Introducción al Diseño de CIs

Universitat Autònoma de Barcelona

Curso académico 2009-10

Elena Valderrama

Carles Ferrer

Capítulo 9 : Conceptos Básicos del Test de CIs

Introducción

En los últimos años se ha asistido a un crecimiento espectacular en la complejidad de los circuitos integrados, como consecuencia tanto de las mejoras tecnológicas como de los avances en las herramientas de CAD ("Computer Aided-Design"), que han permitido alcanzar un aumento en la escala de integración debido a la reducción en las dimensiones del transistor. Por otro lado, el desarrollo de mejores herramientas de ayuda al diseño ha traído consigo la integración de grandes sistemas VLSI (*Very Large Scale Integration*) y ULSI (*Ultra Large*), necesitando mucho menos tiempo para su diseño.

No obstante, no basta con diseñar y construir un determinado circuito integrado sino que además se precisa que funcione de acuerdo con unas prestaciones preestablecidas desde su concepción. Como consecuencia, el circuito una vez fabricado debe ser sometido a una serie de comprobaciones para garantizar su buen funcionamiento, es decir lo que constituye la fase de verificación o test.

La mayor parte de las veces el coste de comprobación de circuitos VLSI-ULSI es netamente superior a cualquier otra fase de su fabricación, como así refleja la tabla de la figura 1 donde aparece el incremento del coste del test. En la tabla de la figura 1 se puede observar como las dimensiones de los circuitos se han ido reduciendo drásticamente en los últimos años. De acuerdo con esto, el aumento en la densidad de integración ha comportado la duplicación en promedio de la complejidad de los circuitos cada año, cambiando la escala de integración cada cinco.

figura>>01

Este aumento de la complejidad de los circuitos no ha ido acompañada por un aumento similar en el número de entradas /salidas, lo que ha provocado partes clave del sistema, antes fácilmente accesibles, ahora queden inmersas en el interior del circuito. Finalmente, el incremento en la frecuencia de funcionamiento de los circuitos dificulta aun más si cabe su comprobación ya que esto implica la necesidad de desarrollar nuevos sistemas de test que se adapten a este cambio tecnológico.

El coste total de **producción** de un sistema comprende básicamente (1) los costes de diseño, fabricación y encapsulado de los circuitos, (2) los costes de montaje sobre placas

Introducción

de circuito impreso y (3) los costes de verificación tanto de los circuitos integrados como de las tarjetas (placas y circuitos) como del sistema completo. Mientras que el desarrollo de herramientas CAD y el aumento de la densidad de integración han disminuido los costes de diseño (número de puertas diseñadas por hombre*año), fabricación, encapsulado y montaje en placas (menos chips a ubicar), los costes de verificación y test son los únicos que han aumentado, representando cada vez un porcentaje más elevado del coste total del sistema.

El coste de verificación se multiplica por un factor 10 al pasar de una etapa de producción a la siguiente; es decir, un error, cuya detección lleve implícito un coste *c* en una etapa *t* (nivel de CI, p.e.), contribuye en un coste *10*c* al coste total del sistema si se detecta en un nivel *(t+1)* (nivel de placa, p.e.). En consecuencia, el objetivo del test debe ser detectar los posibles defectos ***cuanto antes mejor***, en las etapas más tempranas de la producción del sistema. Sólo así se consigue reducir costes y mejorar la fiabilidad de los productos fabricados por la empresa y por tanto su imagen.

Conceptos básicos

Fases del test.

El ASIC debe ir comprobándose a lo largo de todo el proceso de diseño, fabricación e incluso durante su vida activa.

A grandes rasgos, las comprobaciones se pueden subdividir en tres fases:

1- Durante el diseño del circuito:

Desde el inicio de la concepción del circuito integrado éstos pasan por una etapa de verificación para garantizar las características que habrán de cumplir durante su funcionamiento.

Para ello se utilizan un conjunto de herramientas CAD como son: simuladores a diferentes niveles (lógicos, eléctricos, híbridos, ...), verificadores de reglas de diseño, extractores de parámetros, etc.. El diseñador ha de asegurar, en todo momento, que el circuito cumple con las características especificadas en la fase inicial de concepción.

2- Fabricación del circuito:

Una vez fabricada cada oblea, ésta se somete a diferentes tipos de comprobaciones que aseguren la correcta realización de los procesos tecnológicos. Se miden ciertos parámetros eléctricos (conductividades, movilidades, ...) sobre una circuitos o "motivos de test", especialmente concebidos y incorporados en la oblea, fuera de los propios circuitos. De esta forma se asegura un comportamiento similar entre circuitos fabricados en diferentes obleas y/o en zonas diferentes de una misma oblea.

A continuación y sobre cada oblea se realiza una comprobación dado a dado del circuito. Cada uno de ellos se comprueba funcionalmente desde los "*pads*" mediante el uso de una **mesa de puntas** que conecta los *pads* del circuito con el sistema de test. Los dados que no pasan el test se marcan con tinta y se desechan después de que la oblea haya sido cortada y los dados (ASICs) separados.

Los dados que si han pasado las pruebas anteriores se encapsulan individualmente;

Conceptos básicos

pero el mismo proceso de encapsulado puede introducir nuevos defectos. Antes de su aceptación como buenos, los chips ya encapsulados son sometidos a una última comprobación unitaria con la ayuda de un sistema automático de test (***ATE: Automatic Test Equipment***).

3- Durante la vida activa del circuito:

Una vez comprobados los circuitos integrados, éstos se montan sobre las placas de circuito impreso (PCBs). Estas placas se testean con sistemas especialmente concebidos para ello, introduciendo señales en determinados puntos de la placa y comparando el estado lógico a que se encuentran los puntos de test.

Finalmente, estas placas o tarjetas formarán partes de un sistema, el cual debe ser revisado antes de poder dar el visto bueno para su comercialización. Asimismo cabe señalar la necesidad de realizar comprobaciones periódicas de cualquier sistema para garantizar su funcionamiento de acuerdo a unas prestaciones predeterminadas.

Test concurrente y no concurrente.

En cierto tipo de entornos en los que se prive la seguridad (espacio, medicina, ...), no se puede permitir que el sistema funcione mal en ningún momento por las consecuencias catastróficas que provocaría. En estos casos, los circuitos integrados pueden incorporar técnicas de **autocomprobabilidad y tolerancia de fallos** que permiten detectar la presencia de funcionamientos incorrectos a la vez que permiten (hasta un cierto nivel) subsanarlos sin alterar las prestaciones del equipo.

Dentro de estas técnicas existen dos aproximaciones, con niveles de dificultad crecientes:

(1) Se incluye en el CI circuitería adicional que permite testear el mismo sin necesidad de una ATE externo. El chip posee dos formas de funcionamiento, en modo sistema (el circuito funciona normalmente) y en modo test (el circuito se auto-comprueba), controlados por una señal externa que le indica el modo de funcionamiento en cada instante. Para testear el circuito es necesario “sacarlo del modo sistema” y ponerlo en modo test => este tipo de funcionamiento recibe el nombre de **test no concurrente o test “off-line”.**

Conceptos básicos

(2) Se incluye en el CI circuitería adicional que permite que los resultados generados por el sistema se vayan comprobando mientras se están obteniendo, iniciándose acciones determinadas (p.e., dando una señal de aviso) cuando el resultado no coincide con el esperado. Este tipo de comprobación reciben el nombre de **test no concurrente o test "off-line".**

Aunque esto induciría a pensar que con estos tipos de test no es necesario el uso de los ATEs, la pregunta es .... la circuitería adicional testea al "circuito-propio", pero ¿quién testea esta circuitería adicional?. La aproximación más frecuente es realizar de todas maneras un primer test convencional de ASIC utilizando un ATE, antes de ser montado en el equipo, y dejar que la circuitería adicional compruebe el funcionamiento de éste durante su vida activa. Evidentemente, la circuitería adicional aumenta el área del circuito y por ende si coste.

Test de prototipos vs. test industrial

Una vez el chip ha salido de la fábrica y ha llegado a nuestras manos, nos encontramos con dos situaciones diferentes:

1- Test de prototipos:

Este tipo de test viene caracterizado por los siguientes aspecto

- Hay que testear pocas unidades, lo cual implica que el tiempo que se dedica al test de cada muestra no será un factor crítico.
- Normalmente se tiene más interés en realizar una caracterización del circuito que una simple comprobación de su funcionamiento. Dicho de otra manera, se pretende comprobar (i) si funciona el circuito o no funciona, (ii) ¿por qué?; ¿dónde está el error?, y (iii) en caso de que funcione, hasta que límites.
- Finalmente hasta es permisible que el test sea destructivo si con ello se consiga extraer información adicional sobre el funcionamiento correcto o incorrecto del circuito.

Conceptos básicos

Si el test de los prototipos da resultados satisfactorios, entonces se puede empezar la fabricación de la serie. Por otra parte el diseñador o el fabricante, a partir de la información obtenida de los prototipos debe elaborar un catálogo de especificaciones que han de cumplir obligatoriamente todos los circuitos que se fabriquen.

2- Test industrial:

El test de la serie por parte de la industria recibe diferentes nombres como el de “Test industrial” o “Test de entrada”. Aunque el fabricante garantiza que los chips de la serie se han testeado, se considera una buena práctica realizar de nuevo un test a la llegada de los ASICs a la industria que los ha diseñado por dos razones (1) para comprobar que el fabricante cumple el contrato y (2) para minimizar el riesgo de montar en los equipos chips defectuosos, con las consecuentes pérdidas de imagen o económicas cuando así sucede. Muy frecuentemente se realizan test aleatorios sobre un cierto porcentaje de las unidades recibidas en vez de pasar el test a todas las unidades.

De nuevo, éste tipo de comprobaciones se caracteriza por:

- Hay que testear muchas unidades (100.000, 500.000, ...), lo cual implica que el tiempo de test es muy crítico y tendrá una repercusión clara en el coste final del producto.
- Habitualmente es suficiente con una respuesta binaria "go / no go" (el circuito funciona o no funciona). En caso de que no funcione el circuito es rechazado.
- Por razones obvias el test nunca debe ser destructivo.

Solamente los circuitos que pasan todas las comprobaciones se utilizan en la construcción de los sistemas electrónicos. De esta manera se reduce notablemente la aparición de errores en fases más avanzadas de la fabricación de un equipo provocadas por un funcionamiento erróneo de alguno de sus componentes, y se reduce el coste implícito que introducen estos errores (reparaciones, equipos desechados, etc.); a la vez que se mejora la imagen de calidad de la empresa que fabrica estos equipos.

Sistemas de test de CIs

El objetivo del "test" de CIs digitales es comprobar su correcto funcionamiento. Ahora bien, entrando un poco más en detalle se puede decir que el objetivo del test de CIs es verificar que:

1- El comportamiento lógico del circuito sea el esperado; esto es, que los valores lógicos obtenidos a la salida para cada secuencia de entrada, coincidan con los especificados (*test de funcionamiento*)..

2- Las salidas alcanzan los valores de tensión e intensidad previstos en las hojas de especificaciones (*test estático o test DC*).

3- El comportamiento dinámico (p.e. tiempos de subida, de bajada y de propagación) se encuentran dentro de los márgenes previstos (*test dinámico o test AC*).

¿Cómo podemos comprobar el circuito?

Las comprobaciones se puede realizar de muchas maneras. La más simple que se nos puede ocurrir se basa en disponer de un sistema formado por un conjunto de instrumentos que permitan forzar valores concretos de tensión e intensidad a las entradas del CI, (pueden ser fuentes de alimentación, generadores de formas de ondas, etc.) y medir los valores obtenidos a la salida (tester, osciloscopio, etc.).

Evidentemente esta es una aproximación que sólo se puede realizar en circuitos pequeños. Cuanto mayor sea la complejidad del circuito, más larga y tediosa será su comprobación, debido a que el número de entradas y de salidas será mucho mayor (esto requiere un mayor número de instrumentos) y por otro lado el número de combinaciones de entrada que habrá que probar será también mayor.

Actualmente los CIs de aplicación industrial suelen involucrar a grandes series (por encima de las 5.000 unidades), lo que hace imprescindible un cierto grado de automatización del test y el uso de sistemas automáticos de test o ATEs (*Automatic Test Equipment*).

ver figura>>02

Sistemas de test de CIs

Sistemas automáticos de test (descripción básica)

Un sistema automático de test está concebido de forma que permita automatizar los tres tipos de comprobaciones básicas descritas anteriormente. Estos sistemas están constituidos por (1) una mesa de test donde se coloca el chip a testear o "DUT" (*Device Under Test*) y que contiene una serie de recursos de test (generadores de estímulos, drivers, comparadores, ...), (2) un controlador (habitualmente un procesador) encargado de gobernar estos recursos para poder establecer las condiciones bajo las que se realizarán las comprobaciones, y (3) una cierta cantidad de memoria (discos, ...) donde almacenar los patrones de test que se pasarán al circuito.

ver figura >> 03

Funciones básicas de un ATE

El sistema de test debe realizar las siguientes funciones:

1- Almacenar los patrones de test o "*patterns*" (conjuntos de vectores de test a aplicar al circuito). Cada vector lleva información de los valores que se han de forzar a las entradas y la respuesta esperada en las salidas. Según la categoría de la máquina se incluyen ayudas más o menos potentes para la generación y edición de las secuencias de test.

Definición_1 : Un **patrón de entrada** (también llamado vector de test) consiste en el conjunto de valores que se fuerzan en los pines de entrada del circuito y el conjunto de valores que se espera que salgan por los pines de salida si el circuito funciona bien, en un ciclo de reloj concreto. Un patrón de test tiene el siguiente aspecto (en azul) :

	Entradas	*Salidas*
	x1 x2 x3 x4 ..... xn	y1 y2 y3 …….. yk
valores	**0 1 1 1 ..... 0**	**0 0 1 ..... 1**

Sistemas de test de CIs

Definición_2 : Un **conjunto de patrones de test** (o vectores de test) es el conjunto de patrones de test como los definidos anteriormente que se pasan secuencialmente al circuito para verificar su funcionamiento. Se dice que un circuito pasa el conjunto de patrones de test cuando, para cada patrón de test aplicado, las salidas que se obtiene coinciden con las salidas especificadas en el patrón. Un conjunto de patrones de test tiene el siguiente aspecto:

	Entradas	*Salidas*
	x1 x2 x3 x4 ..... xn	y1 y2 y3 …….. yk
patrón 1	0 0 1 1 0	0 0 1 1
patrón 2	1 0 1 0 0	1 0 1 0
patrón 3	0 1 0 0 1	0 0 0 1
…….		
patrón r	0 1 0 1 0	0 1 1 0

Suministrar las señales a las entradas del circuito (generar los estímulos). Para cada entrada o entrada/salida del circuito se conecta uno o varios "*drivers*" encargados de forzar la forma de onda correspondiente. El sistema de test es capaz de generar los estímulos a partir de la secuencia de vectores de test de entrada y los valores temporales programados.

Reconocer las salidas y compararlas con las respuestas esperadas obtenidas a partir de modelos funcionales o por simulación del circuito.

El sistema de test compara los valores obtenidos para cada una de las salidas con su valor previsto. La comparación se realiza gracias a unos comparadores asociados a cada señal de salida que son capaces de (1) convertir el nivel de tensión en una señal digital y (2) compara si ese 0 o 1 lógico coincide con el valor esperado.

La figura 4 muestra el sistema driver/comparador (D/C) asociado a cada pin del circuito. Para cada canal (pin del ASIC) existe una pareja D/C que se conecta de la siguiente manera:

Sistemas de test de CIs

- Pines de entrada : se conecta sólo el driver.
- Pines de salida : se conecta sólo el comparador.
- Pines de entrada/salida : se conectan ambos.

ver figura >> 04

Test funcional

El comportamiento lógico se comprueba aplicando un patrón de test a las entradas del circuito. Para ello, se generan los estímulos de entrada mediante un *driver* de entrada (figura 5), el cual para cada ciclo de test fuerza sobre la entrada un nivel de tensión en función del estado lógico que se desea.

ver figura >> 05

Las máquinas de test más sencillas dividen todo el tiempo de test en ciclos (definidos por el reloj de la propia máquina) y son capaces sólo de forzar un nivel 0 o 1 estable sobre cada una de las entradas o E/S del circuito. Las máquinas más complejas (y por supuesto más caras) permiten definir formatos; esto es, señales capaces de cambiar de 0-1 o 1-0 durante un mismo ciclo. Las más usuales de estas señales reciben los nombres especiales como los que aparecen en la figura 6: RZ (retorno a cero), Rl (retorno a uno), RH (retorno a la inhibición), NRZ(no retorno a cero) y DNRZ (no retorno a cero desplazado).

ver figura >> 06

Sistemas de test de CIs

Todos estos formatos vienen caracterizados por dos parámetros: *d* (*delay*) que es el retardo respecto al origen de ciclo (llamado *frontera de ciclo*) y *w* (*witdh*) que es la anchura de pulso. Estos valores pueden ser iguales o diferentes para cada ciclo de test, en función de la máquina de test.

Por otro lado, la comparación de las salidas con la respuesta esperada se hace mediante una circuitería que consta de (1) un comparador analógico, (2) circuitería de comparación (principalmente XORs) y (3) un biestable de error, tal como muestra la figura 7.

ver figura >> 07

Nótese que el comparador sólo comprueba si la señal de salida está por encima o por debajo de una cierta señal de referencia. Si se desea más precisión se puede utilizar el circuito de la figura 8, que define una ventana de comprobación especificando el intervalo de tiempo en el cual la salida debe ser estable (*w* en la figura 8) y los valores de referencia del 0 y del 1 lógico (*Vh* y *Vl* de la figura 9). Durante el intervalo de la ventana, el comparador mira si la tensión de salida está por encima de *Vh (=1-lógico)* o por debajo de *Vl. (=0-lógico),* y la compara con el valor esperado. Cualquier "pico" que apareciera en la ventana de test se interpreta como una salida errónea (figura 9).

ver figura >> 08 ver figura >> 09

Test de parámetros estáticos (Test DC)

Para este tipo de test se conecta una circuitería adicional a la patilla correspondiente donde se quiere efectuar la medida del parámetro. (Esta tarea es más propia del "ingeniero de test" que del diseñador, con lo que no vamos a insistir por salirse un poco de los objetivos de un capítulo que sólo puede introducir los conceptos más básicos).

- Medida de tensión, con la corriente de salida fija (figura 10). ver figura >>10

- Medida de corriente, con la tensión fija (figura 11).

ver figura >> 11

Sistemas de test de CIs

Test de parámetros dinámicos (Test AC)

La medida de tiempos de propagación se realiza de la siguiente forma: (supongamos que cuando la señal IN pasa a 1, la señal OUT pasa a 0, y que nos interesa medir el tiempo de propagación del estado alto al bajo de la señal OUT con respecto a IN). Se hace lo siguiente:

- Se define un nivel de referencia en los comparadores asociados a las señales IN y OUT (Vc en la figura 12).
- El tiempo de propagación t2-t1, se mide cargando parcialmente un condensador y midiendo la tensión a la que este llega (figura 12).

ver figura >> 12

Cuando A=1, S1 se cierra, y la intensidad I comienza a cargar e condensador. Cuando B1, S2 se cierra, e I desaparece con lo que el condensador no recibe ya más carga. La tensión alcanzada en el punto X es proporcional a t2-t1.

Del mismo modo, pero variando los niveles de comparación y las señales sobre las que se aplica, se puede realizar cualquier medida tanto de tiempos de propagación como de subida y de bajada.

Integración diseño-test

Clásicamente la tarea de verificación de circuitos integrados se relegaba a la comprobación de prototipos una vez fabricados, sin ninguna relación con las fases previas de diseño del circuito. Ahora bien, esta forma de trabajar era aceptable cuando la escala de integración no había llegado a los valores actuales (ver figura 01).

Como ya dijimos, la complejidad de los circuitos y el número de entradas y de salidas no han ido aumentando al mismo ritmo; mientras que el primer factor ha aumentado geométricamente, el segundo solo lo ha hecho de forma lineal. Esto ha traído consigo una notable reducción de la testabilidad del circuito (medida en una disminución de la controlabilidad y de la observabilidad).

Definiciones 3 : Para poder testar bien un circuito se requiere que (1) podamos forzar fácilmente los valores que deseemos sobre cualquier nodo interno del circuito, y (2) que podamos observar fácilmente los valores que toman cada uno de los nodos internos del circuito para comprobar si se trata de un valor correcto o erróneo.

Se llama **controlabilidad** del circuito a la facilidad con la que podemos forzar valores en cada uno de los nodos internos del circuito.

Se llama **observabilidad** del circuito a la facilidad con la que podemos ver desde el exterior los valores lógicos que toman cada uno de los nodos internos del circuito.

Un circuito se dice que es fácilmente testeable, o que tiene una **testabilidad** alta cuando es un circuito de alta controlabilidad y de alta observabilidad.

Sólo se puede conseguir una buena testabilidad si la estrategia del test se define y se tiene en cuenta desde las primeras etapas de diseño del ASIC; desde la misma fase de concepción de un CI si fuera preciso.

Así pues, el diseñador se tendrá que responsabilizar de la elaboración del conjunto de patrones de test a partir, normalmente, de los resultados obtenidos en la simulación del circuito. A diferencia de lo que ocurre con una simulación lógica o eléctrica, la simulación de los patrones de test tendrá de cumplir con todos y cada uno de los requerimientos propios de la máquina de test (frecuencia del CK de test, señales cumpliendo los formatos que permita la máquina, etc.)

Integración diseño-test

¿Cómo se testea un circuito?

Una manera conceptualmente fácil de testear un circuito consiste en aplicar todas las posibles combinaciones de entradas y de salidas y medir los valores de salida que se obtienen (es lo que llamamos un test exhaustivo: testear en un 100% el circuito). Pero si pensamos en algo tan simple como un multiplicador de 32 bits, veremos que un test exhaustivo requeriría comprobar al menos 264 combinaciones de entradas; suponiendo que aplicáramos un patrón de test por ns., el test nos llevaría del orden de 1 millón de días, o lo que es lo mismo *más de 500 años*.

¡Y el problema se complica si el circuito es secuencial!. Resulta evidente que el test exhaustivo de los chips es totalmente imposible cuando se trata de chips LSI o VLSI.

A lo largo de este tema veremos como se ha intentado y se intenta resolver este problema (test funcional vs. test estructural, generadores pseudo aleatorios, generadores automáticos de secuencias de test, etc.), pero una consecuencia parece clara: si se quiere llegar a circuitos fácilmente testable o incluso autotestables, es necesario diseñar “pensando en el test”.

El segundo problema es el de la precisión y rapidez necesarias en la máquina de test. El problema puede resumirse en el siguiente razonamiento: Para testar un chip cuyos componentes tengan un tiempo de respuesta de (por ejemplo) 1ns, es necesario que la precisión del ATE sea inferior al nanosegundo. Si el chip está fabricado con una tecnología CMOS, la máquina de test podría utilizar elementos bipolares más rápidos, pero ...¿Cómo testamos un chip realizado en la tecnología más rápida del momento?. Una posible respuesta es prescindir de la máquina de test, y utilizar técnicas de autotestabilidad.

En lo que resta de este capítulo nos centraremos en el problema de cómo enfocar la generación del conjunto de patrones de test, mientras que en el capítulo siguiente daremos unas breves nociones de lo que se conoce con el nombre de Diseño para la Testabilidad.

Test funcional frente a test estructural

Como ya se dijo, una forma de testar un circuito es aplicar todas las posibles combinaciones de entrada, y verificar que las salidas son las deseadas. Como también se dijo, este tipo de test exhaustivo de la función lógica que el circuito debe realizar sólo es posible en circuitos de tamaño pequeño (SSI) o medio (MSI).

Otra alternativa es la de tratar de asegurar que la estructura del circuito lógico no ha sido alterada durante el proceso de fabricación o durante la vida activa del circuito, y que coincide con la especificada en la fase de diseño. Se supone entonces que la correspondencia estructura-función lógica es competencia del diseñador. El objetivo del test estructural es asegurar que la estructura (layout) suministrada por el diseñador ha sido materializada correctamente y está libre de fallos. El problema de la generación de las secuencias de test se centra en seleccionar un conjunto de vectores capaces de detectar el mayor número posible (idealmente, todos) de estos fallos supuestamente introducidos durante la fabricación y encapsulado del ASIC.

Hemos de ir con cuidado con las definiciones imprecisas; la frase "detectar el mayor número posible de fallos" es poco específica. En primer lugar, habrá que explicitar cuales son los fallos que pueden presentarse en un circuito dado... que, de acuerdo con la ley de Murphy, son simplemente TODOS los imaginables. Sin embargo, algunos de los posibles fallos serán más probables que otros. Parece razonable restringir los esfuerzos a la detección de éstos fallos más probables en vez de intentar detectar todos los fallos.

DEFINICIÓN

Un **MODELO de FALLOS** es simplemente un conjunto de fallos seleccionados como los más importantes para el circuito, de modo que cuando se dice que

"estamos trabajando con el modelo de fallos X" estamos significando que *los únicos fallos* que vamos a tener en cuenta son los especificados en X. La definición del modelo de fallos con el que se va a trabajar dependerá, entre otras cosas, del nivel al que nos movamos (lógico, eléctrico...), de la tecnología en la que se trabaje, y de la seguridad que se desee alcanzar con el test.

Test funcional frente a test estructural

En segundo lugar, hay que concretar un poco más lo que quiere decir "el mayor número posible" en la frase a la que hacíamos referencia anteriormente. Aún habiendo seleccionado un conjunto de fallos, definir vectores de test capaces de detectarlos todos es muchas veces imposible: En primer lugar porque no todos los fallos son detectables ....

DEFINICIÓN

Decimos que un **fallo** es **detectable** cuando existe una combinación de entradas que produce al menos una salida distinta si el fallo está presente que si no lo está.

.... y en segundo lugar porque el tiempo dedicado al test del circuito viene habitualmente limitado por cuestiones de disponibilidad de la máquina de test o de otra índole.

DEFINICIÓN

Se define la **COBERTURA DE FALLOS** de un cierto conjunto de vectores de test como el porcentaje de fallos *del modelo* escogido que se detectan al aplicar dichos vectores de test. Para damos una idea de los valores que comúnmente se barajan, una cobertura por debajo del 95% a 98% es habitualmente considerada como mala y el fabricante se negará a entrar en fabricación (por supuesto hay excepciones).

Modelo de fallos

Para abordar la tarea de la detección de defectos, es necesario conocer previamente qué tipos de defectos se van a considerar. Los fallos que se introducen durante la fabricación (defectos fotolitográficos, deficiente calidad del proceso, etc.) afectan directamente a la geometría del circuito. Parece pues sensato que se baje a este nivel para tratar de modelar los fallos; sin embargo, conforme el sistema crece en complejidad, crece también extremadamente el número de posibles fallos a nivel físico, con los cual la tarea de detección resulta intratable. En este punto, resulta útil describir los fallos en función de su efecto a niveles más altos (nivel eléctrico, lógico, o incluso funcional).

Para que un modelo de fallos sea válido es necesario que:

- Sea preciso; es decir, que los fallos modelados se acerquen tanto como sea posible a la realidad, y que

- Sea tratable; esto es, que dichos fallos puedan detectarse aunque el circuito sea grande.

Ambos requerimientos son contradictorios. Los modelos que más se acercan a la realidad son evidentemente los modelos a nivel de físico; pero estos a su vez son los más intratables. El compromiso que se toma es el de utilizar modelos a niveles más altos que reflejen lo más fielmente posible los fallos en los niveles más bajos.

Existen multitud de modelos de fallos en la literatura que no podemos abordar en estos apuntes. Dado el limitado tiempo del que disponemos para tratar estos temas y su carácter introductorio, aquí nos limitaremos a (1) esbozar brevemente algunos de estos modelos y a (2) centrar nuestro interés en el modelo de fallos más frecuentemente utilizado en el test de CIs digitales, el modelo de “stuck-at”.

Modelos a nivel de transistor o interruptor

Los modelos de fallos propuesto a nivel de transistor incorporan algunos de los siguientes fallos:

Modelo de fallos

1) Cortocircuitos (*shorts*) y circuitos abiertos (*opens*) de transistores o líneas de interconexión.

2) Retardos.

3) Acoplos entre nodos.

4) Degradación de elementos.

Estos modelos reflejan el efecto que producen los defectos en la conducción o no de los transistores sobre el estado lógico de la salida.

Modelos de fallos a nivel funcional

Estos modelos depende del nivel al que se haya descrito el circuito. Así según el nivel jerárquico de que se trate, el circuito se puede describir con puertas lógicas, bloques funcionales o por su estructura funcional (transferencia de registros p.ej.).

Los fallos correspondientes a estos modelos simulan el comportamiento funcional de cada uno de estos elementos en presencia de defectos. Para puertas lógicas existe un modelo muy simple para el cual existen muchas herramientas de generación de vectores de test desarrollados al efecto, ya que este fue uno de los primeros modelos de fallos propuestos.

Para bloques funcionales más complejos, los modelos de fallos se obtienen por inferencia a partir de modelos que describen el mal funcionamiento sobre los elementos que constituyen estos bloques más complejos. Este esfuerzo de desarrollo de un modelo particular para cada bloque solo tienen sentido para circuitos muy repetitivos, donde el número de bloques diferentes que lo forman sea relativamente reducido.

Modelos a nivel lógico

A nivel lógico, los modelos más utilizados son los de:

1) **Stuck-at** (bloqueos) : Los más utilizados, con diferencia.

2) Bridges (puenteos)

Modelo de fallos

El modelo de STUCK-Ats

El modelo de stuck-ats ha sido y es el más utilizado por razones históricas. Este modelo supone que los defectos físicos pueden modelarse como líneas del circuito lógico que quedan bloqueadas permanentemente a 0 o a 1. Evidentemente, este modelo es menos preciso que el de "shorts-opens" a nivel de transistor (por ejemplo), pero la detección de faltas es, a cambio, mucho más simple. En realidad, si bien una buena parte de los fallos a nivel de transistor no pueden modelarse como stuck-ats, experimentalmente se comprueba que sí son detectados por los vectores de test desarrollados a partir del modelo de stuck- at. Como ejemplo, consideremos la puerta NAND de 3 entradas de la figura 13. La tabla adjunta a la figura 13 muestra los vectores de test necesarios para detectar cualquier fallo simple de stuck-at de las entradas o la salida.

ver figura >> 13

Supongamos que la puerta ha sido realizada con tecnología NMOS, y que su estructura es la que muestra la figura 14. En esta figura se han indicado a trazos dos posibles "*shorts*" de líneas (fallos a nivel de transistor), el (1) y el (2). En la tabla 2 hemos representado la respuesta de la puerta a cada vector de test cuando la puerta está libre de fallos (F0); cuando está presente el fallo 1 (F1), y cuando está presente el fallo 2 (F2). Nótese que los fallos 1 y 2 quedan detectados al aplicar los vectores 011 y 111 respectivamente; es decir, con los vectores de test generados para detectar los fallos de stuck-at, se detectan también los fallos a nivel de transistor. Dicho de otra manera, muchos fallos que no pueden modelarse como stuck-ats, pueden de todas formas detectarse aplicando un conjunto de vectores de test basado en el modelos de stuck-at.

ver figura >> 14

Puenteos

Este modelo reúne todos aquellos fallos cuyo efecto es la conexión accidental de dos líneas. Dicho cortocircuito se convierte a nivel lógico en una AND cableada o una OR cableada según los casos.

Modelo de fallos

Modelos de fallos en bloques funcionales (PLAs)

La peculiar estructura de las PLAs no sólo induce fallos de stuck o de puenteo que pueden encontrarse en la mayoría de los circuitos combinacionales sino que, además, da lugar a un tipo de contacto que no se produce en tales circuitos. Fundamentalmente, se pueden presentar tres tipos de fallos: los clásicos de stuck, los fallos de cruce y los fallos de puenteo (o de corte).

Un fallo de cruce es la ausencia o la presencia extra de un dispositivo (transistor) entre una línea de entrada (línea de bit, ver figura 15) y una línea producto; o entre una línea producto y una línea de salida. Este tipo de fallos está estrechamente relacionado con el diseño de la PLA.

En los fallos de stuck, una de las líneas queda fijada permanentemente a uno de los dos valores lógicos. Su causa generalmente es un cortocircuito entre la línea en cuestión y alimentación o tierra.

ver figura >> 15

Por último, un fallo de corte es un corte entre dos líneas adyacentes o entre dos líneas que se cruzan. Se puede demostrar que un test que detecte todos los fallos de cruce simples detecta también la mayoría de los fallos simples de stuck y de corte. Dicho test detecta también la mayoría (alrededor del 98%) de los fallos de cruce de multiplicidad menor o igual que 8.

Se puede demostrar que todos estos fallos se pueden detectar a nivel de la función lógica que implementa la PLA buscando si el fallos (1) introduce términos producto que no existen en las funciones originales, (2) introduce variables an algún término producto, (3) elimina términos producto o (4) elimina variables en algún término producto. Con esto se quiere resaltar el hecho de que, utilizando este modelo de fallos al testear una PLA, basta con que miremos qué funciones está implementando exactamente la PLA y las comparemos con las originales, mirando si hay un exceso o defecto de términos producto o de variables.

Modelo de fallos

(lo que viene a continuación es una breve explicación de este último párrafo, sólo para aquél que esté especialmente interesados; podéis saltároslo sin problemas en una primera lectura)

Estudiemos con algo más detalle el efecto de éstos fallos: Un fallo de stuck puede causar la desaparición de un literal de un implicante (s-a-1 sobre una línea de bit); la desaparición de un implicante completo (s-a-0 sobre una línea de bit); o la transformación de una de las funciones en la identidad o la función nula. Un defecto de cruce puede causar 4 tipos de errores sobre los implicantes de la función: la desaparición de un literal (el implicante "crece") debido a un dispositivo no colocado en el plano AND; la aparición de un literal adicional en un implicante (el implicante "se encoge") debido a un dispositivo extra en el plano AND; la desaparición de un implicante debido a la falta de un dispositivo en el plano OR; y la aparición de un nuevo implicante en la función debido a un dispositivo extra colocado en el plano OR. Finalmente, el resultado de los defectos de corte entre líneas depende de la tecnología que se emplee. Por ejemplo, en NMOS, un corte entre dos líneas de salida da lugar a una operación AND entre ellas.

Una clasificación más amplia de los efectos lógicos cubre los tipos de fallos siguientes:

a) Crecimiento-aparición: Se refiere al crecimiento de un término producto o la aparición de algún término producto sobre una o varias funciones de salida.

b) Encogimiento-desaparición: Se refiere al encogimiento de un término producto o la desaparición de un término producto sobre una o varias funciones de salida.

Un fallo simple de cualquier tipo sólo puede producir un error unidireccional; esto es, puede causar bien un defecto de crecimiento-aparición, o bien un efecto de encogimiento-desaparición, pero no ambos.

Reducción del número de fallos

Una vez tenemos definido el modelo de fallos y sabemos exactamente qué fallos (y cuantos) queremos detectar, antes de proceder a generar patrones de test para cada uno de ellos vale la pensa intentar reducir su número echando mano de los conceptos de **equivalencia** y **dominancia** entre fallos.

La idea general viene a ser la siguiente: si dos o más fallos son detectables con el mismo conjunto de vectores de test, a la hora de generar los vectores no hace falta considerar todos estos fallos sino que basta con tener en cuenta uno de ellos. Consideremos de nuevo la puerta NAND de la figura 13 y supongamos, para mayor facilidad, que trabajamos con un modelo de stuck-at. Tanto si la entrada A, o la E, o la C quedan bloqueadas a 0, el efecto sobre la salida OUT es el mismo: OUT estará permanentemente a 1. Se dice entonces que los fallos A s-a-0 (léase “línea A stuck-at 0”, B s-a-0 y C s-a-0 son FALLOS EQUIV ALENTES (producen el mismo efecto).

Supongamos ahora que A queda bloqueada a 1. Para detectar este fallo se debe aplicar un 0 a A y 1s a B y C para que el fallo pueda propagarse a la salida (esto es, que aparezca un 0 en OUT si existe el fallo, o un 1 si no hay fallo). Este mismo vector ABC = 011 detecta también el fallo OUT s-a-0. Se dice entonces que el fallo A s-a-1DOMINA al fallo OUT s-a-0.

DEFINICIÓN: Dos **fallos** A y B son **equivalentes** cuando el todos los patrones de test que detectan el fallo A detectan también el fallo B. Basta por tanto con buscar un patrón de test que detecte uno de los dos fallos, porque seguro que dicho patrón detectará también el otro fallo.

DEFINICIÓN: Un fallo A **domina** a un Fallo B cuando cualquier patrón que detecte el fallo A detecta también al fallo B (pero no al contrario). Si A domina B, basta con buscar un patrón de test que detecta a A.

A través de las relaciones de equivalencia y dominancia es posible reducir el número de fallos a tratar, seleccionando sólo un representante de cada clase. Siguiendo con el ejemplo anterior, la tabla de la figura 16 muestra las clases de equivalencia de fallos para la puerta NAND (modelo stuck-at), y el vector de test que detecta cada uno de ellos.

ver figura >> 16

Reducción del número de fallos

En el caso más general, para cada fallo *Fi* existe un conjunto de vectores de test *Vi* que lo detectan, y entre estos conjuntos *Vi* hay habitualmente intersecciones. Es decir, *existen vectores de test que por sí solos detectan más de un fallo y viceversa, existen varios vectores de test que detectan un mismo fallo*.

Determinación de vectores de test

Llegados a este punto estamos en condiciones de entender cómo se genera el conjunto de vectores (o patrones) de test. El proceso directo sería:

1. Se selecciona el modelo de fallos. En el caso de que estemos desarrollando un ASIC en el ambiente comercial, el fabricante es el que determina qué modelo se debe utilizar, y, si el circuito es digital, en el 99% de los casos este modelo será el de stuck-ats. Recordemos que el fabricante obliga al diseñador a entregar (1) el layout y (2) un conjunto de vectores de test que garanticen un buen nivel de cobertura de fallos (alrededor del 95-98%).
2. La aplicación del modelo de fallos al circuito permite determinar la lista concreta de fallos a detectar.
3. Se buscan fallos equivalente y redundantes en un intento de reducir el número de fallos, y
4. Se procede a buscar, para cada fallo, un vector de test que lo detecte.

Más adelante veremos cómo se puede simplificar este engorroso procedimiento en la práctica, pero antes de eso vamos a ver cómo se puede buscar, para cada fallo, un vector que lo detecte. Cada modelo de fallos tiene su método, por lo que nos vamos a centrar de nuevo en el modelo de stuck-ats.

Determinación de vectores de test

Generación de vectores de test a nivel lógico (modelo de stuck-ats)

La determinación de vectores de test para el modelo de stuck-ats utiliza el método de "**Sensibilización de caminos**", que funciona de la siguiente manera. Para cada fallo se procede como sigue:

1) **Fase de set-up** : Supongamos que en el circuito de la figura 17.a queremos detectar el fallo "línea *a* stuck-at 0". Cualquier combinación de los valores de entrada que aspire a detectar el fallo deberá forzar sobre la línea *a* el valor contrario al del fallo, puesto que, de no ser así, los valores de salida que se producirán en caso de existencia o no del fallos serán indistinguibles. Esto lo vamos a representar diciendo que establecemos que *la señal de error debe ser un 1/0*, indicando de esta manera que deseamos que el valor que tome la línea *a* sea un 1, aunque si ocurre el fallo dicha línea tomará el valor 0. El hecho de establecer la señal de error 1/0 constituye la fase de set-up. (figura 17.b)

2) **Fase de propagación** : Que la línea *a* tome el valor 1 en ausencia de fallo y 0 en presencia de fallo no nos sirve de nada si no podemos ver esta señal desde el exterior (a través de algún pin de salida). La fase de propagación consiste en (1) seleccionar un camino desde la línea *a* hacia una de las salidas del circuito y (2) asignar valores a un mínimo número de líneas de este camino que garanticen que la señal de error 1/0 (o su complementaria 0/1) se propague hasta dicha salida (figura 17.c).

3) **Fase de justificación** : Lo que resta por hacer es establecer valores sobre el resto de las líneas que garanticen que los valores establecido en la fase de propagación se puedan alcanzar (figura 17.d). Los valores de las entradas junto a el valor previsto de salida (esto es, el 0 puesto que es el valor que debe tomar la salida si el circuito funciona correctamente) constituyen el vector de test buscado.

ver figura >> 17

Determinación de vectores de test

El método de sensibilización simple de caminos (simple: solo se define 1 camino) no siempre consigue sus objetivos. Pueden existir fallos testables para los cuales no es posible generar el vector de test si sólo se contempla 1 camino, como por ejemplo el circuito de la figura 18.

ver figura >> 18

En este circuito, todos los caminos simples generan inconsistencias. Sin embargo, la combinación x1 = x2 = x3 = x4 = 1 detecta el fallo.

Para garantizar la detección del fallo (si éste es detectable), se deben considerar la sensibilización de todos los caminos simultáneamente.

Existen herramientas capaces de generar automáticamente los vectores de test (**ATPGs** o *Automatic Test Pattern Generators*) para el modelo de stuck-ats basadas en este método de sensibilización de caminos que datan de 1966, año en el que Roth presentó su conocido “algoritmo D”. El algoritmo D ha quedado obsoleto; por ejemplo, es conocida la ineficiencia del algoritmo D en funciones caracterizadas por poseer árboles de puertas XOR, y se ha visto substituido hoy en día por otros algoritmos como el PODEM (“Path Oriented Decision Making”), desarrollado por Goel, es más eficiente que el algoritmo D. El PODEM es un algoritmo de implícitamente o el FAN (“Fan-out-oriented test generation algorithm”), que no vamos a explicar aquí.

Aproximación práctica a la generación de vectores de test.

Debido a la complejidad que presentan los circuitos VLSI, tal y como se viene indicando desde el principio de este capítulo, cada vez se hace más necesario utilizar la máxima información que pueda prestarnos la fase de diseño para el test. En este sentido *se pueden aprovechar los vectores de simulación* usados en la fase de diseño, *para la determinación del conjunto de vectores de test*.

Determinación de vectores de test

El máximo inconveniente que plantea su utilización es que, si nos limitamos a coger los vectores de simulación y utilizarlos como vectores de test, se alcanzan coberturas demasiado bajas ya que en la elaboración de los vectores de simulación el diseñador solo ha tenido en cuenta criterios meramente funcionales. Sin embargo, lo que si se puede hacer es tomar este conjunto de vectores como base del conjunto de patrones de test que se utilizará finalmente. En la práctica, la aproximación que se sigue para obtener el conjunto de vectores de test se resume en la figura 19.

ver figura >> 19

Tras decidir el modelo de fallos que se utilizará se construye la lista de fallos a detectar, y se intenta reducir en lo posible mediante las equivalencias. A partir del conjunto de vectores de simulación se evalúa la cobertura de fallos mediante un simulador de fallos, y si no es satisfactoria, se utiliza un generador de vectores de test con el que se obtienen vectores que detectan algunos de los fallos. Cuando la cobertura se considera buena, el conjunto de vectores de test se da por bueno y se pasa a la máquina de test.

Simuladores de fallos

Falta un último punto a considerar : Dado un circuito y un conjunto de vectores de test, ¿cómo podemos conocer la cobertura del mismo. La respuesta a esta pregunta son los simuladores de fallos.

Un simulador de fallos es una herramienta CAD al que se le entra una descripción del circuito libre de fallos, un conjunto de posibles fallos (lista de fallos), la especificación de las entradas y salidas externas del circuito, y la secuencia de vectores de test a simular. Para cada vector de test, el simulador simula la respuesta de la máquina libre de fallos, y la máquina en presencia de cada uno de los fallos posibles (a la máquina que presenta el fallo x se le denomina “máquina errónea x”). Cuando a lo largo de la simulación se detectan valores distintos en las salidas externas para la máquina libre de fallos y para alguna de las máquinas erróneas, el fallo de ésta última se marca en la lista de fallos como detectado y la simulación de ésta máquina se para. Controlando las marcas en la lista de fallos, el simulador determina la cobertura, devolviendo información sobre los fallos no detectados. El "testeador" puede aprovechar estos datos para incluir nuevos vectores de test que detecten los fallos todavía no detectados.

En general, podemos diferenciar dos tipos de simuladores: los "conducidos por tablas", y los "conducidos por compiladores". En los primeros, la descripción lógica del circuito se guarda en unas tablas. Los programas de simulación son independientes del circuito, accediendo a dichas tablas para su simulación. En contraste, el segundo tipo de simuladores tienen el circuito implícitamente descrito como código compilado. Los simuladores conducidos por compilador se emplean usualmente en la simulación de circuitos con modelo de retardo nulo, y de ahí que sólo manejen circuitos lógicos síncronos. Los conducidos por tablas modelan con estas el comportamiento dinámico del circuito, actualizando los valores de cada nodo a intervalos constantes de tiempo (de simulación).

Existen fundamentalmente tres métodos de simulación de fallos:

- Simulación paralela.
- Simulación deductiva.
- Simulación concurrente.

de los cuales vamos a dar simplemente un par de pinceladas ...

Simuladores de fallos

(nota : tomad este apartado como ampliación, y leéroslo sólo si os interesa)

Simuladores paralelos

En la simulación paralela, como su nombre indica, se simulan en paralelo la máquina libre de fallos y las máquinas erróneas. A cada nodo del circuito se le asocian dos o más palabras de ordenador que bit a bit guardan el estado lógico de dicho nodo en la máquina libre de fallos (Bit 0), y en las máquinas erróneas (Bits del 1 al n-1). Puesto que el número de fallos es normalmente mayor que el número de bits por palabra. El proceso de simulación se repite para varios conjuntos de fallos.

ver figura >> 20

La simulación paralela implica la simulación real de todas las máquinas (la libre de fallos y las erróneas), lo que incrementa mucho el coste de la simulación. Según Goel, el coste de la simulación paralelo de un circuito de N puertas es proporcional al cubo de N (N3).

Otro inconveniente de la simulación paralela es que en una simulación se pueden simular un máximo de N máquinas siendo N la longitud de la palabra de ordenador. Si el número de fallos a tratar es M (M*N habitualmente) es necesario realizar (M+1)/N pasadas del simulador.

Finalmente, tras haber aplicado un cierto número de vectores a un mismo conjunto de fallos, solo unos pocos de estos fallos permanecen indetectados y la eficiencia del simulador decrece puesto que éste sigue simulando todos los fallos.

Simuladores deductivos

La idea fundamental de estos simuladores es simular la máquina buena, y, en función de los valores lógicos obtenidos para ésta, *deducir* los valores que se habrían alcanzado con las máquinas erróneas. Brevemente, la técnica deductiva es la siguiente:

Simuladores de fallos

En primer lugar, en lo que se refiere a esta explicación, llamaremos nodos del circuito a los elementos de éste: una puerta es un nodo; un flip-flop son dos nodos, uno para cada salida. El conjunto de nodos que tienen al menos una entrada que es una entrada primaria constituyen el primer nivel de nodos. El conjunto de nodos con al menos una entrada conectada a la salida de un nodo de primer nivel forman el segundo nivel de nodos, y asi sucesivamente.

Para cada vector de test, el simulador calcula el estado lógico de los las salidas de los nodos de primer nivel en la máquina libre de fallos. A continuación calcula las "listas de fallos", una lista asociada a cada salida de los nodos de primer nivel. En cada lista se incluyen aquellos fallos detectables cuando el estado lógico de la línea de salida es el actual; esto es, aquellos fallos que producirían en la línea de salida del nodo el valor lógico complementario del simulado para la máquina buena. A continuación podemos ver como se construyen las listas de fallos para una puerta NOR (figura 21):

ver figura >> 21

El siguiente paso es simular la máquina libre de fallos para los nodos de segundo nivel y calcular las listas de fallos para las salidas de éstos. La lista de fallos de cada nodo contiene ahora los fallos propios del nodo (esto es, los calculados pensando en los fallos a las entradas del nodo que generan la salida complementaria de la alcanzada por la máquina sin fallos, independientemente del resto del circuito), y los fallos consecuencia de la propagación de los fallos contenidos en las lista de fallos de los nodos de primer nivel conectados a éste. El proceso se repite para cada nivel. Finalmente, los fallos contenidos en las listas de los nodos conectados a las salidas primarias son los fallos detectables por el vector de test entrado.

La simulación deductiva es más rápida que la paralela, aunque requiere habitualmente más memoria. El coste de la simulación es proporcional a N (N :número de puertas del circuito). Por otra parte, la simulación deductiva permite simular todos los fallos en una única pasada. La figura 22 compara para varios circuitos el tiempo de CPU en una simulación paralela y en una deductiva. Se observa que la simulación paralela es buena sólo para circuitos pequeños.

Simuladores de fallos

ver figura >> 22

Simuladores concurrentes

Los simuladores concurrentes parten de la observación que el comportamiento del circuito libre de fallos, y el comportamiento del circuito en presencia de un fallo son raramente muy distintos. Los simuladores concurrentes simulan el comportamiento de la máquina libre de fallos y de la máquina con fallos sólo en aquellas partes del circuito en que los valores lógicos obtenidos difieren. La filosofía de estos simuladores es, hasta cierto punto, cercana a la de los simuladores deductivos; pero aquí aparece ya una diferencia importante: los simuladores deductivos calculan las tablas de fallos; los simuladores concurrentes *simulan* explícitamente la máquina errónea.

A cada nodo se le asocia una lista de fallos que incluye todos los fallos posibles que afectan a dicho nodo y los valores que toman las entradas y las salidas de existir éste. Por ejemplo, imaginemos que la puerta AND de la figura 23 recibe en un instante dado las entradas 00. En presencia del fallo A s-a-0 (A/0), las entradas toman los valores 10, y la salida el valor 0. Esto se representa en la lista de fallos como: A/0,00,0. Supongamos que la entrada A cambia a 1 (figura 23). Puesto que se detecta un cambio en una entrada de la puerta AND, se simula el funcionamiento de ésta en presencia de cada uno de los fallos.

Puesto que A = 1, el primer fallo de la lista (A/1,10,0) desaparece, y aparece un nuevo fallo A/0 que se simula y genera la entrada A/0,00,0 en la correspondiente lista. Asimismo, los fallos B/1 y D/1 se convierten en B/1, 11,1 y D/1, 10,1. De esta nueva lista de fallos, sólo el fallo B/1 causa una modificación en la entrada de la siguiente puerta. Para la puerta OR, el fallo B/1 convierte las entradas en 10, y la salida en 1, con lo que se incluye esta línea (B/1,10,1) en la nueva lista de fallos de la puerta OR. El resto de entradas de la lista de fallos no se modifica. En las salidas primarias, aquellas entradas de la lista de fallos cuya salida difiera de la simulada para la máquina libre de fallos representan los fallos detectados por el vector de test entrado al circuito.

ver figura >> 23

Simuladores de fallos

La simulación concurrente comparte con la deductiva algunas ventajas: sólo es necesario una pasada independientemente del número de fallos que se consideren, y las máquinas erróneas no se simulan totalmente. La concurrente es más o menos igual de rápida que la deductiva, aunque requiere en general más memoria y, lo que es peor, no se puede conocer a priori la cantidad de memoria que necesitará. Como contrapartida, la concurrente simula explícitamente las máquinas con fallos (parcialmente), con lo que se pueden modelar más fielmente los comportamiento temporales.

Resumen

En este capítulo hemos introducido los conceptos más básicos del test de circuitos integrados digitales. Por orden hemos visto:

1. Los distintos tests que se le pasan al ASIC, en la propia fase de diseño (simulación), a nivel de oblea y una vez encapsulado. También hemos visto las características diferentes del testado de prototipos y del testado de la serie (test de entrada).
2. La conveniencia de comprobar periódicamente el circuito a lo largo de toda su vida activa, y las posibilidades que ofrece las técnicas de test concurrente y de test no-concurrente.
3. Las máquinas utilizadas en el test de los CIs.
4. La manera de atacar el test de grandes CIs. Es de gran importancia comprender la estrategia de test que hemos denominado “test estructural“ y el concepto de modelo de fallos, puesto que son las piedras fundamentales que soportan las técnicas de generación de vectores de test. De los innumerable modelos de fallos existente nos hemos centrado en el modelo de stuck-at por ser el más utilizado en el test de ASICs industriales.
5. Hemos estudiado el método de sensibilización de caminos.
6. Hemos establecido una aproximación a la generación de vectores de test que aprovecha los resultados de las simulaciones del circuito, facilitando así la tediosa tarea de construir un vector de test para cada uno de los fallos del modelo, y
7. Se han descrito muy brevemente los simuladores de fallos, herramientas CAD que nos permiten, entre otras cosas, calcular las coberturas alcanzadas por el conjunto de vectores de test.

Hay una serie de conceptos que deberían haber quedado claros tras la lectura de este capítulo:

Elena Valderrama; Carles Ferrer

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 9

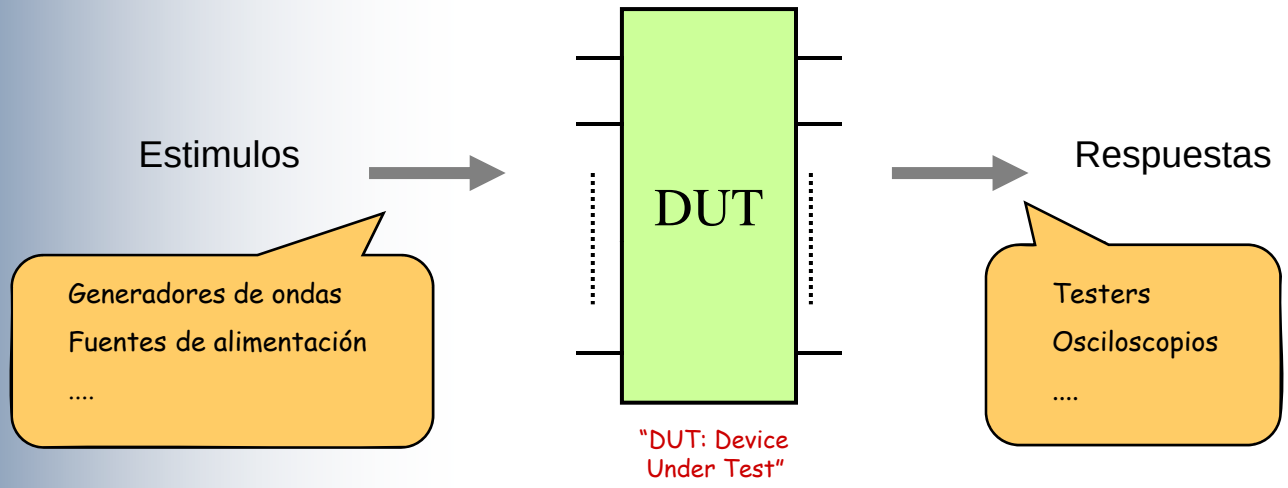
Figura 1

	1970	1975	1980	1985	1990	1995	2000
Complejidad	SSI	MSI	LSI	VLSI	ULSI		SoC
Transistores	100	1K	10K	100K	1M	10M	100M
Memorias	256	1K	4-16K	64-256K	1-16M	64	1-16G
Velocidad	-	1Mhz	10Mhz	30MHz	100MHz	1GHz	10GHz
Pines	-	32	64	128	256	512	1024
Coste test/Coste total	5%	10%	20%	40%	60%		

Evolución de los Cis:

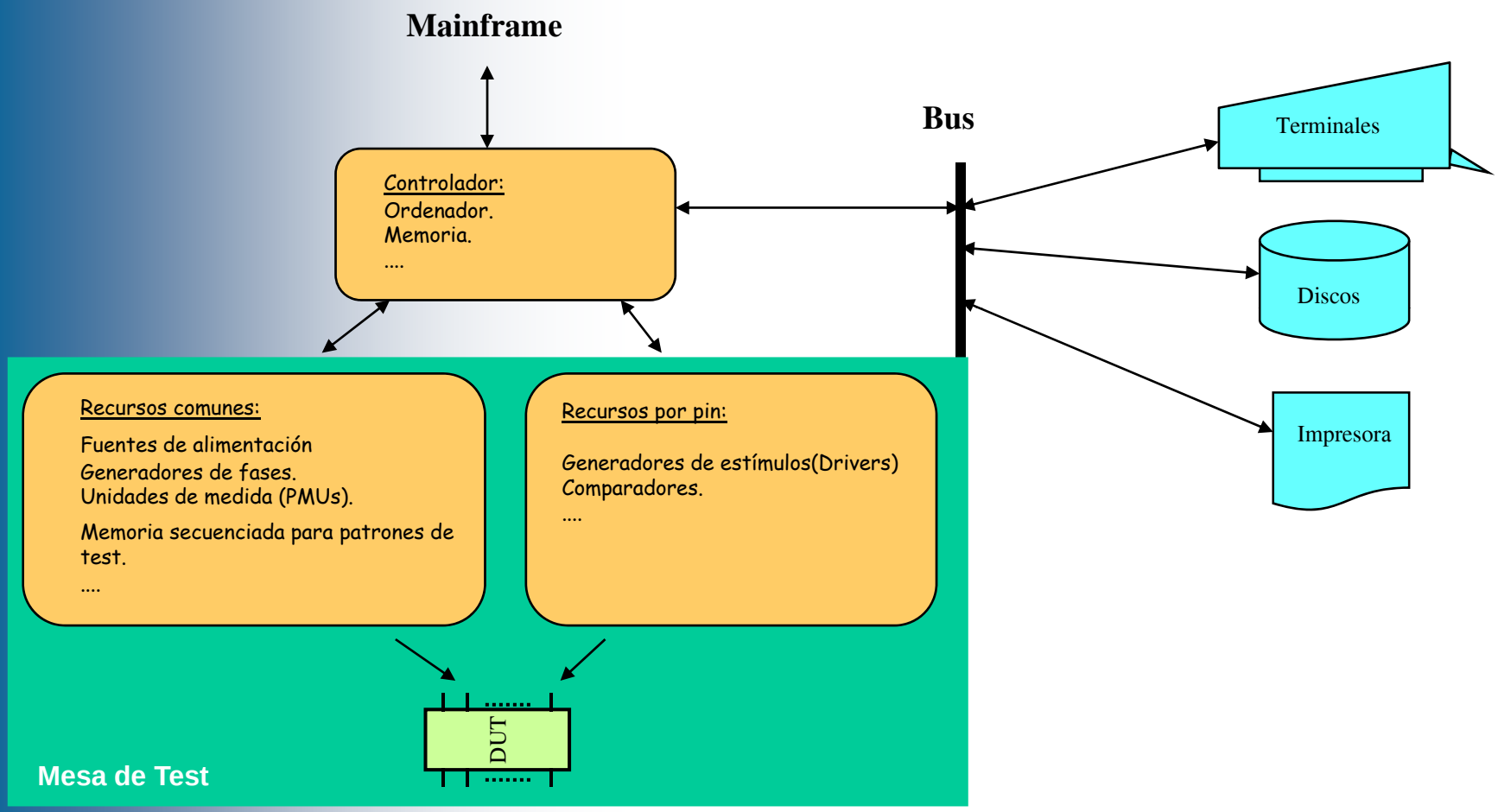
Se evidencia el incremento del coste del test en relación con el coste global de desrrollo de un circuito integrado.

Figura 2



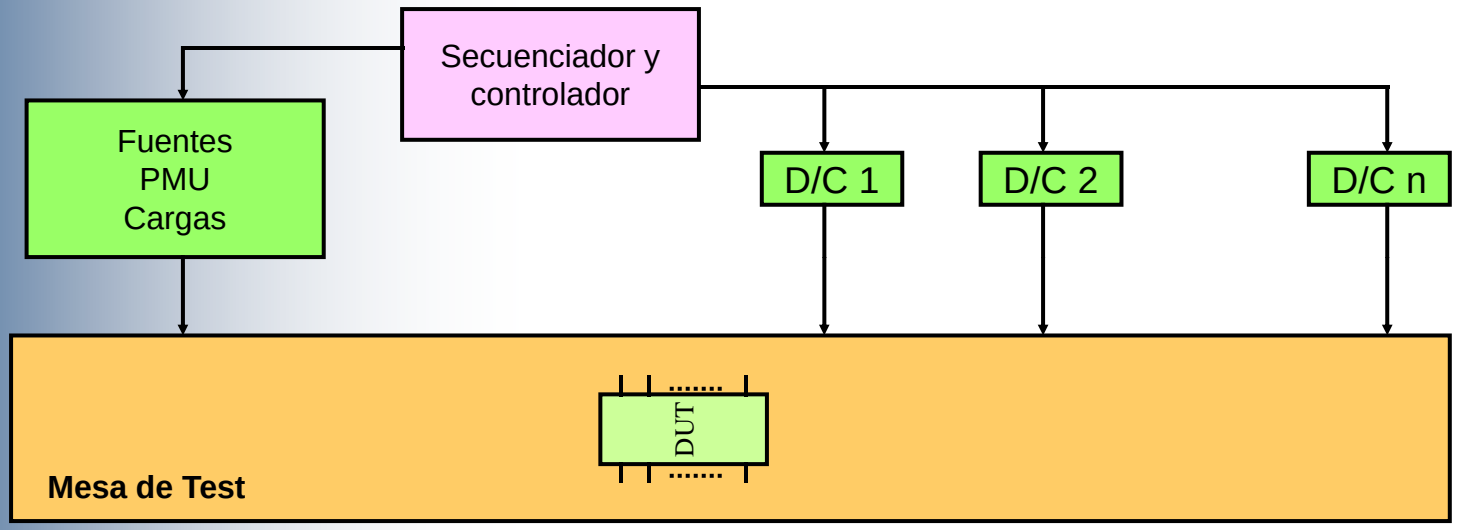
Test de un circuito intergado mediante el uso de instrumetación discreta

Figura 3



"ATE: Automatic Test Equipment"

Figura 4



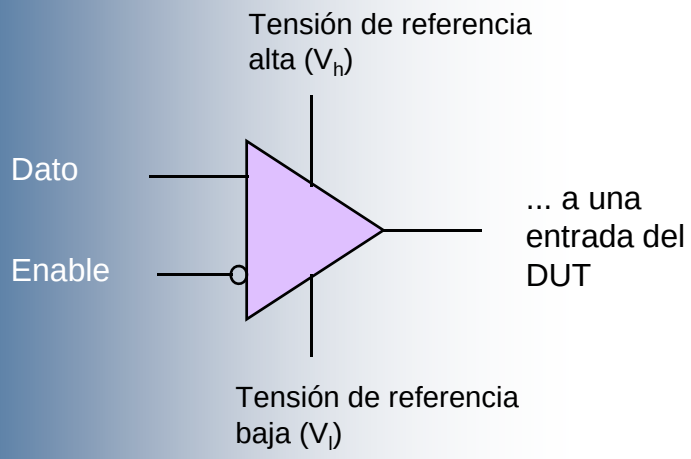
(nota: las parejas D/C y los módulos de PMU, cargas, fuentes, etc., están físicamente dentro de la mesa de test)

Esquema de una máquina de test:

"ATE: Automatic Test Equipment".
"DUT: Device Under Test".
"D/C: Driver/Comparator".
"PMU: Parametric Measurement Unit"

Figura 5

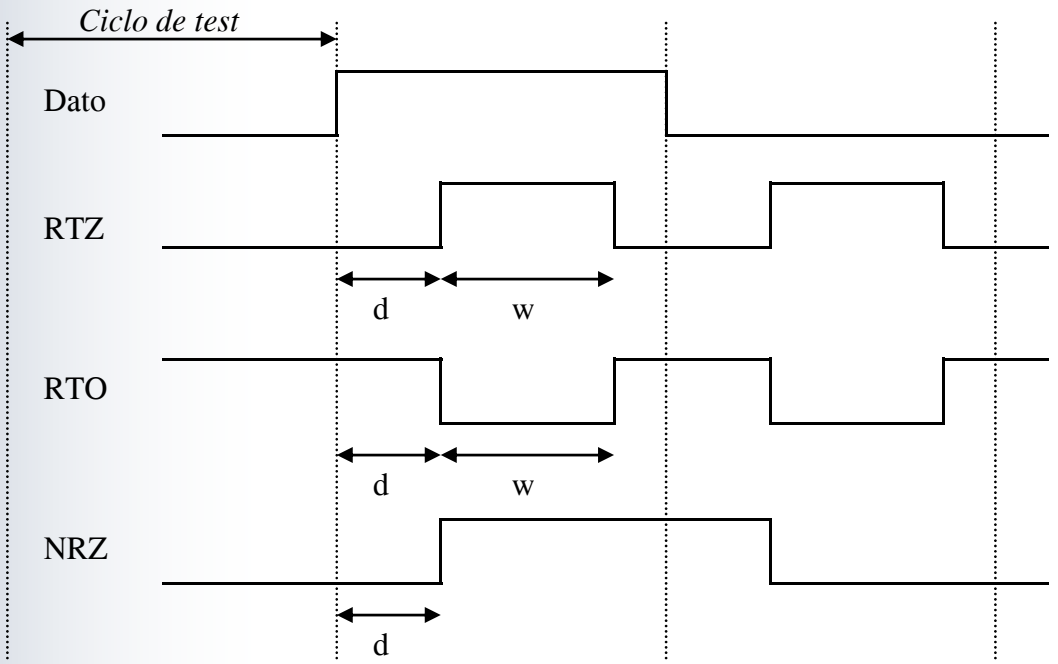
Driver para la generación de estímulos de entrada



Dato	Enable	Salida
0,1	1	Z
0	0	V_l
1	0	V_h

Con *enable* a alta el driver queda en alta impedancia y por lo tanto no se fuerza ningún valor a la entrada

Figura 6

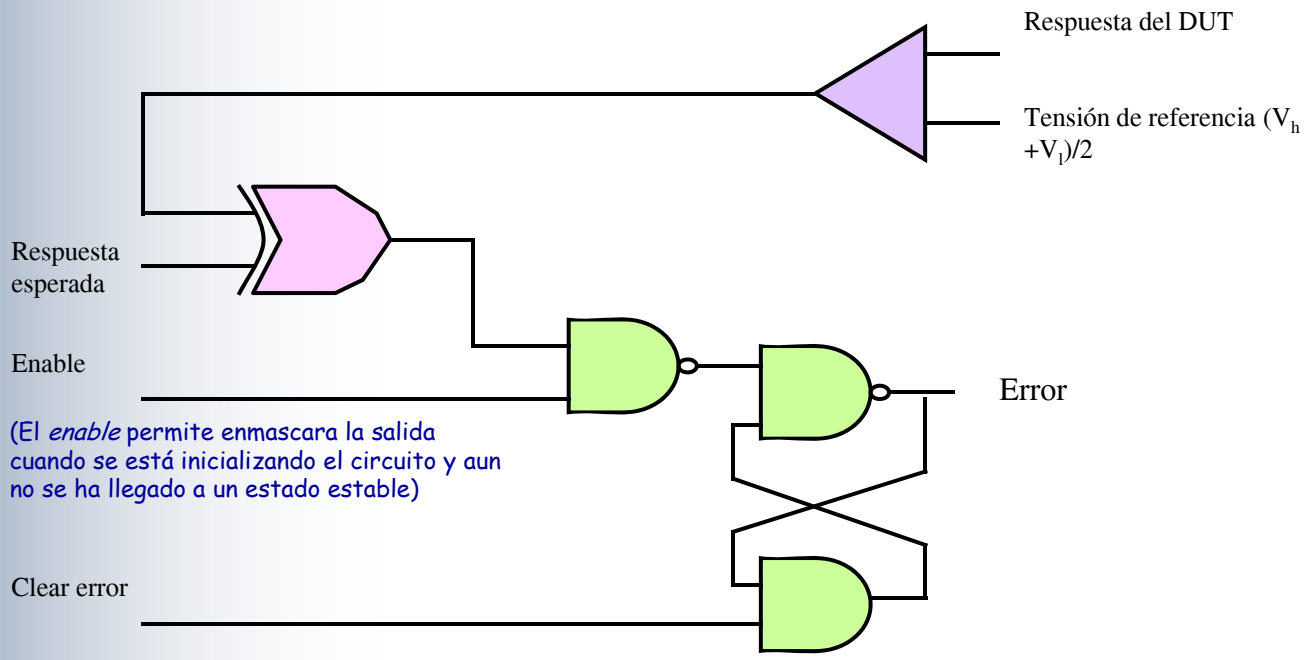


RTZ: "*Return To Zero*", RTO: "*Return To One*", NRTZ: "*Non Return To Zero*", d: *delay*, w: *pulse width*.

NRZ es el formato habitual de datos mientras que RTZ o RTO son los formatos con los que se generan los pulsos de la señal de reloj.

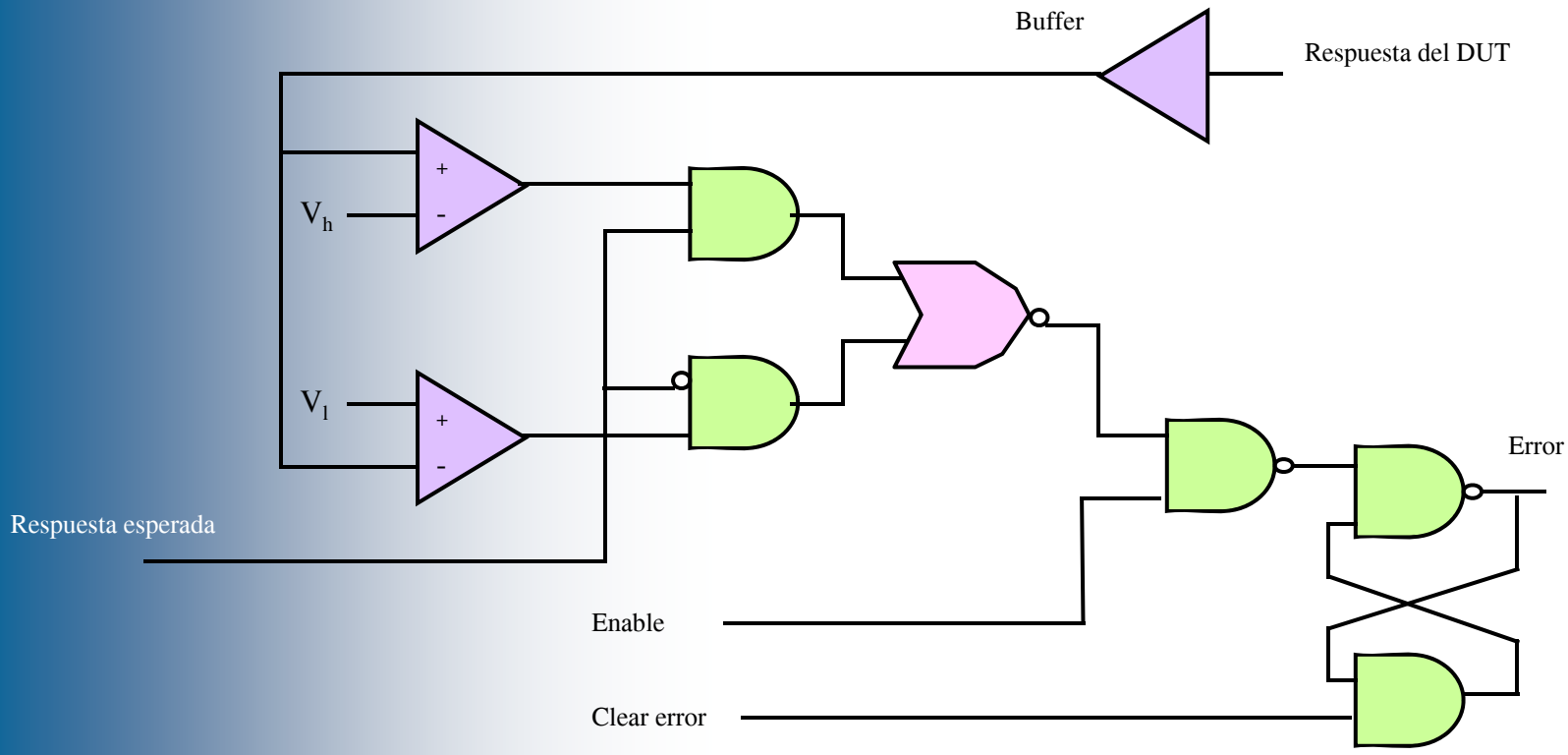
Generación de estímulos de entrada

Figura 7



Comparación de salida con un solo nivel de comparación

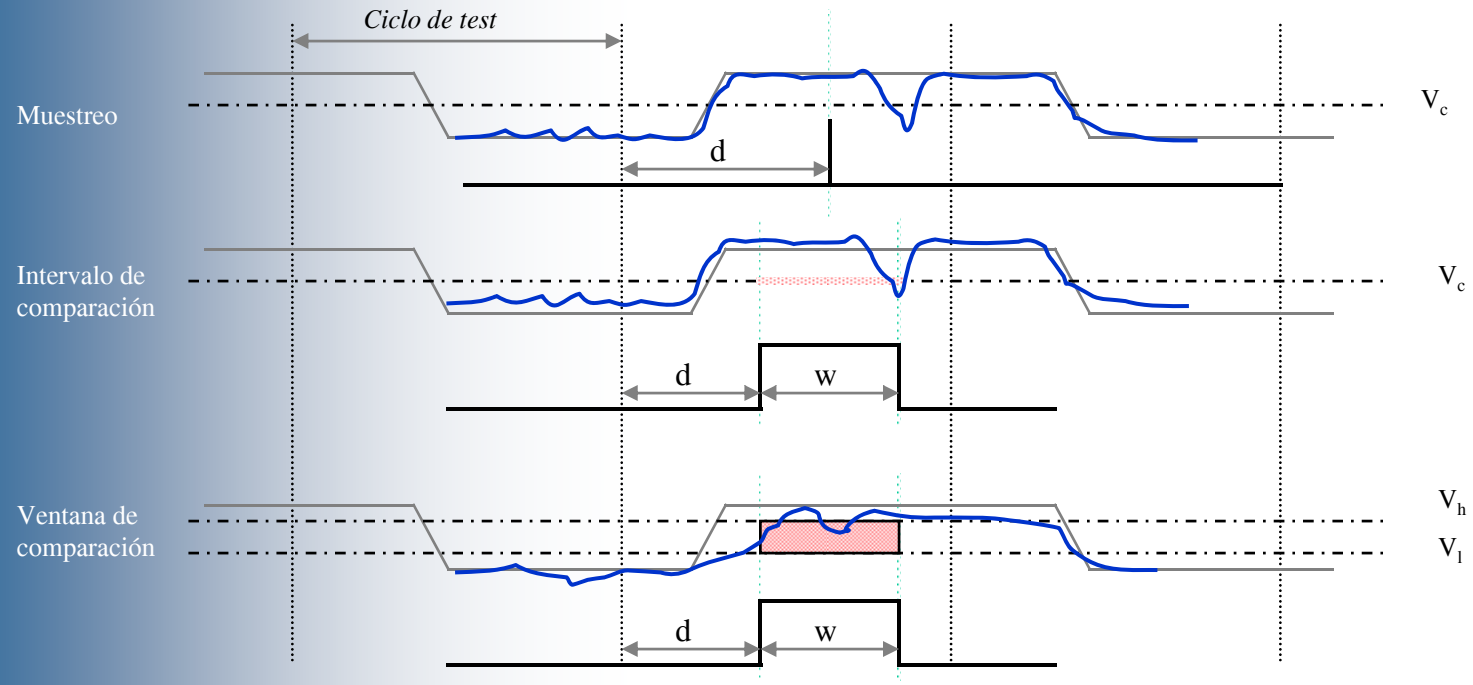
Figura 8



Comparación de salida con dos niveles de comparación

El uso de dos niveles de comparación permite ajustar el test a una situación más estricta y por consiguiente más cercana al catlogo de especificaciones del cirucito.

Figura 9

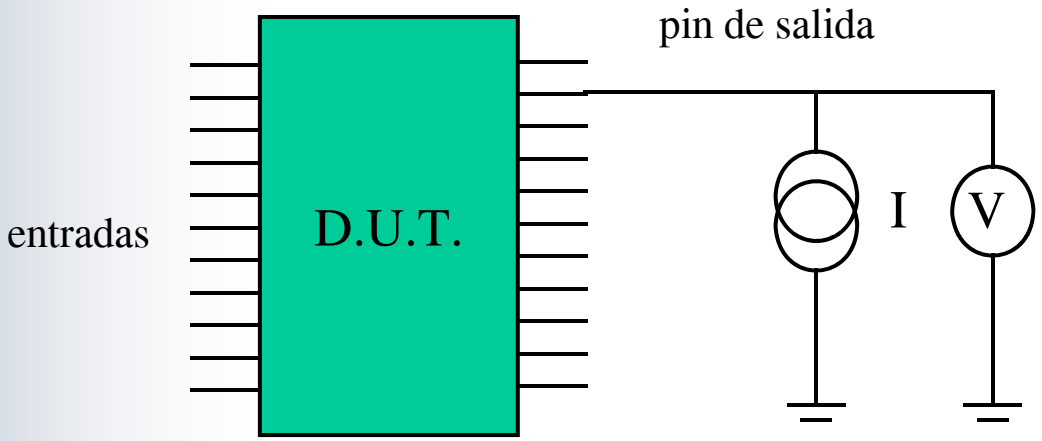


d: *delay*, w: *window width*. (Los formatos estan referenciados al valor de ciclo de test preestablecido por el ATE +capacidad de enmascarar la comparación en un ciclo).

Adquisición de salida

Tener una ventana de comparación con dos niveles de comparación y un ancho de ventana permite disponer de un test más estricto

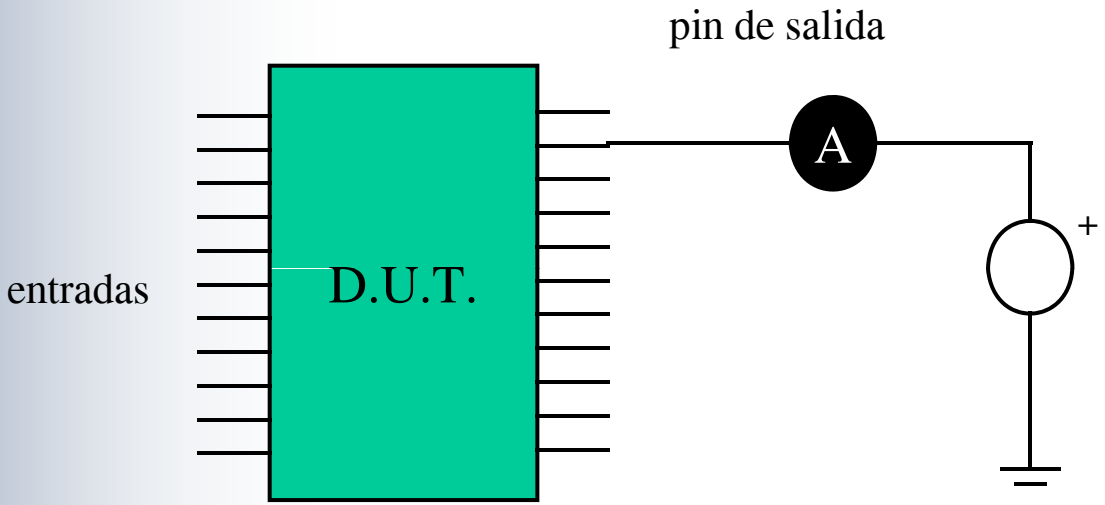
Figura 10



Medida de tensiones

Este esquema es válido tanto para entradas como para salidas

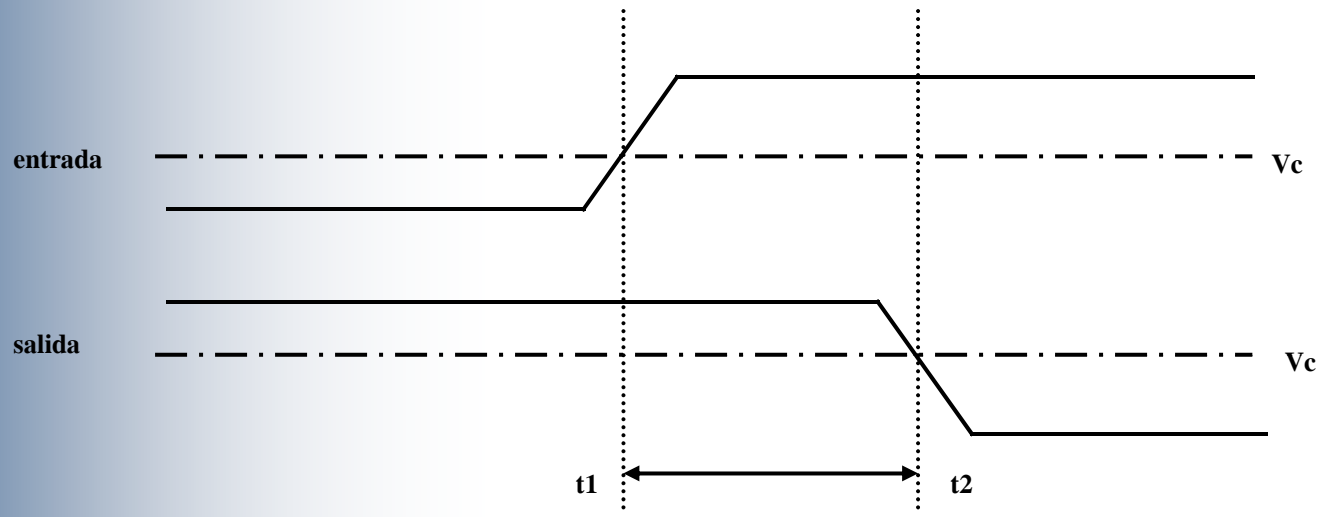
Figura 11



Medida de corrientes

Este esquema es válido tanto para entradas como para salidas

Figura 12.a



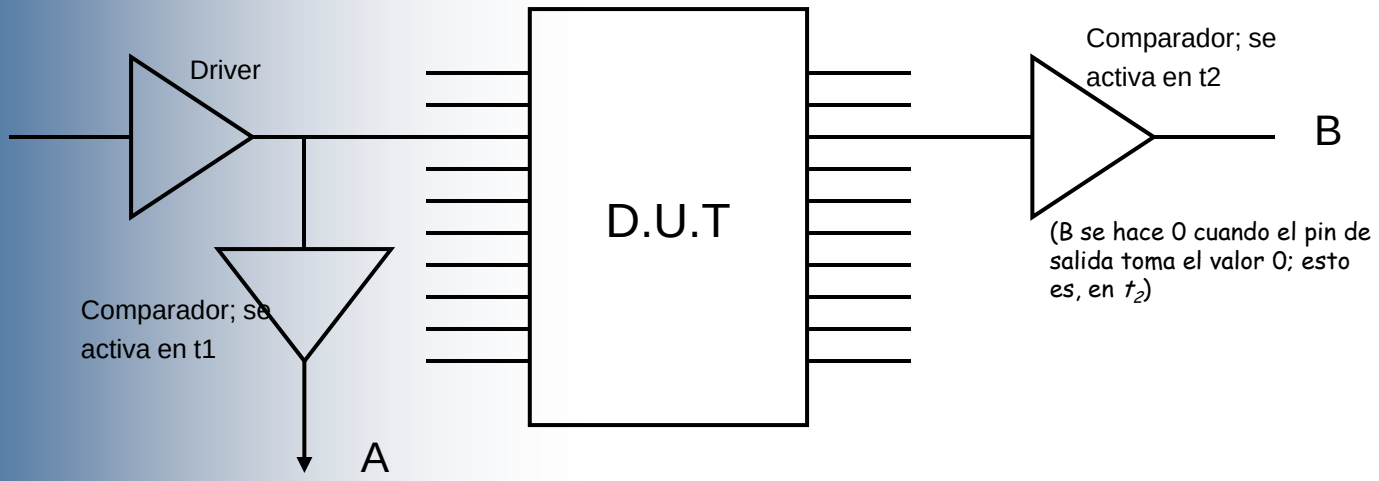
Vc: Tensión de comparación

Medida de tiempos de propagación

Tiempo comprendido entre los dos pasos de la entrada y de la salida por la tensión de comparación

Ver figura 12.b donde se muestra el circuito que realiza la medida del tiempo de propagación

Figura 12.b

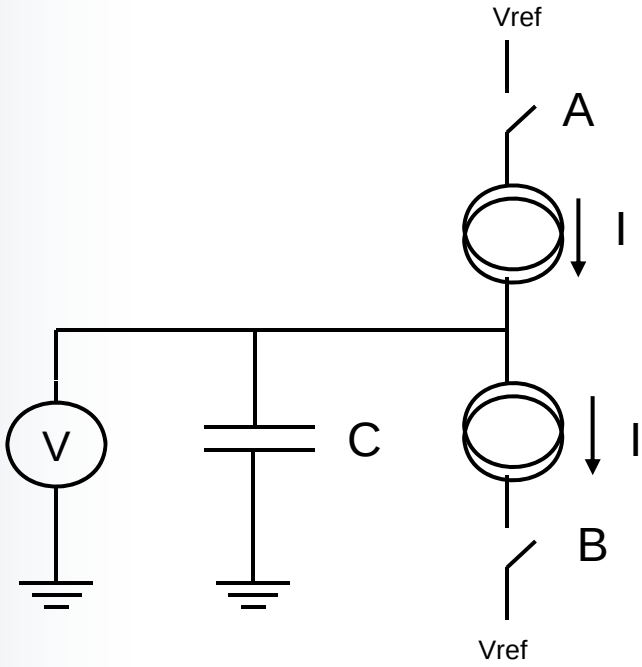


Medida de tiempos de propagación

Las señales A y B se generan a partir de los pasos de la entrada y la salida por el nivel de comparación.

Ver figura 12.c donde se muestra el circuito que mide el tiempo de propagación a partir de A y B.

Figura 12.c



(A=1 significa *switch* cerrado; A=0 significa *switch* abierto).

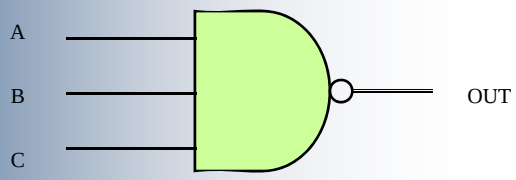
Se supone que C está inicialmente descargada.

Medida de tiempos de propagación

Los dos switches gobernados por las señales A y B controlan la carga de la capacidad C. El valor final de la carga será proporcional al tiempo de propagación.

Figura 13

Detección de fallos stuck-at en una puerta NAND



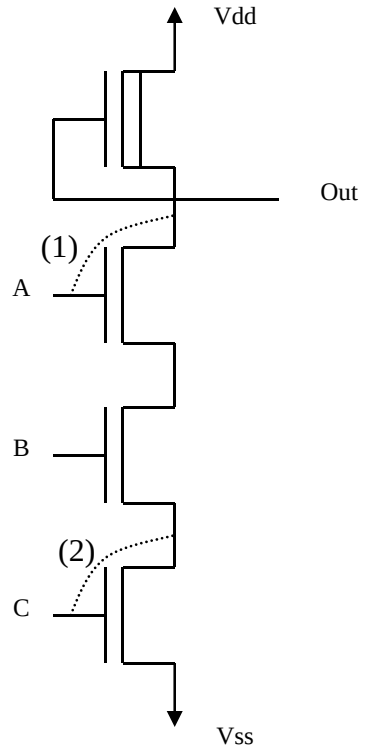
A	B	C	F0	F1	F2
1	1	1	0	1	0
0	1	1	1	1	0
1	0	1	1	1	1
1	1	0	1	1	1

A,B y C entradas del circuito
F0: salida correcta del circuito sin fallos
F1: salida (A stuck-at 1
F2: salida (A stuck-at 0)

El primer vector permite detectar el fallo (A stuck-at 0) mientras que el segundo detecta el fallo (A stuck-at 1)

Figura 14

Detección de fallos en una puerta NMOS



(1) Y (2) representan fallos de inteconexión entre la línea de entrada y un nodo de la red de transistores

Figura 15

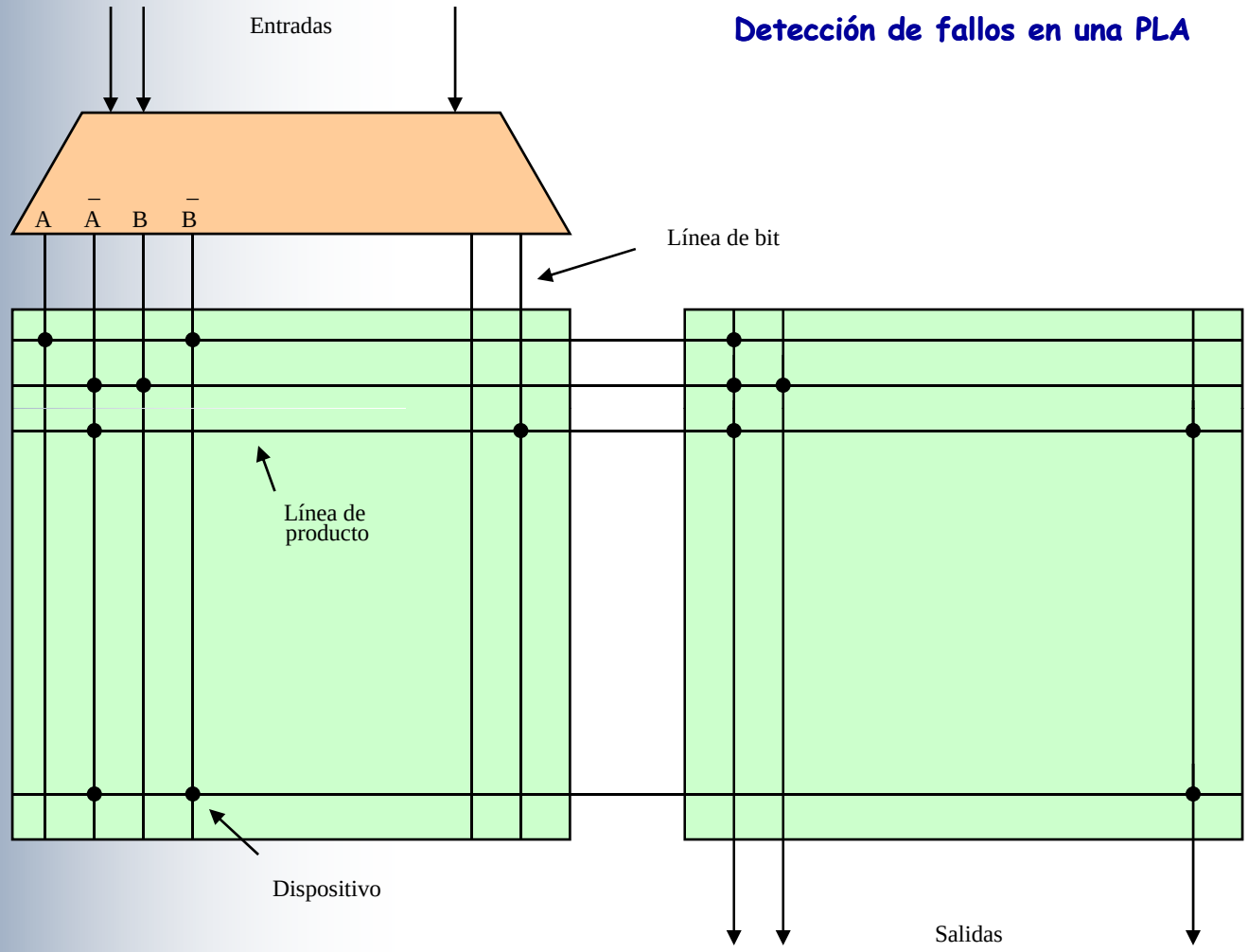

Detección de fallos en una PLA
Entradas
A
$\bar{A}$
B
$\bar{B}$
Línea de bit
Línea de producto
Dispositivo
Salidas

Figura 16

Reducción de fallos

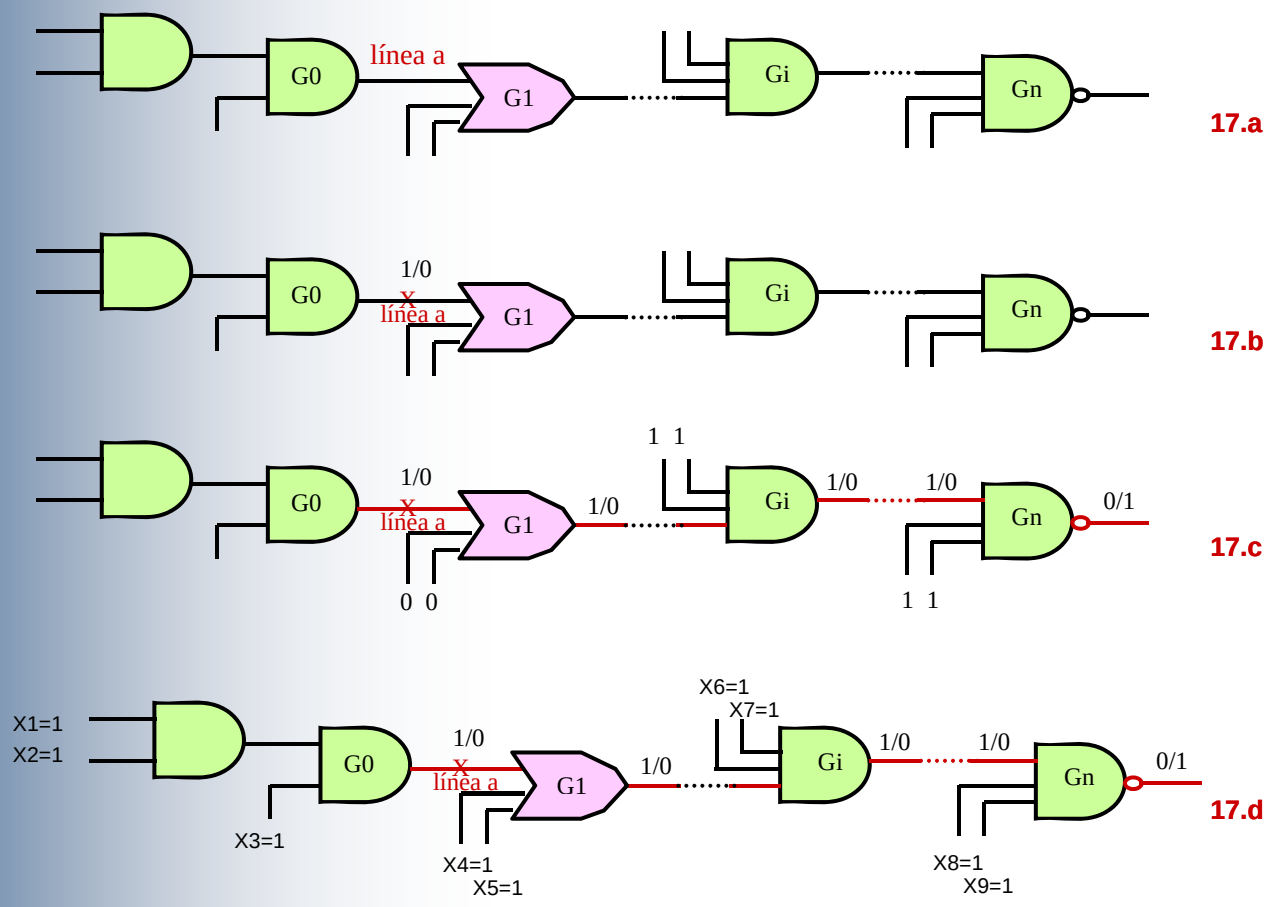
Classes	A	B	C	OUT
1) A/0, B/0, C/0, OUT/1	1	1	1	0/1
2) A/1, OUT /0	0	1	1	1/0
3) 6/1 t OUT /0	1	0	1	1/0
4) B/1, OUT/0	1	1	0	1/0

X/0 significa línea X stuck-at 0
X/1 significa línea X stuck-at 1

0/1 en la columna OUT significa:
OUT=0 en ausencia de fallo.
OUT=1 en presencia de fallo

Sensibilización de caminos

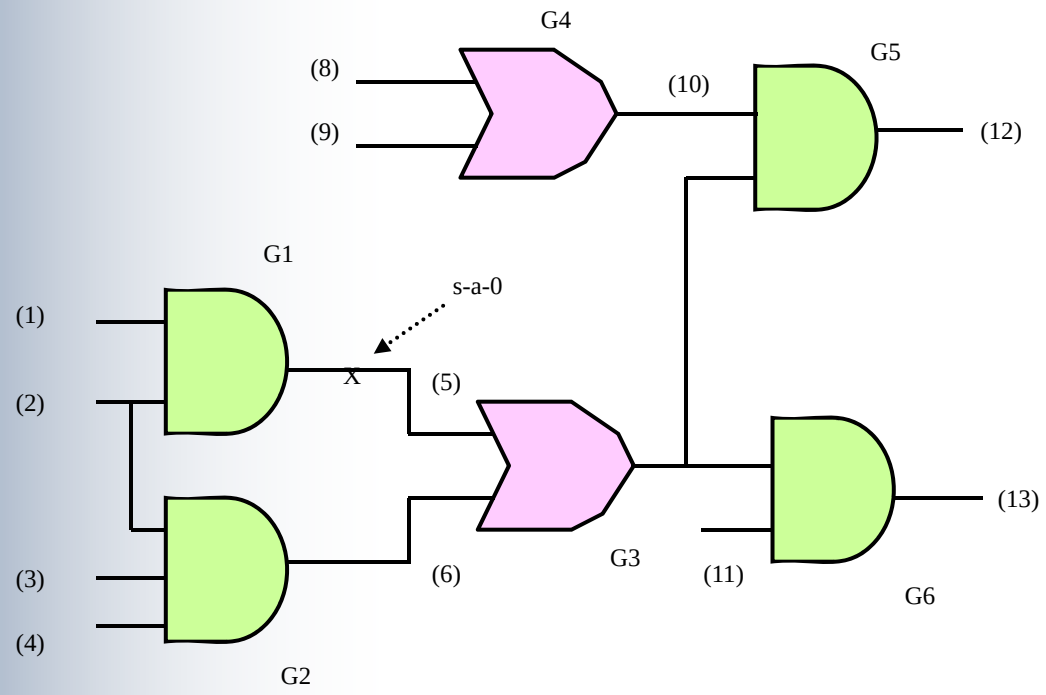
Figura 17



	x1	x2	x3	x4	x5	x6	x7	x8	x9	out
Vector de test:	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**

Figura 18

Ejemplo de generación de vectores de test



Obtención de los vectores de test

Figura 19

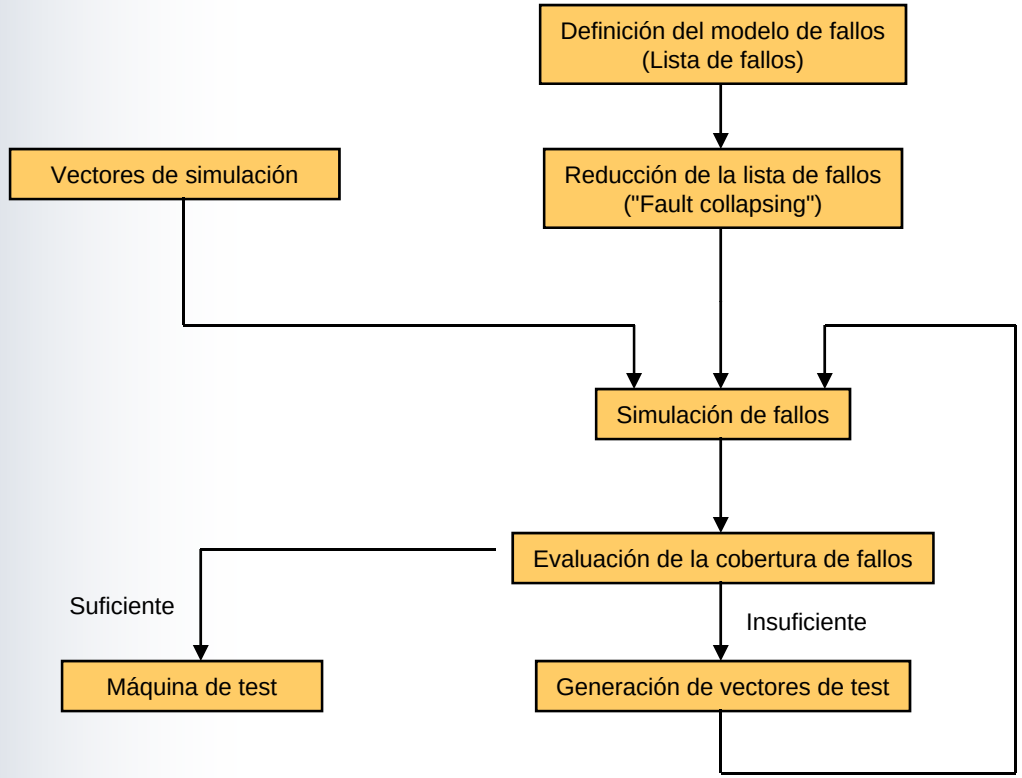


Figura 20

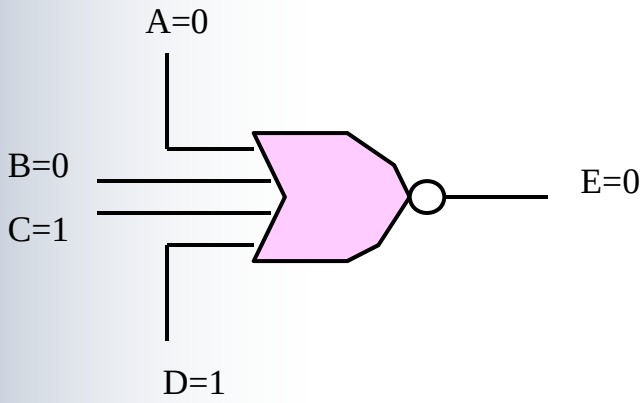
Simulador paralelo

Bit 0: máquina libre de fallos.
Bit i-ésimo: máquina con la presencia del fallo i-ésimo

Palabra 1	Palabra 2	Estado
0	0	0
1	0	1
1	1	X

Figura 21

Simulador deductivo



$$L_A = \{a, e\}$$
$$L_B = \{b, c\}$$
$$L_C = \{a, b, c, d\}$$
$$L_D = \{a, d, f\}$$
$$L_E = \left(\left(\overline{L_A \cup L_B}\right) \cap L_C \cap L_D\right) \cup \{E/1\}$$

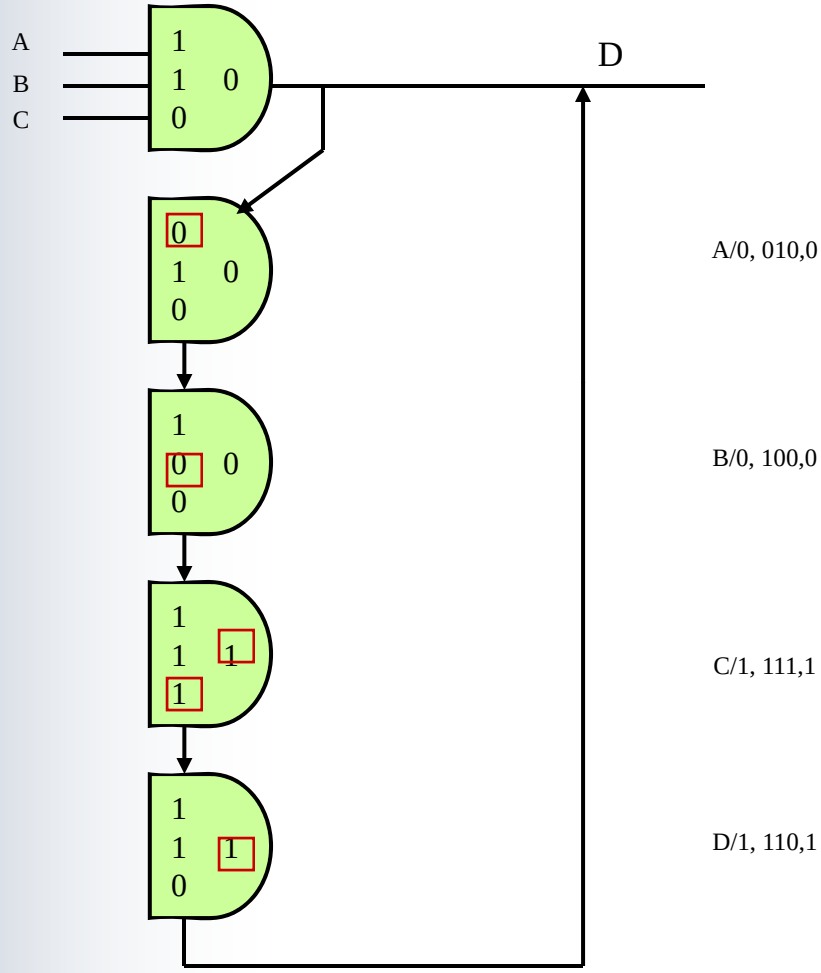
Figura 22

Comparación entre la simulación de fallos paralela y deductiva

Circuito	Nº de fallos simulados	Nº de vectores de test	Segundos CPU Deductiva	Segundos CPU Paralela
1) Conversor serie-paralelo	572	427	433	321
2) Corrector errores	894	412	642	201
3) Conversor paralelo-serie	559	348	252	245
4) Decodificador + secuenciador	886	893	352	360
5) Dial pulser sequencer	295	254	32	--
6) Decoder + match unit	1065	161	43	97
7) ALU	2147	377	510	--
8) Unidad memoria I	2582	200	8361	--
9) Unidad memoria II	2361	16	326	790
10) Procesador	9469	134	8673	--

Simulación concurrente de una NAND

Figura 23



Fin del capítulo 9