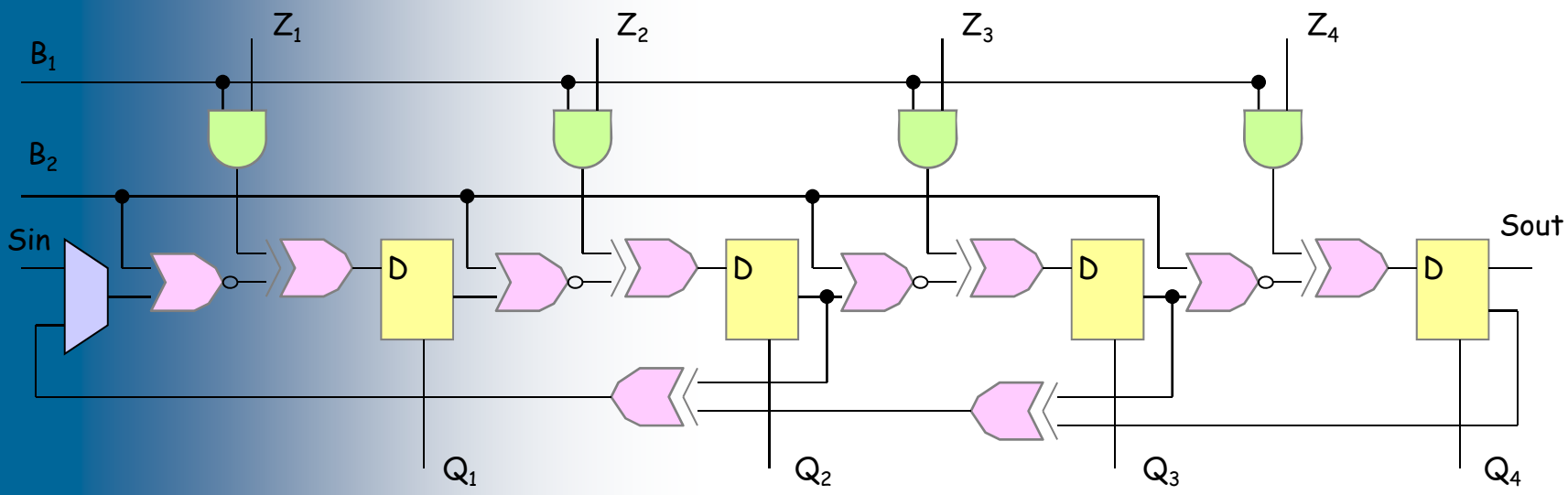
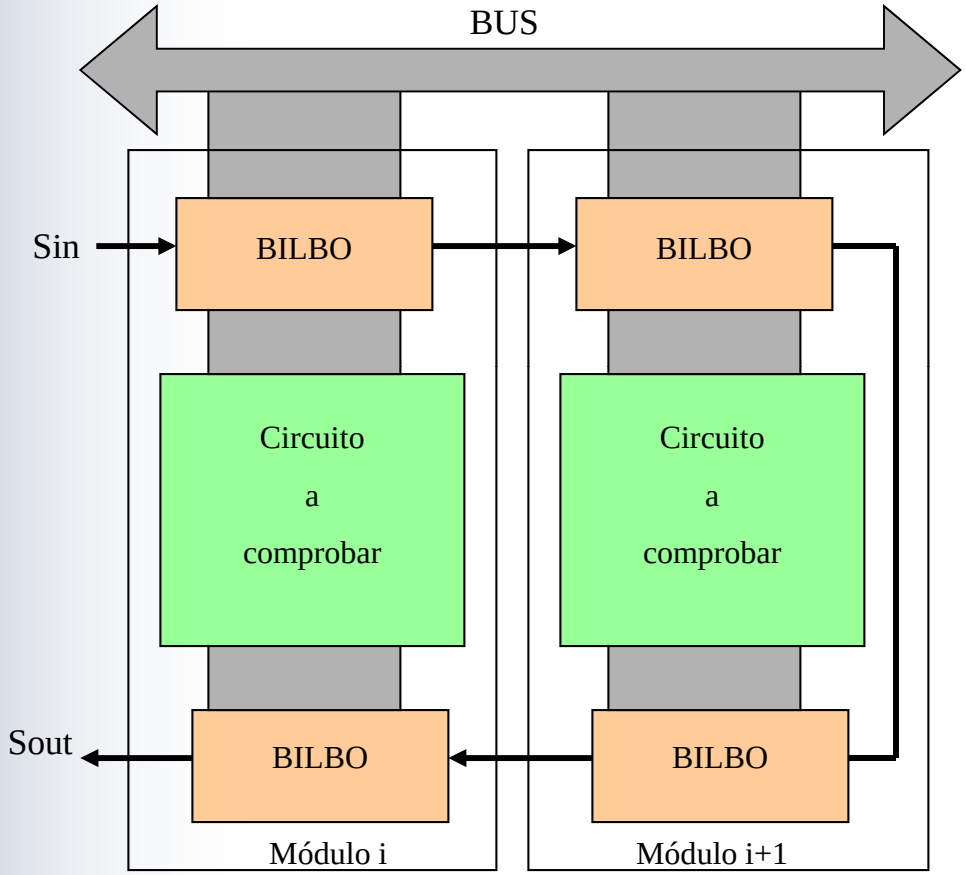


Figura 9


BUS
Sin
BILBO
BILBO
Circuito
a
comprobar
Circuito
a
comprobar
Sout
BILBO
BILBO
Módulo i
Módulo i+1

Fin del capítulo 10