Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Diseño de las estructuras de prueba para un microprocesador RISC de 32 bits

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Eddy Morales Rodríguez

Cartago, Febrero de 2005

INSTITUTO TECNOLOGICO DE COSTA RICA

ESCUELA DE INGENIERIA ELECTRONICA

PROYECTO DE GRADUACIÓN

TRIBUNAL EVALUADOR

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Ing. Jose Alberto Diaz García

Profesor lector

Ing. Roberto Pereira Arrollo

Profesor lector

Ing. Alfonso Chacón Rodríguez

Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Cartago, 27 de enero de 2005

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Cartago, febrero de 2005

Eddy Morales Rodríguez
Céd: 1-1036-0570

RESUMEN

El objetivo de este proyecto consistió en estudiar los métodos y el uso de las herramientas que intervienen en el diseño de pruebas para aplicarse al microprocesador Tesla, diseñado en la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica como parte de la iniciativa del desarrollo de proyectos en el área de circuitos integrados de alta escala de integración, llevados a cabo en el laboratorio VLSI. Para esto se siguió un flujo de diseño de acuerdo a las tendencias modernas de Design-for-Test (DFT) y se utilizaron las herramientas destinadas para este fin de la compañía Mentor Graphics. También se requería que las herramientas quedaran listas para su uso y que se dejara un procedimiento básico a seguir para aplicar a futuros proyectos de diseño.

Palabras clave: Design-for-Test o DFT; flujo de diseño digital; VLSI

ABSTRACT

This project is inscribed within a plan devoted to the motivation of research in the area of Very Large Scale Integration (VLSI) design. To accomplish this, the School of Electronics Engineering at the Costa Rica Institute of Technology has created the VLSI Laboratory for Development of Integrated Circuits. The objective of the current project was to study the methods and tools issues related to the factory testing of a microprocessor (Tesla) designed in the School. The procedure follows a modern design flow that fits into the current Design-for-Test trends. To properly achieve this task, DFT tools from Mentor Graphics were used, and also the setup of this tools and a basic procedure of DFT process was required for a further use in design projects.

Keywords: Design-for-Test, digital design flow, VLSI

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1: Introducción

1.1 Definición del problema y su importancia

La Escuela de Ingeniería en Electrónica, con el fin de involucrarse en el área de diseño en Microelectrónica, ha llevado a cabo un proyecto en el laboratorio de VLSI que consiste en el diseño de un microprocesador RISC basado en arquitectura MIPS.

Generalmente este tipo de proyectos sigue un flujo de diseño como el mostrado en la figura 1.1 y que es recomendable seguir ya que marca la metodología hasta la implementación física del diseño. Ya se habían concluido dos etapas que sirvieron de antecedentes al presente trabajo: la primera consistió en el desarrollo de un modelo de transferencia de registros (RTL) del microprocesador. La segunda etapa consistió en la depuración de este modelo y la implementación de una interfaz que se encargara del control de memoria y puertos, así como también del manejo de periféricos. Con todo esto se quería demostrar que el diseño del microprocesador es funcional al probarlo en módulo FPGA.

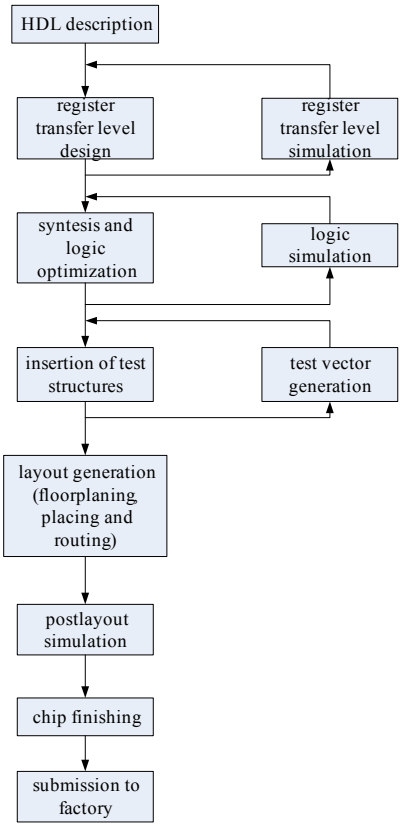


Figura 1.1 Flujo de diseño digital [6]

Siguiendo con el flujo de diseño y fabricación de circuitos integrados de muy alta escala de integración (VLSI) se encuentra la inserción de estructuras de prueba. Como es sabido, en la manufactura de productos semiconductores es importante proveer un método para medir la calidad y la confiabilidad del producto terminado, ya sea que el proceso de prueba se dé en el producto mismo, proceso conocido como *built in self-test*, o a través de un sistema de prueba (*tester*). Por lo tanto esta etapa tiene como objetivo funcional identificar las unidades defectuosas que puedan aparecer por el proceso de fabricación. Antes de su implementación física, el diseño original debe modificarse para agregar las estructuras de prueba; después de la fabricación, estas estructuras son utilizadas en la etapa de pruebas que, bajo ciertas condiciones de entrada, determinan diferentes tipos de fallas físicas, todo esto esta de acuerdo con las prácticas modernas de *testing* y a la aplicación de los conceptos de *Design-For-Test* (DFT) que se utilizan en la actualidad.

La importancia de este proyecto es de nivel académico y busca que la Escuela de Ingeniería Electrónica promueva los proyectos de investigación y desarrollo, en este caso, en el área del diseño de circuitos de alta escala de integración y que al mismo tiempo dé a los estudiantes capacidades que lo vuelvan más competitivo en el mercado tecnológico mundial.

1.2 Solución seleccionada

El Laboratorio de VLSI cuenta con una base computacional ya montada y las herramientas para DFT son de la casa de Mentor Graphics Corporation por lo tanto el proyecto se limita al uso de estas herramientas para realizar todo el proceso de DFT.

Como estrategia para alcanzar un buen diseño de las estructuras se siguió un flujo de diseño contemporáneo que aplica principios de DFT que mejoran las posibilidades de la circuitería de tener un proceso de prueba satisfactorio además de hacerla controlable y observable.

Entre las estrategias de DFT conocidas la que da mejores resultados y la que se utilizó es la conocida como DFT estructurado cuya meta es incrementar la controlabilidad y observablilidad de un circuito. Existen varios métodos para cumplir con esto: el más común es la técnica

llamada *scan design*, pero para poder cumplir la misma se deben realizar previamente algunas tareas propias del DFT, que se muestran en la figura 1.2.

Lo primero que se debe hacer antes de empezar el proceso es llegar a comprender lo relacionado con los conceptos básicos de DFT, las herramientas y el *testing* para poder tomar decisiones importantes y que mejor convengan al diseño. También conocer las herramientas que se utilizan provee mayor ventaja. Así que el proyecto comenzó con un estudio general sobre el tema y de las herramientas para luego continuar con la aplicación de lo aprendido al diseño.

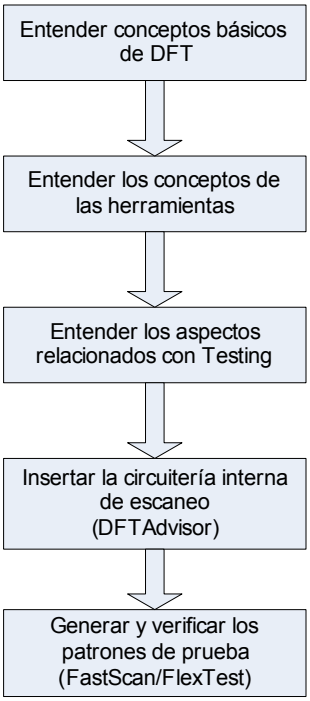


Figura 1.2 Tareas de DFT aplicadas en el diseño de circuitos integrados

Capítulo 2: Meta y objetivos

2.1 Meta

Este proyecto tiene como meta obtener un procedimiento de DFT válido que pueda ser aplicado a dispositivos digitales diseñados en el laboratorio VLSI.

2.2 Objetivo general

Diseñar e introducir las estructuras de prueba en el microprocesador Tesla RISC de 32 bits y generar los vectores correspondientes que van a ser aplicados al mismo.

2.3 Objetivos específicos

1. Obtener una versión optimizada y sintetizada del microprocesador Tesla aplicando para ello las herramientas de síntesis seleccionadas.
2. Obtener una versión modificada del microprocesador Tesla que incluya las estructuras prueba.
3. Generar y obtener los patrones de prueba para la versión modificada del microprocesador Tesla.
4. Escribir los scripts necesarios que permitan la aplicación del flujo DFT a un diseño.
5. Escribir una guía rápida de uso de las herramientas para completar un procedimiento DFT.

Capítulo 3: Marco Teórico

3.1 Métodos de diseño de circuitos integrados

Cuando se empieza el diseño de un chip se tienen muchas opciones: generalmente se empieza con la descripción del diseño a nivel de comportamiento y se pasa a nivel RTL para luego llegar al nivel lógico y, posiblemente a nivel estructural para finalizar con el nivel de layout [15].

3.1.1 Síntesis de Comportamiento

A nivel de comportamiento, la operación del sistema es establecida sin tener que especificar la implementación. Por ejemplo, no se necesita especificar el *pipline* requerido para alcanzar cierta velocidad.

En principio, un compilador de comportamiento debe lograr las siguientes operaciones:

- Decidir y asignar los recursos basados en los requerimientos de área y temporización.
- Insertar los registros de *pipeline* para cumplir con las restricciones de temporización.
- Crear el microcódigo y la lógica de control.

3.1.2 Síntesis RTL

Un programa de síntesis RTL toma un diseño en descripción RTL y lo convierte en un conjunto de registros y lógica combinacional. En esta parte del proceso de diseño la arquitectura ha sido capturada.

Comúnmente, la descripciones RTL son capturadas usando un Lenguaje de Descripción de Hardware (HDL). En general, en un diseño RTL usando HDL se deben especificar los siguientes atributos:

- El flujo de control usando sentencias if-then-else y case
- Iteraciones
- Jerarquía
- Anchos de palabra, vectores de bits

- Operaciones secuenciales vs. paralelas
- Especificación y distribución de registros
- Operaciones aritméticas, lógicas y de comparación

3.1.3 Optimización de la lógica

Programas para la optimización de la lógica toman las descripciones lógicas como las generadas por una síntesis RTL o generadas directamente desde un nivel de lógica y optimiza la red de compuertas requerida para implementar la función especificada por la descripción lógica o por una biblioteca.

El objetivo de la optimización lógica es manipular la lógica para alcanzar las especificaciones de velocidad y área o una combinación de las dos. Generalmente, los sistemas de optimización dividen el problema en dos etapas:

- Una etapa independiente de la tecnología en la cual la lógica es optimizada de acuerdo con técnicas algebraicas o booleanas.
- Una fase de mapeo en la que la descripción derivada del paso anterior es traducida a las celdas estándar de una tecnología específica o a elementos programables (FPGA).

3.1.4 Síntesis de estructural a *layout*

Una vez que la red de compuertas lógicas y registros está disponible, el diseño es automáticamente convertido a *layout*. Los distintos softwares disponibles para esta tarea están muy bien desarrollados y han sido refinados durante los últimos 15 o 20 años. En este proceso se identifican dos fases: colocación y enrutamiento.

La colocación es una tarea que trata de ubicar los módulos adyacentes uno junto al otro para minimizar el área o los ciclos de tiempo.

El enrutamiento toma los módulos ya colocados y una lista de conexiones y los interconecta con cables.

3.1.5 Síntesis de *layout*

El layout de una estructura regular como registros, RAM, ROM, PLA, multiplicadores es generado por software. Estos programas toman una serie de parámetros de entrada y auto-

máticamente crean un layout físico. Algunos sistemas crean una máscara real ajustada para un proceso en particular, otros crean un layout simbólico que puede ser ajustado a una tecnología en particular. Algunos ejemplos de *layouts* se muestran en los anexos B.2, B.3 y B.4

3.2 Definición de *testing* [1]

Testing es cuando un estímulo de entrada conocido es aplicado a una unidad en estado conocido y una respuesta conocida o predecible puede evaluarse. En general la respuesta conocida de un circuito es comparada contra la respuesta esperada ideal. Este concepto es aplicado tanto a chips como a tarjetas y sistemas.

Cualquier tipo de *testing* requiere de aplicar un estímulo conocido en los pines de la unidad que esta en prueba. Estos estímulos de entrada pueden ser aplicados en el simulador o en un *tester*; en sistemas digitales, estos estímulos son los unos y ceros lógicos conocidos como vectores. La capacidad de aplicar los estímulos a la unidad bajo prueba es conocida como controlabilidad.

Para que un *testing* sea válido, la unidad en prueba debe estar en un estado conocido. La unidad debe de actuar como un operador lineal simple. En otras palabras, si "A" es aplicado, siempre ocurre "B". Para poner un circuito en un estado conocido se pueden aplicar una serie de valores de inicialización que incondicionalmente resulten en un estado conocido, o se puede aplicar alguna clase de *sets* y *resets* al hardware que dejen la lógica en un estado conocido.

También, para que un *testing* sea válido, se debe poder evaluar la respuesta esperada. En sistemas digitales, esta determinación requiere de tener una noción preconcebida del funcionamiento del sistema, por ejemplo por medio de una tabla de la verdad, o por simulación de vectores para obtener los vectores de la respuesta esperada contra los cuales se puede comparar la respuesta del circuito. La capacidad de evaluar la respuesta de salida del circuito en prueba es conocida como observabilidad.

3.3 Tipos de *testing*

3.3.1 Prueba de funcionalidad

La prueba de funcionalidad (o comportamiento) es usada para verificar que el modelo o la lógica se comportan como fueron diseñados. También se conoce como prueba de verificación del diseño. Esta prueba puede incluir temporización o consumo de potencia como parte de la funcionalidad estándar.

3.3.2 Prueba estructural

La prueba estructural es usada para verificar la topología del chip manufacturado. Dicho de otra forma, dado un circuito "bueno" antes del proceso de manufactura, la prueba estructural puede usarse para verificar que todas las conexiones están intactas, y que todas las tablas de la verdad a nivel de compuertas son correctas después del proceso de manufactura.

3.3.3 Prueba combinacional exhaustiva y seudo-exhaustiva

La prueba combinacional exhaustiva o 2^n es usada para verificar cómo la porción combinacional del modelo se comporta cuando cada juego posible de valores es aplicado a los puertos de entrada, aun si algunos de los vectores aplicados no tienen significado funcional. La prueba seudo exhaustiva es la aplicación de una porción de todos los valores lógicos 2^n posibles.

3.3.4 Prueba exhaustiva completa

La prueba exhaustiva completa o $2^{(n+m)}$ es similar a la prueba combinacional exhaustiva excepto que existen algunos elementos secuenciales que mantienen un estado dentro del circuito. Aplicando todos los valores combinacionales posibles a los pines de entrada simplemente no es suficiente para caracterizar el diseño secuencial. Una máquina de estados con M elementos requiere 2^m pruebas para probar todas las secuencias. Para probar un circuito combinacional se requiere aplicar 2^n valores, considerando también todas las posibles secuencias. Entonces, se deben de multiplicar las entradas combinacionales y los estados secuenciales para

obtener $2^{(n+m)}$ pruebas. Este tipo de prueba se debe aplicar sólo para propósitos de caracterización, o cuando los vectores son aplicados sin el entendimiento del circuito.

3.4 Generación automática de patrones de prueba (ATPG)

La generación automática de pruebas (ATPG por sus siglas en inglés, *Automatic Test Pattern Generation*) es la aplicación de un software basado en un algoritmo para generar vectores y conjuntos de vectores. El término ATPG es generalmente aplicado a la generación de vectores tipo estructural (vectores generados bajo el criterio de detección de fallas), sin embargo también se ha aplicado (de forma incorrecta) a la generación de vectores de tipo funcional (por ejemplo: IEEE 1149.1 "JTAG") basados en la descripción de comportamiento del modelo.

En general, el proceso de ATPG incluye tareas que tienen que cumplirse antes y después de la aplicación de la herramienta ATPG. Estas tareas abarcan la preparación del ambiente computacional para la herramienta, la preparación de la herramienta para aceptar la descripción del diseño, la preparación de la descripción del diseño para el análisis, y el procesamiento de los vectores creados para alinearlos con el formato de datos y control del *tester*. En medio de todo esto esta la herramienta que genera los vectores a partir de la descripción del diseño. Esta parte es la que se identifica como el proceso ATPG.

La parte central del proceso ATPG tiene una secuencia o flujo. Este flujo ha sido desarrollado a través del tiempo aplicando optimizaciones para continuamente mejorar la "cobertura de fallas", el "tiempo de ejecución" y la "simplicidad de uso" del proceso.

El primer paso después de procesar la descripción del diseño tiene que ver con establecimiento del modelo de fallas que se va a usar, y enumerar las fallas que se van a probar.

La lógica que una falla puede contener, tiene que ser valorada, evaluada o analizada para buscar una ruta de propagación de la falla hacia un punto de observación. El paso de buscar un punto de observación es generalmente el análisis más difícil ya que involucra trazar hacia adelante a través de toda la lógica desde la localización de la falla hasta un punto donde el efecto de la falla puede ser observado. Luego, la lógica es analizada de nuevo pero ahora hacia atrás,

desde donde se localiza la falla hasta los puntos de control con el objetivo de establecer los valores lógicos necesarios para excitar la falla.

3.5 Modelado de fallas

Los defectos físicos son lo que realmente le ocurre a un dispositivo cuando se habla de fallas. Estos se pueden categorizar como defectos de interconexión, defectos de encapsulado, problemas en el proceso y demás. Dichos defectos tienen su fuente en algunos aspectos como contaminación local, máscaras incorrectas, errores en el proceso, insuficiencia de dopado, daños en los dados de silicio, y errores de interconexión.

El modelado de fallas es la interpretación de los defectos físicos como modelos matemáticos que pueden ser aplicados en algoritmos y comprendidos por un software simulador con el propósito de proveer un valor para la medición de calidad. Los modelos de fallas más comunes para VLSI son el modelo en CD o "*stuck-at*", el modelo en CA o de "retardo", y el modelo basado en corriente o "*pseudo-stuck-at*". En el presente trabajo se utiliza únicamente el modelo en CD o *stuck-at* por ser el modelo de uso más común y el más sencillo de implementar en un futuro *tester*.

3.5.1 La falla stuck-at

Los primeros modelos de fallas fueron los que describían los posibles errores en los transistores. Cuando los diseños se hicieron tan grandes como para realizar análisis de transistores, se adoptó el modelo de fallas a nivel de compuertas, donde uno de los primeros modelos satisfactorios fue el de *stuck-at*.

Cuando una prueba estructural es usada para verificar la topología, y no la temporización, de un chip manufacturado, el modelo de fallas utilizado es el modelo *stuck-at*, ya que este verifica que cada conexión en un nodo, en una red o en una compuerta no está cortocircuitada a un 1 lógico o a un 0 lógico.

En realidad, la existencia de un defecto tipo *stuck-at* combinacional en el silicio puede o no puede afectar inmediatamente la lógica implícita, y el efecto de la falla puede o no trasladarse a través del circuito y modificar la salida esperada. Este tipo de falla es activada cuando

el pin defectuoso, nodo o conexión de compuerta, durante una prueba funcional, adquiere un valor lógico que es opuesto al valor descrito por la falla; por ejemplo si se dice que un nodo está cortocircuitado a un 0 lógico ("*stuck-at* 0") se debe de propagar un 1 lógico, la información no es provechosa si se propaga un 0 lógico. Este tipo de fallas son detectadas cuando el comportamiento de un circuito en el punto de observación es diferente al valor esperado debido a la propagación del efecto de la falla *stuck-at* y no a otra razón. Lo que esto significa es que una compuerta puede funcionar erróneamente, pero la falta no es detectada hasta que el efecto de la misma se refleje en el punto de observación.

Capítulo 4: Procedimiento metodológico

Para resolver el problema principal se decidió dividir las actividades en tres etapas; la primera consistió en la familiarización con el ambiente y el uso de los programas; la segunda fue la aplicación de las herramientas de *Design For Test* (DFT) al modelo RTL del microprocesador; en la última etapa se trataría de dar una idea para el diseño inicial del un tester capaz de utilizar los resultados de la etapa anterior.

4.1 Familiarización con el ambiente y programas.

1. Diseño de un modelo sencillo en lenguaje Verilog que contenga poca cantidad de elementos secuenciales y combinacionales.
2. Aplicación al modelo simple de las herramientas de síntesis.
3. Inserción de las cadenas de escaneo al modelo sintetizado
4. Generación de los vectores de prueba.

4.2 Aplicación de las técnicas DFT al microprocesador Tesla

1. Síntesis del modelo con Leonardo Spectrum y las bibliotecas de AMS requeridas.
2. Identificación de los elementos secuenciales escaneables e inserción de las cadenas de escaneo en el microprocesador Tesla con DFTAdvisor.
3. Simulación con Model Sim del modelo modificado para comprobar si se mantiene su funcionalidad con los nuevos elementos insertados.
4. Generación y simulación de los patrones de prueba en el modelo modificado con FlexTest o FastScan.

4.3 Inicio del diseño del tester

1. Interpretación de los patrones de prueba generados para su posterior aplicación en el *tester*.
2. Empezar con el diseño del hardware y software necesario para la aplicación de las pruebas al microprocesador manufacturado, basado en los patrones de prueba obtenidos.

Capítulo 5: Descripción detallada de la solución

5.1 Análisis de soluciones

Para llegar a una solución primero se analizaron las diferentes técnicas de *testing* y más específicamente para este caso las técnicas que soporta las herramientas Mentor Graphics. Estas herramientas soportan las estructuras de prueba de escaneo completo, escaneo parcial, escaneo dividido y puntos de prueba. La escogencia de qué técnica utilizar está determinada por restricciones del diseño, del nivel de controlabilidad y observabilidad que se desee alcanzar entre otros. A continuación se describe de forma general en que consiste cada uno de estas técnicas y como puede o no aplicarse al diseño en cuestión.

5.1.1 Puntos de prueba

Se pueden dar casos, aún en circuitos con algún tipo de estructura de prueba, que un diseño pueda contener cierto número de puntos que son difíciles de controlar y observar, agregando circuitería especial estos puntos se pueden convertir en observables y controlables. En la figura 5.1.a se muestra un caso [7] donde se pueden aplicar los puntos de prueba, se observa que una de las entradas de la compuerta OR esta cortocircuitada a Vcc lo que bloquea la propagación del efecto de cualquier falla de la circuitería que alimenta la otra entrada a través de la compuerta. Al mismo tiempo existe un valor de 1 constante en la salida de la compuerta lo que provocaría que cualquier circuito conectado a ésta quedaría sin control.

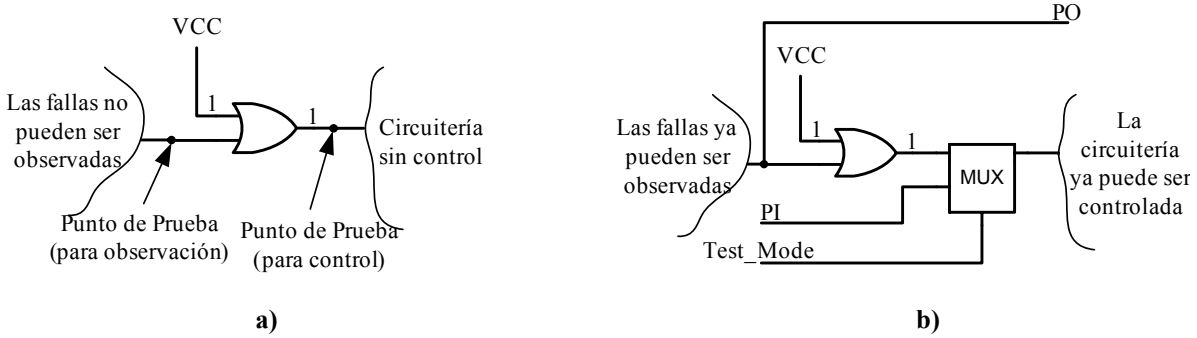


Figura 5.1 a) Circuito con problemas de observación y control; **b)** Circuito con puntos de prueba

Entonces, como se muestra en la figura 5.1.b, al agregar una salida primaria en el punto de prueba para observación se tiene acceso directo al valor de la señal. En el punto de prueba

de control se agrega un multiplexor controlado por la señal Test_Mode y la entrada primaria lo que permite tener control de la circuitería asociada.

Esta técnica mejora las posibilidades de detectar fallas y se usa como complemento de cualquiera de las otras técnicas; nunca se usa como única estructura de prueba. Unas de las desventajas es que hace uso de entradas y salidas primarias que en ciertos casos es indeseable. En el presente proyecto se activó la identificación de los puntos de prueba tal y como se observa en la línea 4 de la figura 6.11, sin embargo no se identificó ninguno.

5.1.2 Escaneo dividido

El proceso ATPG en diseños muy grandes y complejos puede ser impredecible, este problema es cierto cuando existen diseños con estructuras de escaneo o secuenciales muy grandes. En estos casos se prefiere dividir las estructuras de prueba según la jerarquía del diseño; los diseños que ya están fragmentados en bloques son los que se benefician más de esta técnica.

El escaneo dividido agrega observabilidad y controlabilidad por medio de una cadena de escaneo de partición jerárquica. Esta cadena de escaneo de partición consiste en una serie de celdas de escaneo conectadas en la periferia de una partición del diseño y se forma convirtiendo los elementos secuenciales en celdas de escaneo en las entradas que tienen poca controlabilidad o salidas que tienen poca observabilidad desde afuera del bloque.

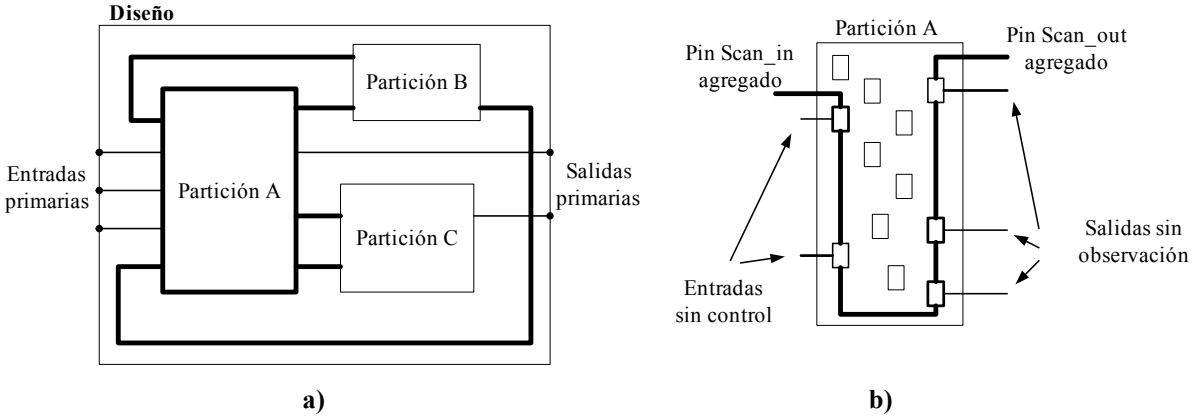


Figura 5.2 a) Ejemplo de un diseño dividido; **b)** Circuitería de escaneo dividido agregada a la partición A

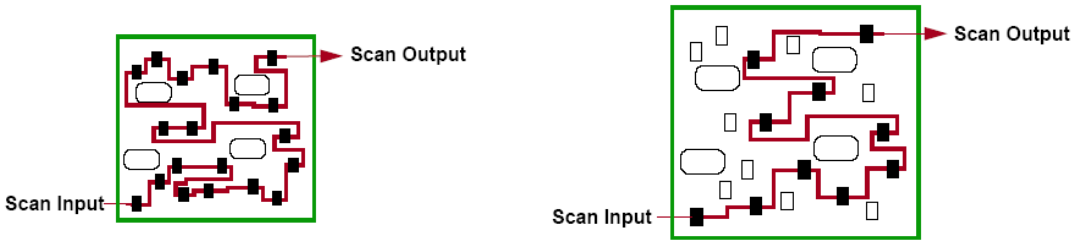
En la figura 5.2.a se muestra el ejemplo [7] de un diseño dividido en tres partes A, B y C. Las líneas más resaltadas indican las entradas y salidas de la partición A que no son directamente controlables u observables a nivel del diseño. La cadena de escaneo de partición consiste fundamentalmente en dos tipos de elementos: los elementos secuenciales conectados directamente a las entradas primarias sin control de la partición, y los elementos secuenciales conectados directamente a las salidas sin observación de la partición como se puede observar en la figura 5.2.b. También la partición adquiere dos nuevos pines, scan_in y scan_out.

Esta técnica por sí solo da muy bajos niveles de observabilidad y controlabilidad por lo que es común usarla en conjunto con el escaneo completo o parcial aprovechando los elementos secuenciales que no se utilizan en la cadena de escaneo de partición. También, como se mencionó antes, esta técnica funciona mejor aplicada en diseños muy complejos y grandes como en sistemas embebidos, es por estas razones que se descarta su uso como estructuras de prueba para el microprocesador Tesla.

5.1.3 Escaneo Completo y escaneo parcial

El escaneo parcial y el completo son técnicas similares y consisten en reemplazar los elementos secuenciales por celdas de escaneo, éstas se conectan entre si para formar una cadena de escaneo que se distribuye a través de todo el diseño.

El escaneo completo consiste en utilizar todos los elementos secuenciales del diseño para formar la cadena de escaneo como se observa en la figura 5.3.a, en esta figura los elementos de escaneo son los rectángulos negros y los cuadros redondeados es la lógica combinacional.



a) **b)**

Figura 5.3 [7] Representación de **a)** escaneo completo y **b)** escaneo parcial

En el escaneo parcial sólo un porcentaje de los elementos secuenciales son convertidos a elementos de escaneo ya que en algunos casos existen restricciones de área o de temporización. En la figura 5.3.b se muestra el caso de un escaneo parcial, los rectángulos son elementos secuenciales presentes en el diseño, los rectángulos negros son elementos secuenciales que han sido convertidos en celdas de escaneo para formar la cadena de escaneo, y los cuadros redondeados es la lógica combinacional.

Una de las principales ventajas que tiene el escaneo parcial sobre el completo es que no causa tanto impacto en el área usada cuando se agregan los elementos de escaneo, cuando el área es una limitante entonces se prefiere su uso; igual pasa cuando existen restricciones de temporización muy exigentes, si por ejemplo no se quiere afectar la ruta crítica del diseño entonces los elementos secuenciales que pertenecen a esta ruta se dejan por fuera de la estructura de pruebas.

Pero para el caso del microprocesador Tesla el área a utilizar no es problema, por lo tanto se decidió hacer uso del escaneo completo, además esta técnica tiene ventajas adicionales muy importantes para el proyecto. Primero es un proceso altamente automatizado, cuando se utilizan las herramientas orientadas a insertar un escaneo completo, la intervención manual durante el proceso es muy poca. Segundo, es fácil de usar, se puede insertar la circuitería de escaneo y correr las herramientas de ATPG sin necesidad de la intervención de un experto en testing. Tercero, el escaneo completo es un método altamente efectivo, predecible y aceptado ya que da los mejores resultados en la cobertura de fallas y testing. Entonces con el uso de esta técnica se puede estar seguro que cualquier persona usuaria del laboratorio VLSI que aplique correctamente este procedimiento va a obtener resultados muy confiables, de aplicación simple y en poco tiempo.

5.1.4 Proceso ATPG

Debido a la utilización de la técnica de escaneo completo para las estructuras de pruebas la herramienta que se debe utilizar para la generación de los patrones de prueba debe ser FastScan, este software se utiliza sólo para estructuras de prueba de este tipo.

En la generación de los patrones de prueba se identificó únicamente la falla tipo *stuck-at* (también llamada de cd ya que su función es verificar los niveles de voltaje). Se decidió utilizar únicamente el reconocimiento de este tipo de falla pensando en una futura construcción del *tester,* debido a la naturaleza de la falla el *tester* tendría una construcción más sencilla y menos costosa que para uno destinado a medir corrientes o fallas de temporización.

Capítulo 6: Análisis de resultados

6.1 Familiarización con el ambiente y programas

Como primer paso en la solución, se diseñó un modelo a nivel RTL que fuera pequeño y simple con el fin de familiarizarse y entender el uso y manejo las herramientas de software, así que se optó por escoger un registro LFSR[1] de 16 bits ya que contiene sólo 16 *flip-flop* y unas pocas compuertas, lo que fue muy conveniente porque una estructura pequeña hizo manejable la exploración y comprensión de los detalles en el uso de las herramientas DFT y síntesis.

El registro LFSR escogido era de cuatro taps conectados en los *flip-flop* 1, 2, 4 y 15, tal y como se muestra en la figura 6.1; en esta figura se muestran los cuatro *flip-flop* tipo D que intervienen en la retroalimentación y su interconexión con compuertas xor, los demás *flip-flop* se omiten para simplificar el diagrama ya que su conexión es como la que existe entre los *flip-flop* 0 y 1 ó entre 3 y 4.

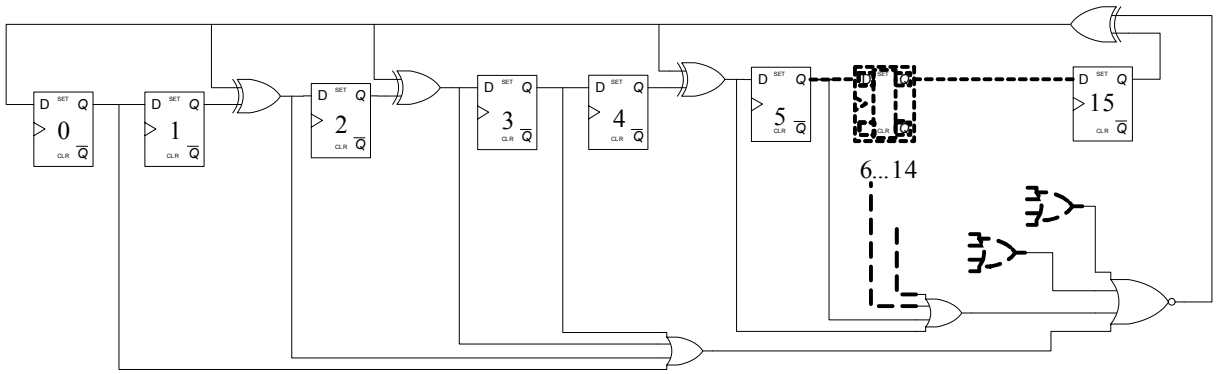


Figura 6.1 Modelo de registro LFSR de 16 bits (los *flip-flop* de 6 a 14 se omiten por simplificación en el dibujo)

Este registro fue modelado a nivel RTL utilizando para ello el lenguaje de descripción de hardware Verilog, el mismo que se utilizó para diseñar el microprocesador Tesla. Utilizando este modelo se simuló el funcionamiento del registro con el programa Model Sim y se ob-

[1]El registro LFSR es un registro de desplazamiento con retroalimentación lineal o, por sus siglas en inglés, Linear Feedback Shift Register, ver el Anexo B.1 para una explicación general de su funcionamiento.

tuvieron los resultados que se muestran en la figura 6.2. Se observa que el registro provee en su salida valores completamente distintos en cada flanco positivo del reloj. Estos se dicen que son valores pseudos aleatorios porque en cierta forma son predecibles y queda claro en los primeros ciclos de reloj según la figura 6.2; también se probó el registro en un módulo FPGA con una pequeña modificación y es que se disponía de solo una salida de 8 bits y para poder ver lo 16 bits del registro se agregó una lógica que permitía intercambiar en la salida entre los 8 bits más significativos y los 8 bits menos significativos del registro.

Lo importante aquí es saber que después de las modificaciones que sufre el modelo debido a todo el proceso de DFT, el comportamiento final del mismo debe coincidir con el comportamiento de este primer diseño.

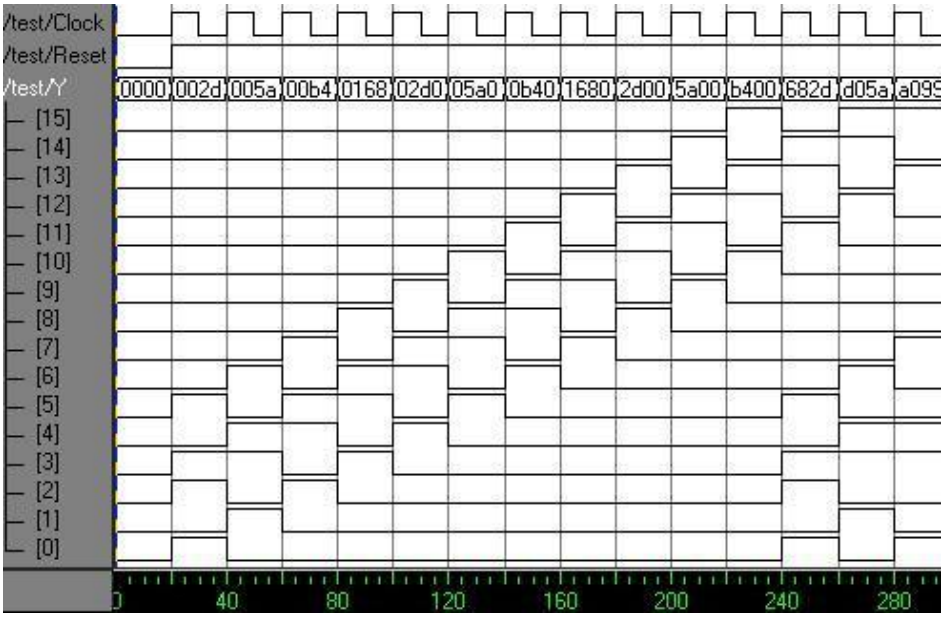


Figura 6.2 Resultados de la simulación para el registro LFSR (los valores de la salida Y están en hexadecimal)

El registro fue sintetizado con Leonardo Spectrum utilizando la biblioteca ami05 típica que utiliza para el análisis una temperatura de 27º C y 5 V. El proceso de síntesis toma un diseño digital a nivel RTL lo interpreta y reconoce cada elemento combinacional y secuencial y lo optimiza, al final se obtiene una descripción del diseño a nivel de compuertas contenido en un archivo Verilog.

Una vez sintetizado el modelo, se está listo para iniciar el proceso de DFT. En esta etapa se utilizó la interfaz gráfica (GUI, graphic user interface) de los programas para facilitar su uso

y la inspección de las múltiples opciones, también sirvió como base para elaborar una guía donde se explican los pasos a seguir para completar el flujo de DFT.

Los programas utilizados para DFT fueron diseñados para correr en plataforma UNIX y en este caso se utilizó el sistema operativo RED HAT LINUX 8. Este primer diseño también sirvió para poner a punto las variables del sistema necesarias para utilizar los programas de la familia de Mentor Graphics Corporation, más específicamente, las variables involucradas para la utilización de los programas DFTAdvisor, FastScan, FlexTest, VSim. En el archivo mgc_location_map es donde se definen estas variables.

El primer paso en el flujo DFT es la identificación e inserción de los elementos de escaneo. Para esto se utiliza el programa DFTAdvisor y su interfaz gráfica se invoca desde una ventana terminal con el comando:

```
>dftadvisor
```

Este comando sin ninguna opción adjunta presenta la ventana de opciones previa mostrada en la figura 6.3 que captura los argumentos para invocar al programa DFTAdvisor.

Figura 6.3 Ventana previa a la invocación de DFTAdvisor

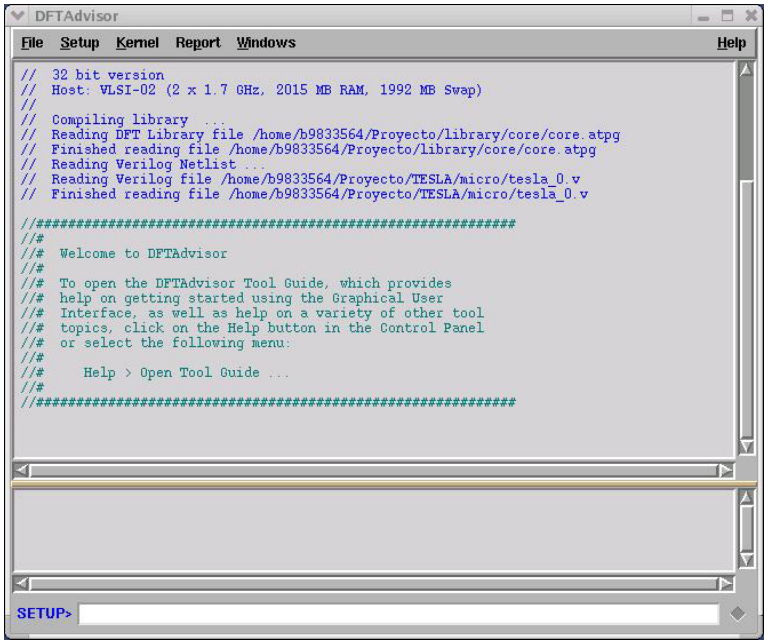
Para utilizar dicho programa hay que proveer cierta información imprescindible como el nombre del diseño y formato y la correspondiente biblioteca ATPG. Adicionalmente se puede

definir el módulo principal del proyecto, un archivo de comandos que el programa ejecutará al momento de iniciar y se puede indicar el nombre de un archivo para que el programa guarde todos los mensajes y comandos que utiliza.

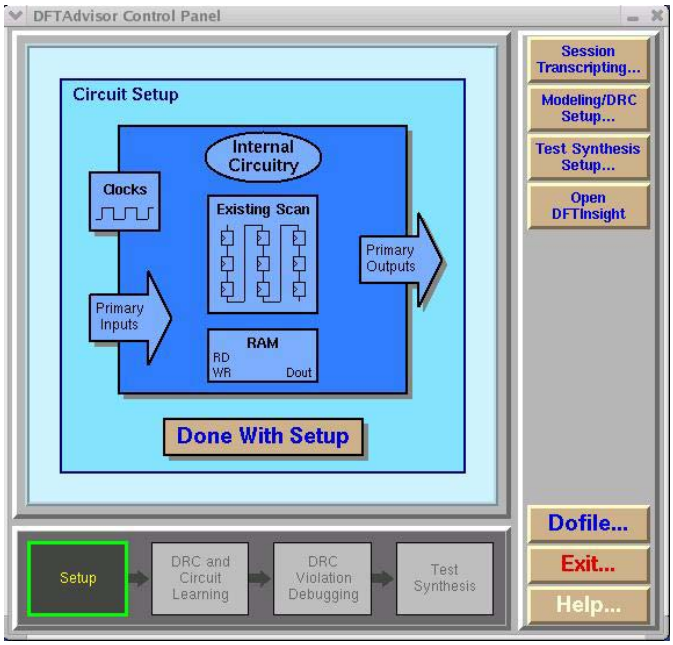
Para el caso de este diseño se utilizó la biblioteca ATPG incluida con el programa, ya que es compatible con la tecnología ami05 que se usó en la síntesis del modelo, esto es importante porque la biblioteca ATPG es la que describe cuáles son los elementos escaneables y por cuáles otros elementos se sustituirán en el diseño sintetizado. Por lo general esta biblioteca se ubica en la siguiente ruta.

```
$MGC_HOME/ADK/technology/adk.atpg
```

Una vez definido todo lo anterior se invoca al programa y si no hay problemas con la lectura del diseño y de la biblioteca se ingresa a modo de *setup*. En este momento la interfaz gráfica cambia a dos ventanas principales, una es la ventana de comandos y la otra es el panel de control tal y como se muestra en la figura 6.4 a y b.



a)



b)

Figura 6.4 Ventanas principales del programa DFTAdvisor

Un paso primordial en el modo de *setup* es definir las señales de reloj. Si se esta utilizando la GUI las señales de reloj se definen en el menú "Clocks" del panel gráfico en el panel de control (figura 6.4.b), aquí se tienen dos opciones: la primera es navegar en el diseño y definir cuáles pines de las entradas principales son entradas de reloj y definir manualmente su estado inactivo; la segunda es dejar que el programa las identifique automáticamente. Es importante destacar que DFTAdvisor considera como relojes toda señal capaz de alterar el estado de un dispositivo secuencial, ya sean relojes del sistema o señales de *set* y *reset*. Lo anterior también se puede hacer desde la línea de comandos, para mayor detalle de su uso referirse a [8]. Después de completar setup el diseño estaba listo para ser sometido a la verificación de las reglas de diseño (DRC). En este caso no hubo ninguna violación a las reglas por lo que el siguiente paso era configurar las estructuras de prueba. Se escogió como insertar una estructura para escaneo completo ya que este es el método que en general da mejores resultados y el que se proyectaba utilizar en el diseño del microprocesador Tesla. De aquí se obtuvo una versión del registro LFSR modificada y que contenía una cadena de prueba llamada chain1 y formada por los 16 registros del diseño y se agregaron dos entradas scan_in y escan_en, y una salida scan_out.

Para la generación de los patrones de prueba se utilizó el programa FastScan y se obtuvieron patrones de prueba para fallas de tipo stuck-at. Se corrió la verificación de los patrones y la simulación de los mismos y el proceso fue satisfactorio. Este trabajo con el registro LFSR fue importante porque con él se obtuvo un procedimiento básico para ser aplicado luego al microprocesador Tesla. Se sabía que con este modelo sencillo no se iba a poder experimentar las mismas situaciones que con el microprocesador por dos razones; primero el diseño del registro es mucho más pequeño y sencillo, y segundo que no se estaba trabajando con las bibliotecas de la tecnología seleccionada para la manufactura porque no se contaba con ellas.

6.2 Aplicación de las técnicas DFT al microprocesador Tesla

Lo primero que se debe de hacer al empezar el flujo DFT es definir el fabricante que va a manufacturar el dispositivo ya que es éste el que debe suministrar las bibliotecas correspondientes a la tecnología. En el caso del microprocesador Tesla, se utilizó la tecnología de .35 μm del fabricante AustriaMicroSystems (AMS).

Del trabajo anterior con el registro LFSR se determinó que la mejor manera de trabajar es utilizando archivos de comandos y sin utilizar la interfaz gráfica, pues esta es la forma menos tediosa y más rápida para aplicar el proceso. Estos archivos de comando fueron escritos basándose en el trabajo realizado con el diseño del registro y basándose en consultas hechas al personal de soporte en DFT de la compañía Mentor Graphics (ver [2],[3],[4] y [5]).También se determinó que de esta forma algunos programas corrían más eficientemente y que el tiempo de ejecución disminuía considerablemente, en algunos casos una operación que duraba más de 8000 segundos usando la interfaz gráfica disminuía a unos cuantos minutos si no se usaba la GUI.

Al utilizar archivos de comandos se debe de especificar la ruta tanto de los archivos fuente del diseño como el nombre y la ruta de los archivos que son resultado del proceso, por esto se creó una estructura de directorios con la idea de simplificar las futuras modificaciones a los archivos de comandos. La estructura utilizada se muestra en la figura 6.5 y se explicará a lo largo de esta sección.

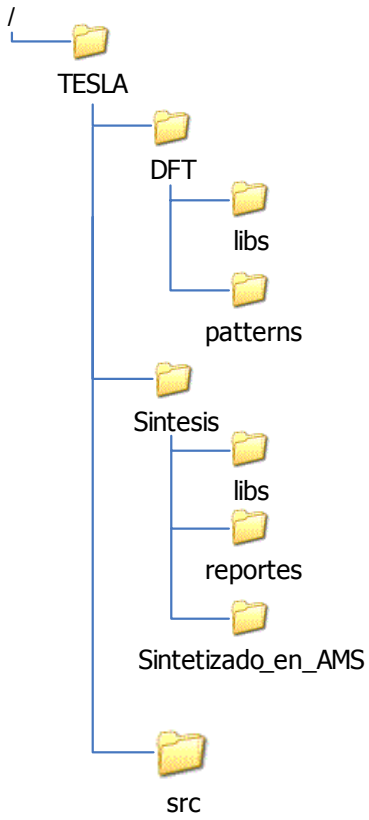


Figura 6.5 Estructura de directorios utilizada para el proceso de DFT

6.2.1 Síntesis y optimización del diseño

La síntesis y la optimización del microprocesador Tesla se realizó en LeonardoSpectrum de Mentor Graphics para plataforma MS Windows por lo que se utiliza una interfaz gráfica. Aun así se pueden utilizar archivos de comandos escribiendo lo siguiente en la ventana de comandos del programa

```
dofile -verbose nombre_del_archivo
```

Para la síntesis se utilizaron los archivos listados en la tabla 6.1. A continuación se hace una breve explicación de estos archivos y de los comandos más relevantes que se utilizan.

El archivo *Compila_Sis_AMS* es el único que hay que ejecutar utilizando el comando `dofile`, desde este se llaman a todos los demás archivos. En *Compila_Sis_AMS* se cargan las bibliotecas, se configura el proceso y se leen y compilan los archivos del proyecto, en esta parte se encontraron algunos errores menores en el proyecto que fueron corregidos para que la

compilación fuera exitosa y así continuar con la síntesis. Después de la lectura del proyecto se utilizó el comando:

```
set_attribute .work.TeslaControl.INTERFACE.clk -name PRESERVE_SIGNAL -value
"1" -net
```

Este tiene como objetivo conservar la señal clk (señal de reloj del sistema) sin sintetizar, para asegurarse que la señal sea la misma en todos los módulos del sistema. En otras palabras, si la señal es sintetizada es probable que se "interrumpa" y se agregue alguna lógica (compuertas, buffers) que produzca retardos y por lo tanto se pierda la sincronización entre los módulos.

Tabla 6.1 Archivos de entrada y salida para la síntesis del microprocesador Tesla

Localización	Nombre del archivo	Descripción
/TESLA/Síntesis/	Compila_Sis_AMS.tcl	Entrada. Archivo de comandos principal
	load_libs.tcl	Entrada. Carga las bibliotecas AMS
	setup.tcl	Entrada. Opciones de configuración
/TESLA/src/	tesla_bus_if.v BusController.v Counter.v etapa_ex.v etapa_id.v etapa_IF.v etapa_mem_wb.v tesla.v	Entrada. Archivos fuente del proyecto
/TESLA/Sintetizado_en_AMS	tesla_bus_if.v	Salida. Es el diseño sintetizado
/TESLA/reportes	area.txt delay.txt	Salida. Indica el área y retardos por celda y el área total ocupada por el diseño

Es importante también indicar la frecuencia de trabajo, para este caso se escogió 50MHz y los siguientes comandos indicando el periodo en nanosegundos.

```
set register2register 20
set input2register 20
set register2output 20
```

Entre los archivos de salida estan *tesla_bus_if.v* que es el diseño sintetizado y optimizado en términos de las celdas de la tecnología en uso y es el que se ocupa para seguir con el

proceso de reconocimiento e inserción de las cadenas de prueba. También están los reportes de área y de retardo, en el primero se informa del área total acumulada por el diseño que en este caso fue de 2661322 μm^2 ó 2.66 mm^2 utilizando un total de 5564 instancias de las bibliotecas; en el segundo se reporta el retardo total de la ruta crítica del sistema y se lista cada instancia que pertenece a esta ruta y el retardo que aporta cada una, para este caso la ruta crítica tiene produce un retardo de 64.62 ns, esto condiciona la frecuencia máxima con la que el diseño puede trabajar e igualmente en este archivo se reporta una frecuencia de trabajo de máximo 15.3 MHz, como se puede observar no se alcanzó la frecuencia inicialmente escogida para este diseño que fue de 50 MHz.

6.2.2 Inserción de las cadenas de prueba

DFTAdvisor es el programa utilizado para el reconocimiento y conversión de los elementos secuenciales a elementos escaneables. Una de los puntos primordiales para este proceso es la biblioteca ATPG, que contiene la descripción de cada celda básica del diseño y se relaciona con la tecnología en uso. Lo ideal es que el fabricante supla esta biblioteca, en este caso no fue así, pero sí provee las bibliotecas escritas en Verilog las cuales se deben de convertir en bibliotecas ATPG con ayuda de la herramienta libcomp [9]. Esta herramienta no realiza una conversión completa, lo que haga falta para que la biblioteca convertida sea equivalente a la biblioteca Verilog debe de editarse a mano.

```
1    Libcomp
2    load library nombre_archivo... -verilog
3    add model -all
4    set learning on
5    set optimization on
6    set system mode translation
7    run
8    write library nombre_archivo
```

Figura 6.6 Secuencia de comandos utilizados para la conversión de la biblioteca

En la figura 6.6 se muestra la secuencia de comandos utilizados para convertir las bibliotecas Verilog en bibliotecas ATPG. Para completar la conversión se debe de verificar ma-

nualmente que el comportamiento de las compuertas básicas sea el equivalente con el descrito en las bibliotecas originales. En este caso se encontró una diferencia en la tabla de verdad de una de las primitivas [2] y [3]. En la biblioteca *udp.v* provista por el fabricante se define la primitiva mostrada en la figura 6.7.

```
1     primitive JK_UDP (JK, J, K, Q);
2      output JK;
3       input J,K,Q;
4       table
5       // J K Q : JK
6          0 1 ? : 0 ; // reset
7          1 0 ? : 1 ; // set
8          0 ? 0 : 0 ; // no change
9          1 ? 0 : 1 ; // toggle
10         ? 1 1 : 0 ; // toggle
11         ? 0 1 : 1 ; // no change
12      endtable
13    endprimitive
```

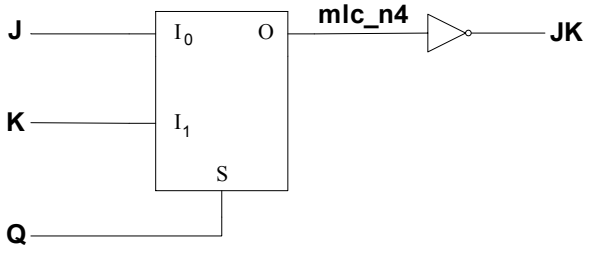
Figura 6.7 Definición de la primitiva JK_UDP provista por el fabricante

El programa Libcomp toma esta descripción la interpreta y la conversión resultante se muestra en la figura 6.8.a. En esta figura se muestra el listado en formato ATPG y como se observa se construye modelo basado en primitivas propias para este formato, a partir de aquí se puede representar el modelo en cuestión como un circuito lógico y su correspondiente tabla de la verdad para poder comparar el comportamiento con el modelo original escrito en Verilog. En este caso se utiliza un multiplexor de dos entradas y un inversor para representar el modelo de flip-flop JK tal y como se muestra en la figura 6.8.b. Para encontrar la diferencia entre el modelo original y el modelo convertido se pueden analizar los modelos como sigue: si se observa en la figura 6.7 cuando la entrada Q es igual a cero sin importar el valor de la entrada K, el valor obtenido en la salida JK es igual al valor que tiene la entrada J en ese momento. Además cuando la entrada Q vale 1 sin importar el valor presente en la entrada J, la salida obtenida es igual a la entrada K negada. Ahora si se observa la tabla de la verdad de la figura 6.8.c se puede notar que esto se cumple que cuando Q = 0 la salida $JK = \overline{J}$ y cuando Q = 1 la salida $JK = \overline{K}$. Entonces esta claro que para corregir este modelo se debe de agregar un inversor en la entrada J. Esta corrección se muestra en el listado de la figura 6.9, donde se puede

observar en la línea 5 que se agrega una instancia de la primitiva _inv a la cual se conecta la entrada J y cuya salida iJ se conecta a la entrada de la primitiva _mux según se aprecia en la línea 6.

```
1    model JK_UDP (JK, J, K, Q) (
2      input(J, K, Q) ()
3      output(JK) (fault=boundary;
4        primitive = _inv (mlc_n4, JK);
5        primitive = _mux (J, K, Q, mlc_n4);
6      )
7      intern(mlc_n4) ()
8    )
```

a)



b)

J	K	Q	JK
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

c)

Figura 6.8 Resultado de la conversión a biblioteca ATPG del modelo JK_UDP. a) listado en formato ATPG; b) interpretación del modelo como circuito lógico; c) tabla de la verdad

```
1    model JK_UDP (JK, J, K, Q) (
2      input(J, K, Q) ()
3      output(JK) (fault=boundary;
4        primitive = _inv (mlc_n4, JK);
5        primitive = _inv (J, iJ);
6        primitive = _mux (iJ, K, Q, mlc_n4);
7      )
8      intern(mlc_n4, iJ) ()
9    )
```

Figura 6.9 Modelo JK_UDP corregido en la biblioteca ATPG

Como paso primordial en la conversión de las bibliotecas en bibliotecas ATPG está la identificación de los modelos escaneables, esto es definir cuáles modelos descritos en la biblioteca pueden ser usados como celdas de escaneo y que por lo general incluyen las entradas scan_data, scan_enable y una salida scan_out para poder formar las cadenas de prueba. En el caso de las bibliotecas de AMS se incluyen elementos de este tipo y son todos aquellos elementos secuenciales que en su nombre se agrega una S y además tienen las entradas SD y SE; por ejemplo el modelo DFS1 que se basa en el modelo DF1.

El programa libcomp no identifica ninguno de estos elementos por lo que se debe de inspeccionar cada modelo de la biblioteca ATPG para distinguirlos y además se debe agregar el código mostrado en la figura 6.10. Con esto se define cuáles son las entradas de escaneo la entrada de datos y las salidas, también cuál es el modelo no escaneable.

```
1    scan_definition (
2        type = mux_scan;
3        data_in = D;
4        scan_in = SD;
5        scan_enable = SE;
6        scan_out = Q, QN;
7        non_scan_model = DF1 (C, D, Q, QN);
8     )
```

Figura 6.10 Código utilizado para definir los elementos de escaneo en la biblioteca ATPG

Ahora se puede continuar con la inserción de las cadenas de prueba al microprocesador, para esto se utilizaron archivos de comandos y otros scripts para el programa DFTAdvisor. En la tabla 6.2 se listan todos los archivos necesarios para la inserción e identificación de las cadenas de prueba así como los archivos que se obtienen o archivos de salida.

El primer archivo *dfta.run* se utiliza para invocar a DFTAdvisor y cargar las bibliotecas y el diseño sintetizado. Los comandos utilizados en dicho programa se encuentran en el archivo dfta.dofile y su contenido se muestra en la figura 6.11, el comando de la primera línea de esta figura se utiliza para reconocer automáticamente la señal de reloj del sistema. En la línea cuatro se muestra el comando utilizado para insertar la lógica de las cadenas de prueba. En este momento se reconocen los elementos no escaneables que serán sustituidos por las celdas de escaneo definidas anteriormente en la biblioteca para luego formar las cadenas de prueba, como el método utilizado es el escaneo completo no hace falta más para reconocer e insertar las estructuras de prueba ya que este es un método bastante automatizado. Importantes son también los comandos de las líneas 5 y 6, se encargan de guardar la versión modifica del microprocesador Tesla que contiene las estructuras de prueba y el *procedure file* que se utiliza como archivo de configuración para FastScan y la simulación de los vectores de prueba.

Tabla 6.2 Archivos de entrada y salida utilizados en la inserción de las cadenas de prueba

Localización	Nombre del archivo	Descripción
/TESLA/Síntesis/DFT/	dfta.run	Entrada. Archivo de comandos principal (invoca a DFTAdvisor)
	dfta.dofile	Entrada. Archivo de comandos para DFTAdvisor
	dfta.log	Salida. Guarda los mensajes producidos por DFTAdvisor
	tesla_bus_if_si.v	Salida. Diseño con las cadenas de prueba
	tesla_bus_if_si.dofile	Salida. Contiene información de las cadenas para el programa FastScan
	tesla_bus_if_si.testproc	Salida. Contiene información de temporización para la simulación
/TESLA/Síntesis/DFT/libs/	c35.atpg c35_IOLIB.atpg c35_CORELIB.atpg	Entrada. Bibliotecas atpg
/TESLA/Sintetizado_en_AMS	tesla_bus_if.v	Entrada. Diseño sintetizado

El archivo más importante producido por el DFTAdvisor es el *tesla_bus_if_si.v*. Este es el diseño con las cadenas de prueba insertadas y además se le han agregados los pines de entrada *scan_in* y *scan_en* y el pin de salida *scan_out* , estos pines son utilizados únicamente en el proceso de pruebas.

En el archivo *dfta.log* se guarda información como por ejemplo la cantidad de elementos secuenciales del diseño y cuántos de éstos son identificados como escaneables, en este caso se encontraron 1443 elementos secuenciales y todos han sido identificados como escaneables por lo tanto fueron sustituidos por los elementos de escaneo.

```
1    analyze control signals -auto_fix
2    set system mode dft
3    run
4    insert test logic -scan on -test_point on
5    write netlist ./tesla_bus_if_si.v  -replace -verilog
6    write atpg setup ./tesla_bus_if_si -replace -procfile
7    exit -d
```

Figura 6.11 Contenido del archivo *dfta.dofile*

6.2.3 Generación de los patrones de prueba

Como al diseño se le aplicó la técnica de escaneo completo (*full scan)*, esto es, se utilizaron todos los elementos secuenciales en las cadenas de prueba; el programa recomendado para generar los vectores de prueba es el FastScan.

Con este programa también se decidió no utilizar la interfaz gráfica y hacer uso de archivos de comandos. En la tabla 6.3 se muestran los archivos necesarios para la generación de los vectores de prueba para fallas tipo *stuck-at*, también se muestran los archivos que son producto de este proceso así como su ubicación en la estructura de directorios.

Tabla 6.3 Archivos involucrados en la creación de los patrones de prueba

Localización	Nombre del archivo	Descripción
/TESLA/Síntesis/DFT/	fscan_SA.run	Entrada. Archivo de comandos principal (invoca a FastScan)
	fscan_SA.dofile	Entrada. Archivo de comandos para FastScan
	tesla_bus_if_si.v	Entrada. Diseño con las cadenas de prueba
	fscan_SA.log	Salida. Guarda los mensajes producidos por FastScan
	tesla_bus_if_si.dofile	Entrada. Contiene información de las cadenas de prueba
/TESLA/Síntesis/DFT/libs/	c35.atpg c35_IOLIB.atpg c35_CORELIB.atpg	Entrada. Bibliotecas atpg
/TESLA/Síntesis/DFT/patterns/	pat_SA_par.v pat_SA_serial.v pat_SA.ascii	Salida. Patrones de prueba

El archivo *fscan_SA.run* es utilizado para invocar al programa FastScan y aquí es donde se asignan las bibliotecas atpg y el diseño modificado que contiene las cadenas de prueba.

En el archivo *fscan_SA.dofile* primero se debe de llamar al archivo *tesla_bus_if_si.dofile* que es resultado del proceso anterior y contiene la información acerca de las cadenas insertadas. También en este archivo se utiliza el comando mostrada en la línea 2 de la figura 6.12 y que es utilizado para tratar específicamente errores de la regla de diseño C3, este error es relativamente común pero puede ocurrir que existan falsos reportes de error de la regla C3 [10], si el problema persiste y hay errores en la simulación de los patrones entonces la regla C3 se debe de tratar con otros métodos. En la línea 8 se guardan los patrones de prueba en formato

ASCII que es un formato propio de Mentor Graphics y facilita la comprensión de la aplicación de los vectores y es el único formato que se puede utilizar con FastScan para la búsqueda de errores; también en las líneas 9 y 11 se guardan bancos de prueba en formato Verilog y los correspondientes vectores que se utiliza en simulación.

```
1    dofile tesla_bus_if_si.dofile
2    set split capture_cycle ON
3    set system mode atpg
4    set fault type stuck
5    add faults -all
6    create patterns -auto
7    report statistics
8    save patterns ./patterns/pat_SA.ascii -ascii -rep
9    save  patterns  ./patterns/pat_SA_par.v  -verilog  -
     parallel -rep -param fscan.param
10   set pattern filtering -sample_per_type 2
11   save  patterns  ./patterns/pat_SA_serial.v  -verilog  -
     rep -serial -param fscan.param
12   report faults -class AU -noeq >fscan_SA_AU_Faults.rpt
```

Figura 6.12 Contenido del archivo *fscan.dofile*

En la línea 12 se guarda un reporte de las distintas clases de faltas y las instancias donde fueron encontradas en el diseño, estas distintas faltas son en las que se basa el programa FastScan para crear los vectores de prueba. Una lista de las distintas clases de faltas que existen y su jerarquía se muestran en el apéndice a.3 .

6.2.4 Simulación de los patrones de prueba

Para la simulación de los patrones se utilizó el programa VSim que es un simulador basado en tiempo de Mentor Graphics. Los archivos utilizados en la simulación se tabulan en la tabla 6.4.

El archivo *vsim.run* contiene todos los comandos necesarios para la simulación, en las líneas de 1 a 7 de la figura 6.13 se compilan las bibliotecas AMS Verilog, el diseño con las cadenas de prueba y los bancos de prueba obtenidos con FastScan.

Tabla 6.4 Archivos utilizados en la simulación de los vectores de prueba

Localización	Nombre del archivo	Descripción
/TESLA/Síntesis/DFT/patterns/	vsim.run	Entrada. Archivo de comandos principal (invoca a VSim)
	fscan_SA.dofile	Entrada. Archivo de comandos para FastScan
	pat_SA_par.v pat_SA_serial.v pat_SA_par.v.0.vec pat_SA_par.v.chain1.name pat_SA_par.v.chain2.name pat_SA_par.v.po.name pat_SA_serial.v.0.vec pat_SA_serial.v.po.name	Entrada. Bancos de prueba y vectores de prueba
	c35_CORELIB.v c35_IOLIB_4M.v udp.v	Entrada. Bibliotecas verilog
	tesla_bus_if_si.v	Entrada. Diseño con las cadenas de prueba
	transcript_ms	Salida. Guarda los mensajes producidos por VSim

Como resultado en la simulación de los vectores de prueba se obtuvo un error de correspondencia en la salida con respecto a la respuesta esperada, en otras palabras la respuesta obtenida después de aplicar algunos de los vectores al diseño no coincide con lo esperado. Este error se intentó corregir resolviendo primero el error con la regla C3 obtenido en FastScan, para esto se probó con el método descrito a continuación.

Primero se toma nota de la salida primaria donde se da el error de correspondencia entre vectores y el valor obtenido; con FastScan se empieza a trazar hacia atrás en el circuito hasta encontrar la compuerta exacta donde ocurre el error, esto puede hacerse manual o automáticamente [11]. En el trazado manual se puede usar el programa Insight View que tiene una interfaz gráfica y así tener un mejor entendimiento del proceso; en la figura 6.14 se muestra una parte del circuito utilizando Insigth View en el trazado hacia atrás del circuito. Una vez encon-

trada la compuerta donde se origina el error y se usa el comando `set split capture_cycle ON` (ver también [13]) después se vuelve a simular los vectores de prueba y si se mantienen los errores de simulación se debe intentar otro método. En este caso el método no resolvió el problema.

```
1   vlib work
2   vlog ../libs/c35_CORELIB.v
3   vlog ../libs/c35_IOLIB_4M.v
4   vlog ../libs/udp.v
5   vlog ../tesla_bus_if_si.v
6   vlog pat_SA_par.v
7   vlog pat_SA_serial.v
8   vsim +nowarnTSCALE TeslaControl -sdfmax TeslaControl=
    tesla_bus_if.sdf -c <<!
9   vsim +nowarnTSCALE TeslaControl_pat_SA_par_v_ctl -c <<!
10  run -all
```

Figura 6.13 Contenido del archivo *vsim.run*

Otro método recomendado [5] es el llamado *backannotation* [12] que es la simulación de los vectores con archivos SDF (Standard Delay Format). En este archivo se especifican los retardos correspondientes a cada compuerta de las bibliotecas de la tecnología en uso. Este archivo puede ser obtenido con Leonardo Spectrum después de sintetizar el diseño original, sin embargo con este tipo de simulación no se obtuvieron resultados diferentes a los anteriores.

También se combinó la simulación gráfica y el uso de FastScan [14] para comparar las respuestas de los modelos en busca de alguna diferencia que permita establecer la no correspondencia entre los modelos de las bibliotecas ATPG y las bibliotecas Verilog. Con este método tampoco se obtuvieron resultados que aclararan el problema.

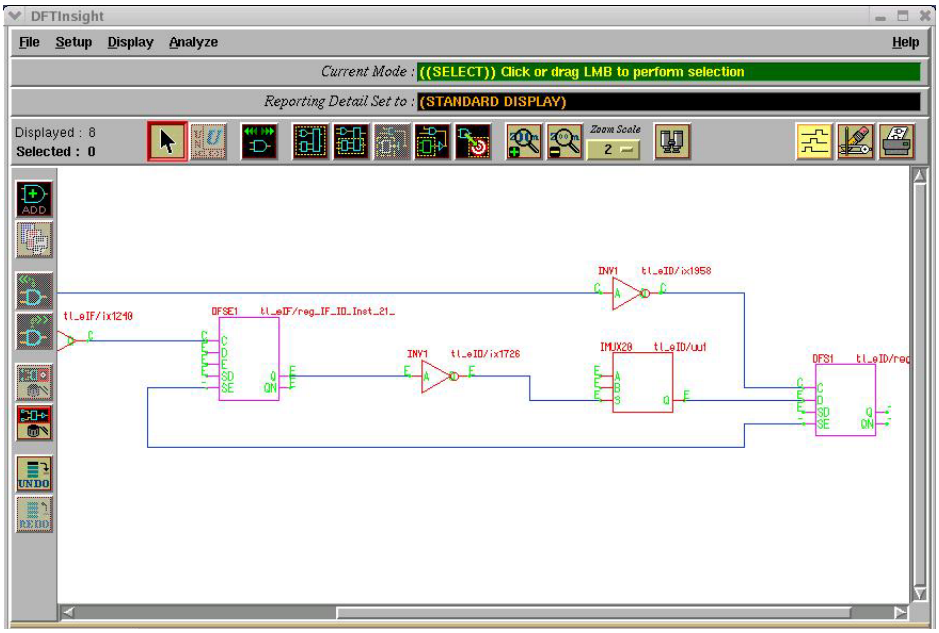


Figura 6.14 Uso del Insight View en el proceso de búsqueda de errores

El problema puede estar sucediendo por un error encontrado [4] en los modelos ATPG, es un error de conversión de las bibliotecas Verilog a bibliotecas ATPG y consiste en un problema del programa LibComp al convertir un multiplexor utilizado para modelar un *flip-flop* JK. Se ha trabajado para este error por medio de consultas con el personal de soporte en DFT de Mentor Graphics, actualmente este error es tratado por el departamento de Ingeniería de la empresa por tratarse de un supuesto error del software de conversión, además en el departamento de DFT se trabaja en una nota técnica para la conversión de bibliotecas motivada por estas consultas. Una vez resuelto este problema se debe de aplicar el flujo de DFT completo para obtener los patrones de prueba correctos. Ahora bien si el problema persiste y se determina que no se debe a las bibliotecas ATPG habrá que retroceder más en el proceso por ejemplo en el proceso de síntesis habrá que reevaluar si existe una alguna restricción de temporización adicional o de otra índole y que se dejó por fuera, también se puede retroceder al diseño del microprocesador y reevaluarlo. También puede darse la posibilidad que la técnica DFT escogida no sea la mejor para este diseño, si fuera el caso entonces por ejemplo podría utilizarse la técnica de escaneo parcial dejando por fuera los elementos secuenciales que estén causando problemas ó podría utilizarse una combinación del escaneo parcial o completo junto con la técnica de escaneo dividido.

Lo que implica que este error exista es que la versión del microprocesador Tesla que posee las cadenas de prueba no se asegura que sea la definitiva, también los patrones de prueba obtenidos fueron generados basándose en esta versión del microprocesador por lo que tampoco se asegura que sean los definitivos además de que no pudieron ser comprobados con la simulación

Lo importante aquí es saber que se tiene la base necesaria para aplicar el flujo de DFT a cualquier diseño, con los scripts obtenidos y aplicados al diseño del microprocesador Tesla se tiene un procedimiento DFT completo y se muestran en el apéndice a.4 , estos scripts pueden ser aplicados a cualquier otro diseño para la identificación e inserción de las estructuras de prueba de escaneo completo además de la generación y comprobación de los patrones de prueba para faltas de tipo stuck-at. Para ello las modificaciones que deben sufrir los scripts se limitan a lo concerniente con el nombre del diseño. Alguna otra modificación, restricción o detalle que falte dependerá de las características de cada diseño y como es obvio su estudio y aplicación deberá estudiarse por aparte y no están al alcance de este proyecto.

6.3 Inicio del diseño del *tester*

Esta sección del proyecto no pudo llegar a cumplirse debido a los retrasos sufridos en la etapa anterior relacionados con el error encontrado en la conversión de las bibliotecas, este error consumió más tiempo del planeado incluso el plazo del proyecto se tuvo que extender, lo que no dejó tiempo para trabajar en el *tester*. Sin embargo esta parte fue propuesta como proyecto para otros estudiantes y expuesta por ellos en el curso de Formulación de Proyectos dejando así abierta la posibilidad de su posterior continuación como proyecto de graduación.

Es importante destacar también que el proyecto fue presentado en el primer Encuentro de Investigación y Extensión realizado en noviembre de 2004 y organizado por la Vicerrectoría de Investigación y Extensión del Instituto Tecnológico de Costa Rica. En dicha actividad se expusieron algunos de los resultados que se tenían hasta el momento junto con el microprocesador TESLA funcionando en un módulo FPGA y como parte del proyecto Diseño de circuitos integrados CMOS a cargo de los profesores Ing Roberto Pereira Arrollo e Ing. Alfonso Chacón Rodríguez profesores en la Escuela de Ingeniería Electrónica.

Capítulo 7: Conclusiones y recomendaciones

7.1 Conclusiones

El presente proyecto llevó a las siguientes conclusiones:

1. El diseño del microprocesador Tesla pudo sintetizarse satisfactoriamente utilizando la tecnología de 0.35μm de AMS y se obtuvo una descripción del diseño a nivel de compuertas en esta tecnología, resultado que es base para la aplicación de la aplicación de las técnicas DFT.
2. No se puede asegurar que el diseño modificado del microprocesador Tesla que contiene las estructuras de prueba sea el definitivo ya que presenta un error en la regla de diseño C3.
3. Se presume que el error en la regla de diseño C3 es debido a un error en la biblioteca ATPG ya que los métodos aplicados para resolver este error fallaron además de que en la conversión de las bibliotecas ATPG se presentó un aparente error de software que no permite que se realice una conversión correcta.
4. Los patrones de prueba obtenidos no se asegura que sean posdefinitivos ya que la simulación de los mismos falló al ser aplicados al diseño con las estructuras de prueba.
5. Los scripts obtenidos para el flujo DFT están basados en un procedimiento que se completó satisfactoriamente, por lo tanto se puede asegurar que son válidos en condiciones donde se tenga un biblioteca ATPG sin errores
6. La estructura de directorios así como los archivos de comandos utilizados facilitan la aplicación de los mismos a otros diseños teniendo que hacer únicamente pequeñas modificaciones a los archivos.
7. El flujo DFT obtenido es independiente al diseño y la tecnología de fabricación utilizada por lo que puede aplicarse a otros diseños.

7.2 Recomendaciones

Debido a que el trabajo ya está avanzado es recomendable continuar con la conversión de las bibliotecas ATPG asegurándose más detalladamente que cada modelo corresponda en funcionamiento a su equivalente en la biblioteca Verilog. Si se resuelve que el problema no es debido a la biblioteca ATPG lo que queda es revisar la síntesis del modelo y en última instancia retomar el diseño del microprocesador Tesla. También es válido pensar en cambiar el tipo de estructura de prueba, en este caso por el tipo de diseño cambiar a escaneo parcial o puede también calzar una combinación de escaneo parcial con el escaneo dividido

También cabe recomendar, si es el caso y se decide cambiar de tecnología, cerciorarse que la nueva tecnología sea cien por ciento compatible particularmente con las herramientas DFT de Mentor Graphics, esto ahorraría mucho tiempo y esfuerzo que se dedica a la conversión de las bibliotecas y además se puede asegurar mejores resultados.

En el caso de tener una biblioteca ATPG válida, ya sea que se cambie de tecnología o estas cambien de nombre, se debe de asegurar de realizar los cambios pertinentes en los scripts. Primero se debe colocar tanto las bibliotecas ATPG y Verilog en el directorio /TESLA /DFT/libs, luego se debe cambiar los nombres correspondientes en los archivos dfta.run, fscan_SA.run y vsim.run. Para más información de cómo correr los scripts y otros detalles se pueden leer los archivos LEAME.txt incluidos en el directorio /TESLA del disco de archivos suministrado con el proyecto.

Referencias bibliográficas

[1] Crouch L., Alfred. “Design-for-Test for Digital IC’s and Embedded Core Systems”. New Jersey. Prentice-Hall PTR. 1999.

[2] Jones, Mike <mike_jones@mentorg.com>. “RE: SR235264801 When performing DRC on our design we get a C2 rule failure in our Clk signal”. [Consulta 27 oct. 2004 07:06:58 AM]

[3] Jones, Mike <mike_jones@mentorg.com>. “RE: SR235264801 When performing DRC on our design we get a C2 rule failure in our Clk signal”. [Consulta 27 oct. 2004 12:09:53 AM].

[4] Jones, Mike <mike_jones@mentorg.com>. “SR237250207 LibComp fails to convert a Mux for JK flop”. [Consulta 9 nov. 2004].

[5] Jones, Mike <mike_jones@mentorg.com>. “SR235264801 Re: vector mismatches”. [Consulta 7 dic. 2004].

[6] Kaeslin, Hubert. “Digital Design Flow”. Swiss Federal Institute of Technology, Zurich. Last Change: December 14, 2004. http://dz.ee.ethz.ch/index.en.html.

[7] Mentor Graphics Documentation. “Scan and ATPG Process Guide”. Chapter 2: Understanding Scan and ATPG Basics. April 2004.

[8] Mentor Graphics Documentation. “Scan and ATPG Process Guide”. 5-12. April 2004.

[9] Mentor Graphics Documentation. “Design-For-Test Common Resources Manual”. Chapter 5: Using LibComp. Apr. 2004.

[10] Mentor Graphics Documentation. “Design-For-Test Common Resources Manual”. 2-53 a 2-55. Apr. 2004.

[11] Mentor Graphics Documentation. “Scan and ATPG Process Guide”. 6-136. Apr. 2004.

[12] Mentor Graphics Documentation. “ATPG Tools Reference Manual”. 2-495. Apr. 2004.

[13] Mentor Graphics Supportnet. TechNote mg27517: “A circuit showing C3 DRC errors”. Jan. 2004.

[14] Mentor Graphics Supportnet. AppNote 3002: “Debugging Simulation Mismatches in FastScan”. Julio 2003.

[15] Weste, Neil. y Eshraghian, Kamran. “Principles of CMOS VLSI Design, *a system perspective*”. Second edition. New York. Addison Wesley Longman. 1994.

APÉNDICES

Apéndice A.1 Glosario

ASIC: *Application Specific Integrated Circuit* – circuito integrado especializado diseñado y optimizado para una función o propósito específico.

ATE: *Automatic Test Equipment* – equipo utilizado para aplicar vectores a un diseño de chip manufacturado.

ATPG: *Automatic Test Pattern Generation* – es el uso de una herramienta de software que por medio de algoritmos genera vectores de prueba.

ATPG combinacional: tipo de generación de patrones que está diseñado para operar más eficientemente en un circuito comprendido sólo por lógica combinacional.

Cadena de escaneo: conjunto de varias celdas de escaneo conectadas formando un registro de desplazamiento conectando el puerto de salida Q o SDO de una celda de escaneo, al puerto de entrada dedicado para escaneo SDI de otra celda.

Celda de escaneo: elemento secuencial conectado como parte de una cadena de escaneo. Por ejemplo un flip-flop tipo D con un multiplexor que permita la selección entre la entrada funcional D y la entrada dedica a pruebas SDI.

Controlabilidad: capacidad que se tiene de establecer un estado lógico conocido sobre nodos, redes, entradas o salidas de compuertas, o elementos secuenciales. Se aplica a la capacidad que tiene la herramienta ATPG de colocar valores conocidos donde sea necesario durante la generación de vectores.

DFT: *Design-for-Test* –acción de incluir características a un chip durante el proceso de diseño para alcanzar cierto nivel de calidad o reducir el costo por pruebas.

Full-Scan: arquitectura de pruebas de escaneo en la cual todos los elementos secuenciales son conectados en cadenas de escaneo.

HDL: *Hardware Description Language* – formato basado en texto para descripción de diseños con objetos de software de muy alto nivel comparativamente al uso de compuertas y objetos secuenciales en RTL.

JTAG: *Joint Test Action Group* –nombre del grupo que empezó el estándar IEEE 1149.1 para Boundary Scan y que ahora es asociado a todo lo concerniente con este estándar.

LFSR: *Linear Feedback Shift Register* –dispositivo comprendido por un registro de desplazamiento y un arreglo de compuertas XOR que lo realimentan. Es utilizado para generar patrones pseudo-aleatorios.

Observabilidad: capacidad de observar nodos, redes, compuertas o elementos secuenciales después de haberlos llevado a un estado lógico conocido. Esto es aplicado a la habilidad que tiene la herramienta ATPG de observar puntos en el diseño durante la generación de vectores.

Patrón: agrupamiento de vectores individuales y generalmente tiene un propósito como "patrón de verificación de la lógica".

RTL: *Register Transfer Level* – descripción de diseños basada en texto donde el comportamiento de un circuito es modelado como un flujo de datos y control de registro a registro en referencia a una señal de reloj u otra señal de sincronización aplicada.

Stuck-at: representación matemática de un comportamiento defectuoso basado en conexiones cortocircuitadas de compuertas o cables a VDD o VSS. Este es un modelo de fallas CD que es aplicado independientemente del tiempo y la frecuencia

Testing: observación de una respuesta esperada como resultado de la aplicación de un vector de entrada a un circuito en estado conocido. El propósito es medir alguna respuesta contra su estándar.

Lógica de pruebas: lógica incluida en el diseño de un chip para implementar funciones específicas de prueba.

Vector: colección de 1s y 0s lógicos aplicados a un chip en determinado punto y tiempo. El vector para escaneo es un caso especial y es la colección de 1s y 0s lógicos y de señales de reloj necesarios para cargar una cadena de escaneo.

Apéndice A.2 Información sobre la institución

A2.1 Descripción del Instituto Tecnológico de Costa Rica

El Instituto Tecnológico de Costa Rica (TEC) fue creado en 1971, es una institución dedicada la docencia, la investigación y la extensión de la tecnología y ciencias conexas necesarias para el desarrollo de Costa Rica.

El campus central del TEC se encuentra en Cartago, 26 kilómetros al sureste de la ciudad capital; cuenta también con un Centro Académico en el Barrio Amón en San José, un Centro de Transferencia Tecnológica en Zapote y una Sede Regional en Santa Clara de San Carlos.

Más de 500 hectáreas de terreno y 80 000 metros cuadrados de construcción albergan las aulas, laboratorios, centros de investigación, instalaciones deportivas y culturales, bibliotecas, bosques y parcelas experimentales, talleres y librerías, que el TEC pone a disposición de sus estudiantes, funcionarios y comunidades circunvecinas.

El presente proyecto se desarrolla en el Campus de la Sede Central, ubicado un kilómetro al sur de la Basílica de Nuestra Señora de Los Ángeles en la ciudad de Cartago, que se encuentra a una altura promedio de 1.414 metros sobre el nivel del mar. Aquí se ubica el grueso de las instalaciones. En un área de más de 90 hectáreas hay un total de 4.185 metros cuadrados de construcciones, entre las que sobresalen edificios de aulas, laboratorios y talleres, servicios (biblioteca, librería, soda-comedor), gimnasio y edificios para las oficinas de los funcionarios docentes y administrativos.

A2.2 Descripción del Escuela de Ingeniería Electrónica

La Escuela de Ingeniería Electrónica inició formalmente sus labores de docencia en 1976; cuenta actualmente con 23 profesores y cerca de 800 estudiantes. Imparte un diplomado en Electrónica y una licenciatura en Ingeniería Electrónica. Además, la Escuela se encuentra en la actualidad en un proceso de acreditación académica internacional que significa nuevos retos para sus docentes y estudiantes, sobre todo con el rápido avance de las tecnologías electrónicas y que han puesto al país en la necesidad de generar nuevos graduados con un mayor grado de conocimientos especializados en alta tecnología.

Una de dichas áreas, en las que se busca precisamente incursionar la Escuela, es en la microelectrónica, en el desarrollo de circuitos integrados de alta escala de integración (Very Large Scale Ingegration, VLSI), para cuya investigación se decidió equipar un laboratorio con equipo donado en gran parte por Intel Inc. Este laboratorio es dirigido por el Msc. Roberto Pereira, graduado en la Universidad Técnica de Dinamarca en Ingeniería de Computadores, y con experiencia profesional en el modelado en alto nivel y producción de circuitos electrónicos integrados. Además, labora en el laboratorio el Lic. Alfonso Chacón, ingeniero en electrónica, quien actúa como asesor del proyecto. Las labores típicas del laboratorio incluyen el mantenimiento de las herramientas, soporte a los estudiantes y profesores usuarios de las mismas, e investigación en el área de diseño VLSI por medio de herramientas de software.

A2.3 Antecedentes prácticos

En la Escuela de Ingeniería en Electrónica se inició, en el 2002, la puesta en marcha del laboratorio de Investigación y Desarrollo en VLSI (Very Large Scale Integration). Con el fin de generar una ruta de trabajo efectiva que tuviese al laboratorio funcionando, a principios del segundo semestre del 2003, el encargado del laboratorio, Ing. Pereira, sugirió la realización de un modelo de circuito electrónico avanzado que, por un lado, sirviera de prueba para los equipos a instalar y, por otro, proveyera de una base académica sobre la cual construir nuevos proyectos.

En un flujo de diseño típico para circuitos integrados VLSI, se debe pasar por diversas etapas antes de lograr la implementación del dispositivo. Un proyecto denominado *Modelo de transferencia RTL de un microprocesador RISC*, desarrollado en el año 2002 y efectuado por el Ing. Chacón, se encargó de una primera etapa, cuyos objetivos fueron establecer las especificaciones del diseño, programar la descripción de comportamiento, establecer el modelo de transferencia RTL y efectuar pruebas de simulación que evidenciaran algunas de las capacidades del diseño.

Los resultados de esta primera etapa fueron satisfactorios, pues las pruebas de simulación evidenciaron que el diseño cumplió con el comportamiento esperado. Así pues, el diseño estaba listo para comenzar una nueva etapa, en la cual debían realizarse pruebas de funcionali-

dad. De esta manera surgió la idea de implementar un sistema mínimo, que pusiera a prueba las capacidades del diseño para controlar dispositivos de memoria y de entrada/salida. Lo anterior sugería también la posibilidad de tener que realizar algunas modificaciones al diseño original para que este se ajustase a los requerimientos de un sistema mínimo.

Esta etapa fue realizada con la ayuda de varios estudiantes y se obtuvieron los resultados deseados ya que se comprobó que el microprocesador es funcional y capaz de interactuar con elementos de entrada/salida, memoria externa y que respondía adecuadamente al conjunto de instrucciones que fueron implementadas. De aquí se concluyó que el microprocesador era lo suficientemente confiable y seguro como para continuar con etapas posteriores.

Apéndice A.3 Jerarquía de las clases de fallas para FastScan

(FU) Full

(UT) Untestable		Son aquellas fallas para las cuales no existen patrones ya sea para detectarlas o posiblemente detectarlas. No causan fallas funcionales.
	(UU) Unused	Incluye todas las fallas en la circuitería que no estan conectadas a un punto de observación.
	(TI) Tied	Cuando el punto de la falla está "atado" a un valor idéntico al de la falla *stuck-at.*
	(BL) Blocked	Incluye las fallas para las cuales la lógica "atada" bloquea todas las rutas hacia un punto de observación.
	(RE) Redundant	Son aquellas fallas que el generador de pruebas considera indetectables.
(TE) Testable		Son aquellas fallas que no pueden ser probadas como UT.
	(DT) Detected	Son todas las clases que el proceso ATPG identifica como detectables.
		(DS) det_simulation. (DI) det_implication. (DR) det_robust. (DF) det_functional.
	(PD) Posdet	Son fallas posiblemente detectadables y son identificadas en la simulación de fallas. Pueden resultar de una diferencia 0-X o 1-X en el punto de observación.
		(PT) posdet_testable. (PU) posdet_untestable.
(AU) Atpg_untestable		Son aquellas faltas que el generador de pruebas no puede encontrar un patrón y crear una prueba, y tampoco puede probar la redundancia de la falla.
(UD) Undetected		Son todas aquellas fallas indetectables que no pueden ser probadas como UT o AU.
	(UC) Uncontrolled	Son aquellas que durante la simulación de patrones nunca alcanza el valor requerido para la detección de la falla.
	(UO) Unobserved	Fallas cuyo efecto no se propaga a un punto observable.

Apéndice A.4 Scripts utilizados

Scripts para Leonardo Spectrum

Archivo load_libs.tcl

```
load_library ./libs/c35_CORELIB.syn
#load_library ./libs/c35_CORELIB_3B.syn
load_library ./libs/c35_IOLIB_4M.syn
load_library ./libs/c35_IOLIBV5_4M.syn
#load_library ./libs/c35_IOLIB_3B_4B.syn
set sdf_write_flat_netlist TRUE
```

Archivo setup.tcl

```
load_library        c35_CORELIB
load_library        c35_IOLIB_4M

set optimize_target "-target c35_CORELIB -io_target c35_IOLIB_4M"

set wire_table      "10k"
set wire_tree       balanced
set process         worst
set temp            125
set voltage         3.0
```

Archivo Compila_Sis_AMS.tcl

```
set_working_dir "E:/TESLA/Sintesis"
dofile -verbose "./load_libs.tcl"
dofile -verbose "./setup.tcl"
read  { "../src/tesla_bus_if.v" "../src/BusController.v" "../src/Counter.v"
"../src/etapa_ex.v"          "../src/etapa_id.v"          "../src/etapa_IF.v"
"../src/etapa_mem_wb.v" "../src/tesla.v" }
pre_optimize -common_logic -unused_logic -boundary -xor_comparator_optimize
pre_optimize -extract
```

```
set_attribute .work.TeslaControl.INTERFACE.clk -name PRESERVE_SIGNAL -value "1" -net

set_attribute .work.TeslaControl.INTERFACE.clk -name USELOWSKEWLINES -value "0" -net

set register2register 20

set input2register 20

set register2output 20

optimize -target c35_CORELIB -io_target  c35_IOLIB_4M

report_area "./reportes/area.txt"  -cell_usage -all_leafs

report_delay   "./reportes/delay.txt"   -num_paths   1   -longest_path   -clock_frequency

auto_write -format Verilog "./Sintetizado_en_AMS/tesla_bus_if.v"
```

Scripts para DFTAdvisor

Archivo dfta.run

```
#!/bin/csh -f

#------

${MGC_HOME}/bin/dftadvisor  ../Sintesis/Sintetizado_en_AMS/tesla_bus_if.v  -verilog -lib ./libs/c35.atpg \

-dofile dfta.dofile -log dfta.log -rep –nogui
```

Archivo dfta.dofile

```
analyze control signals -auto_fix

set system mode dft

run

insert test logic -scan on -test_point on

write netlist ./tesla_bus_if_si.v  -replace -verilog

write atpg setup ./tesla_bus_if_si -replace -procfile

exit -d
```

Scripts para FastScan

Arvhivo fscan_SA.run

```
#!/bin/csh -f
#------
${MGC_HOME}/bin/fastscan tesla_bus_if_si.v -verilog -lib ./libs/c35.atpg \
 -dofile fscan_SA.dofile -log fscan_SA.log -rep –nogui
```

Archivo fscan.param

```
WGL_VECTOR_ANN 1;
VERILOG_VECTOR_COMM 1;
VERILOG_SIM_STATUS 1 ;
```

Archivo fscan.dofile

```
dofile tesla_bus_if_si.dofile
set split capture_cycle ON
set system mode atpg
set fault type stuck
add faults -all
create patterns -auto
report statistics
save patterns ./patterns/pat_SA.ascii -ascii -rep
save patterns ./patterns/pat_SA_par.v -verilog -parallel -rep -param
fscan.param
set pattern filtering -sample_per_type 2
save patterns ./patterns/pat_SA_serial.v -verilog -rep -serial -param
fscan.param
report faults -class AU -noeq > fscan_SA_AU_Faults.rpt
```

Scripts para Vsim

Archivo vsim.run

```
#!/bin/csh -f

vlib work
vlog ../libs/c35_CORELIB.v
vlog ../libs/c35_IOLIB_4M.v
//#vlog ../libs/c35_IOLIB_ANA_3M.v
//#vlog ../libs/c35_IOLIBV5_4M.v
vlog ../libs/udp.v
vlog ../tesla_bus_if_si.v
vlog pat_TR.v
vlog pat_SA_par.v
vlog pat_SA_serial.v
vsim +nowarnTSCALE TeslaControl -sdfmax TeslaControl=tesla_bus_if.sdf -c <<!
vsim +nowarnTSCALE TeslaControl_pat_SA_par_v_ctl -c <<!
run -all
!
vsim +nowarnTSCALE TeslaControl_pat_SA_serial_v_ctl  -c <<!
run -all
!
```

Script para cargar las bibliotecas ATPG

Archivo c35.atpg

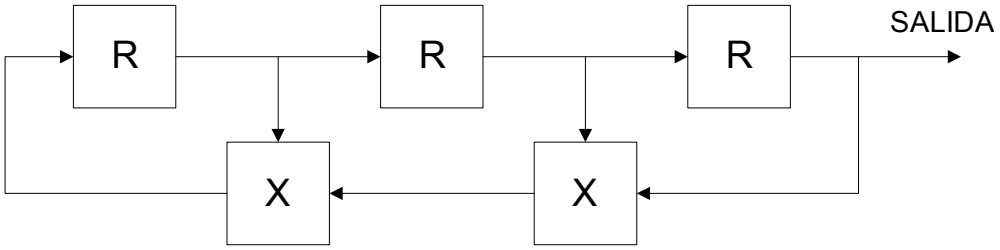
```
#include "c35_IOLIB.atpg"
#include "c35_CORELIB.atpg"
```

ANEXOS

Anexo B.1 Linear Feedback Shift Registers

Los circuitos LFSR se utilizan en infinidad de sistemas electrónicos para generar secuencias pseudoaleatorias con un período garantizado arbitrariamente largo, de forma simple y eficiente.

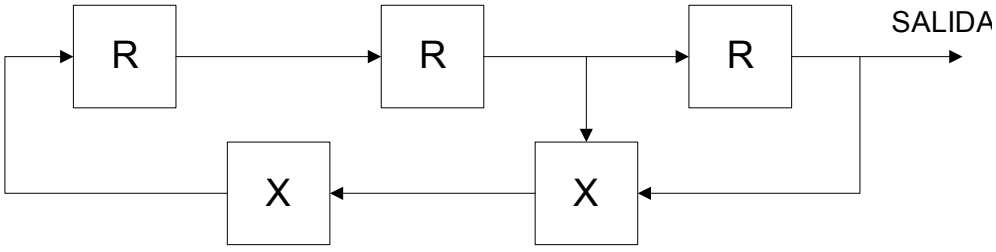
El esquema más habitual de un circuito LFSR es el siguiente:



Las cajas R son biestables, y las cajas X son funciones XOR. Como se puede ver, un registro de desplazamiento con realimentación lineal consta de un sencillo registro de desplazamiento cuyos bits se van desplazando hacia la derecha, en el dibujo, y el último de los cuales proporciona la salida binaria del registro. Al mismo tiempo, por la izquierda en el dibujo, se van inyectando bits cuyo valor es una combinación lineal (módulo 2) de los bits que forman el estado interno del registro.

Aunque en el dibujo se ha dibujado un XOR para cada bit de estado, en la práctica sólo se emplean algunos de ellos. La forma de elegir cuáles se usan no es arbitraria, sino que se basa en principios matemáticos bien definidos. Se verá con detalle más adelante.

Veamos un ejemplo de un circuito concreto:



Supongamos que, de izquierda a derecha, los biestables contienen el valor 001. La tabla con la evolución de estados a partir de ese punto sería:

Estado Interno	Salida
001	1
100	0
010	0
101	1
110	0
111	1
011	1

Dado que el estado interno de un registro de desplazamiento tiene m bits, la longitud máxima de la salida de un LFSR es de 2^m-1 bits, ya que el estado "todo ceros" no es válido (genera una salida continua a cero). ¿Cómo debemos elegir las posiciones de los XOR para asegurarnos alcanzar la lóngitud máxima 2^m-1 bits?

Polinomios primitivos

Un LFSR tendrá longitud máxima si su polinomio característico (polinomio de realimentación) es primitivo en GF(2n).

Un polinomio de grado m es primitivo en GF(2^n) si y solo si a) es irreducible (no es factorizable en polinomios menores) y b) su orden es 2^m-1 (divide exactamente a $X^{2^m-1}+1$ en GF(2^n)). Por definición, todos los polinomios primitivos de grado m tienen la forma x^m+...+1.

Algunos polinomios primitivos en GF(2^n):

Grado	
1	x
2	x^2+x+1
3	x^3+x^2+1, x^3+x+1
4	x^4+x^3+1, x^4+x+1

El número de polinomios primitivos de grado m en GF(b^n) está definido por la fórmula Q(b^m-1)/m, donde Q es la función COCIENTE de Euler (número de enteros menores que su argumento que son coprimos con su argumento).

No existe una fórmula sencilla para obtener polinomios primitivos de grado arbitrario, recurriéndose típicamente a manuales de referencia con tablas o a búsqueda exhaustiva mediante programas informáticos.

Por ejemplo, el siguiente código en Python nos proporciona un listado de todos los polinomios primitivos en GF(2n) para un grado determinado:

```
def polinomios_primitivos(m) :
  rango=pow(2,m)-1
  pol2=pow(2,rango)+1
  p=pow(2,m+1)
  for i in xrange(pow(2,m)+1,pow(2,m+1),2) :
    v=2 # generador
    for j in xrange(rango-1) :
      v*=2
      if v>=p : v^=2*i
      if v==2 :
        break
    if v!=2 : print i
```

La salida de este programa, convertida a código binario, nos indica la presencia o ausencia de cada potencia de x. Por ejemplo, según este programa, un polinomio primitivo de grado 8 sería el 285 que, pasado a binario, es 100011101. Dicho valor representa el polinomo $x^8+x^4+x^3+x^2+1$.

Una propiedad interesante de los polinomios primitivos GF(2n) es que el escribirlos al revés también genera un polinomio primitivo. Es decir, si tomamos el polinomio primitivo 100011101 y lo escribimos al revés, 101110001, el polinomio resultante, $x^8+x^6+x^5+x^4+1$, también es primitivo.

Propiedades

Algunas de las propiedades de los LFSR:

- El periodo de la secuencia es 2^m-1.
- El número de unos generados es 2^m-1.
- El número de ceros generados es 2^{m-1}-1.

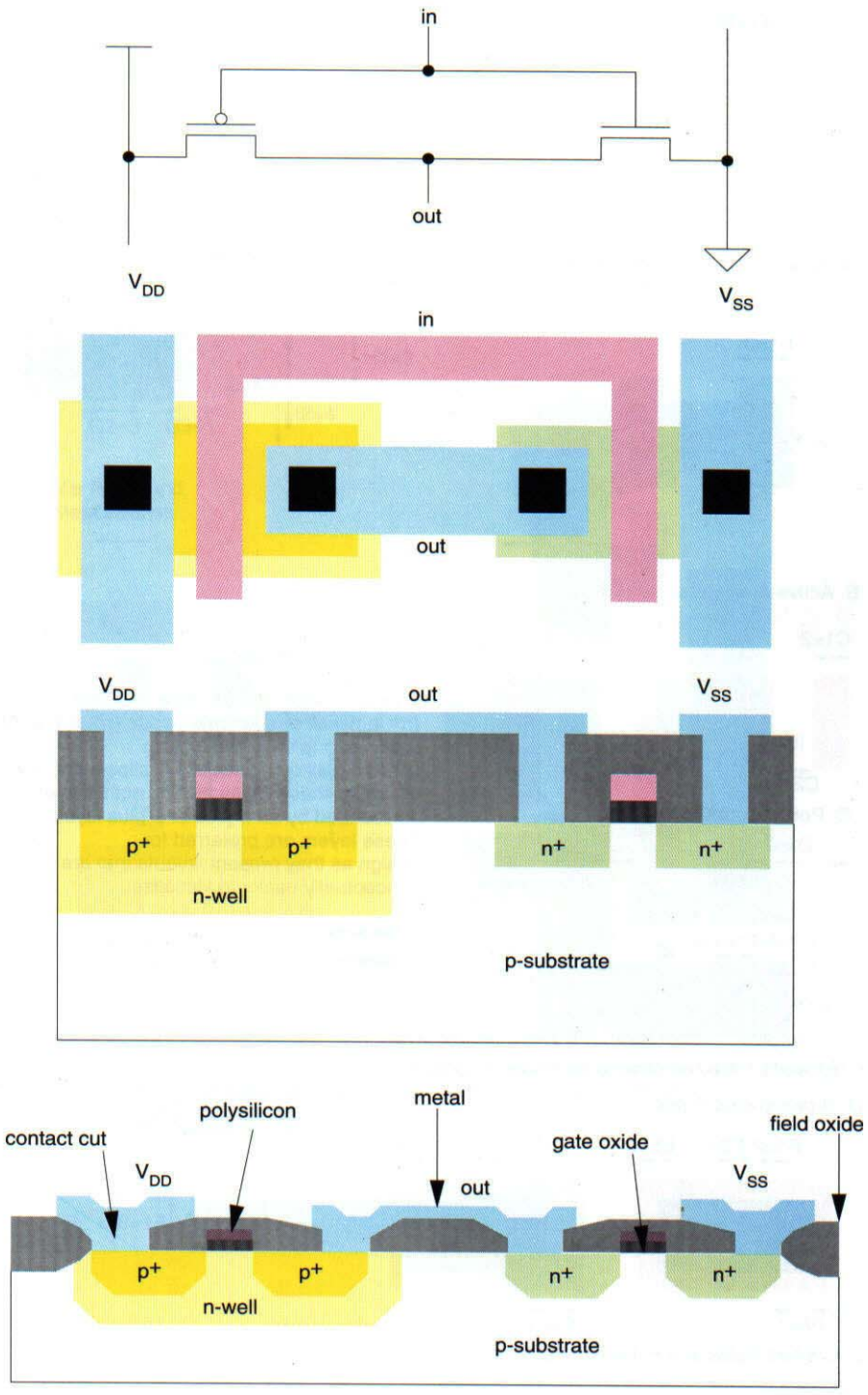
Aplicaciones

La principal aplicación de los circuitos LFSR es la generación de una secuencia de bits pseudoaleatoria, con buenas propiedades matemáticas. La correlación cruzada, por ejemplo, es óptima a medida que el período de la secuencia crece.

Los LFSR NO deben ser utilizados en criptografía como fuentes de cifrado stream, ya que es posible deducir el estado interno y el polinomio característico del LFSR resolviendo una ecuación obtenida a partir de observar 2m bits de salida. Esta técnica ha sido utilizada, por ejemplo, para atacar numerosos sistemas de cifrado analógico de video.

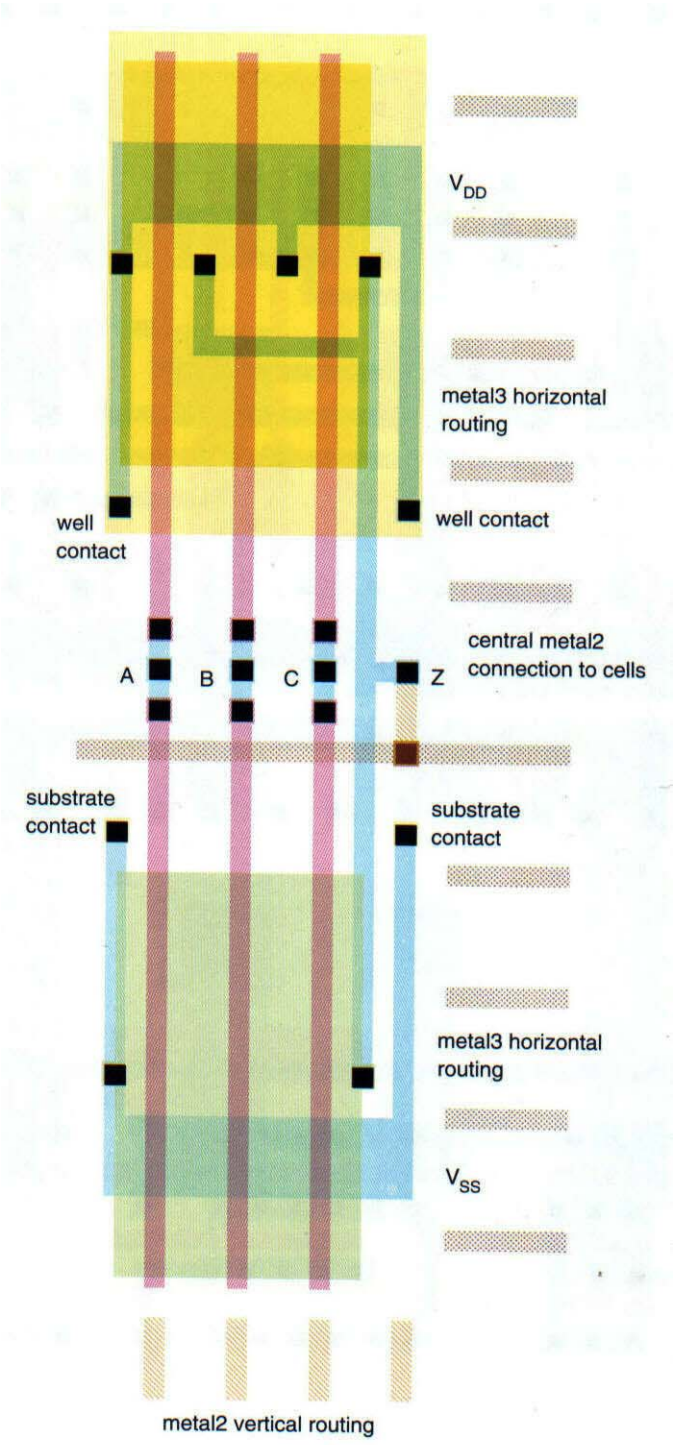
Fuente: `http://www.argo.es/~jcea/artic/lfsr.htm`
`Artículos sobre matemática y teoría de la información`

Anexo B.2 Sección transversal de un inversor CMOS en un proceso n-well



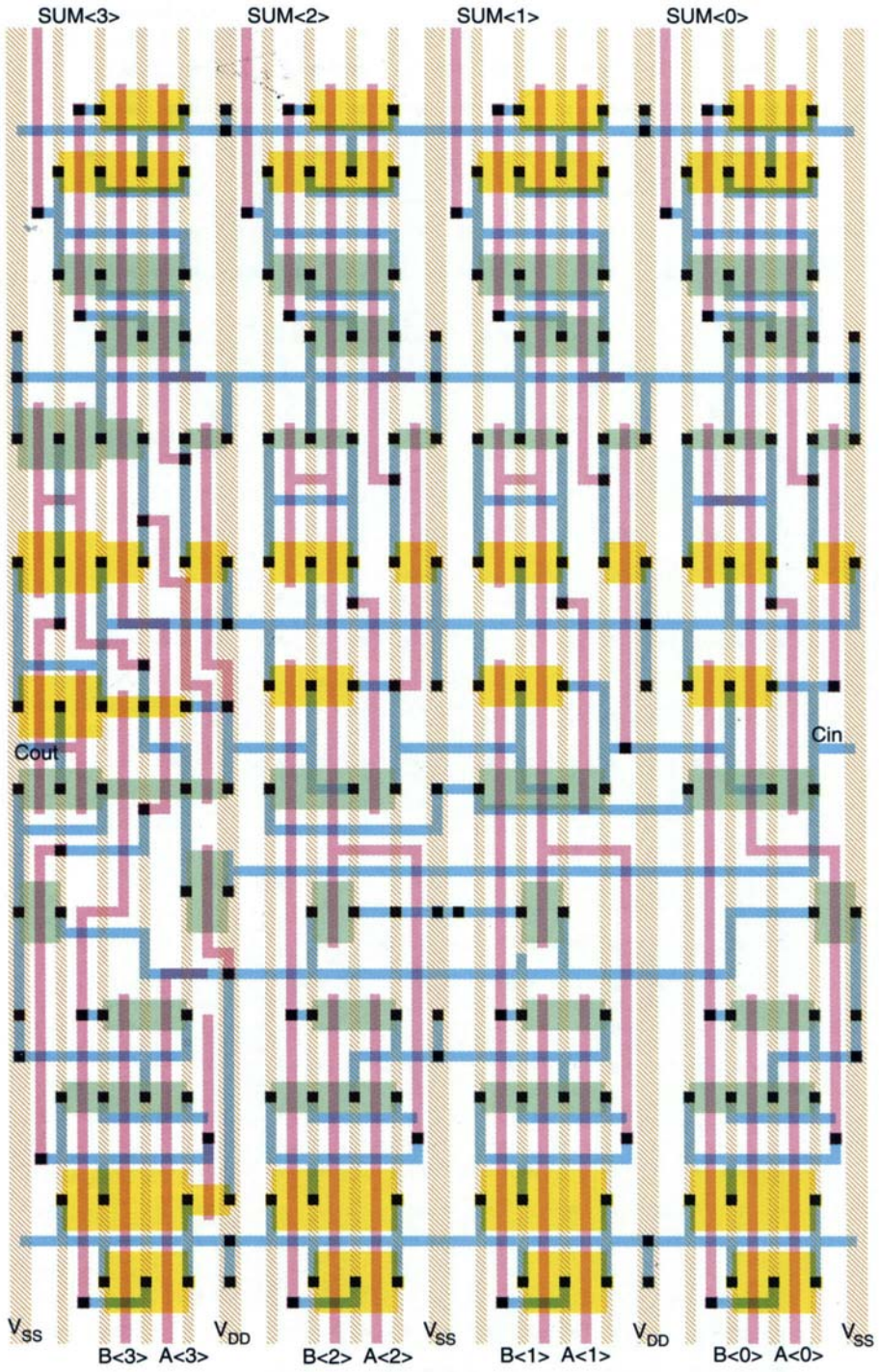
Fuente: Weste, Neil. y Eshraghian, **Kamran. "Principles of CMOS VLSI *Design, a system*** perspective". Second edition. New York. Addison Wesley Longman. 1994.

Anexo B.3 Layout simbólico para una celda estándar de tres metales



Fuente: Weste, Neil. y Eshraghian, **Kamran. "Principles of CMOS VLSI *Design, a system*** perspective". Second edition. New York. Addison Wesley Longman. 1994.

Anexo B.4 Layout de celda estándar Metal3 para un controlador boundary scan tap



Fuente: Weste, Neil. y Eshraghian, **Kamran. "Principles of CMOS VLSI *Design, a system* perspective". Second edition. New York. Addison Wesley Longman. 1994.