ANALIZADOR LÓGICO PARA EL FPGA XC2S200E DE XILINX® BASADO EN BOUNDARY-SCAN PARA EL SISTEMA DE DESARROLLO DIGILAB 2E DE DIGILENT®

CATALINA RIVAS RODRÍGUEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE FISICOMECÁNICAS
ESCUELA DE INGENIERIA ELECTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2005

ANALIZADOR LÓGICO PARA EL FPGA XC2S200E DE XILINX® BASADO EN BOUNDARY-SCAN PARA EL SISTEMA DE DESARROLLO DIGILAB 2E DE DIGILENT®

CATALINA RIVAS RODRIGUEZ

Trabajo de Grado presentado como requisito para optar al
título de Ingeniera Electrónica

Director MSe. Jorge Hernando Ramón Suárez

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE FISICOMECÁNICAS
ESCUELA DE INGENIERIA ELECTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2005

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE ANEXOS

pág.

GLOSARIO DE SIGLAS, ACRÓNIMOS Y TÉRMINOS

ASIC: **Application Specific Integrated Circuit** [Circuito Integrado para una Aplicación Específica]. Circuito Integrado diseñado para una aplicación particular, pues contiene todos los elementos lógicos necesarios para realizar una función específica. Pueden utilizarse para su fabricación bloques creados previamente y contenidos en librerías de funciones lógicas o pueden ser diseñados desde el nivel de transistor. Un ASIC puede ser reconfigurable (llamado **semi- custom**) como los FPGAs, o puede ser fabricado para una única función que no se puede modificar (ASIC **full- custom**).

BOUNDARY SCAN: Nombre dado al estándar 1149.1-1990 del IEEE desarrollado por el grupo JTAG para sondeo serial de datos y exploración por el contorno. El estándar fue aprobado en 1990. También se llama de esta manera a la arquitectura definida en el estándar y que contiene la descripción de todos los elementos (registros, puertos) e instrucciones para su implementación.

BSC: **Boundary-Scan Cell** [Celda Boundary- Scan]. Elemento fundamental de la arquitectura Boundary- Scan. Está constituida por elementos de memoria (flip-flops) y multiplexores; tiene una entrada y salida serie con las que se conecta una celda con otra y una entrada y una salida paralelas.

BSDL: **Boundary Scan Description Language** [Lenguaje de Descripción de Boundary Scan]. Es un sub-juego de instrucciones del lenguaje VHDL que permite describir la arquitectura Boundary-Scan. Un archivo BSDL contiene toda la información acerca de la arquitectura Boundary- Scan de un dispositivo en particular.

BSR: **Boundary-Scan Register** [Registro Boundary- Scan]. Registro de desplazamiento serie y carga paralela de la arquitectura Boundary- Scan, conformado por las celdas Boundary- Scan conectadas de manera serie. Permite el desplazamiento, captura y actualización de los estados lógicos de los terminales del dispositivo.

CLB: **Configurable Logic Block** [Bloque de Lógica Configurable]. Nombre que Xilinx da a la unidad lógica estructural de los **FPGA**s.

IOB: **Input/Output Block** [Bloque de Entrada/Salida]. Bloque CLB especializado de los FPGAs de Xilinx que permite conectar los terminales del circuito integrado con la matriz de CLBs interna.

CPLD: **Complex Programmable Logic Device** [Dispositivo Lógico Programable Complejo]. Dispositivo Lógico programable compuesto por un número de bloques funcionales de menor complejidad conectados entre sí a través de una red central de interconexiones en un solo encapsulado.

DR: **Data Register** [Registro de Datos]. Todo registro de datos que se conecta entre TDI y TDO y que pertenece a la arquitectura Boundary- Scan de un dispositivo.

FPGA: **Field Programmable Gate Array** [Arreglo de Compuertas Programables en el Campo]. Dispositivo lógico programable de arquitectura matricial; el FPGA es un arreglo bidimensional de bloques lógicos programables conectados entre sí a través de conexiones igualmente reconfigurables.

IR: **Instructions Register** [Registro de Instrucciones]. Registro de la arquitectura Boundary- Scan de una longitud mayor o igual a 2 bits y en el cual se cargan de manera serie las instrucciones.

IEEE: **The Institute of Electrical and Electronics Engineers** [Instituto de Ingenieros Eléctricos y Electrónicos]. Este instituto Internacional agrupa a los estudiantes y profesionales de la Ingeniería Eléctrica, Electrónica y afines. Se encarga entre otras cosas de estudiar y aprobar estándares relacionados con las tecnologías de la información y eléctrica, estándares que son una referencia importante en la industria y en la academia a nivel mundial.

JTAG: **Joint Test Action Group** [Grupo de Acción Conjunto para Pruebas]. Grupo de trabajo creado en la década de los ochenta y que propuso la arquitectura Boundary Scan que sería aprobada como estándar del IEEE en 1990. Se llama JTAG también a esta arquitectura, en referencia al grupo que la propuso.

LABS: **Logic Analyzer by Boundary- Scan** [Analizador Lógico mediante Boundary- Scan]. Nombre dado en este documento al analizador lógico desarrollado. Este analizador se vale de la implementación del estándar **Boundary- Scan** en un FPGA para capturar los estados lógicos de sus terminales sin utilizar sondas físicas.

SVF: **Serial Vector Format** [Formato de Vector Serie]. Es un formato estándar para el intercambio de información a través de un bus de datos con el fin de realizar operaciones **Boundary- Scan.** Sus derechos son controlados por Asset InterTech®.

TAP: **Test Access Port** [Puerto de Acceso para Pruebas]. Puerto de la arquitectura Boundary- Scan conformado por las señales TDI, TMS, TCK, TDO y opcionalmente TRST. Está definido en el

estándar 1149.1- 1990 del **IEEE.** El controlador de este TAP se representa como una máquina de estados sincrónica a TCK.

TCK: **Test ClocK** [Reloj de Prueba]. Señal imperativa de entrada del Puerto de Acceso para Pruebas, definida en el estándar 1149.1- 1990 del IEEE. Todas las operaciones que se realizan en el TAP son sincrónicas a este reloj.

TDI: **Test Data In** [Entrada de Datos de Prueba]. Señal imperativa de entrada del Puerto de Acceso para Pruebas, definida en el estándar 1149.1- 1990 del IEEE. Es la entrada serie de los Registros de Datos y del Registro de Instrucciones.

TDO: **Test Data Out** [Salida de Datos de Prueba]. Señal imperativa de salida del Puerto de Acceso para Pruebas, definida en el estándar 1149.1- 1990 del IEEE. Es la salida serie de los Registros de Datos y del Registro de Instrucciones.

TMS: **Test Mode Select** [Selección de Modo de Prueba]. Señal imperativa de entrada del Puerto de Acceso para Pruebas, definida en el estándar 1149.1- 1990 del IEEE. Junto con la señal TCK controla las operaciones que se realizan en los registros internos.

TRST: **Test Reset** [Reset de Prueba]. Señal de entrada optativa del Puerto de Acceso para Pruebas, definida en el estándar 1149.1- 1990 del IEEE. Señal de reset asíncrono.

VHDL: VHSIC **Hardware Description Language** [Lenguaje de Descripción de Hardware para VHSIC].Este lenguaje de alto nivel fue establecido en el estándar 1086- 1987 del IEEE. Es utilizado para la descripción funcional, estructural o de comportamiento de sistemas digitales complejos y para su verificación y síntesis.

VHSIC: **Very High Speed Integrated Circuit.** [Circuito Integrado de Muy Alta Velocidad].

VLSI: **Very Large Scale of Integration** [Muy Grande Escala de Integración].

XSVF: **Xilinx Serial Vector Format** [Formato de Vector Serie de Xilinx]. **XSVF** es un formato binario más compacto que SVF pero que tiene en esencia la misma forma y las mismas funcionalidades.

Título: ANALIZADOR LÓGICO PARA EL FPGA XC2S200E DE XILINX® BASADO EN BOUNDARY-SCAN PARA EL SISTEMA DE DESARROLLO DIGILAB 2E DE DIGILENT®*

Autor: Catalina Rivas Rodríguez**

Palabras claves: Boundary- Scan, JTAG, Analizador Lógico, verificación, BSDL, SVF, IEEE, FPGA

Se realiza un estudio del estándar 1149 del IEEE (Instituto de Ingenieros Eléctricos y Electrónicos), conocido como Arquitectura de Puerto de Acceso para Pruebas y Exploración por el Contorno; como aplicación del estándar se desarrolla un Analizador Lógico para el FPGA XC2S200E de Xilinx® de la tarjeta de desarrollo Digilab 2E de Digilent®. En el diseño y desarrollo de sistemas electrónicos es de vital importancia el proceso de verificación; se realiza en varias etapas del diseño, desde la creación de un código o esquema descriptivo, hasta su implementación física (fabricación o programación). Permite detectar errores físicos o funcionales. La base de muchas técnicas de verificación post- configuración es el acceso físico a los nodos o terminales.

Cuando debido a la alta densidad e integración de los Circuitos Integrados (ICs) el acceso físico se fue limitando, se crea un grupo de trabajo adjunto al IEEE que crea en 1990 el estándar 1149, Boundary- Scan; este describe un arquitectura que al ser implementada en un IC mediante una lógica adicional dentro del encapsulado permite acceder al estado lógico de todos sus terminales, a través de sólo cuatro terminales dedicados.

Utilizando este estándar se desarrolla el analizador Lógico mediante Boundary- Scan o **LABS**; a través de una interfaz gráfica desarrollada en LabVIEW, el usuario puede visualizar el estado lógico de los terminales que utilizó el diseño implementado en el FPGA. El algoritmo desarrollado utiliza la información de la arquitectura Boundary-Scan del dispositivo que se encuentra en su archivo BSDL por Boundary- Scan Description Language, para crear y ejecutar instrucciones en Formato de Vector Serial o SVF, que a su vez realizan operaciones en los terminales dedicados de Boundary-Scan que permiten la captura de los estados lógicos y opcionalmente la aplicación de estímulos a los terminales de entrada.

---

* Trabajo de Grado.

** Facultad de Fisicomecánicas
Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones
Director: MSc. Jorge Hernando Ramón Suárez.

Title: LOGIC ANALYZER FOR THE XILINX® XC2S200E FPGA BASED ON BOUNDARY-SCAN FOR THE DIGILENT® DIGILAB 2E DEVELOPMENT SYSTEM [*]

Author: Catalina Rivas Rodríguez[**]

Keywords: Boundary- Scan, JTAG, Logic Analyzer, verification, BSDL, SVF, IEEE, FPGA

A study of the 1149 IEEE (Institute of Electrical and Electronics Engineers) standard, known as Test Access Port and Boundary Scan Architecture, is pursued; as an application of the standard, a Logic Analyzer for the Xilinx's XC2S200E FPGA from the Digilent's Digilab 2E Development System is unfolded. The verification process is very important for the design and development of electronic systems; it is made in several stages of the design process, from the creation of a source code or descriptive schema, to its physical implementation (manufacture or programming) It allows the designer to detect physical or functional errors. The base of most post- configuration verification techniques is the physical access to pins or nodes.

When this access got limited because of the high density and integration of Integrated Circuits (ICs), an IEEE joint action group (JTAG) was formed; it created in 1990 the 11149 Standard, known as Boundary Scan; it describes an architecture that, when implemented into an IC using some additional logic inside the package, it allows to access the logical state of all its pins, by means of only four dedicated pins.

A Logic Analyzer by Boundary Scan (**LABS**) is developed, using this Standard; the user can view through a graphical interface developed in LabVIEW the logical state of the pins used in the implemented design in the FPGA. The created algorithm uses the Boundary Scan architecture information of the FPGA that is found in its BSDL (for Boundary- Scan Description Language) file, in order to create and execute instructions in Serial Vector Format or SVF, that at the same time execute operations in the Boundary Scan dedicated pins; it allows the logic status capture and eventually apply stimuli on the input pins.

---

[*] Final Project.

[**] Faculty of Physical- mechanical Engineering
Department of de Electrical, Electronics and Telecommunications Engineering
Director: MSc. Jorge Hernando Ramón Suárez.

INTRODUCCIÓN

El concepto de Ingeniería ha ido evolucionando a través del tiempo permanentemente y en la actualidad abarca una amplia gama de acciones y tecnologías diferentes; en sus inicios, hace más de 100 años, se comenzaban a lograr avances tecnológicos y científicos que abrieron a las personas posibilidades antes desconocidas y que iban más allá de la imaginación, como ocurrió con la telegrafía que revolucionó las comunicaciones, o el motor a vapor en el transporte. La ingeniería consistía en explorar las posibles aplicaciones prácticas de los principios científicos que se conocían; a prueba y error, o muchas veces accidentalmente, se hicieron descubrimientos y se concibieron inventos que cambiaron radicalmente el curso de la historia de la humanidad.

Este concepto ha cambiado mucho desde el siglo diecinueve hasta nuestros días. Los avances que la ingeniería ha permitido en tantas áreas del conocimiento, han dejado de ser privilegios exclusivos que sólo unos pocos llegaban a conocer y comprender su funcionamiento, para llegar a tocar todos los aspectos de nuestra vida cotidiana, en campos tan diversos como las comunicaciones, la medicina, el entretenimiento, la vivienda o el transporte. La gran mayoría de personas sabe cómo hacer una llamada en un teléfono celular, conducir un vehículo, utilizar un ascensor.

La misión de un ingeniero en la actualidad consiste ahora más que en patentar un invento, en perfeccionar y optimizar el diseño y operación de sistemas, procesos o máquinas según su área de especialidad; es por esto que continuamente se han ido mejorando los elementos de la industria y la vida diaria para hacerlos más fáciles de utilizar y mejores. Se ha ido aprendiendo de los errores del pasado que inclusive han llegado a cobrar vidas humanas y con cada nuevo hallazgo, los sistemas (en particular sistemas electrónicos) cada vez son más eficientes, más rápidos y ante todo más seguros.

En la actualidad, se da por sentado la responsabilidad de los ingenieros de la calidad de los sistemas que crean, en especial en áreas de aplicación de la electrónica como el transporte aéreo, los equipos electrónicos para la medicina o la seguridad, pues la vida de muchas personas alrededor del mundo

depende del funcionamiento virtualmente libre de fallas de cada uno de los elementos que componen los sistemas diseñados y operados por ellos.

En estas áreas mencionadas y en muchas otras, los Sistemas Digitales tienen una gran importancia y son utilizados con mayor frecuencia a medida que se necesitan para aplicaciones que requieren altas velocidades, un menor consumo de energía, un tamaño más pequeño o una mayor compresión de datos, cualidades que tienen los sistemas digitales sobre los analógicos.

Este proyecto fue desarrollado como una herramienta más del uso de los Sistemas Digitales y que permita detectar errores en un sistema de desarrollo que utiliza lógica programable; de igual manera busca ser un aporte al estudio de estos sistemas en la Universidad Industrial de Santander y motivar la continuidad en el tema que promete ser de gran importancia para tantas áreas en los años venideros.

1. ASPECTOS GENERALES

1.1. JUSTIFICACIÓN

Para el diseñador de sistemas electrónicos análogos o digitales la comprobación de resultados es una necesidad muy importante que permite saber cuándo se debe corregir un diseño o su implementación. Para el diseño o estudio de cualquier sistema se necesita una realimentación para poder ser corregido o mejorado a partir de un diagnóstico de su funcionamiento.

Aunque la simulación de resultados puede ser realizada con diversos programas ofrecidos en el mercado y es un paso útil que permite saber de manera aproximada cuál será el comportamiento de un diseño, después de la configuración puede haber algún error que no fue detectado en las simulaciones (errores físicos: cortocircuitos, circuitos abiertos, problemas de continuidad en los terminales o funcionales); en particular, en los sistemas electrónicos digitales es muy útil para su análisis poder conocer el nivel lógico de los terminales ('0' o '1') pues al saber cuál debe ser el nivel de un terminal específico se detectan errores.

Para detectar estos niveles lógicos en los terminales de un circuito integrado se utiliza un método clásico que son las sondas de prueba. Sin embargo, en un dispositivo con una alta escala de integración, particularmente un **FPGA** o Arreglo de Compuertas Programables en el Campo**,** existe el problema de falta de espacio físico para acceder a cada terminal.

Para continuar con el estudio de los Sistemas Digitales y la lógica programable en la Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones de la UIS E^3T, se propone realizar un estudio del estándar 1149 del **IEEE** (Instituto de Ingenieros Eléctricos y Electrónicos) que describe la arquitectura **Boundary-Scan,** y desarrollar un Analizador Lógico Virtual basado en este estándar y utilizando los archivos **BSDL** (por Lenguaje de Descripción de Boundary- Scan) y el formato **SVF** o Formato de Vector Serial. El estándar **Boundary-Scan** es una herramienta alternativa a las sondas de prueba debido a que la escala de integración y alta densidad de los circuitos integrados hace difícil el acceso físico a cada terminal.

El "Analizador Lógico mediante Boundary- Scan- LABS" permitirá diagnosticar los problemas que puedan existir en un diseño lógico que se ha programado en el **FPGA** Spartan IIE de Xilinx® del sistema de desarrollo DIGILAB 2E de Digilent® así como la respuesta en los terminales de salida a señales de estímulo en los terminales de entrada, convirtiéndose también en una herramienta para

los estudiantes que construyan sistemas electrónicos utilizando **FPGA**s. La implementación se hará completamente en software sin necesidad de sondas físicas de ningún tipo en los terminales a analizar y con comunicación a través del puerto paralelo del computador. Será operable en el sistema operativo Windows XP, mediante una interfaz gráfica confiable y fácil de utilizar.

1.2. PLANTEAMIENTO DEL PROBLEMA

En la Universidad Industrial de Santander existe en la actualidad un gran interés de impulsar la investigación hacia la línea de los Sistemas Digitales; en el desarrollo de sistemas con Dispositivos Lógicos Programables aparece la necesidad de explorar todas las herramientas que estos ofrecen, y la arquitectura **Boundary- Scan** es una de ellas. Por otro lado, en los sistemas de desarrollo que se usan actualmente en la Universidad, se observa que la alta integración de los circuitos integrados dificulta la verificación física y de funcionamiento a través de sondas. La creación de un Analizador Lógico para la verificación de la tarjeta de desarrollo XC2S200E de Xilinx® que haga uso de **Boundary- Scan** soluciona específicamente esta dificultad.

1.3. OBJETIVOS

1.3.1. Objetivo general.

Desarrollar un analizador lógico mediante software para los **FPGA**s XC2S200E de Xilinx® para ser utilizado en el sistema de desarrollo DIGILAB 2E de Digilent® que posee la E^3T, basado en el estándar 1149 (**Test Access Port and Boundary-Scan Architecture** o Arquitectura del Puerto de Acceso de Prueba y Exploración de contorno) del **IEEE.**

1.3.2. Objetivos específicos.

- Redactar un documento con la información recopilada acerca de **Boundary-Scan** para comprender su funcionamiento e implementación así como la manera en que puede ser aplicado en sistemas de desarrollo. Este documento podrá servir más adelante como elemento de consulta.
- Desarrollo de un algoritmo que permita realizar el análisis lógico del **FPGA** XC2S200E en la tarjeta de desarrollo DIGILAB 2E de Digilent® utilizando el estándar **Boundary- Scan** que posea las siguientes características:

- Hacer uso de las herramientas de software existentes y disponibles entregadas por el fabricante (Xilinx®).
- La interfaz gráfica debe permitir la visualización del nivel lógico en los terminales del **FPGA** de manera gráfica como un diagrama de tiempos; así mismo debe permitir aplicar señales de prueba en las entradas del circuito que se evalúa para realizar diagnóstico del funcionamiento.
- El software de análisis lógico debe funcionar en entorno del Sistema Operativo Windows XP y utilizar el puerto paralelo del computador para comunicarlo con el **FPGA** de la tarjeta.
- Utilizar el formato **SVF** para desarrollar el algoritmo de comunicación con la tarjeta utilizada mediante **Boundary- Scan**.
- Facilitar la detección de fallas en sistemas con **FPGA**s que los estudiantes del curso de Sistemas Digitales realizarán en el futuro.
- Garantizar la confiabilidad de la aplicación y hacer que sea fácil de operar.

- Elaboración de un algoritmo para verificar el correcto funcionamiento del analizador lógico desarrollado, a manera de protocolo de prueba.

1.4. ORGANIZACIÓN DEL DOCUMENTO

El contenido del documento fue organizado en seis capítulos; el capítulo dos presenta datos generales acerca de los circuitos lógicos programables así como el proceso de implementación de un diseño en un circuito; el tercer capítulo compila toda la información acerca del estándar **Boundary- Scan** y los estándares relacionados que permiten llevarlo a la práctica en sistemas de desarrollo. En el capítulo cuatro se presentan las características del Analizador Lógico desarrollado y una explicación de cada una. Los capítulos cinco y seis describen el algoritmo desarrollado para el análisis lógico: el quinto detalla el proceso de análisis y captura y el sexto la interfaz gráfica para la visualización de resultados. El séptimo capítulo presenta los resultados de las pruebas realizadas. Al final del documento se plantean las conclusiones y recomendaciones de la investigación. Adicionalmente, se incluye un glosario con el fin de facilitar el acceso a las definiciones de algunos términos que pueden ayudar en la comprensión del texto.

2. GENERALIDADES DE LOS CIRCUITOS LÓGICOS PROGRAMABLES

2.1. CIRCUITOS VLSI

La creciente popularidad de los equipos electrónicos portátiles y de consumo masivo, desde computadoras, PDAs, teléfonos celulares o reproductores de MP3 hasta dispositivos que integran muchas de las funciones desempeñadas por estos (como teléfonos celulares con cámara digital incorporada y conexión inalámbrica a Internet), además de otras circunstancias, ha impulsado la investigación y el desarrollo en miniaturización de los circuitos electrónicos.

Este ha sido uno de los factores que han permitido que a través de los últimos años se haya estado desarrollando una revolución en el campo de la integración de los circuitos electrónicos. Son precisamente los circuitos integrados **VLSI** o de Muy Alta Escala de Integración los que se utilizan en los dispositivos en la actualidad especialmente en sistemas digitales.

En la actualidad, las soluciones que existen para estos circuitos, y que son presentadas en el Congreso internacional de Circuitos de Estado Sólido (**International Solid-State Circuits Conference***)* del **IEEE**, en la gama de 0.18-0.25 micrones hay 3 clases de enfoques o arquitecturas: 1- Microprocesadores de propósito general; 2- **DSPs** (**Digital Signal Processors**) o en español Procesadores de Señal Digital y 3- **ASICs** (**Application Specific Integrated Circuits**) o en español Circuitos Integrados para una Aplicación Específica. Los **FPGAs** o Arreglo de Compuertas Programables en el Campo se encuentran en la tercera categoría, y son una propuesta relativamente nueva para los circuitos electrónicos digitales de diferentes aplicaciones en las cuales son críticos el tamaño, la flexibilidad o el funcionamiento en tiempo real.

A finales de los años 40, los circuitos tenían un solo transistor, utilizado en un principio como amplificador; una década más tarde aparecen los circuitos integrados, compuestos por todos los elementos que componen un circuito en una sola pastilla de material semiconductor, por ejemplo una memoria o un regulador de voltaje.

Diferentes tecnologías en la fabricación de transistores permitieron crear circuitos integrados que implementan una función lógica: una función AND, una XOR. Estos circuitos de *lógica discreta* fueron utilizados hasta hace pocos años; sin embargo, en las últimas décadas aparecen los dispositivos lógicos programables que consisten en arreglos de cientos y hasta millones de

compuertas lógicas en una sola pastilla. Para el año 2005 vale anotar que Xilinx®, una de las empresas más grandes a nivel mundial en el área de lógica programable, fabrica **FPGAs** compuestos por hasta 8 millones de compuertas (el XC2V8000 de la familia Virtex de Xilinx que trabaja a más de 420 MHz[1] y con ratas de entrada y salida de datos de hasta 840 Mbits/s[2]), siguiendo además la ley de Moore que predice que este número se duplica cada 18 meses. Por otro lado, el precio al público de estos circuitos ha ido decreciendo a medida que su demanda aumenta.

El campo de la lógica programable, en desarrollo a partir mediados de la década de los 80s, ha sido utilizado últimamente gracias a su gran potencial, a su flexibilidad y adaptabilidad, en sistemas más amigables y con lenguajes de desarrollo de más alto nivel y por lo tanto más sencillos de manejar.

2.2. LÓGICA PROGRAMABLE

Los dispositivos lógicos programables pueden definirse en general como arreglos de elementos lógicos con una memoria configurable. La lógica programable se refiere a los circuitos en los cuales se puede personalizar la lógica una o varias veces, modificando las conexiones entre sus elementos. Las celdas son la unidad básica de todos los dispositivos lógicos programables. Las celdas utilizadas junto con las conexiones entre ellas, definen la función lógica que desempeña una configuración particular. Dicha función (que puede representar circuitos digitales bastante complejos) es implementada temporalmente en hardware y no en software, como ocurre con otros elementos electrónicos como por ejemplo los microprocesadores.

El hecho de permitir una flexibilidad en el diseño de sistemas digitales, así como el uso de lenguajes de más alto nivel, son algunas de las características más importantes de estos dispositivos. Dependiendo de su tamaño (en número de compuertas) pueden contener desde unas pocas ecuaciones hasta la implementación completa de un sistema de microprocesador con sus periféricos. Algunos de ellos se basan en tecnología de memoria re- escribible lo cual permite a los diseñadores hacer pruebas hasta encontrar el resultado deseado, mientras que otros pueden ser configurados una sola vez.

Los **PLDs** que existen en la actualidad son seudo dinámicos, en función a la forma en que permiten la configuración temporal por hardware o software de una función lógica. La ventaja de un **PLD**

[1] Xilinx®. Virtex™-II Platform **FPGA**s: Complete Data Sheet, Module 3. [s.l.]: Xilinx, 2004. p. 35.
[2] --------. Virtex™-II Platform **FPGA**s: Complete Data Sheet, Module 1. [s.l.]: Xilinx, 2004. p. 1-2.

frente a un **ASIC** full- custom (el cual es un sistema dedicado y fabricado a la medida para una aplicación), es su bajo costo teniendo aun la flexibilidad de ser configurado por el usuario.

Salcic y Smailagic[3] mencionan algunas técnicas de programación:

- Memoria Estática: Programación de células de memoria EPROM, EEPROM, RAM o FLASH lo cual permite modificar varias veces la programación de la estructura lógica.

- Establecimiento de conexiones mediante la técnica de anti- fusibles que consiste en la aplicación de un voltaje muy elevado para formar un corto circuito permanente entre dos nodos.

- Ruptura de fusibles, interrumpiendo conexiones.

Las últimas dos técnicas permiten programar una sola vez; en cuanto el fusible es quemado, no se puede volver al estado inicial.

Existen diferentes arquitecturas para implementar estos dispositivos, de tal manera que se encuentran disponibles muchos tipos de lógica programable. En general, acerca de arquitecturas que corresponden a dispositivos de alta complejidad pueden definirse en la actualidad las arquitecturas **CPLD** y **FPGA**, y son bastante representativas como dos enfoques diferentes de los fabricantes.

2.2.1. Arquitectura PLD

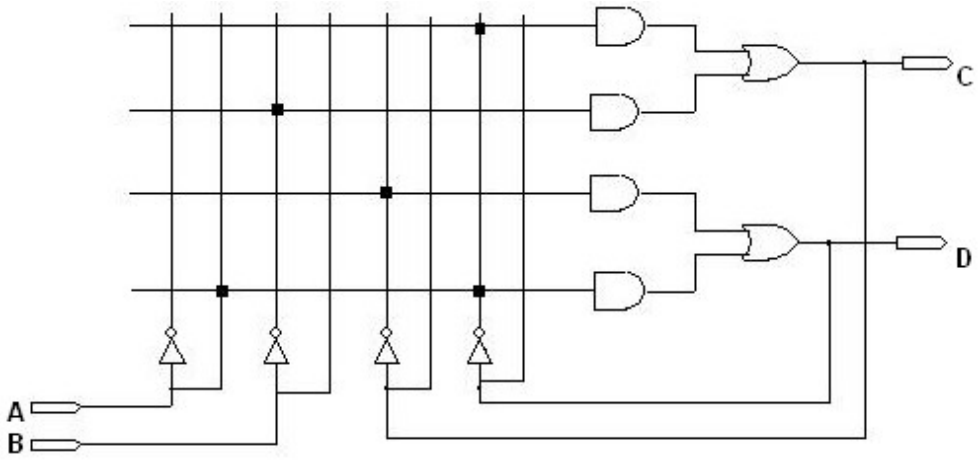
Corresponde a la arquitectura base de los **CPLDs** que se usan en la actualidad: los **CPLDs** utilizan en su arquitectura los PLDs como unidad básica, de manera que es necesario explicar inicialmente la arquitectura PLD para poder pasar a la descripción de la arquitectura **CPLD**.

Un PLD es un dispositivo de baja complejidad; cada circuito integrado correspondiente a un PLD contiene una operación lógica representada en forma de función booleana (suma de productos en este caso). Son implementados en tecnología CMOS. En la figura 1 puede observarse un esquema de la arquitectura de un PLD, con 2 señales de entrada, A y B, y dos señales de salida, C y D.

El circuito integrado es un arreglo de compuertas AND y OR sin conexiones fijas entre ellas; un PLD puede programarse abriendo una o varias conexiones a la entrada de cada compuerta AND, o dejándolas cerradas. Estas funciones pueden ser modificadas internamente, cambiando el cableado.

[3] SALCIC, Zoran y SMAILAGIC, Asim. Digital Systems Design and Prototyping: Using Field Programmable Logic and Hardware Description Languages. 2 ed. Norwell, Massachusetts: Kluwer Academic Publishers, 2000. p. 10- 17.

Figura 1. Arquitectura PLD



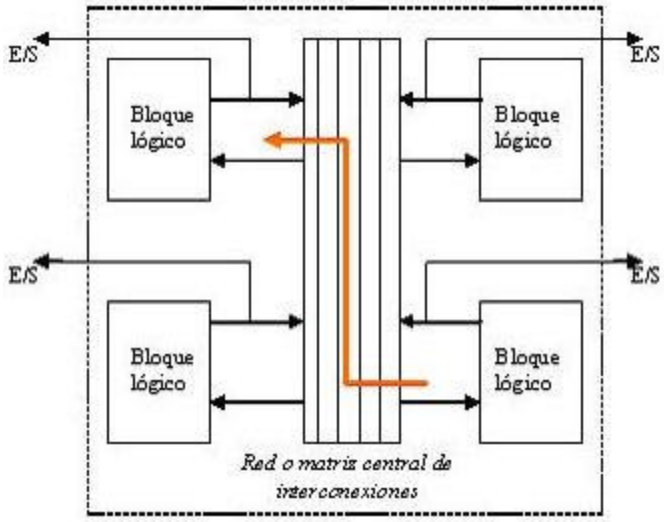
Fuente: XESS CORPORATION. RIVAS, Catalina, (Adaptación de). What are CPLDs and FPGAs? [en línea]. [s.l.]: Xess Corp, [s.f.]. [Consulta 24 May. 2004]. Disponible en Internet: <http://www.xess.com/fpgatut.htm>.

Como inconveniente, se puede mencionar el costo elevado por unidad de cada circuito integrado, teniendo además en cuenta que para circuitos de mayor tamaño y con más de 20 funciones lógicas, se necesitaba más de un PLD.

La arquitectura anteriormente presentada corresponde sólo a una de las formas en las cuales se implementan las funciones lógicas en un PLD. Cada fabricante da un nombre a la arquitectura que creó y lo patenta, por lo cual se habla de PAL, VPAL, GAL o PLA; todos ellos son dispositivos lógicos programables que tienen en común una baja complejidad, es decir, un número relativamente pequeño de compuertas.

A medida que aparecieron circuitos más complejos y se necesitaron funciones de mayor tamaño en cuanto al número de funciones lógicas, se creó una estructura compuesta por varios PLDs utilizados a manera de bloques lógicos: el **CPLD,** cuya estructura puede observarse en la figura 2 en la cual cada bloque que lo compone es un PLD.

Figura 2. Arquitectura CPLD



Fuente: XESS CORPORATION. RIVAS, Catalina, (Adaptación de). What are CPLDs and FPGAs? [en línea]. [s.l.]: Xess Corp, [s.f.]. [Consulta 24 May. 2004]. Disponible en Internet: <http://www.xess.com/fpgatut.htm>.

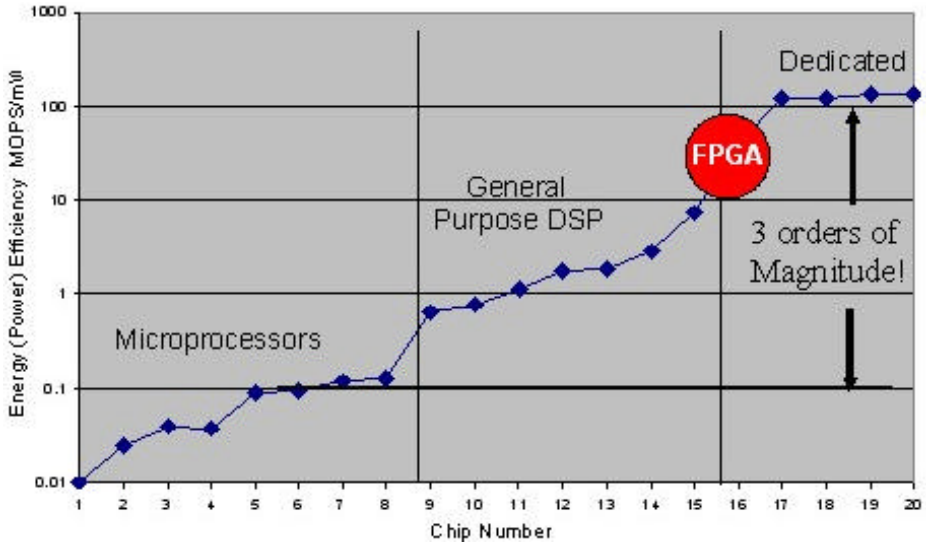
Un **CPLD** tiene una arquitectura regular con comportamiento predecible, ya que se puede saber cual es su desempeño; el retardo de propagación es conocido, e igual al retardo introducido por la propagación de la señal a través de la red central de interconexiones. Esta red hace que la distancia entre 2 compuertas cualesquiera sea fija, y además que la propagación de una señal entre ellas sea muy rápida. Sin embargo, debido a esta misma distribución, no permite la integración de un gran número de compuertas pues todos los bloques deben estar a una distancia similar de la matriz central de interconexiones.

2.2.2. Arquitectura Matricial

Un **FPGA** implementa otra arquitectura de los componentes VLSI íntegramente reconfigurables que es de estructura matricial. La integración que permite un **FPGA** actualmente está aumentando a una gran velocidad, y es precisamente un **FPGA** el que posee el mayor número de compuertas entre los circuitos lógicos programables. En un solo **FPGA** se encuentra implementada una gran cantidad de elementos sencillos de procesamiento, a lo cual se puede llamar *paralelismo espacial* pues pueden procesar datos de manera simultánea. Data Flux Systems, empresa que trabaja en el área de diseño y tecnologías de implementación en hardware, hace una descripción acerca de este paralelismo que permite implementar hasta 86000 millones de operaciones por segundo. Esta

cantidad medida como la eficiencia puede describirse como entre 10 y 100 Millones de operaciones por segundo por miliwatts, lo cual corresponde a 10 veces la eficiencia de un DSP[4]. La figura 3 muestra una comparación realizada por Data Flux Systems de la eficiencia de energía entre las diferentes arquitecturas implementadas en los circuitos **VLSI.** Se puede observar en la figura que la eficiencia de un FPGA se encuentra varios órdenes de magnitud por encima de las otras arquitecturas en la categoría de circuitos completamente configurables.

Figura 3. Eficiencia de energía de las diferentes arquitecturas VLSI



Fuente: DATA FLUX SYSTEMS INC. Scalable Computing Fabric [en línea]. [s.l.]: Data Flux Systems Inc., 2003. Disponible en Internet: <http://www.datafluxsystems.com/scf.htm>.

Cada **FPGA** posee una memoria interna en la cual se carga la configuración del dispositivo y que puede ser modificada. La tecnología utilizada para los elementos de la estructura matricial es SRAM, la cual es por naturaleza re-escribible, de allí que un **FPGA** puede ser programado varias veces.

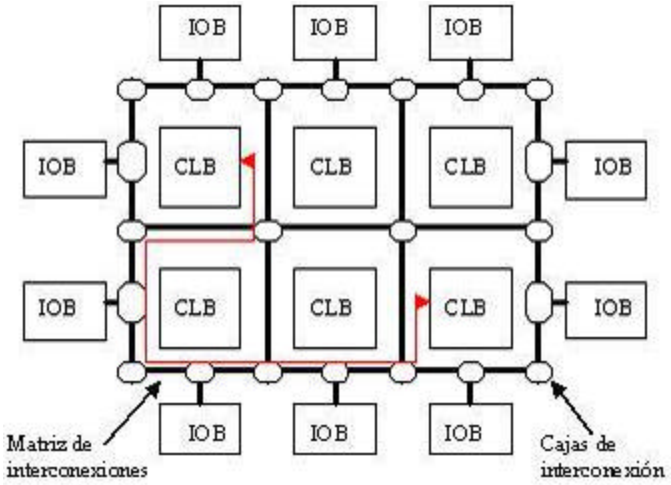
El fabricante Xilinx diferencia dos tipos diferentes de bloques estructurales: CLBs o en español Bloque de Lógica Configurable e IOBs[5] o en español Bloque de Entrada Salida; aunque este nombre puede variar según el fabricante, la estructura del FPGA no cambia; es un arreglo bidimensional de bloques lógicos programables e interconexiones entre ellos en una estructura matricial de interconexiones. El número máximo de **CLBs** que tiene un **FPGA** está alrededor de los 100 000,

[4] DATA FLUX SYSTEMS INC. Scalable Computing Fabric [en línea]. [s.l.]: Data Flux Systems Inc., 2003. Disponible en Internet: <http://www.datafluxsystems.com/scf.htm>.
[5] Xilinx®. Spartan-II 1.8 V **FPGA** Family: Complete Data Sheet, Module 2. [s.l.]: Xilinx, 2004. p. 1-2. Disponible en Internet: <http://www.xilinx.com/partinfo/ds031.pdf>..

duplicándose cada 18 meses (ley de Moore), con más de 3 000 000 de compuertas lógicas y más de mil millones de transistores.

Los CLBs conforman una matriz en el centro del circuito integrado, son idénticos y se encuentran distribuidos uniformemente; cada bloque es configurable independientemente, pudiéndose implementar en ellos funciones combinacionales; los IOBs (que son CLBs especializados) los rodean, conectándose con los pines e interconectando la lógica interna con el exterior del circuito integrado. Esta estructura se observa en la figura 4.

Figura 4. Arquitectura Matricial



Fuente: GONZÁLEZ, Juan. RIVAS, Catalina (adaptación de). Convirtiendo el hardware en Software: FPGAs [en línea]. En: x- ezine revista electrónica, No. 2 (Oct. 2002). [Consulta: 19 Dic. 2004]. Disponible en Internet: <http://x-ezine.todo-linux.com/x2/2x011-fpga.html>.

Las cajas de interconexión o **magic boxes** que se observan en la figura 4 son el otro elemento presente en un **FPGA**; también pueden ser configuradas para establecer la conexión entre varios CLBs, mientras que la matriz de interconexiones es simplemente el conjunto de líneas metálicas que conectan un elemento con otro. Tanto los bloques como las conexiones entre ellos pueden ser configuradas tantas veces se desee hasta obtener el comportamiento deseado y optimizado en términos de velocidad, flexibilidad, confiabilidad, etc.

Se observa en la figura 4 que la interconexión entre 2 bloques es segmentada (cada segmento corresponde al camino entre dos cajas de interconexión), y puede variar cada vez que se configura pues hay más de un camino para ir de uno a otro bloque. Por lo tanto, el tiempo de interconexión es imprevisible. Sin embargo, existen herramientas para optimizar esta distancia; el fabricante, entre

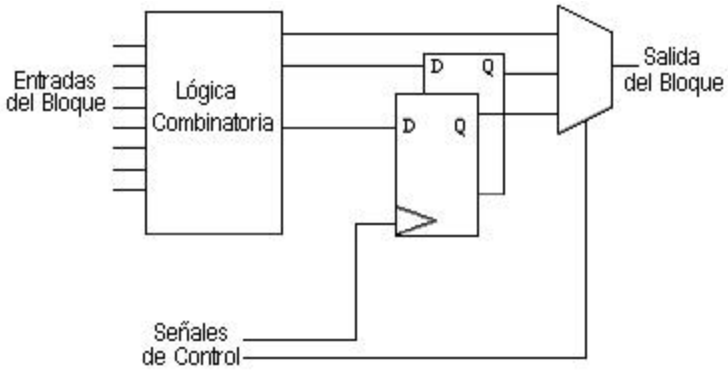
las múltiples herramientas de software para configuración de sus circuitos integrados, provee de software para optimizar automáticamente el "cableado" interno y realizar un mapeo para un desempeño óptimo.

La regularidad en la estructura de un **FPGA** es una ventaja de diseño ya que facilita su construcción así como el avance hacia una nueva generación con una escala de integración mayor. Su flexibilidad hace que según el diseño a realizar, la implementación, mapeo y enrutamiento se adapte al diseño particular y por lo tanto pueda optimizarse, llegando muy cerca del máximo desempeño para el circuito integrado que se está utilizando.

Las salidas de cada bloque lógico dependen de sus entradas y de señales de control. La lógica combinatoria del bloque se compone por compuertas lógicas básicas AND, OR, XOR; así mismo hay unos elementos de memoria (biestables).

Una representación simplificada de cada una de las celdas o bloques puede observarse en la figura 5. En ella se observa una lógica combinatoria en función de las entradas del bloque, y las señales producidas por esta lógica van a unos elementos de memoria de los cuales irán a la salida del bloque, según las señales de control. Un grupo de multiplexores permiten seleccionar la salida según las señales de control.

Figura 5. Esquema de un bloque lógico de un FPGA



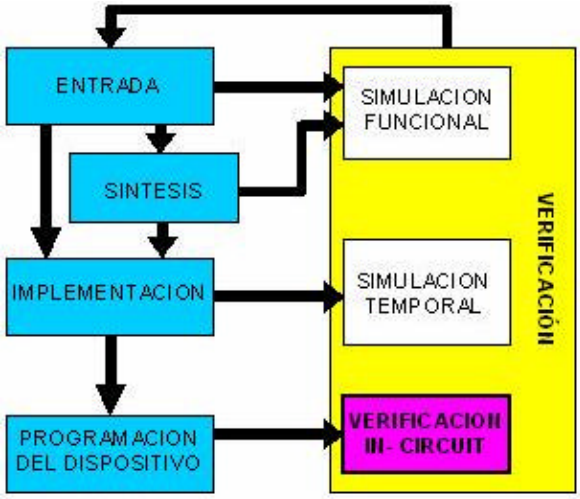
Fuente: Investigación del Autor

En cuanto a la tecnología utilizada para la configuración de los **FPGA**s se utilizan la técnica de anti-fusible y la de SRAM o memoria estática. Algunos vendedores representativos de **FPGA**s a nivel mundial son Xilinx, Actel, Altera, Cypress y Lucent Technologies. Xilinx y Actel utilizan el proceso de memoria estática para la configuración. Actel también utiliza la técnica de anti- fusible.

2.3. LA IMPLEMENTACIÓN EN UN DISPOSITIVO LÓGICO PROGRAMABLE

Sea cual sea el fabricante o el tipo de PLD que se utilice, el proceso de implementación de un diseño lógico desde su concepción hasta la programación en el circuito integrado se compone de un número de pasos, que pueden esquematizarse en el diagrama de la figura 6.

Figura 6. Diagrama de flujo de programación de un PLD



Fuente: XILINX®. RIVAS, Catalina (adaptación de). Xilinx 5 Software Manuals [cd-rom]. [s.l.]: Xilinx®, 2002. p. 1.

ENTRADA: la entrada del diseño puede hacerse mediante captura esquemática o algún tipo de descripción en lenguaje de alto nivel (**VHDL** o Lenguaje de Descripción de Hardware para Circuitos Integrados de Muy Alta Velocidad, **Verilog**, etcétera); incluso se puede describir un diseño como la conexión de varios bloques prediseñados y ofrecidos por el fabricante junto con el software.

SÍNTESIS: es el proceso de traducción de la entrada a un archivo en un formato que contiene la descripción de la implementación del diseño con compuertas lógicas.

IMPLEMENTACIÓN: corresponde a la fase del diseño en la cual se realiza el mapeo, enrutamiento y la ubicación de las **CLBs** que se van a utilizar. Este es un proceso crítico en el cual se analiza la distancia de cada ruta o los retardos críticos y el número de **CLBs** a utilizar (partición) y la función que cada cual va a realizar; de una buena implementación depende el buen desempeño del diseño, para lo cual es necesario tener en cuenta los parámetros de diseño: velocidad, densidad del número

de compuertas. Al final de este paso se crea un archivo de configuración para que el circuito integrado pueda ser programado.

PROGRAMACIÓN: Cuando se ha logrado el resultado deseado, se descarga en el circuito integrado el programa que se obtuvo en la etapa de implementación. Si está disponible la verificación **in- circuit**, en la siguiente etapa correspondiente a la verificación puede verse el resultado de la programación una vez realizada para confirmar que fue correcta y que los resultados obtenidos son similares a los esperados.

VERIFICACIÓN: como se puede observar en la figura 6, la verificación se realiza a varios niveles del proceso. La simulación da una idea del comportamiento del diseño, tanto después de la etapa de creación del circuito (para verificar su comportamiento funcional) como después de la etapa de implementación (para observar cuál sería el comportamiento tomando en cuenta los retardos reales de las señales internas), antes de la programación real del circuito integrado. Finalmente, existe una forma de verificación del funcionamiento del circuito integrado luego de haberse programado; se conoce como verificación **in- circuit** y permite comparar los resultados esperados con los reales de la configuración del circuito integrado. Esto es posible en los dispositivos programables **in- circuit** o **ICP**. Se verá en la siguiente sección con mayor detenimiento el proceso de verificación.

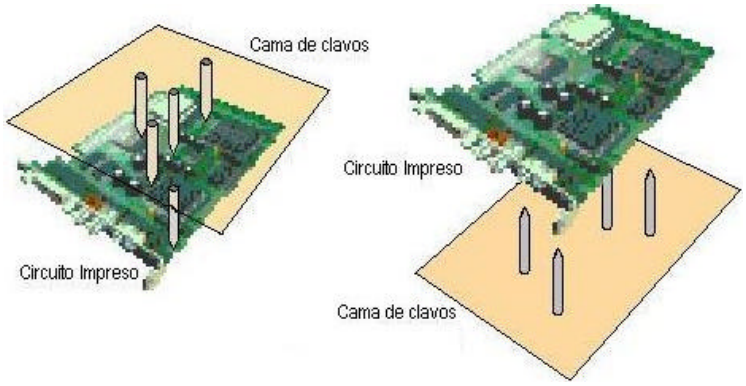
2.4. LA VERIFICACIÓN

Desde mediados de los años 70, se han utilizado varias técnicas para verificar el funcionamiento de las tarjetas electrónicas. Además de realizar simulaciones que permiten mediante software obtener una idea aproximada de lo que será el funcionamiento de un diseño antes de su montaje real, se deben realizar pruebas que permitan examinar un diseño que ya ha sido fabricado (o programado, en el caso de los circuitos programables).

Una manera de verificar, es observar los registros o memorias de configuración de los dispositivos si son programables; en los circuitos lógicos, se observa el nivel lógico de las señales presentes en el diseño, para comprobar que sean las esperadas cuando el sistema se encuentra en funcionamiento. También se pueden aplicar señales como estímulo a las entradas y observar las respuestas obtenidas en las salidas. La verificación entonces diagnostica problemas funcionales que pueden ser debidos al diseño implementado, pero también a problemas físicos como cortocircuitos, conexiones abiertas, etc.

Inicialmente, la técnica utilizada para las pruebas del funcionamiento físico de una tarjeta electrónica en la etapa de prototipo, fue la cama de clavos (**bed-of-nails**). La base de esta técnica es el acceso físico a cada uno de los pines que se desean sondear por lo que debe haber puntos de contacto entre las interconexiones de cobre del circuito eléctrico, y una superficie fija que contiene los "clavos" o sondas, como se observa en la figura 7. De esta manera y aplicando señales de prueba como estímulos, puede saberse la presencia o ausencia de señal, su orientación y la manera en que los elementos están unidos entre sí. La cama de clavos aún es utilizada en circuitos en los cuales la escala de integración no es tan alta, para probar un prototipo antes de fabricar en masa, pero a medida que los circuitos integrados se achican y tienen un mayor número de pines, el espacio físico necesario para aplicar esta técnica se reduce demasiado. Además, el costo de un sistema de verificación de este tipo es muy elevado ya que debe construirse "a la medida" para la tarjeta específica.

Figura 7. Diagrama de una cama de clavos



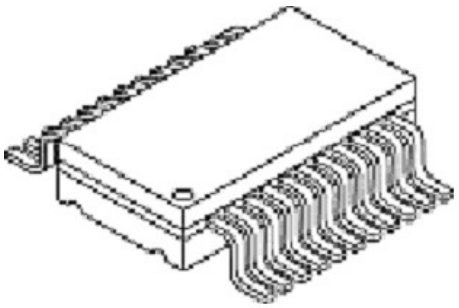
Fuente: Investigación del Autor.

Otra alternativa, que consiste en conectar sondas a cada terminal a analizar; es utilizada por osciloscopios y analizadores lógicos; aunque puede utilizarse para cualquier sistema, no resuelve el problema del espacio físico; al contrario, es aun menos operable si el espacio físico es limitado.

En segundo lugar, los circuitos integrados ya no sólo se instalan en la tarjeta con sus pines a través de agujeros, sino que aparecen los nuevos estilos de encapsulados de circuitos integrados, para montar sobre la superficie de la tarjeta electrónica, como los encapsulados SOIC, TSOP o QFP

entre otros como lo menciona Bennets[6] como el de la figura 8. Además, se utilizan los dos planos de la tarjeta, o aun más, aparecen los circuitos **multi- layer***, y las sondas que antes se ubicaban en el lado de la tarjeta donde estaban las soldaduras de cobre ya no pueden colocarse de esa manera.

Figura 8. Encapsulado SOIC (Small Outline Integrated Circuits)



Fuente: BENNETTS, R G. Boundary- Scan Tutorial [en línea]. V. 2.1. [s.l.]: Asset InterTech, 2002. p. 6. [Consulta: 5 Ago. 2004]. Disponible en Internet: <http://www.asset-intertech.com/PDFs/boundaryscan_tutorial.pdf>.

El avance en los equipos externos de prueba no es el mismo de los circuitos que ellos deben probar, y son de un mayor tamaño en varios órdenes de magnitud; otro problema en el uso de estos equipos es la alta frecuencia a la que trabajan los nuevos circuitos integrados y que crea problemas en la interfaz entre el equipo de prueba y el elemento que está siendo probado. El método de prueba utilizado por estos equipos requiere de vectores de prueba en un número proporcional al número de transistores del circuito, de manera que con las altas escalas de integración se hace cada vez más difícil realizar un control de pruebas adecuado.

Surgen entonces otros enfoques para realizar las pruebas **in- circuit**. Sus objetivos son minimizar los costos del equipo necesario para la realización de las pruebas, así como reducir el tiempo que se necesita para verificar el funcionamiento de un circuito integrado.

La verificación **in- circuit** no reemplaza la simulación; es una manera de comprobar el funcionamiento o de detectar los errores de un sistema electrónico, después de su configuración. Además, se deseaba encontrar un método que solucionara el problema del difícil acceso físico a las señales a medir.

---

[6] BENNETTS, R G. Boundary- Scan Tutorial [en línea]. V. 2.1. [s.l.]: Asset InterTech, 2002. p. 6. Disponible en Internet: <http://www.asset-intertech.com/PDFs/boundaryscan_tutorial.pdf>.

[*] La tecnología MLB o Multi- Layer Board, de tarjeta de múltiples capas, se encuentra en pleno desarrollo, como respuesta a la necesidad de circuitos cada vez más complejos y de mayor densidad.

La solución para estos problemas fue utilizar sistemas de prueba incrustados dentro del silicio en el momento de la fabricación del integrado. Uno de estos sistemas fue estandarizado con el nombre de **Boundary- Scan**; en 1990 se publica protocolo que describe su funcionamiento y requerimientos, con la publicación del estándar 1149.1 del **IEEE** creado por el grupo **JTAG** (**Joint Test Action Group**). La aplicación de este protocolo requiere que los fabricantes diseñen y elaboren nuevos circuitos integrados con la implementación de la arquitectura **Boundary- Scan**, lo cual implica agregar lógica adicional, por lo cual ha sido un proceso lento. Inicialmente pocos fabricantes estaban en capacidad de implementar las partes adicionales que se necesitaban, además de los archivos de descripción de su implementación. Ahora se puede escoger entre una amplia gama de circuitos integrados que poseen **Boundary- Scan** o **JTAG**, como también se le conoce. El estándar será descrito en la siguiente sección.

3. EL ESTÁNDAR BOUNDARY- SCAN

3.1. GENERALIDADES

A mediados de los 80, y a raíz de las dificultades que ya empezaban a presentarse para realizar medidas lógicas a nivel de tarjeta electrónica si los sistemas eran altamente complejos, debido a la falta de espacio para el acceso físico a los terminales a sondear, aparece una entidad llamada **Joint Test Action Group** (**JTAG**), grupo de trabajo adjunto al **IEEE** y derivado de un grupo europeo que se encontraba investigando para encontrar soluciones al mismo problema. Fue así como se desarrolló el estándar que existe en la actualidad y que compila las investigaciones propuestas por algunas compañías que ya trabajaban en la idea.

Definido en 1990 por el estándar 1149.1-1990 del **IEEE**[7], el **Test access Port and Boundary-Scan Architecture** o Arquitectura del Puerto de Acceso para Pruebas y Exploración por el Contorno es un método integrado para probar interconexiones en tarjetas electrónicas, implementado a nivel de cada circuito integrado; aunque no reemplazó las tecnologías utilizadas anteriormente (cama de clavos, sondas), sí se posiciona como una alternativa útil y cada vez más necesaria para suplir la necesidad de acceso físico a los nodos. Al estándar se le conoce en la bibliografía como **Boundary- Scan** o sencillamente como **JTAG**. A partir de ahora, a la arquitectura definida en el estándar 1149.1 del IEEE se le llamará en este texto simplemente **Boundary- Scan**, y a un circuito integrado que tenga esta arquitectura implementada se le llamará circuito **Boundary- Scan** o dispositivo **Boundary- Scan**. La última actualización de la norma fue realizada en 2001; en ella se define claramente la arquitectura y el Puerto de Acceso para Pruebas con todas las señales necesarias para su implementación, así como el juego de instrucciones obligatorias y optativas. Entre los problemas que busca solucionar la implementación de **Boundary- Scan** están:

- En la actualidad muchas tarjetas electrónicas tienen 2 caras e incluso son **multi- layer**; los caminos son inaccesibles a nivel físico para una sonda. Además muchos **CPLDs** y memorias **FLASH** están soldadas a la tarjeta electrónica sin sockets.

[7] IEEE WORKING GROUP P1149.1. IEEE Standard Test Access Port and Boundary-Scan Architecture [base de datos en dvd]. Piscataway, New Jersey: The Institute of Electrical and Electronics Engineers Inc, 2001. 200 p. Disponible en IEEE/IEE Electronic Library (IEL) 3.0, Versión en DVD, IELDVD040, Base de Datos Biblioteca Central Universidad Industrial de Santander.

- Con el desarrollo de la nanotecnología se pierde el acceso físico pues los componentes son tan pequeños que no tienen puntos de prueba; si es necesario probar un nodo sospechoso es imposible o al menos bastante complicado.
- Los circuitos integrados tienen una alta densidad, lo que se traduce en un gran número de terminales de entrada/salida, por lo cual no es tan evidente diferenciar un problema de diseño de uno de fabricación.
- No sólo el tamaño de los circuitos integrados se reduce, sino también el de las tarjetas electrónicas en las que se montan, lo cual también es un problema de falta de acceso físico.
- Adicionalmente a suplir estas necesidades por las cuales se creó, la creación e implementación del estándar también reduce la dificultad para comprobar un gran número de señales simultáneamente, ya que el número de canales de un analizador lógico que utiliza sondas siempre es limitado mientras que con **Boundary- Scan** se permite el acceso a todos los terminales del circuito integrado con lo cual puede extenderse el diagnóstico.

Algunos de los beneficios de la implementación y utilización de **Boundary- Scan** son: un mayor cubrimiento de los posibles defectos físicos de la tarjeta, una reducción del tiempo de pruebas, un menor costo de los equipos (un dispositivo de cama de clavos puede acceder a muchos integrados en una tarjeta al mismo tiempo; un analizador lógico también- dependiendo del número de sondas y siempre que exista un acceso físico a los nodos que se desean comprobar, pero sus precios son muy elevados), lo cual lleva a un incremento en la capacidad de diagnóstico para empresas pequeñas o a nivel académico.

Por otro lado, algunas de sus limitaciones son: la necesidad de una lógica adicional dentro del circuito integrado (lo cual eleva el precio de fabricación), una limitación en la velocidad a la cual se puede muestrear los datos limitada por el reloj que se utilice y por el número de celdas **BSC**, el acceso a las señales es dependiente de las instrucciones que cada fabricante incluya en su implementación de la arquitectura.

3.2. FUNCIONAMIENTO

El funcionamiento de **Boundary- Scan** es independiente de la función del circuito integrado, y por esto es necesario implementar una lógica adicional e independiente de la lógica interna correspondiente a la función del circuito integrado.

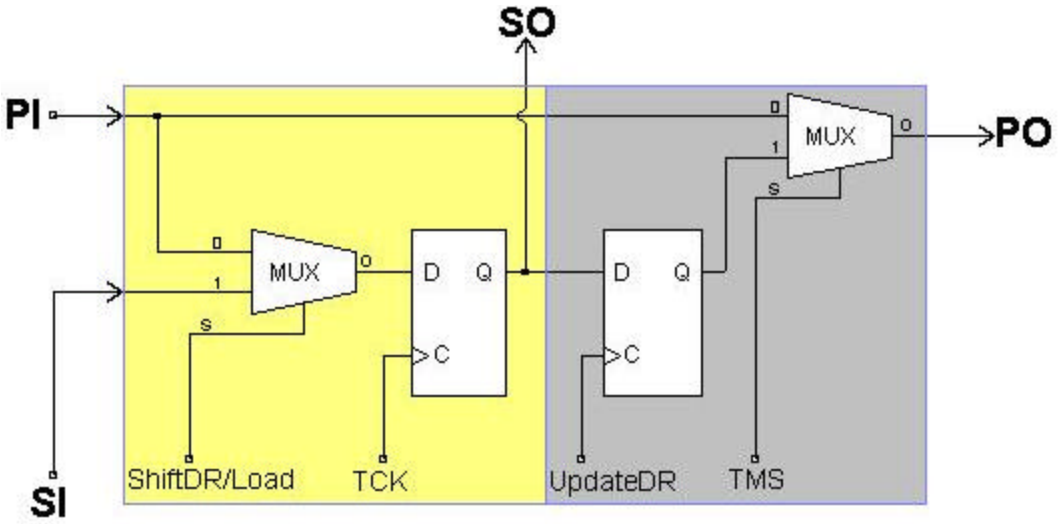
3.2.1. La celda Boundary- Scan

Para conocer los estados lógicos de los terminales sin utilizar sondas u otro tipo de elementos físicos, por cada terminal del dispositivo se agregan una o varias Celdas de **Boundary- Scan** (**BSC** por sus siglas en inglés); la celda es un elemento de memoria multipropósito que incluye básicamente multiplexores y latches.

Las **BSC** pueden ser vistas como un bus de sondas virtuales que permite observar y aplicar señales teniendo acceso a las estructuras de interconexión en una tarjeta electrónica. Las celdas en las entradas del dispositivo son las celdas de entrada y las de las salidas se llaman celdas de salida; las celdas de entrada y salida tienen la misma estructura básica. Un terminal bidireccional necesita de más de una celda **Boundary- Scan** para poder tener acceso a ella.

En la figura 9 puede observarse un diagrama básico de una posible implementación de una **BSC.** Sin embargo, hay que tener en cuenta que esta figura es sólo un esquema que sugiere el funcionamiento de la **BSC** pues el estándar no establece una arquitectura física determinada para ser implementado, dejándolo a decisión del fabricante. Xilinx y National Semiconductor utilizan esta estructura básica en sus implementaciones de la arquitectura **Boundary- Scan**.

Figura 9. Diagrama de una celda Boundary- Scan:



Fuente: IEEE WORKING GROUP P1149.1. RIVAS, Catalina (adaptación de). IEEE Standard Test Access Port and Boundary-Scan Architecture [base de datos en dvd]. Piscataway, New Jersey: The Institute of Electrical and Electronics Engineers Inc, 2001. p. 3. Disponible en IEEE/IEE Electronic Library 3.0, Versión en DVD, IELDVD040, Base de Datos Biblioteca Central Universidad Industrial de Santander.

Cada celda tiene dos partes: a la izquierda la compuerta de CAPTURA (en amarillo) y a la derecha la de ACTUALIZACIÓN (en gris)[8].

Cada celda está compuesta por dos flip- flops de tipo D que actúan como elementos de memoria, y tiene 4 modos de funcionamiento: normal, actualización, captura y desplazamiento serie. Debe garantizarse que tanto los modos de captura y de desplazamiento no interfieran con el paso normal de datos de las entradas a las salidas paralelas. A la captura de datos durante el funcionamiento normal del dispositivo se le conoce como captura "**on the fly**", esencial para el control y seguimiento en tiempo real del estado y funcionamiento de un elemento en un sistema[9]. Las señales que conectan la celda con el exterior son 4: la Entrada Paralela (**Pi**), Salida Paralela (**Po**), Entrada Serie (**Si**) y Salida Serie (**So**). La forma en que se conectan estas 4 señales, depende del tipo de celda: así, si la celda es de entrada, la Entrada Paralela **Pi** se conecta con el terminal de entrada correspondiente y la salida Paralela con la lógica interna; si la celda es de salida, la Entrada Paralela **Pi** se conecta con la lógica interna y la Salida Paralela **Po** se conecta con el terminal de salida correspondiente. Las celdas se conectan entre sí de manera serie, de manera que la Salida Serie **So** de cada celda se conecta con la Entrada Serie **Si** de la celda adyacente. Las señales **TMS** y **TCK** controlan la operación que se realiza en las celdas.

3.2.2. Arquitectura del estándar

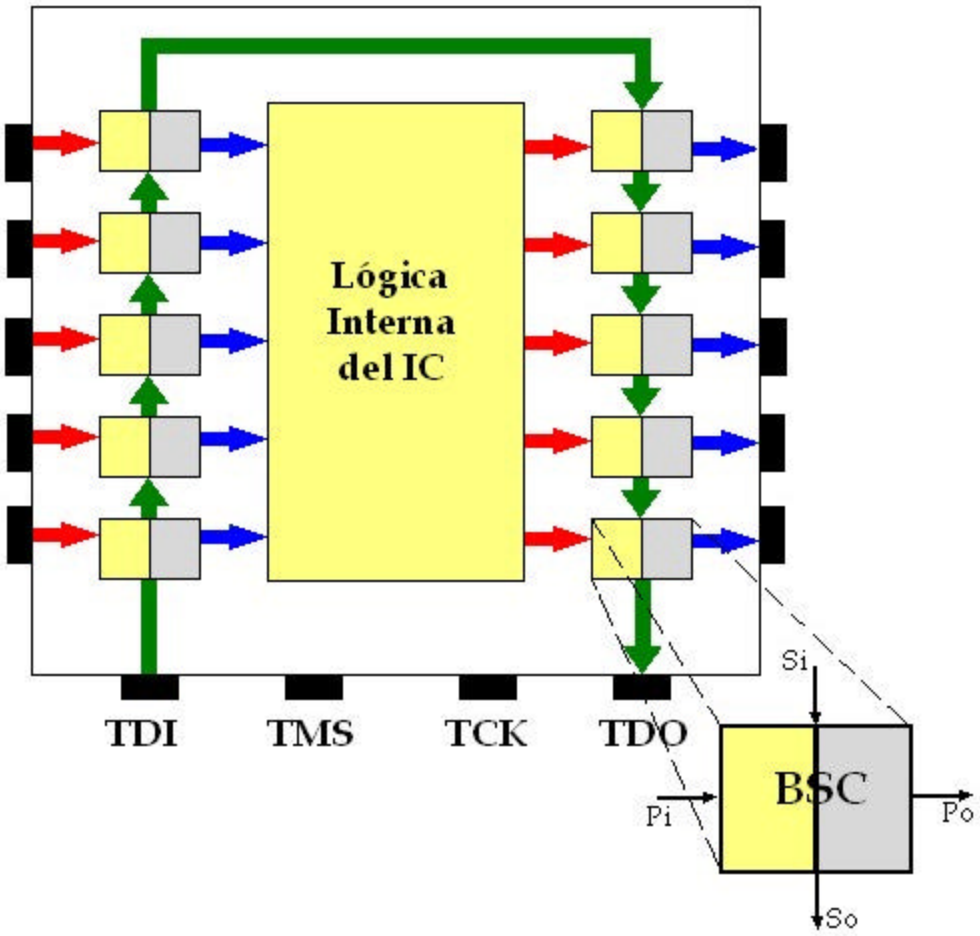
*** El Registro Boundary- Scan** Las celdas **Boundary- Scan** son dispuestas en un registro de desplazamiento en una configuración serie, con entrada paralela y salida paralela llamado Registro **Boundary- Scan** o **BSR**.

Entre cada uno de los terminales del Circuito Integrado (**IC**) y su lógica interna, se agregan una o varias celdas; al conectar la Salida Serie So de una celda con la Entrada Serie Si de la celda contigua, se forma el registro.

La configuración de este registro puede observarse en la figura 10.

[8] IEEE WORKING GROUP P1149.1. IEEE Standard Test Access Port and Boundary-Scan Architecture [base de datos en dvd]. Piscataway, New Jersey: The Institute of Electrical and Electronics Engineers Inc, 2001. p. 160. Disponible en IEL 3.0, Versión en DVD, IELDVD040, Base de Datos Biblioteca Central UIS.
[9] BENNETTS, R G. Boundary- Scan Tutorial [en línea]. V. 2.1. [s.l.]: Asset InterTech, 2002. p. 13. Disponible en Internet: <http://www.asset-intertech.com/PDFs/ boundaryscan_tutorial.pdf>.

Figura 10. Diagrama del principio de funcionamiento



Fuente: Investigación del autor.

Con más detalle los modos de funcionamiento de las celdas **BSC** son:

- **Modo Capture - Capturar datos en su Entrada Paralela Pi** (operación en rojo figura 10)**.** Esta operación de carga paralela causa que los valores de señal en los terminales de entrada del dispositivo se carguen en las celdas de entrada, y que los valores que pasan de la lógica interna del IC a sus terminales de salida, se carguen en las celdas de salida.

- **Modo Update - Actualizar datos en su Salida Paralela Po** (operación en azul figura 10)**.** Esta operación causa que los valores que se encuentran ya presentes en las células de salida pasen a los terminales de salida del dispositivo. Los valores de señal presentes en las células de entrada, pasan a la lógica interna del dispositivo. Permite aplicar estímulos al núcleo lógico.

- **Modo Shift - Sondear de manera serie datos desde su salida Serie So hacia la Entrada Serie Si de la celda contigua** (operación en verde figura 10)**.** Los datos o valores de respuesta pueden ser desplazados de manera serie a través del registro desde la salida serie So de una celda hacia la entrada serie Si de la siguiente, comenzando en el terminal Test Data In (**TDI**) que es la entrada serie del registro y finalizando en el terminal de salida Test Data Out (**TDO**) que es la salida serie del registro. Este principio funciona igualmente para varios dispositivos **Boundary- Scan** conectados entre sí de manera serie; esta concatenación crea un solo registro, así como un **TDI** y un **TDO** global.
- En el modo normal, la celda permite el paso de la Entrada Paralela **Pi** hacia la Salida Paralela **Po** de manera que no realiza ninguna operación en los flip-flops.

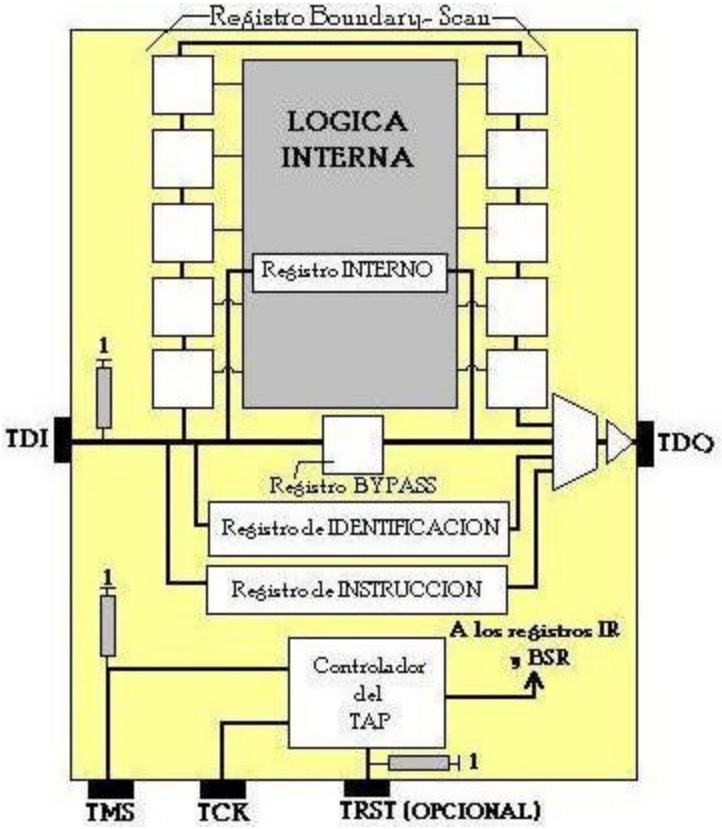
La presencia de este sistema de registro debe ser transparente, es decir, **Pi** pasa sin ningún cambio a **Po**. Estos registros adicionales a la lógica del dispositivo no afectan en nada su funcionamiento. Además de los terminales adicionales **TDI** y **TDO** (entre los cuales se conecta el registro **Boundary- Scan**), el estándar tiene otros 2 terminales, **Test ClocK** (**TCK**) que es la señal de reloj de prueba y **Test Mode Select** (**TMS**). **TCK** es totalmente independiente de todos los relojes del sistema, para garantizar que las operaciones de sondeo no interfieran con el funcionamiento del sistema digital. Los modos de operación se controlan mediante el terminal **Test Mode Select** (**TMS**).

De esta manera, para realizar pruebas en un dispositivo o en un conjunto de ellos (tarjeta electrónica), se cargan los valores de los estímulos necesarios en las entradas de manera serie a través de **TDI** hacia las celdas, aplicar estos estímulos y enviar las señales hacia las celdas de salida, capturar las respuestas y leer las celdas de entrada, y finalmente desplazar las respuestas hacia **TDO**.

La gran importancia de esta tecnología para el desarrollo de pruebas, es que permite el análisis tanto de un circuito aislado, como de todo un sistema en el lugar preciso donde se presentan mayor número de fallos, que es en la periferia (terminales de entrada y salida) de los dispositivos, sin interferir en su funcionamiento normal. El Analizador Lógico desarrollado, en adelante LABS por *Logic Analyzer by Boundary Scan*, se enfoca en el caso de una tarjeta electrónica con un solo dispositivo **Boundary- Scan** (FPGA) lo cual simplifica la comprensión del funcionamiento de **Boundary- Scan** así como la elaboración de los algoritmos necesarios para ejecutar operaciones **Boundary- Scan**.

*** El Puerto de Acceso para Pruebas** Además del Registro **Boundary- Scan**, la implementación del estándar **IEEE** 1149.1 dentro de un Circuito Integrado exige otra lógica adicional que controle su funcionamiento así como unos registros adicionales. Esta arquitectura se presenta a continuación en la figura 11.

Figura 11. Diagrama de la arquitectura interna de Boundary- Scan



Fuente: BENNETTS, R G. RIVAS, Catalina (adaptación de). Boundary- Scan Tutorial [en línea]. V. 2.1. [s.l.]: Asset InterTech, 2002. 58 p. [Consulta: 5 Ago. 2004]. Disponible en Internet: <http://www.asset-intertech.com/PDFs/boundaryscan_tutorial.pdf>.

Los cuatro terminales que fueron descritos en la sección anterior (**TDI**, **TMS, TCK** que son entradas y **TDO** que es salida) son obligatorios en la implementación y son las señales que controlan el funcionamiento del estándar. Una quinta señal, **TRST**, Señal de Reset asíncrono es optativa, activa en nivel bajo. Si no está presente, existe un reset sincrónico disponible: si **TMS** se mantiene en "1", con cinco pulsos consecutivos de **TCK** se garantiza el estado de Reset. Estas señales conforman el Puerto de Acceso para Pruebas o **TAP.**

A la lógica de control de toda la arquitectura se le conoce como controlador de **TAP**. Se representa como una máquina de estados sincrónica a **TCK** cuyo diagrama se puede observar en la figura 12 y cuya señal de control es **TMS**, esto indica que **TRST** (optativo), **TMS** y **TCK** son las señales que definen el estado del controlador del **TAP**. En la especificación se indica que las señales que genera controlan la operación de los registros y circuitería asociada con el estándar[10]**.**

En la figura 12 puede observarse también que si **TMS** = 1 durante 5 ciclos de reloj, el **TAP** llega al estado **Test-Logic-Reset** que es el estado inactivo; es decir, al aplicar un 1 lógico durante 5 ciclos de reloj se produce un Reset sincrónico. En el diagrama presentado sólo se observa el cambio de estado debido a la señal **TMS**; si la señal **TRST** está presente, un 1 lógico en esta señal (durante cualquiera de los estados) produce un cambio de estado a **Test-Logic-Reset**.

El estado **Run- Test/ Idle** es un estado inactivo pero de espera, previo a la selección de un registro (de instrucciones o de datos); dicha selección se realiza en los estados Select- Scan para iniciar el desplazamiento de una secuencia de bits.

Pueden observarse en la máquina de estados dos lazos, el de la izquierda corresponde a los registros de Datos (DR) y el de la derecha al Registro de Instrucciones (IR) y los estados en los 2 lazos son equivalentes. Según esto, **Select- DR Scan** selecciona un registro de Datos, mientras que **Select- IR Scan** selecciona el Registro de Instrucciones.

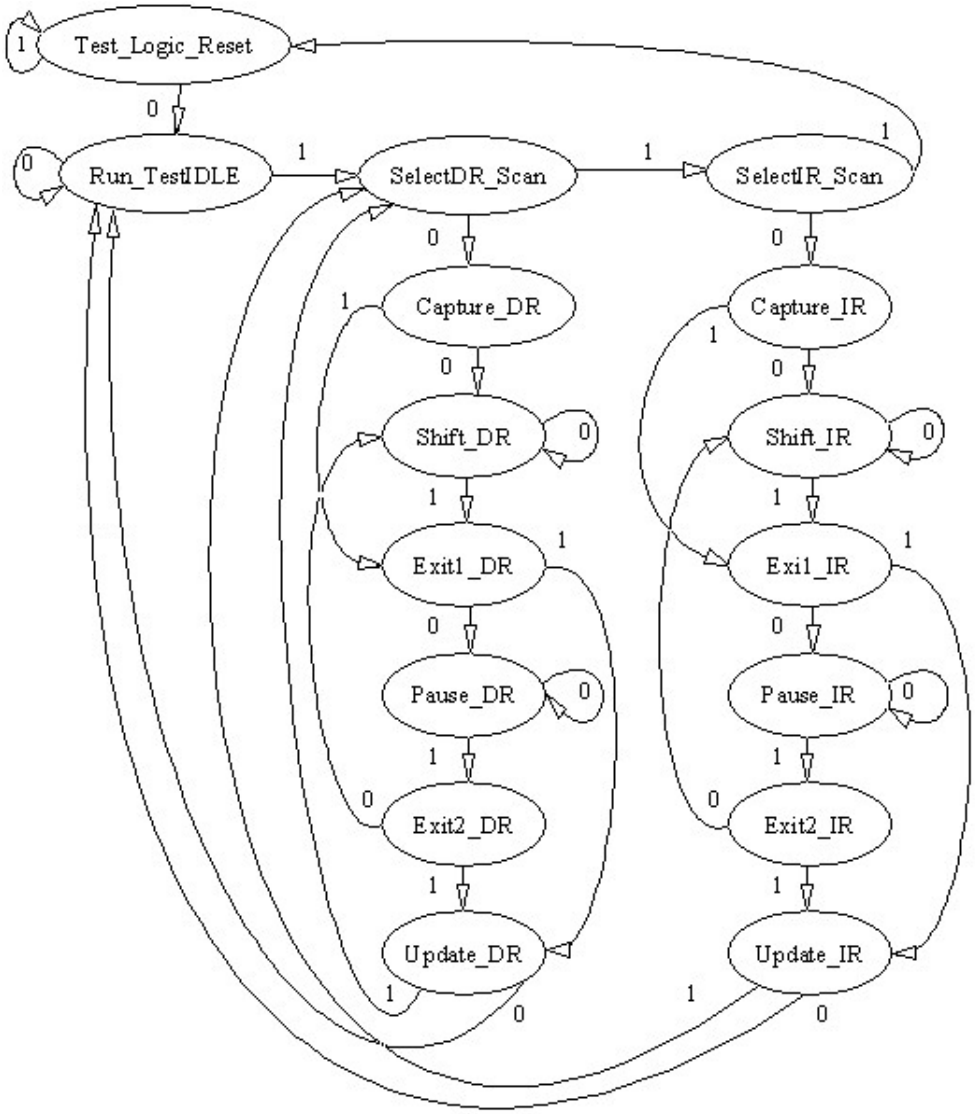
Durante el estado **Capture - DR** o el **Capture - IR**, los datos paralelos se cargan en el registro de desplazamiento; en el caso de el Registro de Instrucciones, los datos pasan del registro estático al de desplazamiento y en el caso de ser un Registro de datos, los datos se cargan a las celdas del Registro **Boundary- Scan**.

En el estado **Shift- IR** o el **Shift- DR** los datos se desplazan de manera serie a través del registro de desplazamiento. Al pasar al estado **Exit1**, se finaliza el desplazamiento de los datos.

El estado **Pause** detiene la operación transitoriamente. Durante **Exit2** puede realizarse el desplazamiento de datos adicionales. El registro estático correspondiente (para **Exit2- DR** el de datos y para **Exit2- IR** el de instrucciones) conserva los datos precedentes.

[10] IEEE WORKING GROUP P1149.1. IEEE Standard Test Access Port and Boundary-Scan Architecture [base de datos en dvd]. Piscataway, New Jersey: The Institute of Electrical and Electronics Engineers Inc, 2001. p 24. Disponible en IEL 3.0, Versión en DVD, IELDVD040, Base de Datos Biblioteca Central UIS.

Figura 12. Estados del TAP



Fuente : BENNETTS, R G. RIVAS, Catalina (adaptación de). Boundary- Scan Tutorial [en línea]. V. 2.1. [s.l.]: Asset InterTech, 2002. p. 34. [Consulta: 5 Ago. 2004]. Disponible en Internet: <http://www.asset-intertech.com/PDFs/boundaryscan_tutorial.pdf >.

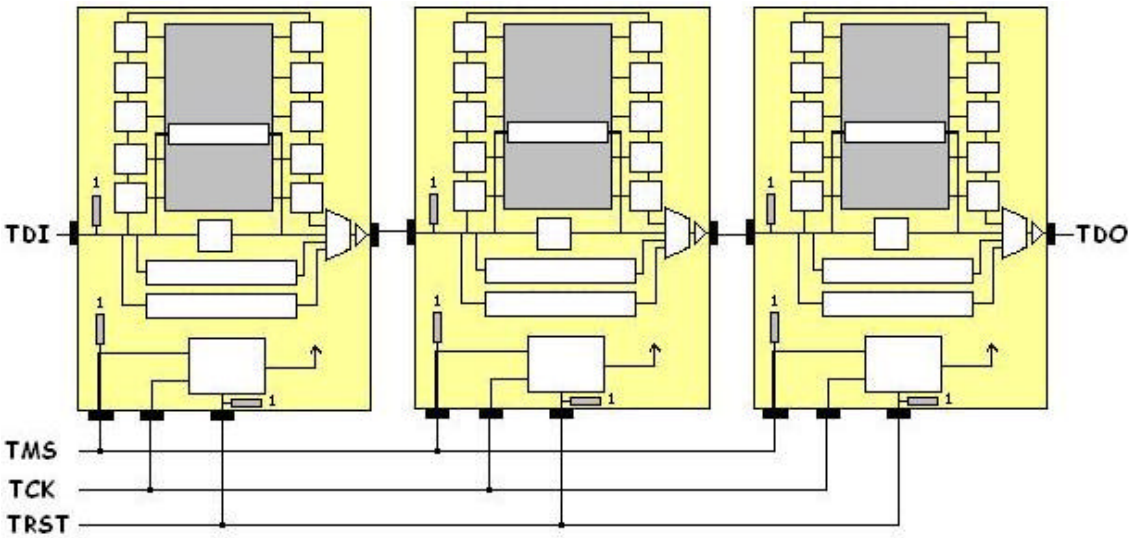
Durante el estado **Update** se realiza la actualización de los datos desplazados de manera serie a través del registro de desplazamiento, cargándolos en los registros paralelos. National Semiconductor, describe detalladamente cada uno de estos estados en un documento acerca de Boundary- Scan[11].

[11] NATIONAL SEMICONDUCTOR, Description of Boundary - Scan [en línea]. [s.l.]: National Semiconductor, 1996. p. 3- 5. Disponible en Internet: <http://www.national.com/ms/DE/DESCRIPTION_OF_SCAN-MISC.pdf>.

Las señales que genera el **TAP** llegan al Registro de Instrucciones y a cada una de las celdas **BSC** para controlar las operaciones que éstas realizan y la manera en que se pasa de uno a otro estado.

La arquitectura está definida de manera que en una tarjeta donde haya más de un circuito integrado con la implementación del estándar 1149.1, se mantenga la misma estructura de cuatro (o cinco) señales: **TDI, TDO, TMS, TCK** y (si está presente) **TRST**. Para lograr esto, los registros de todos los integrados se conectan de manera serie y las señales **TMS, TCK** y **TRST** de todos ellos se conectan en paralelo. Así se puede realizar el sondeo de los puertos de todos los circuitos integrados de manera serie. Tal como muestra la figura 13 puede observarse que el terminal **TDO** de cada circuito integrado se conecta a **TDI** del integrado contiguo de manera que se crea una sola cadena, en la cual existe un registro de desplazamiento conectado entre **TDI** y **TDO** globales.

Figura 13: Conexión de Circuitos Integrados con Boundary- Scan



Fuente. ASSET INTERTECH INC. RIVAS Catalina (adaptación de). Boundary- Scan Tutorial [en línea]. [s.l.]: ASSET InterTech, 2000. p. 15. [Consulta: 19 Ene. 2005]. Disponible en Internet: <http://www.ezytech.com/asset/pdfs/boundaryscan_tutorial.pdf>.

*** Otros Registros de la arquitectura Boundary- Scan** Todos los registros descritos en la arquitectura se sitúan de manera que su entrada serie es **TDI** y su salida serie es **TDO** y en un instante determinado solo uno de ellos se encuentra conectado entre **TDI** y **TDO**. Los registros que describe la arquitectura y que se pueden observar en la figura 11 son:

- El Registro **Boundary- Scan** anteriormente descrito.

- El Registro Bypass, de 1 bit de longitud. En cadenas de **Boundary- Scan** compuestas por más de un circuito integrado, este registro permite desplazar bits de **TDI** a **TDO** a través de la cadena **Boundary- Scan,** sin interferir con la lógica interna del circuito integrado.
- El Registro de Identificación, cuya implementación es optativa, es un registro de 32 bits en el cual se encuentra un código fijo de identificación correspondiente a cada dispositivo. Por ejemplo, para el FPGA Spartan 2S2000E_PQ208 de Xilinx®, este registro de identificación contiene el código XXXX 0000 1010 0001 1100 0000 1001 0011, o en hexadecimal X0A1C093.
- Algunos registros internos de la lógica del dispositivo.
- El Registro de Instrucciones que contiene la instrucción actual, registro de n bits (n=2), que se expondrá con detalle más adelante.
- Otros registros según fabricante.

Cuando se selecciona uno de los registros entre **TDI** y **TDO**, pueden realizarse 3 operaciones en él: la captura paralela, el desplazamiento en serie y la actualización en paralelo. El orden de estas 3 operaciones se establece mediante el diseño de la secuencia de los estados en el controlador de **TAP**. Según el registro seleccionado, algunas de estas operaciones son nulas (es decir, que no pueden ser ejecutadas) debido a la naturaleza del registro.

La instrucción que se carga y decodifica en el Registro de Instrucciones o IR, determina cuál es el registro escogido. Este Registro tiene 2 secciones: la primera de desplazamiento que está conectada entre **TDI** y **TDO**, y la segunda conectada en paralelo con la primera que es estática y que mantiene la instrucción vigente en cada instante determinado; esta sección puede tener una lógica de decodificación, dependiendo de la cantidad de instrucciones y del número de bits que tenga el Registro.

Las señales de control para este Registro IR se originan en el controlador de **TAP** y causan un desplazamiento en cualquiera de las 2 direcciones a través de la sección de desplazamiento (Shift) del Registro de Instrucciones, o causa que el contenido de la sección de desplazamiento pase a la sección estática (actualización paralela o Update). También puede realizarse la operación contraria (captura o Capture), desplazándose la instrucción estática a la de desplazamiento. La importancia del IR radica en que selecciona y controla las señales que vienen del controlador de TAP para seleccionar cuál será el Registro de Datos activo, de manera que la decodificación de la instrucción que se encuentra en la sección estática permite seleccionar uno sólo de los registros en un instante

determinado. Debe tener una longitud de al menos dos bits para poder codificarse las cuatro instrucciones imperativas (o tres en el caso de las versiones del estándar anteriores a 1994).

En el modo de captura, los 2 bits menos significativos (más cercanos a **TDO**) deben capturar **01**, ya que es exigido en el estándar. Un diagrama de este Registro de Instrucciones puede observarse en la figura 14.

Figura 14: Diagrama del Registro de Instrucciones

Fuente: BENNETTS, R G. RIVAS, Catalina (adaptación de). Boundary- Scan Tutorial [en línea]. V. 2.1. [s.l.]: Asset InterTech, 2002. p. 21. [Consulta: 5 Ago. 2004]. Disponible en Internet: <http://www.asset-intertech.com/PDFs/boundaryscan_tutorial.pdf >.

Se pueden señalar algunos aspectos en cuanto a la conexión física de las señales del TAP necesarias para el correcto funcionamiento de **Boundary- Scan:**

Las señales de entrada **TDI**, **TMS** y **TRST** en circuito abierto están conectadas a un "1" lógico mediante resistencias o transistores internos por seguridad:

- Para **TDI**, la razón es que si se selecciona el Registro de Instrucciones y el dispositivo tiene a **TDI** en circuito abierto, se carga y se ejecuta una instrucción de todo 1s, la cual es una instrucción de seguridad.
- Para **TMS**, en circuito abierto –con un valor lógico de 1- en un máximo de 5 ciclos de **TCK**, el controlador TAP de este dispositivo se va a su estado ***Test_Logic Reset***. En ese estado (estado de seguridad), y con circuito abierto, la lógica de **JTAG** está inactiva, pero el dispositivo continúa funcionando normalmente.
- Para **TRST**, un 1 lógico es el estado inactivo, entonces el dispositivo puede estar funcionando en cualquiera de los modos. Si **TRST** está presente, la señal de reset sincrónica (**TMS = 1; 5 x TCK**), es preferible a la señal **TRST** asíncrona.

3.2.3. Juego de Instrucciones

En la tabla 1 se enumeran las instrucciones estándar, conocidas como instrucciones públicas y que define el estándar 1149.1- 2001; cuatro de ellas son imperativas y deben ser implementadas; las otras seis son optativas. Las instrucciones más adelante serán descritas con mayor detalle.

Además de estas instrucciones enunciadas, pueden implementarse otras no definidas en el estándar, según criterio del fabricante.

El código binario correspondiente a cada una de las instrucciones y que se observa en la tabla depende del fabricante, aunque para la instrucción BYPASS el código imperativo es todo- unos, lo cual está definido en el estándar.

Cada una de las instrucciones hace que la máquina de estados del TAP pase de uno a otro estado, dependiendo del código binario que cada instrucción represente, de manera que para realizar una secuencia deseada de estados, debe ejecutarse una secuencia determinada de instrucciones.

Tabla 1: Instrucciones de **Boundary Scan** para Spartan-II de Xilinx®

	Instrucción	Registro Activo	Código Binario
Imperativas	EXTEST	**Boundary- Scan**	00000
	BYPASS	Bypass	11111
	SAMPLE	**Boundary- Scan**	00001 (funcionamiento normal)
	PRELOAD*	**Boundary- Scan**	No existe en Spartan II
Optativas	INTEST	**Boundary- Scan**	00111
	IDCODE	Identificación	01001
	USERCODE	Identificación	01000
	RUNBIST	Reg. de Resultado	No está implementado
	CLAMP	Bypass	No está implementado
	HIGHZ	Bypass	01010 (terminales de salida en alta impedancia)

Fuente: XILINX. RIVAS, Catalina (adaptación de). Application Note 188: Configuration and Readback of Spartan-II FPGAs Using Boundary-Scan [en línea]. V. 2.1. [s.l.]: Xilinx®, 2002. p. 3. [Consulta 25 May. 2004]. Disponible en Internet: <http://direct.xilinx.com/bvdocs/appnotes/xapp188.pdf>.

* En la última actualización del estándar (1149.1- 2001) la instrucción imperativa SAMPLE/PRELOAD fue separada en 2 instrucciones independientes, SAMPLE y PRELOAD. En el FPGA XC2S200E de Xilinx®, se implementa la versión de 1993 del estándar por lo que exite sólo la instrucción SAMPLE/PRELOAD.

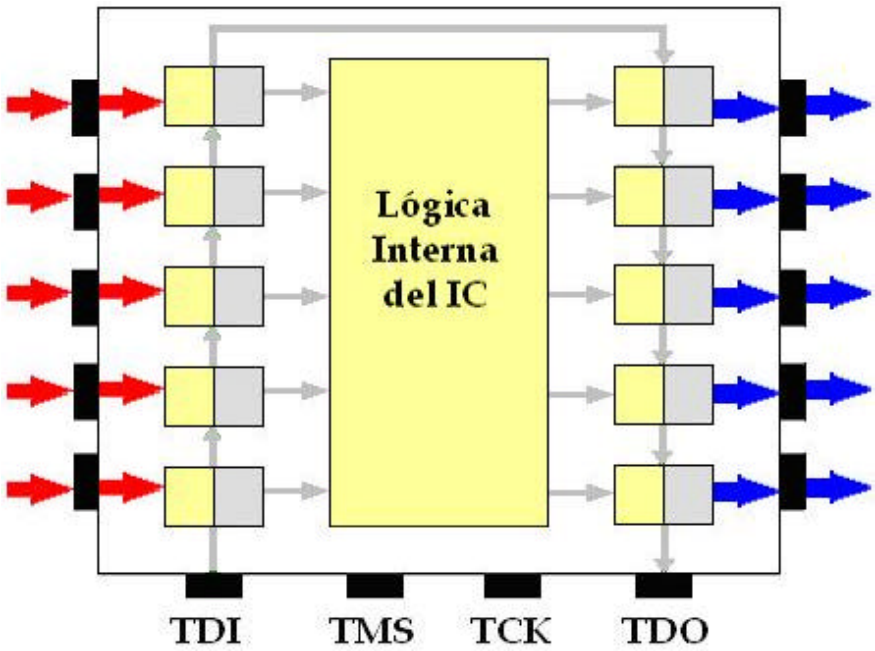
Las instrucciones **Boundary- Scan** son de dos tipos: invasivas o no invasivas. Las instrucciones no invasivas son aquellas que al ejecutarse permiten que el circuito integrado continúe en su modo de funcionamiento normal; las instrucciones no invasivas son BYPASS, SAMPLE, PRELOAD, IDCODE y USERCODE.

Por otro lado, las instrucciones invasivas interrumpen momentáneamente durante su ejecución la operación normal del circuito integrado, y son EXTEST, INTEST, RUNBIST, CLAMP Y HIGHZ.

A continuación, se presenta una explicación detallada de cada una de las instrucciones imperativas, así como de la instrucción INTEST.

***EXTEST (External TEST)** Esta instrucción es utilizada para probar la estructura de interconexión entre dos dispositivos en la tarjeta electrónica. Su diagrama se observa en la figura 15.

Figura 15. EXTEST



Fuente: Investigación del Autor.

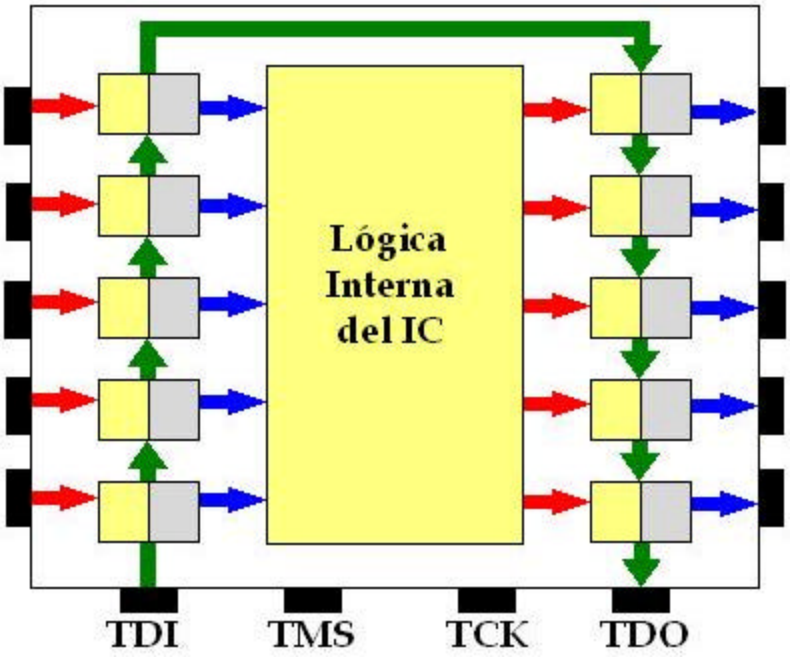
El uso de esta instrucción constituye la aplicación más utilizada de las estructuras **JTAG**, que es la prueba de circuitos abiertos, cortocircuitos y daños en la periferia de los circuitos integrados en una

tarjeta electrónica. En esta aplicación a las celdas se les conoce como “clavos virtuales”. Permite detectar fallas físicas en la conexión entre un circuito integrado y su periferia o entre dos circuitos integrados que tengan implementado el estándar.

En la instrucción ExTest (por External Test) se selecciona el Rregistro **Boundary- Scan** entre **TDI** y **TDO** al prepararse la ejecución de la prueba. Aunque ahora el código binario para la instrucción es definido por el fabricante, hasta antes de la revisión 2001 del estándar, el código para definirla era todo-ceros. EXTEST se utiliza para aplicar patrones de prueba a las estructuras de conexión alrededor de los circuitos integrados en la tarjeta. Las celdas tienen permiso para escribir en sus salidas (modo de prueba, mas no de funcionamiento normal). En la figura 15, las flechas azules indican que las celdas de salida escriben a las salidas del circuito integrado y hacia el circuito integrado contiguo durante el estado Update, y como indican las flechas rojas, se reciben las respuestas del circuito integrado anterior a través de las celdas de entrada durante el estado Capture.

***SAMPLE/PRELOAD** Se observa un diagrama de esta instrucción en la figura 16.

Figura 16. SAMPLE



Fuente: Investigación del autor.

Durante las instrucciones SAMPLE y PRELOAD, o la instrucción SAMPLE/PRELOAD de las versiones anteriores del estándar, todos los flip-flops de captura en las celdas de entrada y de salida cargan los estados lógicos de las señales a las cuales se encuentran conectados (esto durante el estado Capture). No se desconecta la lógica interna y el dispositivo permanece funcional: las señales que llegan de los terminales a las entradas de la lógica del dispositivo, pasan a éstas de manera normal e igualmente de las salidas lógicas a los terminales de salida durante el estado Update. De esta manera, se pueden observar las señales lógicas del sistema en cualquier momento mientras que éste se encuentra operativo, gracias a que en el estado Shift el registro conectado entre TDI y TDO es el Registro **Boundary- Scan.**

En el caso de la instrucción SAMPLE, los valores de los flip-flops de captura se desplazan de manera serie a través de **TDO** pudiendo ser capturados en ésta salida. Es ésta instrucción no invasiva la que puede ser utilizada como analizador lógico. Si la instrucción es PRELOAD, se utiliza para cargar un valor determinado en las celdas de entrada antes de ejecutar una instrucción invasiva.

Debe tenerse en cuenta que las operaciones de sondeo del estándar **Boundary Scan** pueden realizarse antes y después pero no durante la configuración. Después de la configuración (que es cuando se necesita la funcionalidad de analizador lógico) todas las instrucciones se encuentran disponibles para los dispositivos Xilinx®.

***BYPASS** BYPASS es una instrucción de inicialización o seguridad. El código binario correspondiente a esta instrucción debe ser siempre todo- unos. La razón de esta limitación es que -como se definió anteriormente- una desconexión involuntaria de **TDI** que produce una entrada de todo-unos, debe ejecutar una instrucción de seguridad o inicialización; ésta es la instrucción BYPASS, que no afecta el funcionamiento normal del sistema.

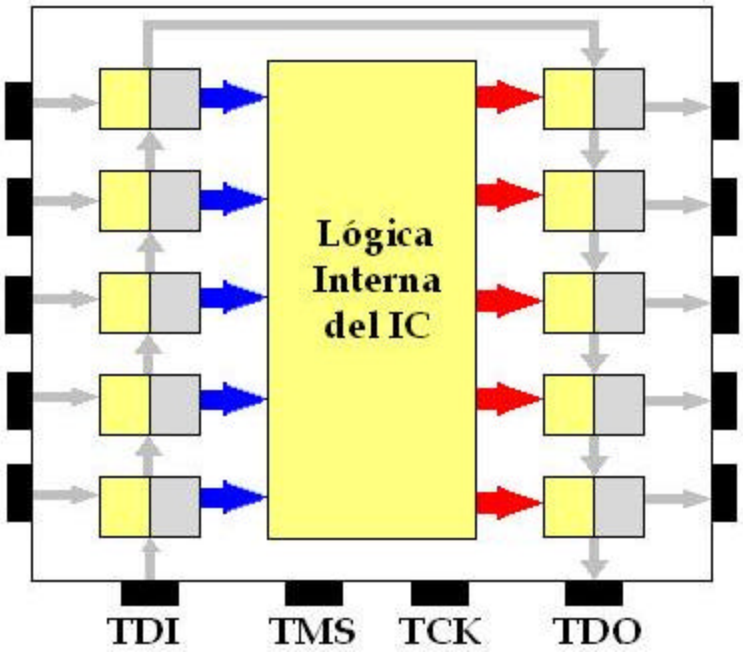
Al ser ejecutada, el registro Bypass (de 1 bit) se conecta entre **TDI** y **TDO**. El circuito integrado continúa operacional y se crea un camino de longitud mínima en la cadena de sondeo en el caso de que exista más de un circuito integrado en ella.

***INTEST (Internal TEST)** Como en EXTEST, el registro seleccionado también es el Registro **Boundary- Scan**. INTEST es una instrucción optativa.

INTEST permite probar el funcionamiento interno de un dispositivo. Se usa para identificar defectos como una variable incorrecta, o detectar algún defecto interno importante y se puede observar un diagrama en la figura 17.

Durante el estado Update, los latches de actualización permiten aplicar las señales que se encuentran almacenadas en las celdas a la lógica interior como estímulos y en el estado Capture, las señales de salida pasan a las celdas. De esta manera, al desplazar los valores cargados en el registro se pueden observar las respuestas en las salidas del circuito.

Figura 17. INTEST.



Fuente: Investigación del autor.

El hecho de que en las entradas de la lógica interna se están aplicando los valores que fueron cargados en las celdas **BSC**, y no los valores que llegan externamente a través de los terminales del circuito integrado, indica que al realizar esta operación el sistema **no es operativo**.

3.3. APLICACIONES

Existen aplicaciones de los Dispositivos Lógicos Programables en productos de consumo masivo como computadores personales y periféricos, así como productos del sector de las telecomunicaciones, procesamiento de imágenes entre otros. **Boundary- Scan** es un protocolo de gran utilidad para facilitar y acelerar la etapa de desarrollo y pruebas de sistemas que tienen estos circuitos Lógicos Programables, como DSPs, FPGAs, CPLDs o memorias. A pesar de que en los años siguientes a la definición del estándar hubo resistencia en la industria a implementar el

estándar debido a que se aumentan los costos de fabricación, por sus ventajas económicas en comparación con probadores físicos ésta tecnología ya está siendo utilizada por pequeñas empresas. Con los circuitos integrados diseñados específicamente para facilitar la ejecución de pruebas (del tipo **Design-for-test**), se busca que se pueda observar la presencia y orientación de los diferentes componentes de la tarjeta a probar y el vínculo entre ellos.

Boundary- Scan no sólo puede ser usado en la fase de fabricación de un producto para realizar las pruebas físicas y de funcionamiento, sino para su configuración y parar la eliminación de errores en los prototipos luego de realizar la configuración de circuitos integrados **VLSI** como son los **FPGAs**, **CPLDs**, **DSPs** (Procesadores de Señales Digitales) o microcontroladores.

La mayoría de los fabricantes de dispositivos lógicos programables de alta complejidad (Altera, Lattice, Xilinx entre otros) han incorporado lógica de **Boundary- Scan** en sus componentes (**FPGAs**, **CPLDs**, memorias, **DSPs**, microprocesadores o **ASICs**) lo cual permite que en la actualidad pueda escogerse entre una variedad de circuitos integrados con la arquitectura implementada y esta gama de dispositivos aumenta permanentemente.

Los principales proveedores de dispositivos con implementación del estándar Boundary- Scan son**:** Advanced Research Technology, Altera, Asset Intertech, Corelis, Goepel Electronic, Intellitech, JTAG Technologies B.V., National Semiconductor, Texas Instruments, Teradyne y Xilinx.

3.4. ESTÁNDARES UTILIZADOS RELACIONADOS CON BOUNDARY- SCAN

3.4.1. BSDL (Boundary- Scan Description Language)

Aunque el estándar 1149.1 no definió la implementación física, al advertirse que los fabricantes empezaban a implementarlo se vio la necesidad de crear un subcomité para desarrollar un lenguaje que estandarizara la descripción de dicha arquitectura, sus registros e instrucciones. Al lenguaje se le llamó BSDL o Lenguaje de Descripción de **Boundary- Scan**, por sus siglas en inglés. fue aprobado como el estándar 1149.1b del **IEEE** en 1994.

BSDL es una porción del lenguaje **VHDL** pues como se indica en el documento publicado por IEEE y que define el estándar **Boundary Scan**[12]**,** usa algunos de sus elementos sintácticos y comparte con éste lenguaje un grupo de palabras reservadas, y describe todos los elementos de la arquitectura y operación de **Boundary- Scan** en un dispositivo y además incluye información de sus terminales (si son de entrada o salida, cuáles son tierras, etc.). Su formato es ASCII lo que lo hace legible y modificable en un editor de texto, y su estructura está enfocada a la automatización de pruebas, ya que saber cómo se implementó la arquitectura de **Boundary Scan** permite desarrollar herramientas para conocer o controlar los estados del TAP o inclusive los estados lógicos de los terminales del dispositivo. De esta manera, un archivo **BSDL** puede ser usado como entrada de un proceso que involucre enviar datos a través de los terminales **Boundary Scan** hacia los registros internos del dispositivo.

Cada fabricante proporciona los archivos **BSDL** (antes de configuración del circuito integrado) correspondientes a cada una de sus referencias (**FPGA**s, **CPLD**s, elementos de almacenamiento). En el caso de Xilinx®, se dispone de los archivos **BSDL** tanto en el software de desarrollo (en este caso se utilizó el sistema de desarrollo Xilinx® ISE 5) como en Internet*.

***Partes de un archivo BSDL** En esta sección se describen los elementos que componen un archivo BSDL. Para cada uno de ellos se dará un ejemplo tomado del archivo **BSDL** xc2s200e_pq208.bsd, correspondiente al FPGA XC2S200E- PQ208 de Xilinx®[13]: , archivo que se presenta en el anexo D de este documento.

- Declaración de la entidad: mediante la estructura *entity* se enuncia el nombre con el cual se identifica el dispositivo.

Ejemplo:

```
entity XC2S200E_PQ208 is
```

- Parámetro genérico. Son atributos por defecto o externos a la entidad, como por ejemplo el tipo de paquete, y dependen de la forma en la cual cada fabricante describa y organice las diferentes referencias que ofrece.

---

[12] IEEE WORKING GROUP P1149.1. IEEE Standard Test Access Port and Boundary-Scan Architecture [base de datos en dvd]. Piscataway, New Jersey: The Institute of Electrical and Electronics Engineers Inc, 2001. p.131. Disponible en IEL 3.0, Versión en DVD, IELDVD040, Base de Datos Biblioteca Central UIS.

* Se encuentran los archivos BSDL de Xilinx en http://www.xilinx.com/support/sw_bsdl.htm.

[13] XILINX®. What is BSDL, and how do I read a BSDL file? [en línea]. Answer Record # 8350 en: Answers Database. Disponible en Internet: <http://www.xilinx.com/xlnx/xil_ans_display.jsp?getPagePath=8350>.

Ejemplo:

```
generic (PHISICAL_PIN_MAP : string := "PQ208");
```

- Descripciones de la entidad: A través de la descripción de la entidad se establece una lista de los puertos que corresponden al dispositivo que se enunció, indicando si son de tipo **bit** (si es un terminal de entrada o salida) o de tipo **bit_vector** o vector de bits, caso en el cual más de un terminal corresponde a la misma señal, como **GND** (tierra) o **VCC** (voltaje de alimentación); además indica su orientación (de entrada, salida o bidireccional).

Ejemplo:

```
CCLK_P155: inout bit;
GND: linkage bit_vector (1 to 24);
TCK: in bit;
TDI: in bit;
TDO: out bit;
```

En el ejemplo anterior, la primera línea corresponde a un terminal de reloj; es bidireccional de tipo bit; la segunda línea corresponde a la tierra o **GND**, de tipo vector de bits; la tercera y cuarta líneas son las señales de **Boundary- Scan** TCK y TDI, son señales de entrada (in) de tipo bit. Finalmente, la quinta línea corresponde también a una señal del **TAP** (**TDO**) pero en este caso es una señal de salida.

- Declaraciones tipo "USE". En un archivo **BSDL** se utilizan constantes, tipos de variables o atributos que corresponden al lenguaje **VHDL** y las definiciones de estos elementos se encuentran en paquetes externos al archivo **BSDL** mismo; estos paquetes se deben declarar con el fin de que el archivo pueda ser interpretado de manera correcta.

Ejemplo:

```
use std_1149_1_1994.all
```

- Atributos. La estructura *attribute* permite describir atributos correspondientes al archivo BSDL, como la versión del estándar que se está utilizando (1993, 2001).

Ejemplo:

```
attribute COMPONENT_CONFORMANCE of XC2S200E_PQ208 : entity is
"STD_1149_1_1993";
```

- Mapeo de los terminales. Se mapean las señales lógicas en los terminales físicos del circuito integrado. A una señal puede corresponder más de un terminal, por ejemplo, en el caso de **GND** pues por lo general más de un terminal corresponde a la tierra del circuito integrado.

Ejemplo:

```
"CCLK_P155:P155," &
"GND:(P1,P12,P19,P25,P32,P39,P51,P65,P72,P79," &
"P85,P92,P103,P117,P124,P131,P137,P144,P158,P170," &
"P177,P183,P190,P197)," &
"TCK:P207," &
"TDI:P159," &
"TDO:P157," &
```

- Identificación del **TAP**. La estructura *attribute* permite además declarar las señales del **TAP**. Xilinx® no implementa el **TRST** sincrónico, por lo cual este fabricante declara las 4 señales **TDI**, **TDO**, **TMS** y **TCK** con sus características; para **TCK** se enuncia además su frecuencia en Hertz.

Ejemplo:

```
attribute TAP_SCAN_IN   of TDI : signal is true;
attribute TAP_SCAN_MODE  of TMS : signal is true;
attribute TAP_SCAN_OUT   of TDO : signal is true;

attribute TAP_SCAN_CLOCK of TCK : signal is (33.0e6, BOTH);
```

- Descripción del **TAP**: El archivo BSDL describe además el resto de la arquitectura de **Boundary- Scan:**

    - Descripción del Registro de Instrucciones. Describe la longitud en bits de este registro expresada como número decimal, así como los códigos operacionales binarios de cada una de las instrucciones.

    - Descripción del Registro de Acceso. En esta sección se describe cuál es el registro "activo" (es decir, que se encuentra entre **TDI** y **TDO**) para cada una de las instrucciones. Presenta una lista de los registros y a continuación las instrucciones que corresponden a cada uno.

    - Descripción de los Registros de Identificación y de Usuario. Indica el código que se almacena en estos registros según la entidad que describe el archivo **BSDL**.

    - Longitud del Registro **Boundary- Scan.** Indica la longitud en bits de este registro expresada como número decimal.

Ejemplo:

```
attribute INSTRUCTION_LENGTH of XC2S200E_PQ208 : entity is 5; --Longitud del IR

attribute INSTRUCTION_OPCODE of XC2S200E_PQ208 : entity is
"SAMPLE (00001)," &
```

```
"INTEST (00111)," &
"USERCODE (01000)," &
"IDCODE (01001)," &
"HIGHZ (01010)," &
"JSTART (01100)," &
"RESERVED (00110)," &
"CFG_OUT (00100)," &
"CFG_IN (00101)," &
"USER2 (00011)," &
"USER1 (00010)," &
"EXTEST (00000)," &
"BYPASS (11111)";   –-Códigos de las instrucciones

attribute IDCODE_REGISTER of XC2S200E_PQ208 : entity is
"XXXX" &-- version
"0000101" &-- family
"000011100" &-- array size
"00001001001" &-- manufacturer
"1";-- required by 1149.1 –- Código de Identificación

attribute USERCODE_REGISTER of XC2S200E_PQ208 : entity is
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"; –- Código de usuario

attribute BOUNDARY_LENGTH of XC2S200E_PQ208 : entity is 1022; –-Longitud de BSR
```

En el lenguaje BSDL como en VHDL, para agregar un comentario se escribe después de dos guiones.

- Descripción del Registro **Boundary Scan**. La información que contiene este campo lo hace uno de los más importantes para poder generar una cadena de bits para realizar el análisis lógico de los estados del circuito; provee información acerca de la cantidad de celdas que tiene este registro, y para cada una de ellas su tipo y señales de control. Por lo general a los terminales de entrada corresponde una celda, a los de salida dos y a los bidireccionales tres. Sin embargo, casi todos los terminales se definen como bidireccionales para poder sondear información en cualquier dirección. Para cada celda, hay una línea en el archivo **BSDL**; cada línea tiene el número de celda y 7 campos: tipo de celda, nombre de puerto, función de la celda (**input**, **output3** o control), estado seguro (1, 0, Z, X), número de la celda de control, valor de control de desactivación y valor al desactivar. Los últimos 3 campos existen sólo en las celdas de salida.

Ejemplo:

```
attribute BOUNDARY_REGISTER of XC2S200E_PQ208 : entity is
-- cellnum (type, port, function, safe[, ccell, disval, disrslt])
"   0 (BC_1, *, controlr, 1)," &
"   1 (BC_1, IO_P160, output3, X, 0, 1, PULL0)," & --  PAD84
"   2 (BC_1, IO_P160, input, X)," & --  PAD84
"1019 (BC_1, *, controlr, 1)," &
"1020 (BC_1, CCLK_P155, output3, X, 1019, 1, PULL1)," &
"1021 (BC_1, CCLK_P155, input, X)";
```

Por cada terminal bidireccional existen 3 celdas **Boundary- Scan**: una de control, una de salida y una de entrada. Los terminales son por defecto bidireccionales para no limitar las pruebas que se puedan realizar y que haya desplazamiento de señales en los dos sentidos. La tercera celda controla justamente la dirección en señales bidireccionales.

- Advertencias. Al final se presenta una sección que enumera los mensajes de advertencia que puedan ser útiles al hacer uso de este archivo.

Ejemplo:

```
attribute DESIGN_WARNING of XC2S200E_PQ208 : entity is
"This BSDL file must be modified by the FPGA designer in order to" &
  "reflect post-configuration behavior (if any)." &
```

Finalmente, es importante recordar que el archivo **BSDL** que entrega cada fabricante refleja la arquitectura **Boundary- Scan** de cada dispositivo antes de ser configurado o programado; es decir, si un terminal puede ser configurado como entrada o salida (lo cual sucede casi siempre), en la descripción del Registro **Boundary- Scan** habrá tres líneas correspondientes a ese terminal (una por cada Celda: de entrada, salida y control); sin embargo, si dicho terminal es configurado en el diseño como una entrada, el archivo BSDL debe ser modificado para reflejar eso. Estas modificaciones se verán más adelante.

3.4.2. SVF (Serial Vector Format)

*** Definición** La especificación SVF o Formato de Vector Serial por sus siglas en inglés, es un formato estándar para el intercambio de información a través de un bus de datos con el fin de realizar operaciones **Boundary- Scan.** Fue desarrollado en 1991. Los derechos de SVF son controlados por Asset Intertech, pero el estándar es distribuido gratuitamente[14].

Aunque **SVF** no describe explícitamente los estados del bus en cada pulso de **TCK**, ni los estados del **TAP**, las operaciones **Boundary- Scan** consisten precisamente en operaciones de sondeo y movimientos entre uno y otro estado estable de la máquina de estados que representa al **TAP**.

El formato de un archivo SVF es ASCII, y consiste en un grupo de instrucciones o declaraciones que se componen de un comando y un grupo de parámetros; cada instrucción puede utilizar más de una línea y debe terminar en punto y coma, pero cada una de las líneas del archivo tiene un máximo

[14] ASSET INTERTECH, INC. Serial Vector Formal Specification [en línea], Revision E. [s.l.]: Asset InterTech, 1999. 26 p. Disponible en Internet: <http://www.asset-intertech.com/support/svf.pdf>.

de 256 caracteres. Cada una de estas declaraciones expresa cadenas de bits que pueden ser estímulos, resultados esperados o máscaras para comparar la salida esperada con la obtenida. Estos datos servirán para controlar el bus de datos del **TAP.**

SVF no diferencia entre mayúsculas y minúsculas, permite insertar comentarios (anteponiendo // a la línea) y los datos a ser sondeados se expresan entre paréntesis en formato hexadecimal.

Los datos a ser sondeados (la cadena de bits que también se conoce como Vector Serial) no pueden tener una longitud mayor a aquella que se especificó. En cuanto al orden de los bits, el menos significativo es el primero en ser desplazado por **TDI** a través de los registros y hacia **TDO**, lo cual es consistente con la convención que utiliza **Boundary- Scan**.

Para poder entender con mayor facilidad la manera en que se definen las instrucciones en formato SVF, debe entenderse que como bien lo define el estándar, la arquitectura **Boundary- Scan** hay dos clases de registros con propósito diferente: por un lado, el Registro de Instrucciones y por otro lado los registros de datos que deben ser al menos dos: BSR y Bypass[15]. El Registro de Instrucciones se diferencia de los demás en que a través de una lógica que decodifica su contenido, se selecciona uno de los otros registros para realizar en él operaciones de captura (Capture), desplazamiento (Shift) o actualización (Update) como ya se anotó. Al Registro de Instrucciones se le identifica como IR, a los demás registros como DR (por Registro de Datos), lo cual puede verse con claridad en los 2 lazos del diagrama de estados que representa al **TAP**.

Con el fin de obtener una secuencia de instrucciones SVF que realicen determinada operación en los estados del **TAP** o para que se sondee cierta cadena de bits hacia alguno de los registros, es necesario conocer algunas de las características de la arquitectura **Boundary- Scan** del dispositivo al cual se van a aplicar las operaciones **Boundary- Scan.** Estas características se encuentran en el archivo **BSDL** correspondiente al dispositivo. Estas características son la longitud del Registro de Instrucciones y de los registros de Identificación y **Boundary- Scan**

*** Juego de instrucciones** El formato **SVF** describe unas instrucciones que describen todas las posibles operaciones en el Registro de Instrucciones o cualquiera de los registros de Datos.

- **SDR (Shift Data Register)** y **SIR (Shift Instruction Register).**

[15] IEEE WORKING GROUP P1149.1. IEEE Standard Test Access Port and Boundary-Scan Architecture [base de datos en dvd]. Piscataway, New Jersey: The Institute of Electrical and Electronics Engineers Inc, 2001. p. 61. Disponible en IEL 3.0, Versión en DVD, IELDVD040, Base de Datos Biblioteca Central UIS.

Estas instrucciones se utilizan para aplicar una serie de bits a uno de los registros de datos o al de Instrucciones. Sus argumentos son el número de bits que se van a desplazar, el valor de esos bits y opcionalmente el valor con el que será comparado lo que se desplace por **TDO**. Estas 2 instrucciones son instrucciones de sondeo.

Sintaxis:

SDR long **TDI** (TDI) **SMASK** (mascara_TDI) **TDO** (TDO) **MASK** (mascara_TDO);
Donde: long es la longitud en bits del Registro de Datos (decimal)
TDI es el valor a desplazar (hexadecimal)
mascara_TDI es la máscara a aplicar a TDI (hexadecimal)
TDO es el valor a capturar, optativo (hexadecimal)
mascara_TDO es la máscara a aplicar a TDO (hexadecimal)
La sintaxis es igual para la instrucción SIR.

Por ejemplo, si se desea aplicar la instrucción INTEST a un FPGA Spartan IIE (XC2S200E- PQ208), esta sería la instrucción correspondiente en el archivo **SVF**:

SIR 5 **TDI** (07);

La longitud de su Registro de Instrucciones es 5 bits y el código operacional de la instrucción INTEST es 0 0111; ya que los bits a desplazar se deben escribir en formato hexadecimal entonces el valor es 07 (hexadecimal), que corresponde a 0 0111 binario.

Si en el mismo archivo **SVF** se desea capturar el valor almacenado en el Registro de Identificación estas son las líneas necesarias:

SIR 5 **TDI** (09);
SDR 32 **TDI** (00000000) **SMASK** (ffffffff) **TDO** (00a1c093) **MASK** (0fffffff);
La primera línea corresponde a la carga de la instrucción IDCODE en el Registro de Instrucciones; su código binario es 0 1001, o 09 en hexadecimal; la segunda línea corresponde a la captura del código de Identificación; para lo cual se desplazan 32 bits pues el Registro de Identificación tiene 32 bits de longitud; se espera capturar el valor X0A1C093 que es el código de Identificación de el dispositivo, para lo cual se enmascara el valor de TDO que es (00A1C093) con el valor de MASK que es (0FFFFFFF).

- **HDR (Header Data Register), HIR (Header Data Register), TDR (Trailer Data Register)** y **TIR (Trailer Data Register)**

Estas instrucciones de encabezado (o **Header**) y cierre (o **Trailer**) corresponden a los bits adicionales necesarios para realizar operaciones sobre un circuito integrado en una cadena de más de un elemento **IEEE** 1149.1. También son necesarios para la sincronización al enviar una cadena serial de bits a través de **TDI**; estas operaciones deben ser realizadas antes de cada operación de sondeo. En el caso de tener un solo circuito en la cadena, la forma de utilizar las instrucciones **Header** y **Trailer** es con la cadena (0) como argumento:

```
TIR (0);
HIR (0);
TDR (0);
HDR (0);
```

En el caso de tener en una misma tarjeta electrónica varios dispositivos en los cuales se vaya a ejecutar una serie de operaciones **Boundary- Scan,** estos encabezados y cierres permiten aplicar las mismas instrucciones a otro elemento en la cadena cambiando únicamente el valor de los argumentos de las instrucciones HIR, HDR, TIR y TDR. Si se va a aplicar una prueba sólo a uno de los elementos de la cadena, los demás elementos se llevan al estado Bypass y se puede sondear una cadena de bits a un Registro de Datos de uno sólo de los elementos agregando algunos bits como encabezados y cierres.

- **RUNTEST**

  Se utiliza para forzar al TAP al estado señalado durante el número de ciclos de reloj señalados, y luego moverse al estado final **Run- Test/ Idle**. Sus argumentos son el estado en el que se desea que permanezca y el tiempo deseado para que permanezca allí. El estado final puede ser diferente a **Run Test/Idle**.

- **TRST**

  Describe la presencia de la señal del **TAP TRST** o **Test Reset,** ya que esta señal es optativa. Si el terminal **TRST** no existe, esta instrucción es permitida sólo al comienzo del archivo **SVF**, antes de cualquier otra instrucción para indicar que no está implementado este terminal; los valores posibles son ON, OFF, Z, ABSENT (si no está presente):

  ```
  TRST OFF;
  ```

Con las instrucciones anteriores se puede realizar prácticamente cualquier operación en el TAP de **Boundary Scan** (de captura, actualización o desplazamiento), así como cualquier instrucción **Boundary Scan** (SAMPLE, EXTEST, INTEST…) que se encuentre implementada en el circuito por el fabricante.

Las demás instrucciones presentes en la especificación **SVF** son:

- **FREQUENCY:** Especifica la frecuencia máxima que puede tener el reloj **TCK** del TAP. Su argumento es el número máximo de ciclos de **TCK** (es decir, su frecuencia en Hz.).
- **STATE**: Envía al **TAP** a un estado estable específico.
- **ENDDR:** Describe el estado final al que se desea que pase el TAP luego de una operación de sondeo de un Registro de Datos.
- **ENDIR:** Como la instrucción anterior, describe el estado final al que se desea que pase el TAP luego de una operación de sondeo, en este caso del Registro de Instrucciones.
- **PIO:** Tanto la instrucción PIO como la siguiente (PIOMAP) permiten realizar operaciones de pruebas paralelas. PIO o Parallel Input/Output, en español Entrada/Salida Paralela permite detectar o aplicar a un grupo de pines un valor lógico mediante un "vector de prueba", el cual es dado como argumento de la instrucción y en el cual cada letra representa un terminal. PIO debe estar precedida de la instrucción PIOMAP que indica a cuáles terminales se enviará este vector de prueba.

  H.....Aplicar 1 Lógico
  L.....Aplicar 0 Lógico
  Z.....Llevar a alta impedancia
  U.....Detectar 1 Lógico
  D.....Detectar 0 Lógico
  X.....Detectar Desconocido
- **PIOMAP:** Realiza un mapeo previo a la operación de prueba paralela PIO. Este mapeo define en una primera columna la dirección (entrada, salida o bidireccional) y el nombre lógico de cada señal a las cuales se va a aplicar el vector en la instrucción PIO. El argumento de esta instrucción es un conjunto de señales y el orden de estos valores determina el orden en que se aplicará el vector en la instrucción PIO, de esta manera:

```
PIOMAP (IN entrada1
IN entrada2
OUT salida1
OUT salida2
OUT salida3
IN reset);
PIO (HZUXDH);
```

En el ejemplo anterior entrada1, entrada2 y reset son señales de entrada (in) y las otras 3 señales son de salida (out). En este caso, el mapeo es:

entrada1< - H
entrada2< - Z
salida1 < - U
salida2 < - X
salida3< - D
reset< - H

Algunas de las instrucciones anteriores tienen como argumento uno de los estados del **TAP**. Cada uno de los estados tiene una denominación en el formato **SVF,** que se pueden observar en la tabla 2 a continuación:

Tabla 2: Estados del **TAP**

Estados según estándar 1149.1	**Nombre de estados en SVF**
Test-Logic-Reset	RESET
Run-Test/Idle	IDLE
Select-DR-Scan	DRSELECT
Capture-DR	DRCAPTURE
Shift-DR	DRSHIFT
Pause-DR	DRPAUSE
Exit1-DR	DREXIT1
Exit2-DR	DREXIT2
Update-DR	DRUPDATE
Select-IR-Scan	IRSELECT
Capture-IR	IRCAPTURE
Shift-IR	IRSHIFT
Pause-IR	IRPAUSE
Exit1-IR	IREXIT1
Exit2-IR	IREXIT2
Update-IR	IRUPDATE

Fuente: ASSET INTERTECH, INC. RIVAS, Catalina (traducción de). Serial Vector Formal Specification [en línea], Revision E. [s.l.]: Asset InterTech, 1999. p. 7. [Consulta: 2 feb. 2005]. Disponible en Internet: <http://www.asset-intertech.com/support/svf.pdf>.

Una vez que se ha creado un archivo SVF que realice las transiciones de estado deseadas o las operaciones de desplazamiento, actualización o captura en los Registros, la lógica asociada al TAP (**Test Access Port**) a partir de las declaraciones de este archivo **SVF,** envía señales a los terminales **TMS** y **TCK** así como entradas a través de **TDI** para ejecutar dichas operaciones.

Como conclusión puede decirse que con el fin de crear este archivo SVF que realice algún tipo de prueba, se deben desarrollar cadenas de bits a desplazar y se debe tener la siguiente información del dispositivo o de la cadena de dispositivos en la cual se va a ejecutar la prueba:

- Su composición (tipo del dispositivo).
- La arquitectura **Boundary- Scan** de cada dispositivo (Longitud del Registro de Instrucciones, códigos de operación, número de bloques de entradas y salidas, y cómo se comportan éstos).
- En el caso de una cadena de varios dispositivos, cómo se conectan los bloques de entrada y salida unos con otros y cuántos dispositivos hay en la cadena.

Por tanto, es necesario el conocimiento tanto del estándar **Boundary- Scan** como del lenguaje BSDL y del formato SVF. Esta información permite utilizar el estándar de manera práctica para verificar el funcionamiento de un dispositivo o un circuito integrado así como para detectar sus defectos físicos.

4. CARACTERÍSTICAS DE UN ANALIZADOR LÓGICO

Este capítulo referirá las características con las cuales se describe un analizador lógico y hará una definición de ellas.

De igual manera hará una descripción del LABS**,** detallando cuáles son sus características particulares.

4.1. DEFINICIÓN

Un Analizador Lógico es un instrumento de laboratorio usado para inspeccionar varias señales digitales o análogas sobre una referencia de tiempo común[16]. Permite capturar y realizar monitoreo de las señales de respuesta ante la presencia de unas señales de entrada; por ejemplo se utiliza para comprobar la presencia y orientación de las señales en los buses de datos y direcciones de un sistema, en el instante en el que se activa una señal. También se utiliza muy comúnmente en caso de un evento determinado como es una transición o un nivel lógico específico en una de las señales. Un analizador lógico elemental muestra simplemente dos estados lógicos: alto y bajo, según se encuentre la señal por encima o por debajo del umbral especificado; sin embargo, en algunos casos también puede mostrar la señal análoga que captura en sus terminales.

Contrario a un osciloscopio que permite analizar una señal analógica en tiempo y amplitud, el analizador permite observar muchas señales al mismo tiempo de una manera organizada; entre otras cosas, esto permite observar los cambios en una señal de manera relativa a otra.

Si un dispositivo tiene la arquitectura del estándar **Boundary- Scan** implementada, no es necesario tener un instrumento físico que mida estos valores, pues permite crear un instrumento "virtual", en el cual las celdas de **Boundary- Scan** pueden funcionar como "sondas", conformando un registro que almacena estos estados lógicos al capturar en paralelo y desplazar en serie el valor en todos los terminales del circuito integrado, que en el caso del **FPGA** son sus salidas y entradas después de ser

[16] SILVA BIJIT, Leopoldo y BACIGALUPO, Virgilio. Aplicación de analizadores lógicos en experiencias de laboratorio [en línea]. Valparaíso: 2003. p. 1. [Consulta: 5 feb. 2005]. Universidad Técnica Federico Santa Maria. Departamento de Electrónica. Disponible en Internet: <http://www.google.com/u/eloUTFSM?q=cache:JE5Od-8xbxAJ:www.elo.utfsm.cl/~lsb/elo311/labs/tut-analogic.pdf+Bacigalupo&hl=es&ie=UTF-8>.

configurado. Es esta posibilidad que brinda el estándar la que fue utilizada para desarrollar un software que funcione como Analizador Lógico para el **FPGA** XC2S200E – PQ208 de Xilinx®.

La ejecución de una prueba mediante una comunicación a través del puerto paralelo de un computador personal basada en el estándar **Boundary- Scan** potencialmente permite realizar la adquisición de los estados lógicos de la tarjeta electrónica analizada, haciendo uso de algunas de las instrucciones del estándar. Adquisición es el proceso en el cual las señales de entrada son muestreadas y reunidas en un registro escrito visualizándose como una lista de valores o en forma de onda[17]. Las señales adquiridas son digitales (unos y ceros) y deben ser almacenadas o registradas junto con el momento en el cual fueron adquiridas para poder visualizarse. Al utilizar **Boundary-Scan** para adquirir las señales, es transparente la manera en la cual se detecta un uno o un cero, ya que la información se recibe a manera de una cadena de bits a través de uno de los terminales del **TAP**, **TDO,** cuyas características fueron discutidas en el capítulo anterior.

4.2. CARACTERÍSTICAS

4.2.1. Tiempo de muestreo

Los parámetros más importantes de una medición son el tiempo de muestreo y la condición de disparo. El tiempo de muestreo es el que transcurre entre dos lecturas consecutivas de los canales de entrada. Es importante que este tiempo se ajuste a la variación que va a sufrir la señal. Es decir, si la señal puede tener pulsos de 1 milisegundo de duración, el tiempo de muestreo no deberá ser mayor que ese valor, pues se podría ocultar ese pulso. Ya que **Boundary- Scan** permite acceder al estado de todos los terminales de manera serie, este tiempo aumentará a medida que haya un mayor número de celdas a sondear, o un mayor número de terminales, pero no se verá afectado si se desea analizar una o cincuenta señales.

La frecuencia con la que se toman las muestras, que es el inverso del tiempo de muestreo, debe ser al menos cinco veces mayor que la de la señal más rápida a muestrear. Tener una frecuencia máxima de 33 MHz para **TCK** que es el reloj de **Boundary- Scan** y con el cual las operaciones en el TAP están sincronizadas limita la resolución, así que no sólo la limita la cantidad de celdas que

[17] TEKTRONIX INC. User Manual: TLA 700 Series Logic Analyzer 070-9775-04 [cd-rom]. U.S.A.: Tektronix Inc, 1998. p. 3-47.

deben ser sondeadas. Este tiempo de muestreo es fijo y en el caso de LABS es un valor bastante elevado, igual a 20,8 microsegundos.

4.2.2. Profundidad de memoria de adquisición

Es el número de muestras que puede almacenar. Las muestras serán almacenadas en un archivo de texto luego de ser capturadas por lo cual el número de muestras se limita por el máximo tamaño de este archivo.

4.2.3. Modo de la adquisición de datos

Un analizador lógico puede funcionar como analizador temporal o analizador de estados. El analizador de estados es el que permite saber los cambios en las señales pero no es sincrónico a ninguna señal de reloj. Los datos son capturados a una velocidad fija limitada por el analizador lógico mientras que un analizador temporal adquiere los datos de manera sincrónica a la señal de reloj del sistema es decir que su resolución es variable.

En este caso, LABS funcionará en modo de analizador de estados.

4.2.4. Visualización

El valor de las señales digitales muestreadas se visualiza en un formato numérico o gráfico. En el formato numérico hay una lista de muestras numeradas y de números (unos y ceros) representando el valor lógico capturado de cada señal para cada muestra, mientras que en un formato gráfico se ve el "diagrama de tiempos", visualizándose las señales de interés de manera paralela y con la misma referencia de tiempo en forma de onda. Para construir este diagrama es útil tener con anterioridad las muestras en formato numérico.

4.2.5. Condición de disparo

El disparo (o **trigger** en inglés) es la condición que indica en qué momento se adquieren los datos. Puede ser una señal o una combinación de señales de entrada o señales externas, que hace que el analizador lógico comience a almacenar las muestras.

La condición de disparo es la señal que inicia la captura y almacenamiento de los datos, hasta que se realice la captura de todas las muestras que se deseen tomar. Esta condición puede ser una transición, un nivel lógico en una o varias señales o una secuencia de eventos. En el caso de LABS, el usuario es quien dispara la captura de los datos y quien determina el número de muestras que necesita.

4.2.6. Ancho de canal

En un analizador lógico físico, hay un número fijo de canales o puntas de prueba y una tierra como referencia. El ancho de canal es el número de canales.

Como en el caso de la implementación con **Boundary- Scan** no hay sondas físicas, no hay un ancho de canal fijo, sino que éste depende del número de señales de interés; a mayor número de señales, el ancho de canal es mayor.

La ausencia de sondas físicas evita los problemas que se pueden presentar debido a éstas, como aumento de la temperatura, dificultad para el acceso físico, o posibles cortos por la presencia de cableado adicional; la tierra es configurada en el proceso ya que toda la comunicación se realiza por el puerto paralelo y es transparente para el usuario de LABS.

El bus de comunicación utilizado es el puerto paralelo del cual se utilizan cuatro terminales correspondientes a las cuatro señales del **TAP** además de las señales de tierra y alimentación.

4.2.7. Tipo de señales muestreadas

El tipo de señal muestreada corresponde a la forma de onda. Un analizador lógico puede mostrar el estado lógico de las muestras capturadas y en algunos casos también su nivel de voltaje (análogo), como en el caso del Analizador Lógico referencia TLA 700 de Tektronix, que puede presentar una forma de onda con la amplitud de la señal capturada. Permitir visualizar la amplitud de la señal es útil en aplicaciones de sistemas Digitales/ Análogos.

En este caso LABS almacenará y mostrará únicamente el estado lógico (uno o cero) que es la forma de onda básica.

5. IMPLEMENTACIÓN DEL ANALIZADOR LÓGICO- LABS

En este capítulo se estudia la manera en que el Analizador Lógico desarrollado- LABS, utilizando el estándar 1149.1 de **IEEE (Boundary- Scan),** captura y analiza los datos desde la tarjeta de desarrollo Digilab 2E de Digilent® y en la cual el dispositivo lógico programable a analizar será un **FPGA** de la familia Spartan 2E de Xilinx®, XC2S200E, paquete PQ208, el cual tiene la arquitectura **Boundary- Scan** implementada. La captura se realiza a través del puerto paralelo de un computador personal. Las características del **FPGA** y de la tarjeta de desarrollo se presentan en los anexos A y B.

El algoritmo desarrollado genera de manera automática un archivo en formato **XSVF** (siglas en inglés de Formato de Vector Serie de Xilinx) el cual se detallará más adelante y que contiene las instrucciones necesarias según el diseño que está configurado en el **FPGA** y el número de muestras deseadas para realizar la captura de los estados lógicos, y posteriormente aplica también de manera automática este archivo ejecutando de esta manera una prueba cada vez que se realiza una captura de datos. Finalmente genera un archivo con las muestras tomadas para que la interfaz gráfica pueda permitir su visualización.

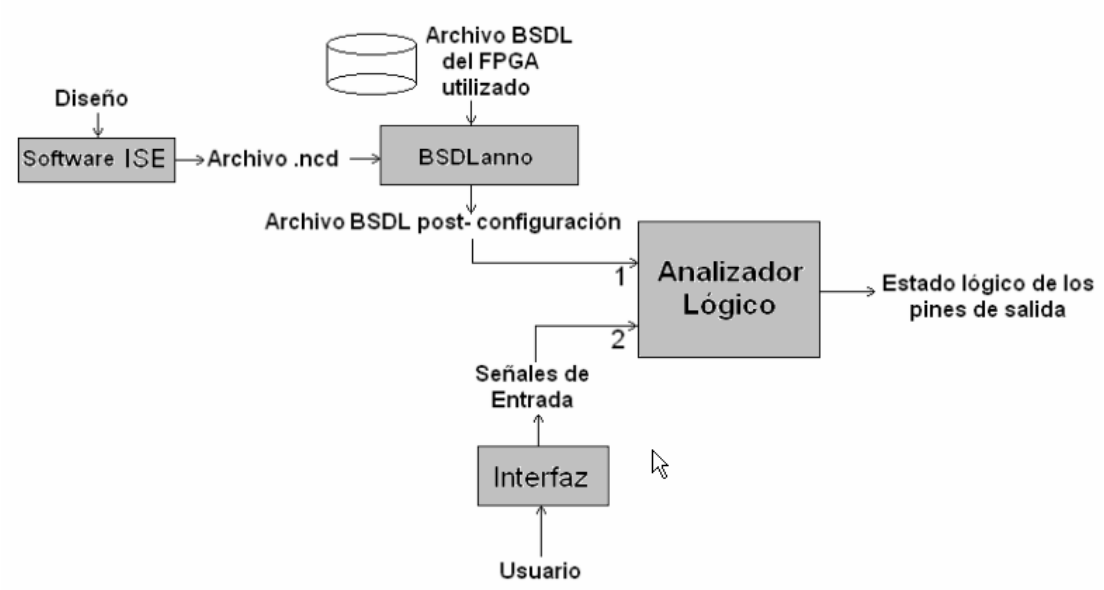
Se detallará cada uno de los pasos que ejecuta el software desarrollado para obtener las muestras de datos capturados.

5.1. ENTRADAS DEL SISTEMA

El primer paso que realiza el software es leer los archivos de entrada en los cuales encuentra información acerca de la arquitectura **Boundary- Scan** del dispositivo, en este caso el **FPGA** XC2S200E de Xilinx®, e información acerca de los datos que pide el usuario. Es necesario entonces observar de qué datos de entrada se dispone inicialmente y que son necesarios para el algoritmo.

En la figura 18 se muestra un diagrama de bloques que representa las entradas y salidas del analizador lógico así como los procesos externos necesarios para obtenerlas.

Figura 18. Diagrama de Bloques- Entradas del Analizador Lógico.



Fuente: Investigación del autor.

Los datos de entrada que se aprecian en la figura son descritos con detalle en las siguientes secciones de este capítulo.

5.1.1. Arquitectura Boundary- Scan del dispositivo después de la configuración

La información acerca de la arquitectura JTAG del dispositivo después de la configuración corresponde al número 1 en la figura 18; el archivo **BSDL** (formato que fue descrito en el capítulo anterior) contiene información acerca de cuáles son los pines del dispositivo y las celdas de su registro **Boundary- Scan** y su orientación. Esta información es parte de la descripción de la arquitectura **Boundary- Scan**.

En el circuito integrado los terminales se encuentran numerados del 1 al 208, y las celdas de 0 a 1021. Por ejemplo, si en el circuito el terminal número 80 (P_80) fue configurado como señal de salida, y se desea capturar dicha salida, entonces debe mostrarse a qué número de celda en el registro **Boundary- Scan** corresponde la salida del terminal 80. Entonces el estado lógico de esta celda debe ser capturado; igualmente sucede con los terminales configurados como entradas del sistema; con esta información se puede crear una cadena de bits a desplazar a través de **TDI** y aplicar en dichas entradas.

La limitación es que se cuenta con un archivo **BSDL** del **FPGA** utilizado que describe esta arquitectura antes de la configuración del dispositivo. En ese archivo todos los terminales aparecen como bidireccionales por defecto en el dispositivo, antes de ser configurado.

Un archivo **BSDL** post- configuración debe tener información acerca de cuáles terminales son entradas y cuáles salidas, indicando a qué bit en el registro **Boundary- Scan** corresponden.

Este archivo describe la arquitectura **Boundary- Scan** después de la configuración y no es suministrado por el fabricante pero puede ser creado mediante la ejecución de una aplicación llamada *BSDLAnno.exe*, incluido en el sistema de desarrollo ISE 5 de Xilinx® con el cual se verifica, sintetiza y configuran los dispositivos lógicos programables del mismo fabricante y que es utilizado para la configuración de la tarjeta de desarrollo.

BSDLanno tiene como argumento el archivo de descripción del circuito de extensión .ncd (siglas de la expresión en inglés **Native Circuit Description**) que es generado automáticamente por el sistema de desarrollo ISE de Xilinx® al realizar el proceso **Map & Route** (en español mapeo y enrutamiento)[*] para describir el diseño (ver figura 18). Además de el archivo .ncd, esta aplicación identifica el archivo **BSDL** que describe el **FPGA** utilizado para el diseño y que se encuentra en el directorio en el cual fue instalado el sistema de desarrollo; para cada una de las referencias fabricadas existe un archivo **BSDL** y estos son agrupados en carpetas según la familia a la cual pertenecen; en el caso del FPGA XC2S200E- PQ208, la carpeta es /Xilinx/spartan2e/data en el directorio raiz en el que se instalo ISE; para cada dispositivo existe en la carpeta /Xilinx un archivo **BSDL** particular.

La salida de esta aplicación es un archivo con la extensión .bsd en lenguaje **BSDL** en el cual se modifica –entre otros- el atributo **BOUNDARY_REGISTER** el cual contiene la descripción del registro **Boundary- Scan**.

Como se describió en el capítulo acerca de **Boundary- Scan**, cada línea en este atributo tiene una información acerca de la celda: su número de celda (entre 0 y 1021 el caso del Spartan 200E) y otros campos: tipo de celda, nombre de puerto, función de la celda, estado seguro, número de la celda de control, valor de control de desactivación y valor al desactivar.

[*] El sistema de desarrollo ISE 5 de Xilinx® ejecuta una serie de procesos para configurar un dispositivo según un diseño determinado, entre ellos, el proceso **Map & Route.** Más detalles en los manuales de Xilinx®, *Xilinx® 5 **Software Manuals***, disponibles en línea http://support.xilinx.com/support/sw_manuals/xilinx5/.

Como muestran algunos ejemplos, esta es la manera en la cual aparecen las líneas del campo **BOUNDARY_REGISTER** del archivo **BSDL** después ejecutar *BSDLanno*:

- Celdas no utilizadas:

```
" 110 (BC_1, *, internal, X)," & -- PAD48.I
```

- Celdas de salida:

```
" 504 (BC_1, IO_P49, output3, X, 503, 1, Z)," & -- PAD253
```

- Celdas de entrada:

```
" 451 (BC_1, IO_P42, input, X)," & -- PAD271
```

En el caso de la celda de entrada, la línea mostrada indica que la celda 451 es de tipo BC_1 (que es la celda básica); esta celda corresponde al terminal de entrada/salida número 42 (IO_P42), fue configurada como entrada (orientación **input)** y el estado seguro es X (lo cual en lenguaje **BSDL** significa que es estado desconocido o indeterminado).

A cada terminal bidireccional del FPGA corresponden 3 celdas **Boundary- Scan** por lo que existen tres líneas en el campo **BOUNDARY_REGISTER** para cada terminal: una línea corresponde a la celda de entrada, otra a la celda de salida y una tercera a la celda de control; hay que tener en cuenta que casi todos los terminales son definidos como bidireccionales antes de la configuración del dispositivo y pueden ser utilizados como entrada o salida, o como terminales para una señal bidireccional.

Si el terminal después de ser configurado corresponde a una señal bidireccional, dos de las tres líneas son modificadas y el terminal aparece tanto de entrada como de salida.

Sin embargo, algunos terminales son la excepción pues no son de uso genérico en el diseño: son los siete terminales de uso específico que aunque no son terminales de entrada del circuito pues no pueden ser utilizadas para enrutar señales del diseño, en el archivo **BSDL** aparecen de tipo **input** y se definen de esa manera para no limitar la funcionalidad de **Boundary- Scan**:

- M0, M1 y M2: son tres terminales que definen el modo de configuración (ya que la configuración del **FPGA** puede realizarse Mediante **Boundary- Scan** o a través de una memoria ROM externa). No son utilizados como entradas ni salidas ya que son terminales dedicados.

- GCK0, GCK1, GCK2 y GCK3: los Relojes Globales (en inglés **Global ClocKs)** son terminales a los que se puede conectar un búfer para implementar una señal de reloj; pero además de utilizarse de esta manera pueden también ser conectados como señales diferenciales junto con uno de sus terminales adyacentes o simplemente como señales de entrada. En el caso específico de la tarjeta DIGILAB 2E, GCK1 que corresponde al terminal P77 se encuentra también conectado a un pulsador (externo al **FPGA**).

A cada uno de estos relojes globales corresponde únicamente una línea en el archivo BSDL pues a los terminales que les corresponden se conecta sólo una celda que es de entrada.

Las celdas que corresponden a las señales M0, M1 Y M2 y a los relojes globales (GCLK) aparecen de tipo entrada o **input** así no sean utilizadas en el circuito, como se muestra en los siguientes ejemplos:

```
" 506 (BC_1, M1_P50, input, X)," & --  Esta celda corresponde al terminal M1
" 635 (BC_1, GCK1_P77, input, X)," &  --  Esta celda corresponde al terminal GCK1
```

Debe tenerse en cuenta que estas celdas pueden ser utilizadas en el proyecto o diseño que se configura, pero así no se utilicen aparecen de tipo **input** en el archivo .bsd, lo cual presentó un pequeño inconveniente a la hora de diferenciar las señales que están presentes en el diseño de las que no.

La sintaxis del proceso *BSDLanno*[18] para crear un archivo BSDL post- configuración es:

bsdlanno <nombre_de_archivo.ncd> <archivo_de_salida.bsd>

El argumento de entrada nombre_de_archivo.ncd corresponde al archivo generado por el sistema de desarrollo ISE 5 al configurar el **FPGA** y que contiene la descripción física del diseño programado en el circuito integrado. El nombre de este archivo se obtiene a través de la interfaz, y es el usuario quien indica cuál es el archivo .ncd del diseño que programó. De esta manera, archivo_de_salida.bsd es el nombre que se desea dar al archivo **BSDL** que describe la arquitectura **Boundary- Scan** después de la configuración, el cual es el archivo de entrada para el analizador lógico y de donde obtendrá sus datos.

[18] XILINX®, Application Note 476: Using BSDL Files for Spartan-3 FPGAs [en línea]. V. 1.0. [s.l.]: Xilinx, 2003. p. 5. [Consulta 26 May. 2004]. Disponible en Internet: <http://direct.xilinx.com/bvdocs/appnotes/xapp476.pdf>.

5.1.2. Datos de la Interfaz

La segunda parte de la información (ver figura 18) la provee el usuario del analizador mediante la interfaz, indicando:

- Número de muestras que desea tomar,
- Operación a realizar (que puede ser realizar captura de estados lógicos, mostrar un conjunto de muestras tomadas anteriormente o aplicar señales de entrada como estímulo para conocer las salidas) y, si es el caso qué estímulos desea aplicar.
- Archivos que describen su diseño (.ncd y .lpc). El usuario sabe que su diseño tiene unas señales de entrada, salida o bidireccionales, que fueron nombradas en la etapa de diseño y que va a sondear; el resto de información es desconocida para el. Es función del software tomar estos archivos y a partir de allí generar una prueba que capture los valores lógicos de esas señales. Acerca del archivo .lpc se hablará mas adelante.

Conociendo tanto la arquitectura **Boundary- Scan** como cuáles son las señales que se desean conocer o modificar, puede crearse una cadena de bits para enviar a través de **TDI** y así mismo cuáles son los bits que son de interés de la cadena de bits capturada a en **TDO**. Por ejemplo, si se quiere aplicar un "0" lógico a la señal de entrada "X" que fue mapeada en el terminal 42, entonces el bit número 451 de la cadena de bits que se enviará por **TDI** será un "0" ya que la celda 451 corresponde al terminal 42 para el **FPGA** XC2S200E.

5.2. DESCRIPCIÓN DE LOS MÓDULOS DEL LABS

El algoritmo desarrollado genera y ejecuta automáticamente una prueba que captura los datos del **FPGA** a través del **TAP** utilizando el estándar **Boundary- Scan**. Esta prueba consiste en la interpretación y ejecución de una serie de instrucciones en lenguaje **XSVF** (por Formato de Vector Serie de Xilinx), lenguaje que definió el fabricante Xilinx en una Nota de Aplicación de la cual se presenta un resumen en el anexo C; estas instrucciones se encuentran contenidas en un archivo binario generado por el algoritmo para la captura de un número determinado de muestras, y según la información de entrada que le es suministrada.

El proceso que realiza el analizador lógico para hacer esta captura puede dividirse en varios módulos cada uno de los cuales ejecuta un sub-proceso; estos son ejecutados de manera consecutiva para realizar la operación deseada. Un esquema de estos módulos puede observarse en la figura 19.

Figura 19. Pasos para el análisis y ejecución de la prueba

Fuente: Investigación del autor.

Los pasos anteriores a la ejecución misma de la prueba son necesarios ya que para poder crear de manera automática el archivo **XSVF** que se ejecuta, se deben crear varias cadenas de bits que son requeridas en las instrucciones y que corresponden a los bits que se van a desplazar a través de los registros y a las máscaras que determinan cuáles de estos bits son utilizados realmente y cuáles no importan para la ejecución de la prueba e interpretación de los resultados (enmascarar los bits): estos valores corresponden a TDI y mascara_TDI en la sintaxis de las instrucciones SDR (ver sección 3.4 acerca del formato **SVF**).

Para ejecutar cada uno de estos pasos, se crearon unos algoritmos en lenguaje **C**; el código fuente de cada algoritmo se presenta en el anexo E de este trabajo. Los módulos son ejecutables o aplicaciones que pueden ser invocadas por línea de comando y posteriormente fueron conectados una con otra para obtener la implementación final del LABS. Este método tiene como ventaja la facilidad para desarrollar y posteriormente verificar el funcionamiento de cada módulo independientemente. En el caso de la ejecución de la prueba, se utilizó como base un grupo de archivos (también en lenguaje **C**) ofrecidos de manera gratuita por Xilinx® que contienen código fuente que permite realizar este tipo de comunicación a través del **TAP** de **Boundary- Scan,** ejecutando las instrucciones de un archivo **XSVF**. Este código fuente fue modificado con el fin de poder ser utilizado para el fin que se requirió. A continuación se detallan cada uno de estos módulos.

5.2.1. Lectura de los Archivos de Entrada

Una vez creado el archivo **BSDL** post- configuración mediante el proceso **BSDLAnno**, se deben leer cuáles son las señales de entrada y de salida que se desea analizar, para crear las cadenas de bits que deberán ser enviados a través de **TDI** así como las señales que se van a visualizar de los bits que se capturan en **TDO**.

Para esto, se creo un módulo, que leyendo cada línea del archivo **BSDL**, y de un archivo adicional de extensión .lpc que se discutirá mas adelante, crea como salida un archivo que indica las celdas correspondientes a cada señal lógica presente en el diseño y si ésta es de entrada o de salida. La aplicación se llama *crearoutput.exe*. El código fuente que se encuentra al final de este documento en el anexo E, se compone de dos archivos con las funciones necesarias tanto para la lectura de las entradas.

La sintaxis de la aplicación *crearoutput.exe* es**:**

crearoutput <archbsdl.bsd> <archivo_lpc.lpc> <opción>

Sus 3 argumentos de entrada son: el archivo BSDL que corresponde al FPGA ya configurado con el diseño realizado, el archivo .lpc que corresponde al mismo diseño y una opción que puede ser:

-i:si se desean aplicar señales de entrada
-s:si se va a capturar el estado de los pines en funcionamiento (función analizador lógico).

Para ejecutar la lectura del archivo BSDL se utilizaron dos funciones:

- crearpuertos (FILE *f)

  Argumento: puntero a un archivo, **archbsdl.bsd**

  Inicialmente mediante la función *crearpuertos (FILE *f)*, se identifican en el archivo **BSDL** (**archbsdl.bsd**), las líneas que corresponden a cada celda de **Boundary- Scan** y que se encuentran en la descripción del Registro **Boundary- Scan;** a partir de la información que tiene cada una de estas líneas (como se vio en la sección 3.4.1) y como paso intermedio, se crea un archivo de texto (pads.txt) de salida de 2 columnas donde se relaciona cada señal presente en el diseño con los terminales del **FPGA** donde estas fueron mapeadas: el nombre lógico de cada terminal aparece en la primera columna (columna PAD) y el nombre que el usuario dio a la señal en el diseño que fue configurado en el **FPGA** aparece al frente en la segunda columna (columna SENAL). Un ejemplo de este archivo es:

```
PAD | SENAL
P49, "in2"
P34, "in1"
P69, "igual"
P71, "out2"
P73, "out1"
P74, "enc"
P77, "cs"
P84, "reset"
```

En el ejemplo, la octava línea relaciona la señal “cs” con el terminal P77.

- crearoutput (FILE *f char tipo)

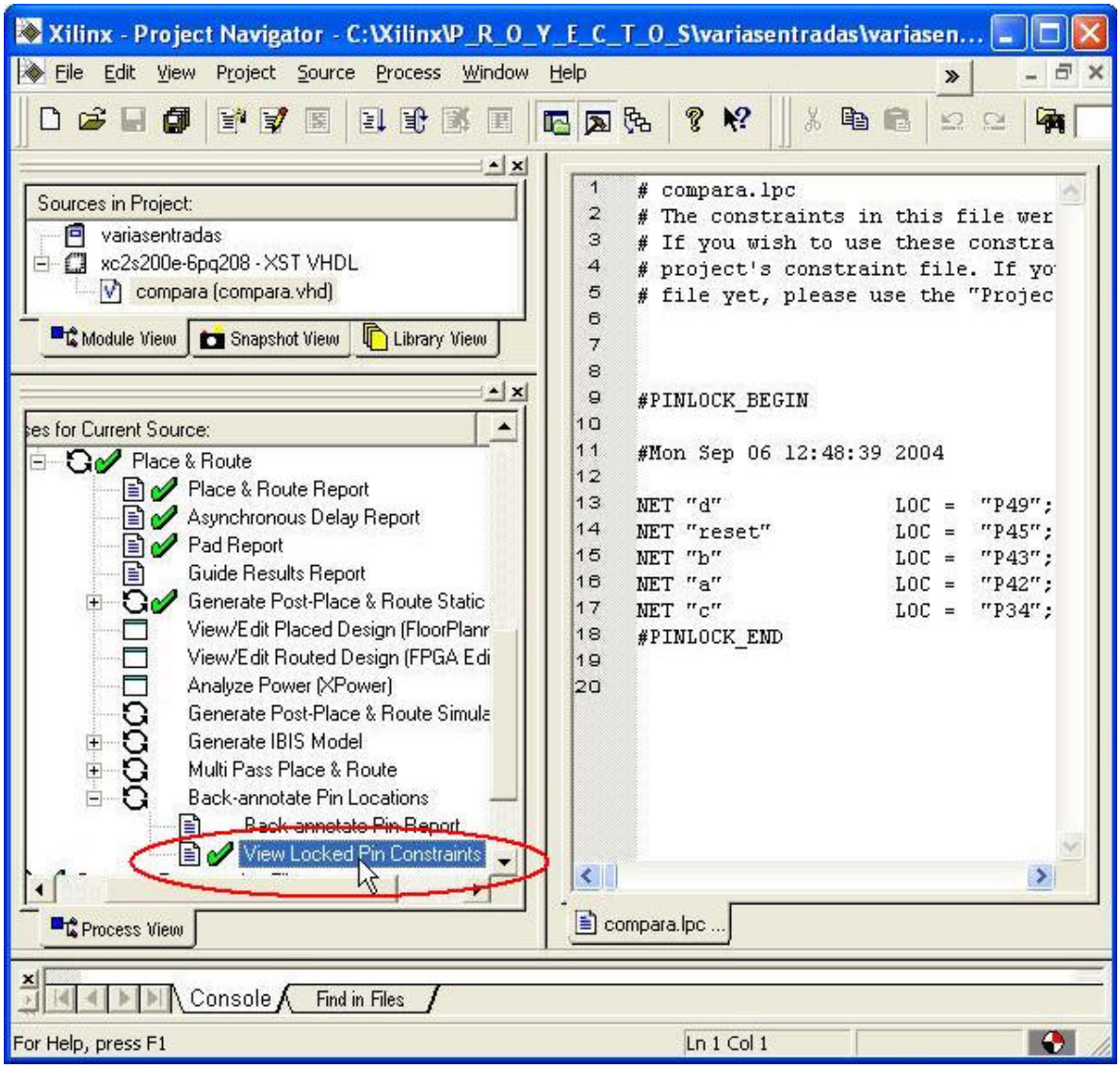
Argumentos: - puntero a un archivo: **archivo_lpc.lpc**
- tipo: una letra que corresponde a ‘i’ si se van a aplicar estímulos o ‘s’ si se va a ejecutar la función de analizador lógico

Para saber a qué señal del diseño implementado corresponde cada celda del registro **Boundary-Scan**, la aplicación ejecuta una segunda función llamada *crearoutput*; esta función utiliza el segundo argumento del ejecutable *crearoutput.exe* (**archivo_lpc.lpc**) y que debe ser un archivo de extensión *.lpc (**Locked Pin Constraints** por su sigla en inglés).

Este archivo contiene información acerca de la ubicación física de cada una de las señales enrutadas en el diseño, e indica en una lista el número de terminal y la señal de entrada o salida que corresponde a cada uno; debe ser generado por el usuario antes o después de la configuración del **FPGA** utilizando el *“Project Navigator”*, que es la pantalla principal del sistema de desarrollo ISE 5 de Xilinx® con el cual se diseña, sintetizan, mapean, enrutan y configuran los dispositivos de Xilinx**.**

En la figura 20 se observa una instancia de la pantalla del *Project Navigator*; la ventana inferior izquierda muestra los procesos correspondientes al elemento que se está diseñando; durante el proceso *Place & Route* uno de los sub- procesos que se encuentra al final de la lista es *“View Locked Pin Constraints”*. Este proceso crea el archivo de extensión .lpc que se puede visualizar igualmente en la figura a la derecha de la pantalla del Project Navigator.

Figura 20. Pantalla del Project Navigator de Xilinx®



Fuente : Investigación del autor.

Esta función también utiliza el archivo de texto creado en el paso anterior pads.txt; realiza una lectura de estos dos archivos y crea un archivo de texto (textout.txt) que contiene una lista de las señales del diseño programado, junto con el terminal correspondiente en el **FPGA** a cada una de ellas; esta información se organiza en el archivo de salida en tres columnas: la primera indica la orientación (es un carácter, **'i'** si es entrada y **'o'** si es salida), la segunda es la celda del Registro **Boundary- Scan** que corresponde a la señal (es un número entre 0 y 1021 ya que en el **FPGA** XC2S200E de Xilinx el Registro **Boundary-**

Scan tiene 1022 celdas), y la tercera es una palabra o serie de caracteres, que es el nombre dado a la señal por el usuario (ejemplo: "reset"). La primera línea del archivo indica la opción escogida por el usuario, s o i ya que esta información será utilizada más adelante. Un ejemplo del texto contenido en este archivo es:

```
s
i 126
i 127
o 414    "c"
i 451    "a"
i 454    "b"
i 469    "reset"
o 504    "d"
i 506
i 507
i 508
i 635
i 636
```

En este archivo de texto mostrado, la cuarta línea indica que la celda del registro **Boundary- Scan** número 414 corresponde a la señal "c" y es una salida. En este archivo, se puede observar la información que extrajo el algoritmo: en este diseño se configuraron cinco señales (tres de entrada y dos de salida), se tiene el nombre de ellas y la celda en el registro **Boundary- Scan** a la cual corresponde cada una.

En este mismo ejemplo se observa que hay siete líneas que no tienen un nombre de señal. Esto se debe a las excepciones de las que se habló anteriormente y que existen debido a que existen terminales dedicados para establecer el modo de configuración y terminales para conectar relojes. Aparecen entonces en este archivo esas celdas que no están presentes en el mapeo del diseño y es por eso que no aparece su nombre de señal, pero sin embargo existen como celdas de entrada. Para la generación del archivo **XSVF** de ejecución de la prueba las celdas que no tienen una señal asociada no son utilizadas.

Acerca del tercer argumento de *crearoutput.exe* (**-i** o **-s**) se hablará más adelante.

5.2.2. Generación del Archivo XSVF

El archivo que se ejecuta y que contiene las instrucciones que se envían a través del **TAP** es de formato **XSVF** (por **Xilinx Serial Vector Format**). Este formato fue creado por Xilinx® y se encuentra información acerca de este en el anexo C de este trabajo, donde se resume la descripción que hace Xilinx® de los formatos **SVF** y **XSVF** para sus dispositivos; como una modificación de

SVF, este formato tiene las mismas funciones y básicamente la misma forma. Casi todas las funciones **SVF** tienen una o varias funciones equivalentes en **XSVF**; la diferencia radica en que **XSVF** tiene formato binario más compacto pues fue creado pensando en sistemas embebidos en los cuales exista una memoria limitada para almacenar estos archivos; cada una de sus instrucciones tiene un byte de longitud y uno o varios argumentos.

El archivo **XSVF** con el cual se realiza el intercambio de información entre el computador y el **FPGA** de la tarjeta electrónica puede ser obtenido fácilmente traduciendo un archivo **SVF** que contenga las instrucciones necesarias para ese intercambio de información, mediante un algoritmo cuyo código fuente puede ser obtenido de manera gratuita del fabricante a través de Internet, por lo que es necesario primero crear el archivo **SVF** en un editor de texto (ya que es formato **ASCII**) para luego traducirlo.

*** Generación del archivo SVF** El archivo **SVF** es creado de manera automática por una segunda aplicación desarrollada, *crearsvf.exe*, la cual a partir del archivo textout.txt generado por el módulo anterior, crea un archivo con las instrucciones **SVF** necesarias para hacer un muestreo o para ejecutar una serie específica de operaciones en las celdas de **Boundary- Scan**.

La sintaxis de la aplicación *crearsvf.exe* es**:**

crearsvf <**opción**> <número_muestras>

Donde <**opción**> es **-i**: Para aplicar entradas a través de TDI
-s: Sólo para capturar las señales tanto de entrada como de salida.
<número_muestras> es el número de muestras a capturar.

Los pasos de este ejecutable que corresponden a esta etapa de generación del archivo de ejecución de la prueba son:

- Se parte del archivo *textout.txt* para continuar con la generación de los vectores de bits que harán parte del archivo **SVF.** Si la opción fue "-i", es decir que se van a aplicar estímulos en las entradas, este ejecutable también lee un archivo que contiene los valores lógicos que se van a aplicar en cada una de las entradas, muestra a muestra. Este archivo llamado *estimul.txt* es generado a través de la interfaz con el usuario pues es este quien selecciona la secuencia de señales que se van a aplicar.

- Inicialmente se generan dos cadenas de caracteres que representan vectores de bits, cada una correspondiente a una de las variables TDI, y SMASK que son necesarias para las instrucciones **SVF.**

Las dos variables son inicializadas en formato **string** como solo ceros (el carácter '0'); cada carácter en la variable representa un bit del registro **Boundary- Scan** por lo que puede ser igual a '1' o '0'; en el caso del **FPGA** XC2S200E este registro tiene 1022 bits de longitud. Los caracteres se numeran de 0 a 1021 de manera que la numeración sea consistente con la del registro: el bit más significativo es el más cercano a **TDI**.

Se comenzó por modificar la variable SMASK que es una máscara que indica cuáles son las celdas que corresponden a entradas; el algoritmo lee el archivo *textout.txt* y de la información allí contenida se puede conocer a cuáles celdas del registro **Boundary- Scan** corresponden señales de entrada (pues están precedidas de un carácter 'i'), de manera que las posiciones correspondientes a estas celdas en la cadena de caracteres se hace igual a '1'. Por ejemplo, para el archivo que se muestra en la página 62 …en 5.2.1 …, en el cual la cuarta línea es la celda 451 y corresponde a una señal de entrada, el carácter número 451 de la variable SMASK debe hacerse igual a '1'.

- Si se desean aplicar señales (es decir, si <opción> = "-i") entonces se modifican los valores de los bits de TDI que lo requieran según los estímulos que haya creado el usuario y que se almacenaron en *estimul.txt.* De esta manera queda generado el vector TDI. Si no se desean aplicar señales, TDI es una cadena de ceros ya que este valor no afecta el funcionamiento del circuito ni los valores que serán capturados.

- Para que estos vectores que fueron generados puedan ser escritos en el archivo **SVF**, fue necesario traducirlos a formato hexadecimal; ya que eran de tipo palabra o **string**, fue necesario crear una función llamada *char2hex* cuyos argumentos son cuatro caracteres que corresponden a cuatro bits de cada una de las variables de 1022 bits; estos 4 caracteres que se convierten en UN carácter hexadecimal. La salida de esta función es este carácter hexadecimal. La función puede ser descrita de esta manera: *char char2hex (char a, char b, char c, char d).*

  La longitud de estos números hexadecimales corresponde a la de las cadenas binarias dividido por cuatro debido a la conversión realizada. Entonces en este caso estas variables TDI y SMASK tendrán 256 caracteres que corresponden al mismo número de cifras hexadecimales. Ya que al multiplicar 256 por cuatro, se obtienen 1024 bits y no 1022 que es la longitud exacta del registro, fue necesario agregar dos caracteres adicionales al inicio de las variables binarias para poder realizar la conversión.

- se genera también un quinto vector de bits llamado TMASK para enmascarar a todas las señales presentes en el circuito, tanto las de entrada como las de salida; TMASK será necesario en la fase de generación de los resultados para saber cuáles son las señales que se visualizarán en el analizador lógico.
- Finalmente se generan y escriben cada una de las líneas en el archivo SVF (en modo texto) llamado *archsvf.txt* y en él se incluyen (en las instrucciones que lo requieran) los valores de SMASK y TDI.

Este archivo **SVF** que va a ejecutar las operaciones necesarias para la captura del estado lógico de los terminales del **FPGA** utiliza para este fin las instrucciones SAMPLE e INTEST principalmente, junto con otras que sirven como encabezados y cierres necesarios para la ejecución de cualquier instrucción **Boundary- Scan**.

En este archivo creado (ver anexo G) se ejecuta inicialmente una instrucción que desactiva el reset. A continuación se define el estado final del TAP y se ejecutan instrucciones de encabezado y cierre para los registros de instrucciones y de datos. Estas instrucciones deben ejecutarse antes de poder ejecutar el sondeo de los datos a capturar.

La función de analizador lógico que captura las señales con el circuito en funcionamiento normal, se hace con la instrucción SAMPLE de **Boundary- Scan**, para la cual se necesitan 2 instrucciones **SVF:**

SIR 5 TDI (01); – carga en el Registro de Instrucciones el código 00001 que corresponde a SAMPLE.

SDR 1022 TDI (000…000) SMASK (XXX…XXX); -- carga en el registro **Boundary- Scan** el valor entre paréntesis después de TDI que puede ser cualquiera ya que este no interfiere en la operación; se desplaza a través del Registro **Boundary- Scan** hacia TDO los niveles lógicos capturados del **FPGA** en operación. Habrá tantas líneas de esta instrucción como muestras se desea tomar.

Si la operación a realizar no es la de analizador lógico sino que se van a aplicar estímulos a través del TAP en las señales de entrada, la instrucción **SVF** utilizada es INTEST:

SIR 5 TDI (07); – carga en el Registro de Instrucciones el código 00111 que corresponde a INTEST.

SDR 1022 TDI (XXX…XXX) SMASK (XXX…XXX); -- carga en el registro **Boundary- Scan** el valor entre paréntesis después de TDI; los valores que serán aplicados a las entradas serán aquellos que indique la máscara SMASK.

Esta instrucción **SDR** se repite en el archivo **SVF** tantas veces como muestras se vayan a capturar**;** A medida que se desplaza el valor de TDI, también se desplazan a través del Registro **Boundary-Scan** hacia TDO los valores almacenados en los flip-flops de salida de las celdas **Boundary- Scan**. Ya que la ejecución de una instrucción INTEST aplica el valor de TDI en las entradas y captura el valor que se encuentra almacenado en las celdas **Boundary-Scan,** para observar la respuesta al primer estímulo hay que observar la muestra número 2, la respuesta del segundo estímulo se captura en la muestra número 3, etcétera.

Finalmente las últimas instrucciones en el archivo SVF son de encabezado y cierre. La razón por la cual el argumento de estas instrucciones es cero, es que solamente hay un elemento a sondear, que es el **FPGA** XC2S200E; si hubiese más de un elemento en la cadena **Boundary- Scan**, esto se vería reflejado en las instrucciones de encabezado y cierre.

*** Generación del archivo XSVF** Una vez se ha generado el archivo **SVF**, se convierte en **XSVF** mediante la aplicación *svf2xsvf.exe*, que es una aplicación que Xilinx® pone a disposición de los usuarios. s*vf2xsvf* .exe es un traductor de un formato al otro.

Sintaxis:

svf2xsvf –i <arch_svf.txt> -o <arch_xsvf.xsvf> -a <arch_ascii.txt> [opciones],

donde <arch_svf.txt> es el archivo **SVF** a traducir y <arch_xsvf.xsvf> es el nombre dado al archivo **XSVF** de salida; <arch_ascii.txt> es un archivo optativo de salida, con las instrucciones XSVF generadas en formato ASCII para que puedan ser leídas.

Las opciones de este algoritmo disponibles para el **FPGA** XC2S200E de la tarjeta Digilab IIE son:

-d= Borrar archivos de salida existentes.
-fpga= Traducción especial para bitstreams de FPGA.
-w=Sobrescribe el archivo XSVF si este existe (similar a -d).
-xprintsrc N = imprime varios niveles de comentarios en el XSVF (N=0-3)

5.2.3. Ejecución de la Prueba

El siguiente paso es ejecutar la prueba; el archivo **XSVF** ya fue generado y contiene todas las instrucciones necesarias para capturar los datos y opcionalmente para aplicar señales de entrada. En este tercer paso se realiza la ejecución del archivo XSVF y la captura de las muestras que se obtienen a través del terminal **TDO** del **TAP.**

Esto se realiza mediante la ejecución de la aplicación *playxsvfsi.exe*, algoritmo que tiene como argumento el archivo **XSVF**; con esta ejecución se aplica el archivo al FPGA que se encuentre conectado al computador a través del puerto paralelo.

Playxsvfsi.exe es una modificación de playxsvf (aplicación distribuida por Xilinx®) creada a partir del código fuente en lenguaje C que Xilinx® distribuye sin restricciones como archivos de soporte de una de sus Notas de Aplicación[19], y que interpreta un archivo **XSVF** dado como argumento (ver código fuente asociado con la aplicación playxsvfsi en anexo E).

Sintaxis:

playxsvfsi [–v <nivel_de_detalle>] <archivo_xsvf.xsvf>

El primer argumento (–v <nivel_de_detalle>) es optativo; nivel_de_detalle es un número entre 0 y 4 que indica qué tan detallada se desea la información que se puede visualizar a medida que se realiza la prueba al ejecutar la aplicación desde la línea de comando; 0 corresponde a ninguna información y 4 a información muy detallada. Su valor por defecto es igual a 0. Con el máximo nivel de detalle (-v 4), a medida que se ejecutan cada una de las instrucciones XSVF, se va imprimiendo en pantalla información acerca de dicha instrucción, el estado en que se encuentra el TAP, así como los valores de TDI, TDO y la máscara de TDO.

Originalmente este código tiene tres archivos de código fuente y tres archivos de encabezado*:

- ports.c : Contiene las rutinas para capturar y enviar señales a través del **TAP**: TDI, TMS, TCK y TDO bit por bit utilizando el puerto paralelo.
- lenvalc: Contiene funciones para el manejo de la estructura de datos lenVal (esta estructura es un objeto creado para almacenar y manipular información de un valor binario; un vector de bits es de tipo lenVal).
- micro.c: Es el código fuente primario; en este archivo se encuentra la función main() junto con otras que ejecutan el archivo **XSVF**.
- ports.h: Incluye declaraciones externas de las funciones de ports.c.

[19] XILINX®. Application Note 058: Xilinx in- system programming using an embebbed mirocontroller [en línea]. V. 3.0. [s.l.]: Xilinx®, 2001. 38 p. [Consulta 7 Dic. 2003]. Archivos asociados disponibles en Internet en: <ftp://ftp.xilinx.com/pub/swhelp/cpld/eisp_pc.zip>.

* El archivo de código fuente tiene extensión .c y el archivo de encabezado o archivo cabecera tiene extensión .h y declara las funciones que se utilizan en el código fuente y almacena información adicional.

- lenval.h: Contiene la definición de las partes de la estructura lenVal y declaraciones de "**procedures**" o procedimientos externos para manipular elementos con dicha estructura.
- micro.h: Contiene los prototipos de las funciones que hay en el código fuente primario; define constantes declara variables externas.

Algunos de estos archivos fueron modificados para lograr implementar mediante ellos el analizador lógico, ya que con el fin de realizar la captura no sólo se debe ejecutar un archivo XSVF sino que también se debe capturar el estado lógico de algunos de los terminales para lo cual se deben capturar y almacenar los bits recibidos a través de TDO.

Además, la adición de un archivo de código fuente y un archivo de encabezado adicionales fue necesaria para poder autorizar el uso del puerto paralelo, ya que al ejecutar *playxsvfsi.exe* en un equipo con sistema operativo Windows XP (Windows NT/2000/XP), el acceso a los puertos para operaciones del usuario se encuentra restringido lo que se refleja en una excepción de acceso (0xc00000096) al ejecutar playsxvfsi.exe, que accede al puerto paralelo. Esto se debe a que el procesador (386 o posterior) funciona en modo protegido en el caso de esos sistemas operativos.

Como no está autorizada una instrucción de tipo inportb() o outportb(), a menos que se tenga un controlador del puerto al que se desea acceder, se implementó como solución la adición de estos archivos, los cuales instalan un controlador y ejecutan una función en el archivo fuente principal, la función openporttalk(); el código fuente para este controlador fue tomado del sitio Web de Beyond Logic[20] el cual presenta artículos y ejemplos de controladores de dispositivos, puerto USB y otras soluciones de software; estos dos archivos son:

- pt_ioctl.c:Contiene funciones para el control de entradas y salidas a través de un puerto (fue definida sólo para el puerto paralelo que es el que se utiliza para recibir y enviar las señales del computador a la tarjeta Digilab 2E)
- porttalk_ioctl.h: Archivo cabecera que contiene definiciones de variables que se utilizan en pt_ioctl.c.

Las modificaciones realizadas a este código fuente fueron:

En ports.c:

[20] Beyond Logic, Sitio Web http://www.beyondlogic.org/porttalk/porttalk.htm

- Se modificó la función *waittime(int microseg)* para que fuera acorde a la velocidad del procesador.

En micro.c:

- Se creó la función *void xsvfFileLenVal(lenVal *plv, *FILE *fi)* que es similar a xsvfPrintLenVal (función que ya existía en micro.c), pero en lugar de mostrar el valor de un objeto LenVal en la pantalla del computador (lo que permite visualizar TDI, TDO capturado, TDO esperado y MASK en pantalla), lo imprime a un archivo dado como argumento. Esto permitirá más adelante crear un archivo con las muestras tomadas en TDO.
- Se agregaron a *xsvfrun ()* las funciones *openporttalk ()* y *closeporttalk ()* definidas en pt_ioctl.c que se definieron en pt_ioctl.c para autorizar el acceso a los puertos durante la ejecución del archivo XSVF y para cerrar el puerto al finalizar esta ejecución.
- Se modificó *xsvfshift ()*, que es la función que desplaza un vector de bits que se aplica a través de TDI, y compara el vector de TDO capturado con el esperado, para que cada vez que se ejecute se agreguen unas líneas a un archivo de texto con el valor de TDO desplazado, es decir, que cada vez que se ejecute *xsvfshift ()* se almacene la muestra de TDO capturada. Esto hará que con la ejecución de las instrucciones SAMPLE o INTEST, se obtenga el valor desplazado por TDO. En la última etapa del proceso se hablará con mayor profundidad de este archivo.
- Se modificó la función *xsvfexecute()*, para que al ejecutar un nuevo archivo **XSVF** se borren las muestras anteriores y se pueda comenzar a escribir en el archivo vacío.

5.2.4. Generación de archivo de Resultados

La aplicación *playxsvfsi.exe* que ejecutó la prueba también se encarga de la primera parte de la generación de resultados; su ejecución genera un archivo con todos los valores capturados a través de **TDO** llamado TDOCapt.txt**.** El archivo de texto está conformado por grupos de tres líneas; cada grupo corresponde a una muestra:

- La primera línea indica si el resultado es válido o no (un 1 en la variable iMisMatch indica que la muestra no es válida). Este valor se tuvo en cuenta únicamente en las pruebas realizadas al algoritmo para su depuración al capturar el valor del registro de Identificación

del **FPGA** analizado, y que es un valor conocido. Cuando la aplicación es ejecutada para ejecutar la prueba en el analizador lógico este valor no se tiene en cuenta.

- La segunda línea es la máscara de **TDO** en el caso de que TDO sea un valor conocido. En el caso del analizador lógico este valor no se tiene en cuenta ya que TDO (es decir, el valor a capturar) no se conoce; la máscara que se utiliza en este caso es TMASK, generada al ejecutar la aplicación *crearsvf.exe* y que enmascara tanto las señales de entrada como las de salida.
- La tercera línea es el valor en hexadecimal de la muestra capturada por **TDO** (1022 bits desplazados a través del registro **Boundary- Scan** que corresponden a las 1022 celdas del Registro).

A continuación se deben seleccionar únicamente las muestras de las señales que serán visualizadas pues son de interés para cada diseño o proyecto particular; de las 1022 celdas del registro **Boundary- Scan** sólo algunas corresponden a los terminales utilizados. Para esto se deben distinguir de todos los bits cuáles son los que se encuentran presentes, lo cual es indicado en la máscara TMASK, que indica con un '1' las posiciones de los bits que se deben mostrar y con un '0' las que no se necesitan.

Para tal fin, hay que leer el archivo de muestras generado anteriormente y aplicar TMASK al valor en hexadecimal de TDO que fue capturado y almacenado en TDOCapt.txt.

La aplicación que efectúa la lectura y captura de las muestras de las señales es *muestreo.exe*.

Sintaxis:

muestreo

A partir del archivo de muestras -TDOCapt.txt- y del archivo que relaciona el nombre de cada una de las señales con el número de celda **Boundary- Scan** que le corresponde -textout.txt-, la ejecución de la aplicación desarrollada *muestreo.exe* imprime en un archivo de salida una lista de las señales presentes junto con las muestras tomadas correspondientes a cada señal.

A cada muestra corresponde una línea del archivo generado llamado muestras.txt, y cada línea tiene tantos caracteres como señales tiene el diseño que fue configurado en el **FPGA**; un carácter '0' es un estado lógico bajo y un '1' corresponde a un estado lógico alto. Se presenta un ejemplo del contenido de este archivo:

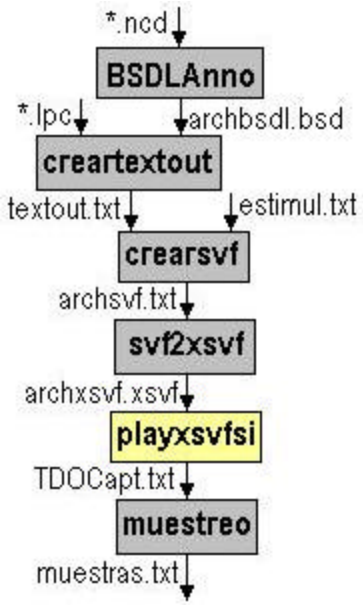
```
“a”
“b”
“reset”
“c”
“d”

00110
00101
10010
01001
10010
10010
```

En este caso, el archivo de muestras corresponde a un sistema con cinco señales de las cuales aparecen sus nombres al comienzo del archivo, y se tomaron seis muestras que se ven a continuación. De los 1022 bits desplazados a través de TDO que corresponden a las 1022 celdas, se muestran únicamente las señales pertinentes a este diseño El carácter más a la derecha en cada línea corresponde a la primera señal y el carácter más a la izquierda corresponde a la última señal de la lista.

5.2.5. Resumen de los procesos del análisis.

En la figura 21 puede observarse un resumen de los anteriores procesos, desde la lectura de los archivos de entrada hasta la generación del archivo de muestras que será interpretado por la interfaz gráfica.

Figura 21. Resumen de los procesos del análisis lógico



Fuente: Investigación del autor.

También se observan en la figura los archivos que intervienen en el proceso. Las aplicaciones *creartextout.exe*, crearsvf.exe, *svf2xsvf.exe, playxsvfsi.exe* y *muestreo.exe* se encargan de realizar los cuatro pasos del análisis lógico de los terminales del dispositivo. El proceso que ejecuta la prueba para el análisis así como la captura de los datos (que son las dos funciones críticas del analizador lógico) es *playxsvfsi.exe*.

En el anexo E se encuentra el código fuente correspondiente a los ejecutables *creartextout.exe, crearsvf.exe, playxsvfsi.exe* y *muestreo.exe*

6. INTERFAZ GRÁFICA DEL LABS

La interfaz gráfica se desarrolló en LabVIEW, un programa de National Instruments para el desarrollo gráfico de aplicaciones para control, mediciones y pruebas; este programa fue seleccionado ya que en sus librerías tiene entre otros un instrumento para visualización de datos digitales a partir de arreglos de números en formato binario, fáciles de generar a partir del archivo muestras.txt, además de que su entorno gráfico lo hace fácil de aprender y las aplicaciones son generadas rápidamente.

Se generó un Instrumento Virtual en LabVIEW que realiza las siguientes funciones:

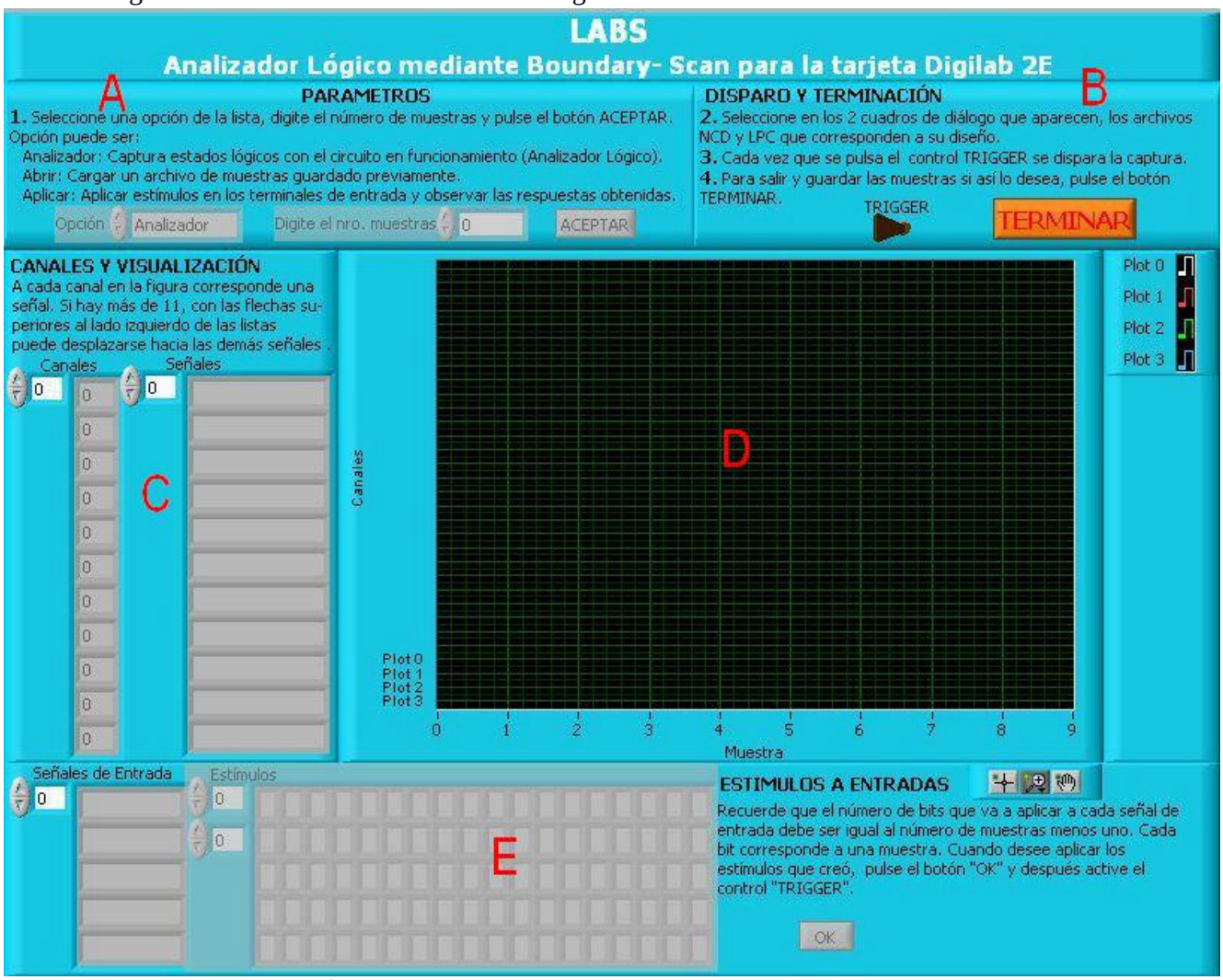
- Interacción del usuario con el Analizador, de manera que éste seleccione los parámetros de la captura a realizar.
- Visualización de resultados como formas de onda. También permite adicionalmente la visualización de capturas realizadas previamente, por lo cual debe ofrecer la alternativa de almacenar las muestras que son tomadas.
- Ejecución de las aplicaciones creadas. El Instrumento Virtual invoca la ejecución de las aplicaciones desarrolladas en lenguaje C de manera secuencial en el orden apropiado para generar los archivos que se utilizan en el proceso y ejecutar la prueba.
- Creación del archivo de texto estimul.txt que será utilizado en uno de los sub-procesos (ver figura 21) a partir de los estímulos a las señales de entrada que cree el usuario.

6.1. ELEMENTOS DEL PANEL FRONTAL DE LA INTERFAZ GRÁFICA

El panel frontal de la interfaz tiene 5 elementos principales, cada uno de ellos en un recuadro. Estos elementos pueden observarse en la figura 22 y son:

A. Selección de los parámetros iniciales.
B. Disparo y Terminación.
C. Canales.
D. Gráfico de visualización de muestras.
E. Señales de entrada y Estímulos.

Figura 22. Panel Frontal del Analizador Lógico



Fuente: Investigación del autor

A continuación se describen detalladamente cada uno de los elementos:

A. SELECCIÓN DE LOS PARÁMETROS. En este primer recuadro aparecen las instrucciones iniciales y el usuario selecciona qué función desea ejecutar.

El primer control que el usuario modifica es “Opción”, que indica la función a ejecutar y puede ser:

- Analizador: Ejecuta el analizador lógico, es decir que realiza la captura del número de muestras deseado del estado de los terminales del FPGA en funcionamiento normal.
- Aplicar: Esta opción es adicional a la de Analizador lógico; permite aplicar un número de estímulos a los terminales de entrada igual al número de muestras.

- Abrir: Con esta opción el usuario puede visualizar un grupo de muestras anteriormente almacenadas; con la opción “Abrir” el usuario debe seleccionar el archivo que corresponde a las muestras que desea ver.

El segundo control que se selecciona es “nro. muestras”. Es un número entero mayor o igual a dos.

En este recuadro finalmente se encuentra el botón “Aceptar”. Este botón envía una señal lógica; al ser pulsado, los valores de “Opción” y “nro. muestras” pasan a ser interpretados por el Algoritmo.

B. DISPARO Y TERMINACIÓN. En la parte superior derecha de la pantalla aparece la continuación de las instrucciones a seguir; también dos botones: “TRIGGER” y “TERMINAR”. Cada vez que se pulsa en control TRIGGER se dispara la ejecución de la prueba, captura y visualización de resultados; puede pulsarse cuantas veces se desee, disparando varias veces la captura.

C. CANALES. A la izquierda, una vez se ha disparado la captura aparece la lista de los canales en el gráfico digital junto con las señales que corresponden a cada uno de los canales.

D. VISUALIZACIÓN DE MUESTRAS. Al lado de la lista de canales se encuentra la zona de visualización; este gráfico tiene en la parte inferior 3 botones; el del medio permite hacer acercamientos con el fin de mirar una zona específica del gráfico para ver con más detalle un grupo de muestras. También permite alejarse. El botón de la derecha permite desplazar los gráficos a derecha e izquierda.

E. SEÑALES DE ENTRADA Y ESTÍMULOS. Si la opción seleccionada fue “Aplicar” entonces en la parte inferior aparece la lista de señales de entrada; el usuario modifica los valores de estímulos de todas ellas para generar una secuencia de bits que se aplicará a cada uno de estos terminales de entrada. Cuando ya seleccionó la secuencia de bits para cada entrada, entonces puede pulsar el botón “OK” que se encuentra a la derecha, abajo, para así estar listo para hacer el disparo.

En cada cuadro aparecen las instrucciones a seguir; además a medida que se van ejecutando los pasos para visualizar las muestras, aparecen ventanas de diálogo que indican al usuario qué hacer.

6.2. ALGORITMO DE LA UTILIZACIÓN DEL LABS

Estos son los pasos a seguir, para capturar y visualizar las muestras:

1. Selección de una opción que puede ser Analizador, Abrir o Aplicar pulsando las flechas abajo a arriba en el control respectivo.

2. Selección del número de muestras; debe ser mayor o igual a dos.

3. Una vez seleccionados los dos parámetros, se pulsa el botón “ACEPTAR”

Los pasos siguientes son diferentes, según la opción seleccionada. Si la opción es **“Abrir”**:

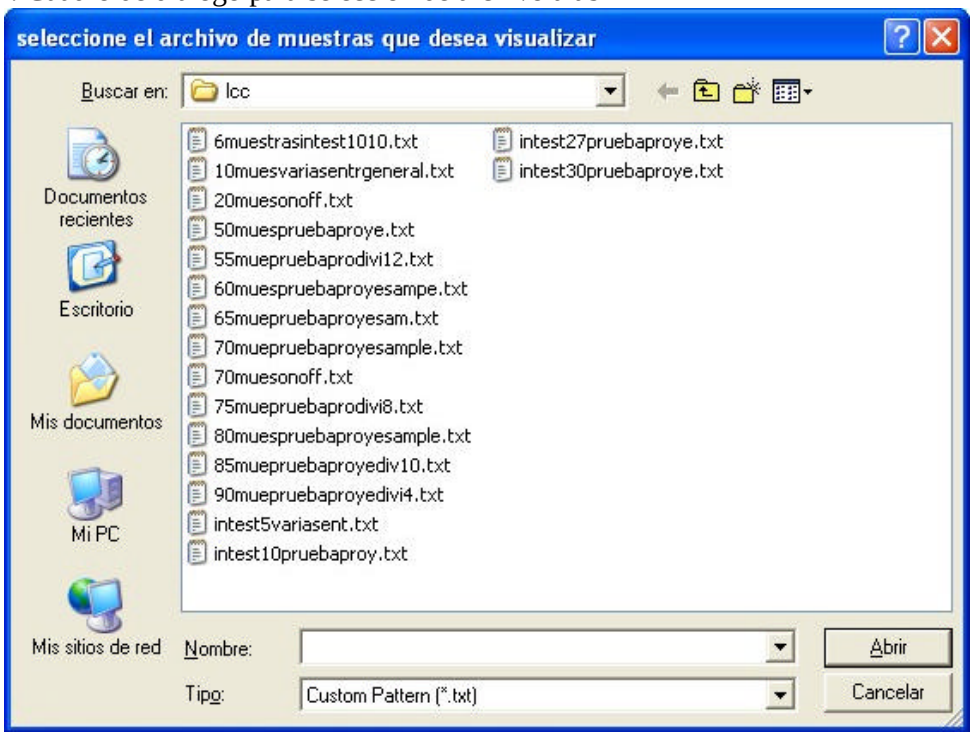
4. Aparece un cuadro de diálogo como el que se observa en la figura 23 que indica al usuario que pulse el control “TRIGGER” para continuar con la selección del archivo:

Figura 23. Cuadro de diálogo de la opción “Abrir”

Fuente: Investigación del autor.

5. Cuando el usuario pulsa el control TRIGGER aparece un cuadro de diálogo en el cual se selecciona el archivo de muestras a visualizar, como el de la figura 24:

Figura 24. Cuadro de diálogo para selección de archivo a abrir



Fuente: Investigación del Autor.

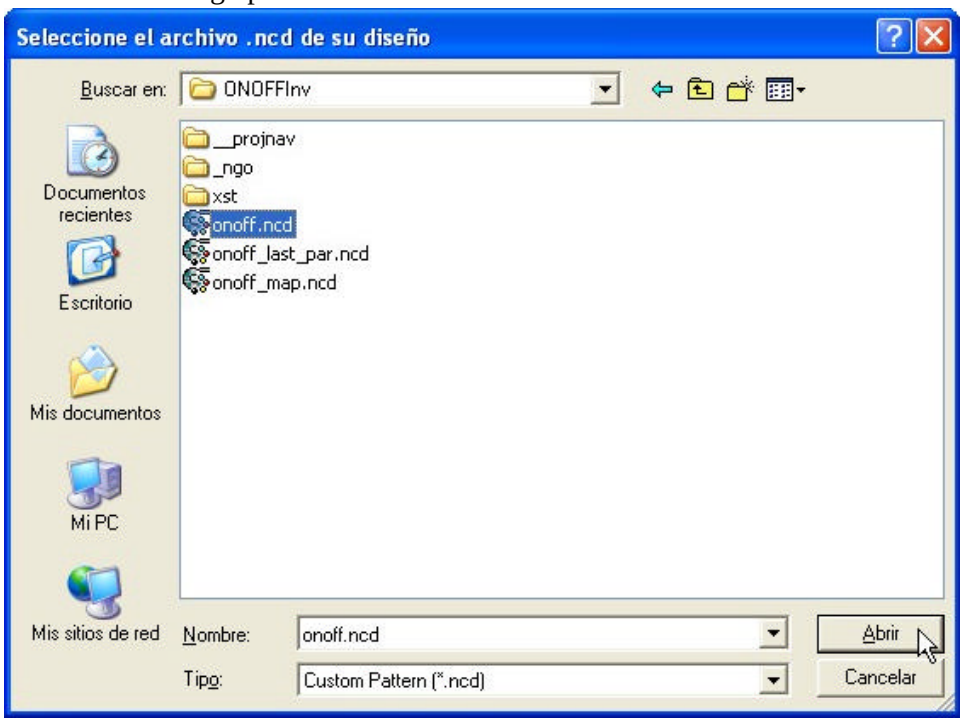
6. Aparecen las muestras en el gráfico de visualización. Para cerrar, se pulsa el botón "TERMINAR". Al pulsarlo la aplicación se termina.

Si la opción es **"Analizador"** o **"Aplicar"**, a partir del paso 4 el algoritmo continúa de la siguiente manera:

4. Aparecen consecutivamente dos cuadros de diálogo para abrir archivos.

En el primer cuadro de diálogo se indica que se debe seleccionar el archivo de extensión .ncd correspondiente al diseño que se configuró. Se puede ver en la figura 25.

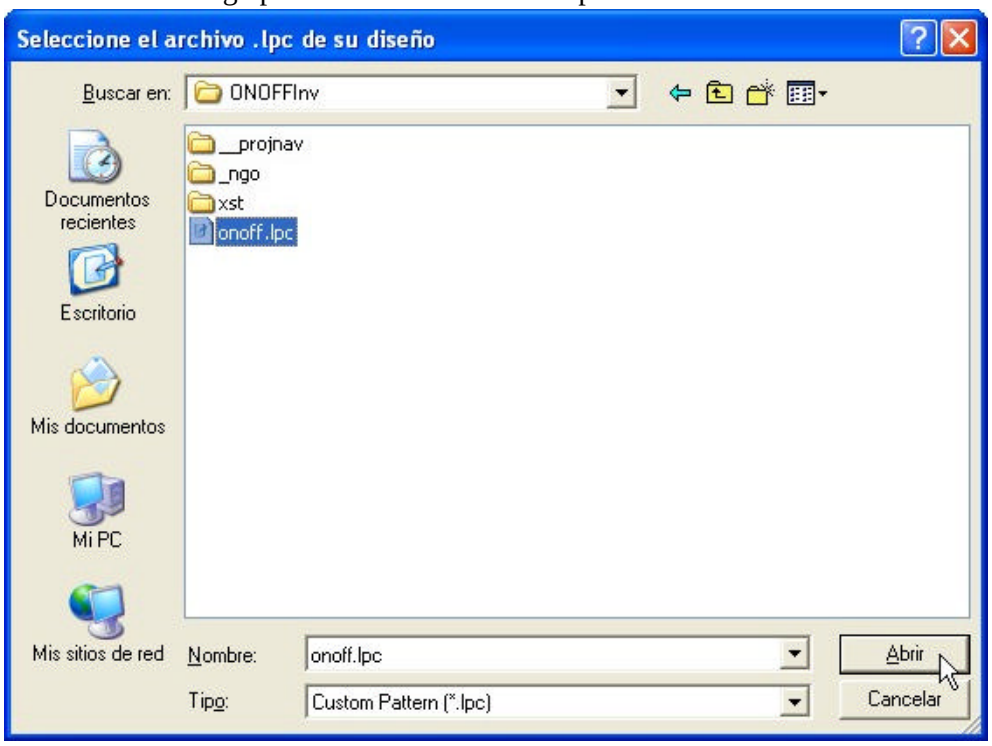
Figura 25. Cuadro de diálogo para selección de archivo .ncd



Fuente: Investigación del Autor.

Otro cuadro aparece a continuación (según el número de muestras puede tardar unos segundos en aparecer) e indica que se debe seleccionar el archivo de extensión .lpc correspondiente al mismo diseño. El archivo tiene el mismo nombre que el anterior y se encuentra en la misma carpeta. La figura 26 muestra un ejemplo de este cuadro de diálogo.

Figura 26. Cuadro de diálogo para selección de archivo .lpc



Fuente: Investigación del Autor.

5. Este paso consiste en la generación de estímulos para aplicar a los terminales de entrada del diseño y se ejecuta únicamente si la opción es **"Aplicar"**. Aparece un cuadro de diálogo que se observa en la figura 27 que indica al usuario que debe crear la secuencia de bits a aplicar en cada terminal de entrada.

Figura 27. Cuadro de diálogo de la opción "Aplicar"

Fuente: Investigación del Autor.

Después de oprimir el botón "Aceptar" de este cuadro, en la parte inferior del panel frontal aparece la lista de señales de entrada y un control por cada señal cuyo valor puede ser modificado lo cual permite crear una secuencia de bits de una longitud igual a la del número de muestras. Si la longitud es menor, entonces los bits que falten a la izquierda se hacen iguales a cero; si es mayor, los bits que

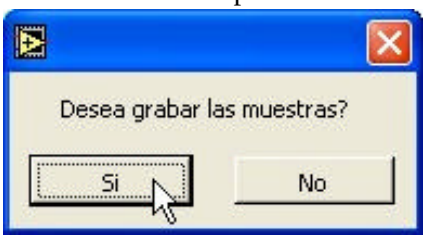
sobran a la izquierda son ignorados. Una vez se ha modificado el valor de cada control para crear los estímulos, se pulsa el botón “OK” que se encuentra en la parte inferior al costado derecho de la lista de señales de entrada.

Finalmente, los últimos dos pasos son iguales tanto para la opción **“Analizador”** como para la opción **“Aplicar”**:

6. Ahora todo ya está listo para realizar el disparo pues ya se generó la prueba. Cada vez que el usuario pulsa el control “TRIGGER” dispara la captura de las muestras y estas se pueden visualizar en el gráfico de visualización.

7. Para guardar estas muestras para su posterior visualización o para cerrar el Analizador, se pulsa el control “TERMINAR” y aparecen un cuadro de diálogo que da la opción de guardar el archivo de muestras para visualizarlas posterior. En la figura 27 se observa este cuadro.

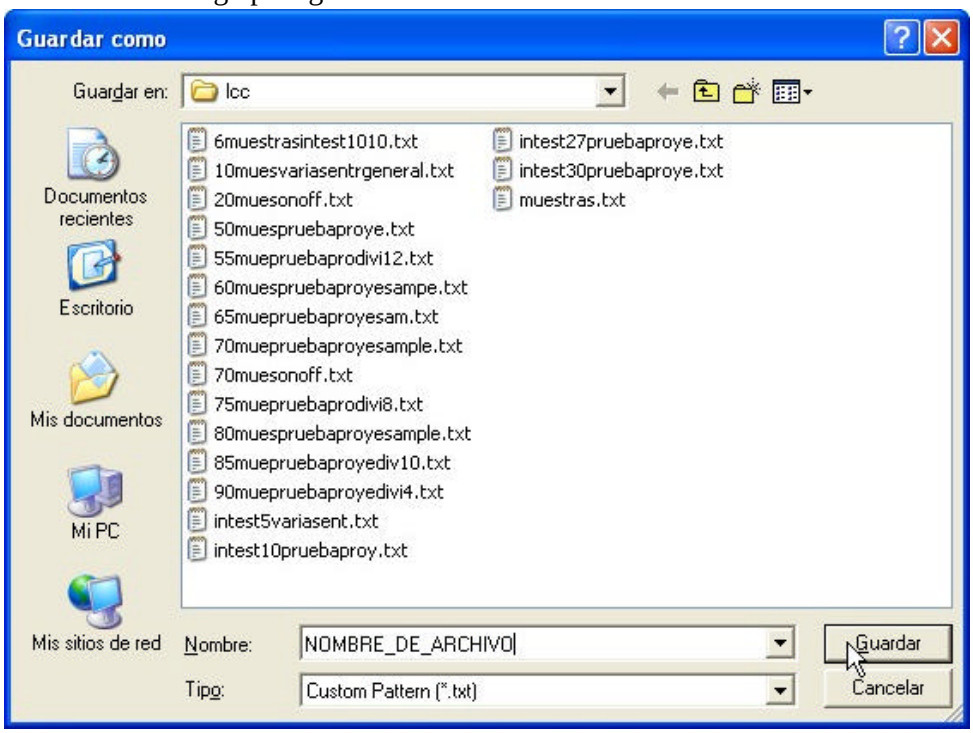
Figura 27. Cuadro de diálogo de terminación de la aplicación.



Fuente: Investigación del Autor.

8. Si el usuario pulsa “No” entonces la aplicación termina. Si pulsa “Si”, se abre otro cuadro de diálogo para guardar el archivo de muestras, como se visualiza en la figura 28. Después de guardar el archivo, la aplicación se termina.

Figura 28. Cuadro de diálogo para guardar las muestras.



Fuente: Investigación del Autor.

En el anexo F se presentan y describen los diagramas de bloques de la interfaz gráfica del Analizador que fue desarrollada en LabVIEW.

7. PROTOCOLO DE PRUEBAS

En este capítulo se consigna una descripción de las pruebas realizadas con el fin de verificar el funcionamiento del LABS así como los resultados obtenidos.

Se generaron varios diseños en el sistema de desarrollo ISE 5 de Xilinx®: inicialmente, se realizó un diseño muy sencillo que consistió un inversor; utilizó para enrutar la señal de entrada el terminal P77 del FPGA, que corresponde al pulsador de la tarjeta de desarrollo Digilab 2E y par la salida el terminal P69 que corresponde al LED de la tarjeta. Además tiene dos salidas adicionales, una que permanece en estado alto y la otra en estado bajo. Los archivos que corresponden a este diseño utilizados por el Analizador son onoff.ncd y onoff.lpc.

Se realizó además un diseño más complejo con un mayor número de señales. Este diseño (secuencial) consistió en un registro de desplazamiento de ocho bits con carga paralela y un divisor de frecuencia. El nombre del archivo que describe el registro es reg8bits.ncd. El archivo .lpc que corresponde a este diseño tiene el mismo nombre, reg8bits.lpc. Los archivos VHDL y los demás archivos que se desarrollaron para la creación de estos diseños se pueden observar en el anexo H.

En principio, se probó cada uno de los módulos desarrollados en lenguaje C independientemente; se ejecutaron de manera secuencial a través de la línea de comandos las seis aplicaciones que se utilizaron en el LABS: *bsdlannno, crearouput, crearsvf, svf2xsvf, playxsvfsi* y *muestreo*; cada una de las aplicaciones generó los archivos necesarios para la ejecución de la siguiente.

Se realizó una prueba que ejecutara la función de Analizador Lógico (es decir, que tomara muestras sin afectar el funcionamiento del FPGA) y que tomara veinte muestras. Inicialmente se configuró el FPGA con el diseño onoff (utilizando el sistema de desarrollo ISE 5 de Xilinx®). A continuación se ejecutaron las aplicaciones; las primeras cuatro generaron el archivo archxsvf.xsvf de la prueba. Las últimas dos aplicaciones permitieron la ejecución de la prueba y la generación del archivo de muestras. A medida que se ejecutaba cada una de las pruebas, se observó cada archivo de salida (si estos son de tipo texto) para verificar los resultados obtenidos.

bsdlanno onoff.ncd archbsdl.bsd –esta aplicación genera el archivo BSDL post- configuración; archbsdl.bsd es el nombre de este archivo.

crearoutput archbsdl.bsd onoff.lpc -s --Genera el archivo textout.txt. Utiliza el archivo BSDL generado con bsdlanno.

crearsvf -s 20 --utiliza el archivo textout.txt generado en la etapa anterior para generar un archivo con las instrucciones SVF necesarias para tomar 20 muestras, llamado salidasvf.txt.

svf2xsvf –w –i salidasvf.txt -o archxsvf.xsvf -a archxsvftexto.txt -- a partir del archivo salidasvf.txt, genera el archivo xsvf que será ejecutado. Con la opción **–a** pudo observarse una versión en caracteres ASCII del archivo generado para verificar su correcta generación.

playxsvfsi –v 4 archxsvf.xsvf --ejecuta la prueba generada; con la opción **–v 4**, a través de línea de comando, se observan con detalle los datos capturados y los estados del TAP.

muestreo --una vez generado el archivo con los valores capturados de TDO (TDOCapt.txt) la ejecución de esta aplicación genera el archivo muestras.txt que contiene los valores lógicos de los estados de las señales.

De igual manera se realizó una prueba configurando el FPGA con el diseño reg8bits, y ejecutando las aplicaciones con los archivos .ncd y .lpc correspondientes a este diseño.

Una vez realizadas estas pruebas, y verificado el funcionamiento de cada uno de los módulos independientemente, se procedió a la ejecución de la prueba del Analizador Lógico desde la interfaz; la interfaz invoca cada uno de estos procesos de manera que genera el algoritmo adecuado para crear y ejecutar la prueba según los parámetros del usuario. Además permite la visualización en un gráfico de los estados que fueron capturados.

Ya que el analizador comprende 3 funciones (“Analizador”, “Aplicar”, “Abrir”) como se describe previamente …véase Sección 6… se realizó una prueba para cada una de ellas.

PRUEBA 1: En la primera prueba se busca verificar la generación y aplicación de estímulos a las señales de entrada, para observar lo que sucede en los terminales de salida. Se configuró el FPGA con el diseño reg8bits (ver anexo H para observar el archivo VHDL que describe este diseño). La frecuencia de la señal de salida *reloj1* es el doble de la frecuencia de *clk*. Los parámetros de la primera prueba fueron: Opción: “Aplicar” y Nro. de Muestras: 80.

Los estímulos aplicados a las señales de entrada fueron:

```
clk=     0101010101010101010101010101010101010101010101010101010101010101010101
din=      1111111111111111111111111111111111111111111111111111111111111111111111
datosp<0>= 00000000000000000000000000000000000000000000000000000000000000000000
datosp<1>= 11111111111111111111111111111111111111111111111111111111111111111111
datosp<2>= 00000000000000000000000000000000000000000000000000000000000000000000
datosp<3>= 11111111111111111111111111111111111111111111111111111111111111111111
datosp<4>= 00000000000000000000000000000000000000000000000000000000000000000000
```

```
datosp<5>= 11111111111111111111111111111111111111111111111111111111111111111111111111111111
datosp<6>= 00000000000000000000000000000000000000000000000000000000000000000000000000000000
datosp<7>= 11111111111111111111111111111111111111111111111111111111111111111111111111111111
cargap=    00000000000000000000000000000000000000000001111000000000000000000000000000000000
```

Este diseño tiene 2 salidas: reloj1 (1 pulso por cada 2 pulsos de clk) y *dout*, que fue igual a cero hasta que hubo 8 pulsos de reloj y luego se hizo igual a *din*; cuando *cargap* (señal que dispara la carga paralela) se hizo igual a uno y hubo un flanco de subida de *reloj1*, *dout* se hizo igual a los valores cargados en el bus *datosp*. El archivo de muestras que corresponde es prueba1.txt en la carpeta Anexos\anexo H. En este caso, las salidas correspondieron a los valores esperados.

```
reloj1=   00110011001100110011001100110011001100110011001100110011001100110011001100110011
dout=     00000000000000000000000000000011111111111111110000111100001111000011110000111111
```

PRUEBA 2: En la segunda prueba se busca verificar el funcionamiento de la función Analizador con la que se hace la captura sin afectar el funcionamiento normal del FPGA. Parámetros: Opción: "Analizador" y Nro. de muestras: 100.

En este caso se busca conocer el tiempo de muestreo, ya que la frecuencia de la señal de entrada *tck* es conocida e igual a 50 MHz., ya que esta señal fue conectada al terminal P80 que corresponde al oscilador externo de la Tarjeta Digilab 2E. En este caso la frecuencia de *reloj1* será igual a la de tck dividida por 2^{24}. La frecuencia de *reloj1* entonces es igual a 50 $(10^6)/2^{24}$= 2,98Hz.

Se ejecutó la prueba y el analizador tomó 79 muestras en 5 pulsos de *reloj1*. Con este dato se puede calcular el tiempo de muestreo (igual al inverso de la frecuencia de muestreo en muestras /segundo).

$$\text{frecuencia de muestreo} = 2{,}98 \frac{\text{pulsos}}{\text{segundo}} * \frac{79}{5} \frac{\text{muestras}}{\text{pulso}} = 47{,}1 \text{ muestras/segundo}$$

Entonces el tiempo de muestreo es igual a 1/47,1= 21,2 ms.

Con esta misma prueba se verificó que los cambios en las entradas se reflejaran en las muestras tomadas. Para esto la entrada *cargap* se conectó al terminal P77 (pulsador). Este fue pulsado varias veces con el fin de observar la captura del estado de esta señal; los resultados son conformes a lo esperado. El archivo de muestras que corresponde es prueba2.txt en la carpeta Anexos\anexo H.

PRUEBA 3: Con la tercera prueba se verifica que se visualizan correctamente muestras guardadas de análisis realizados previamente; en esta prueba los parámetros fueron: Opción: "Abrir" y Nro. de muestras: 100. A continuación se pulsó el control trigger de manera que aparece un cuadro de diálogo en el que se seleccionó el archivo prueba1.txt. Se realizó esta prueba además con otros archivos guardados de la ejecución de diferentes pruebas; se pudo verificar que se observan las muestras correspondientes a la ejecución de dichas pruebas.

8. CONCLUSIONES

Al concluir el desarrollo de esta investigación, se cuenta en la Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones (E^3T) de la UIS con una herramienta que mediante software implementa un Analizador Lógico Virtual para el FPGA XC2S200E de Xilinx® de la tarjeta de desarrollo Digilab 2E de Digilent®. Adicionalmente, el algoritmo desarrollado permite aplicar estímulos en los terminales de entrada del FPGA y capturar las respuestas a estos estímulos sin necesidad de sondas físicas externas, utilizando el puerto paralelo de un computador. El algoritmo desarrollado utiliza como se planteó en los objetivos el estándar 1149.1 del IEEE, conocido como **Boundary- Scan**.

Con el estudio del Estándar **Boundary- Scan** se concluye que puede ser utilizado para realizar pruebas a los dispositivos lógicos en los cuales se implementa; estas pruebas permiten no sólo la verificación funcional sino también física de los dispositivos y su configuración desde un computador, funcionalidad que ha sido utilizada por los fabricantes de estos circuitos integrados. **Boundary- Scan** es una de las herramientas que en la actualidad los fabricantes de FPGAs, CPLDs, DSPs o PROMs implementan en sus dispositivos por lo cual es importante sacar provecho de ella para que no sea desaprovechada.

Existe un gran volumen de información acerca del estándar disponible en Internet. No tanto así en libros que se publican en el país. Ya que la información se encuentra tan dispersa y no es muy específica en cuanto la manera de llevar a la práctica un algoritmo que haga uso de **Boundary-Scan**, es de gran importancia para los estudiantes de la E^3T contar con un documento que recopile la información encontrada y explique su funcionamiento. De esta manera se encontrarán aplicaciones en la academia y la investigación para el estándar. Este documento que fue creado, fue también muy importante para crear el algoritmo del Analizador Lógico Virtual.

La gran flexibilidad de los FPGAs (ya que pueden ser muy fácilmente reconfigurables múltiples veces) junto con una muy alta integración y densidad (muchos pines de salida de difícil acceso en un

espacio muy pequeño) hacen que una herramienta que permite hacer diagnóstico de su funcionamiento sin necesidad de conectar o desconectar sondas, sino a través de una sencilla interfaz en un computador sea de gran utilidad. Además es un complemento al sistema de desarrollo utilizado, ISE 5 de Xilinx®. En el caso de esta tarjeta se utilizó para el diseño, síntesis y simulación de sistemas lógicos, por ser una herramienta poderosa de soporte para los dispositivos programables de Xilinx®.

Se comprobó la utilidad de LabVIEW® para el diseño de sistemas de verificación; en este caso, la facilidad de crear un Instrumento que permitiera la visualización de resultados. Las aplicaciones de LabVIEW® son tan útiles como variadas, y la interfaz creada gracias a este lenguaje es fácil de utilizar para el usuario que está familiarizado con el concepto de analizador lógico.

El desarrollo de un algoritmo de prueba en lenguaje VHDL permitió comprobar que el algoritmo desarrollado tiene un buen desempeño, además permitió verificar la frecuencia a la cual se realiza la captura de muestras. La ejecución de pruebas de manera modular facilitó la tarea de depuración de errores en el código y en la interfaz así como en el producto final, y la optimización de la operación del Analizador Lógico Virtual.

La mayor limitación del algoritmo desarrollado es la frecuencia a la cual se capturan las muestras, pues debido a la misma arquitectura de **Boundary- Scan** que realiza el sondeo de los datos de manera serie, independientemente del número de señales que se desee capturar con el analizador la frecuencia es igual pues el número de bits desplazados es siempre el mismo.

Además de su sistema de desarrollo, Xilinx® pone a disposición de los usuarios de sus productos unas notas de aplicación disponibles en Internet que son de gran utilidad pues explican con detalle las herramientas y utilidades de sus circuitos integrados. Una de ellas constituyó la base para aplicar las instrucciones **Boundary- Scan** y hacer la captura, la Nota de Aplicación 058 o XAPP058 (ver referencia bibliográfica), la cual incluye el código fuente de un intérprete de comandos XSVF el cual fue modificado para tomar las muestras y poder ejecutar el algoritmo. Es de importancia tanto para el diseñador de sistemas digitales que trabaja en la industria como para el investigador que

desde la academia estudia los dispositivos lógicos programables contar con este apoyo del fabricante pues esto fomenta el uso y el estudio de estos dispositivos.

Finalmente, puede concluirse que los objetivos planteados al inicio de la investigación se cumplieron satisfactoriamente.

9. RECOMENDACIONES

El algoritmo desarrollado realiza la comunicación con la tarjeta Digilab 2E a través del puerto del computador; esta tarjeta de desarrollo contiene un puerto paralelo por lo que no hubo que crear una interfaz física; sin embargo, se pone a consideración la posibilidad de extender el protocolo para que utilice el puerto USB, pues tal como es la tendencia en este momento, será el puerto más utilizado en los sistemas informáticos en los años que vienen.

La aplicación puede ser extendida para el análisis de otros dispositivos lógicos que tengan implementada la arquitectura Boundary- Scan; para poder llevarlo a cabo, la información necesaria para la generación de la prueba (longitud del registro de Instrucciones y registro Boundary- Scan, código operacional binario de las instrucciones Boundary- Scan, longitud en bits del BSR, etc.), se encuentra contenida en el archivo BSDL que describe la arquitectura de cada dispositivo. Ya que este lenguaje BSDL se ha establecido y ya es utilizado en la industria, se tiene la información necesaria para la generación de la prueba modificando el código fuente correspondiente al ejecutable crearsvf.exe que es el que genera el archivo de la prueba (SVF).

Puede optimizarse la velocidad a la cual el Analizador desarrollado realiza la captura de los datos, aumentando la frecuencia de muestreo. Esto se puede hacer reduciendo el tiempo de ejecución del algoritmo *playxsvfsi.exe,* que tarda mucho tiempo en escribir las muestras capturadas a un archivo tipo texto. Enviando la información directamente a la aplicación muestreo.exe que es la que toma las muestras correspondientes a las señales del diseño, puede reducirse este tiempo de ejecución.

El Analizador Lógico Virtual desarrollado realiza la captura del número de muestras deseado por el usuario y después de terminar esta captura muestra los datos capturados en la interfaz. A mayor número de muestras, al analizador le toma más tiempo permitir la visualización de los datos. Sin embargo, esta aplicación puede hacerse en tiempo real siempre que se aumente la frecuencia de muestreo. Sin embargo, hay que tener en cuenta que esta velocidad se encuentra limitado por el reloj TCK y la máxima frecuencia que este soporta. Una forma de hacer esto es crear una prueba

que tome una muestra cada vez que se ejecute, y ejecutarla continuamente en una estructura tipo "while" que permite la ejecución del código que se encuentra dentro de ella hasta la aplicación de una señal externa que le indique que se detenga.

Se recomienda la continuidad en el estudio de los dispositivos lógicos programables y las nuevas generaciones que salen al mercado periódicamente, pues ellas presentan funcionalidades que permiten una gran libertad y opciones de diseño. Igualmente, la continuidad en la aplicación del estándar **Boundary- Scan** como herramienta para el desarrollo de sistemas con estos dispositivos.

BIBLIOGRAFIA

ASSET INTERTECH, INC. Serial Vector Formal Specification [en línea], Revision E. [s.l.]: Asset InterTech, 1999. 26 p. [Consulta: 2 feb. 2005]. Disponible en Internet: <http://www.asset-intertech.com/support/svf.pdf>.

--------. Boundary- Scan Tutorial [en línea]. [s.l.]: ASSET InterTech, 2000. 78 p. [Consulta: 19 Ene. 2005]. Disponible en Internet: <http://www.ezytech.com/asset/pdfs/boundaryscan_tutorial.pdf>.

BARR, Michael. Programmable Logic: What's it to ya? [en línea]. En: Embedded Systems Programming Magazine, Vol. 12 No. 6 (Jun. 1999). p. 75-84. [Consulta: 5 Jul. 2004]. Disponible en Internet: <http://www.embedded.com/1999/9906/9906sr.htm>.

BENNETTS, R G. Boundary- Scan Tutorial [en línea]. V. 2.1. [s.l.]: Asset InterTech, 2002. 58 p. [Consulta: 5 Ago. 2004]. Disponible en Internet: <http://www.asset-intertech.com/PDFs/boundaryscan_tutorial.pdf >.

BRIGFORD, Brendan y CAMMON, Justin. Application Note 503: SVF and XSVF File Formats for Xilinx® Devices [en línea]. V. 1.0. [s.l.]: Xilinx®, 2002. 19 p. [Consulta 30 Sept. 2004]. Disponible en Internet: <http://direct.xilinx.com/bvdocs /appnotes/xapp503.pdf>.

CALDAS TORRES, Marcos y DE PRIEGO, Arturo Miguel. Verificación de Hardware mediante Software: El Estándar 1149 y el Desarrollo en la PUCP de un Equipo Automático de Pruebas [en línea]. Lima, 2000, 10 p. Pontificia Universidad Católica del Perú. Grupo de Microelectrónica. [Consulta: 2 Mar. 2005]. Disponible en Internet: <http://www.iberchip.org/VII/cdnav/pdf/62.pdf>.

CORSO SARMIENTO, Jorge Arturo y NEIRA PERDOMO, Ricardo Andrés. Diseño y Construcción de una Tarjeta de Desarrollo para los FPGAs XC4005E /XC4005XL/XC4010XL de XILINX®. Bucaramanga, 2002, 148 p. Trabajo de grado. Universidad Industrial de Santander.

Facultad de Fisicomecánicas. Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones.

DATA FLUX SYSTEMS INC. Scalable Computing Fabric [en línea]. [s.l.]: Data Flux Systems Inc., 2003. [Consulta: 15 Dic. 2004]. Disponible en Internet: <http://www.datafluxsystems.com/scf.htm>.

DIGILENT® INC. DIGILAB 2E Reference Manual [en línea]. Revisión B. [s.l.]: Digilent®, 2002. 15 p. [Consulta: 2 Mar. 2005]. Disponible en Internet: <http://iris.iau.dtu.dk/courses/31028/Supplement/Hardware/d2e.pdf>.

GONZÁLEZ DE RIVERA, Guillermo. Proyecto de analizador lógico y generador de funciones a través del puerto paralelo del PC [en línea]. Madrid: Escuela Politécnica Superior, 2002. Universidad Autónoma de Madrid. Escuela Politécnica Superior. [Consulta: 13 Sep. 2004]. Disponible en Internet: <http://www.ii.uam.es/~gdrivera/labetcii/curso0304/proyecto.htm>.

GONZÁLEZ, Juan. Convirtiendo el hardware en Software: FPGAs [en línea]. En: x- ezine revista electrónica, No. 2 (Oct. 2002). [Consulta: 19 Dic. 2004]. Disponible en Internet: <http://x-ezine.todo-linux.com/x2/2x011-fpga.html>.

IEEE WORKING GROUP P1149.1. IEEE Standard Test Access Port and Boundary-Scan Architecture [base de datos en dvd]. Piscataway, New Jersey: The Institute of Electrical and Electronics Engineers Inc, 2001. 200 p. Disponible en IEEE/IEE Electronic Library 3.0, Versión en DVD, IELDVD040, Base de Datos Biblioteca Central Universidad Industrial de Santander.

PARDO, Fernando y BOLUDA, José. VHDL: Lenguaje para síntesis y modelado de circuitos. Ciudad de México: Alfaomega, 2000. 238 p.

PHILLIPS SEMICONDUCTORS. Data Sheet: 74HC/HCT244 Octal buffer/line driver; 3-state [en línea]. [s.l.]: Phillips Semiconductors, 1990. 7 p. [Consulta: 21 Ene. 2005]. Disponible en Internet: <http://www.standardproducts.philips.com/products/hc/pdf/74hc244.pdf>.

REBAUDENGO, Maurizio. E-learning Resources in Microelectronics: Boundary-Scan Testing [en línea]. En: Microelectronics for Industrialists. E- Learning Resources in Microelectronics. [Torino]: COREP, Politecnico de Torino, 2001. [Consulta: 28 Feb. 2005]. Disponible en Internet: <http://www.bolton.ac.uk/mind/corep/bst/bst-intro.html>.

SALCIC, Zoran y SMAILAGIC, Asim. Digital Systems Design and Prototyping: Using Field Programmable Logic and Hardware Description Languages. 2 ed. Norwell, Massachusetts: Kluwer Academic Publishers, 2000. 620 p. + cd-rom.

SCHILDT, Herbert. C, Manual de Referencia. Madrid: Osborne/McGraw-Hill, 1989. 695 p.

SILVA BIJIT, Leopoldo y BACIGALUPO, Virgilio. Aplicación de analizadores lógicos en experiencias de laboratorio [en línea]. Valparaíso: 2003, 18 p. [Consulta: 5 feb. 2005]. Universidad Técnica Federico Santa Maria. Departamento de Electrónica. Disponible en Internet: <http://www.google.com/u/eloUTFSM?q=cache:JE5Od-8xbxAJ:www.elo.utfsm.cl/~lsb/elo311/labs/tut-analogic.pdf+Bacigalupo&hl=es&ie=UTF-8>.

TEKTRONIX INC. User Manual: TLA 700 Series Logic Analyzer 070-9775-04 [cd-rom]. U.S.A.: Tektronix Inc, 1998. 252 p.

TELMATIC. Analizador Lógico Serie PA20 [en línea]. Barcelona: Telmatic, 1999. [Consulta 10 Jun. 2004]. Disponible en Internet: <http://www.telmatic.com/LA20.htm>.

WAKERLY, John F. Diseño Digital: Principios y Prácticas. Tercera Edición. Ciudad de México: Prentice Hall, 2001. 946 p. + cd-rom.

XESS CORPORATION. What are CPLDs and FPGAs? [en línea]. [s.l.]: Xess Corp, [s.f.]. [Consulta 24 May. 2004]. Disponible en Internet: <http://www.xess.com/fpgatut.htm>.

XILINX®. Application Note 058: Xilinx in- system programming using an embebbed microcontroller [en línea]. V. 3.0. [s.l.]: Xilinx®, 2001. 38 p. [Consulta 7 Dic. 2003]. Disponible en Internet: <http://direct.xilinx.com/bvdocs/appnotes/xapp058.pdf>. Archivos asociados disponibles en Internet en: <ftp://ftp.xilinx.com/pub/swhelp/cpld/eisp_pc.zip>.

--------. Application Note 069: Using the XC9500/XL/XV JTAG Boundary Scan interface [en línea]. V. 3.1. [s.l.]: Xilinx®, 2002. 10 p. [Consulta 2 Ene. 2005]. Disponible en Internet: <http://direct.xilinx.com/bvdocs/appnotes/xapp069.pdf>.

--------. Application Note 188: Configuration and Readback of Spartan-II FPGAs Using Boundary-Scan [en línea]. V. 2.1. [s.l.]: Xilinx®, 2002. 15 p. [Consulta 25 May. 2004]. Disponible en Internet: <http://direct.xilinx.com/bvdocs/appnotes/xapp188.pdf>.

--------. Application Note 476: Using BSDL Files for Spartan-3 FPGAs [en línea]. V. 1.0. [s.l.]: Xilinx®, 2003. 6 p. [Consulta 26 May. 2004]. Disponible en Internet: <http://direct.xilinx.com/bvdocs/appnotes/xapp476.pdf>.

--------. Xilinx® 5 Software Manuals [cd-rom]. [s.l.]: Xilinx®, 2002.

--------. Spartan-II 1.8 V FPGA Family: Complete Data Sheet [en línea]. [s.l.]: Xilinx, 2004. 103 p. [Consulta 13 Ago. 2004]. Disponible en Internet: <http://www.xilinx.com/bvdocs/publications/ds077.pdf>.

--------. Virtex™-II Platform FPGAs: Complete Data Sheet [en línea]. [s.l.]: Xilinx, 2004. 315 p. [Consulta 3 Ene. 2005]. Disponible en Internet: <http://www.xilinx.com/partinfo/ds031.pdf>.

--------.What is BSDL, and how do I read a BSDL file? [en línea]. Answer Record # 8350 en: Xilinx Answers Database. [Consulta 19 Jun. 2004]. Disponible en Internet: <http://www.xilinx.com/xlnx/xil_ans_display.jsp?getPagePath=8350>.

ANEXO A:

TERMINALES Y CARACTERÍSTICAS DEL FPGA XC2S200E- PQ208 DE XILINX®

Este anexo presenta algunos datos generales y la tabla de terminales del FPGA XC2S200E – PQ208 que se encuentran en la **data sheet** entregada por el fabricante. Este FPGA hace parte de la familia Spartan IIE de Xilinx®. Su empaquetado es QFP plástico y tiene 208 terminales. En el cuadro se encuentran las características más significativas de este FPGA:

Celdas lógicas	Rango típico de Compuertas de Sistema	Arreglo de CLBs (filas x columnas)	Número total de CLBs	Máximo número disponible de E/S	Máximo número pares diferenciales	Bits de RAM distribuidos	Bits de Bloque RAM
5292	71000-200000	26 x 42	1176	146	120	75264	56k

Definiciones de los terminales: Esta tabla describe los pines dedicados; además de estos, algunos terminales I/O pueden utilizarse para enrutar señales de configuración, DLLs o voltajes de referencia.

FUNCIÓN	**DIRECCIÓN**	**DESCRIPCIÓN**
M0,M1,M2	Entrada	Terminales de modo que especifican el modo de configuración.
CCLK	Entrada o Salida	Terminal de E/S del Reloj de Configuración.
/PROGRAM	Entrada	Cuando su estado es bajo inicia una secuencia de configuración.
DONE	Bidireccional	Indica que la configuración se completó.
TDI, TDO, TMS, TCK	Mixtos	Terminales del Puerto de Acceso para Pruebas de Boundary Scan (IEEE 1149.1- 1990).
VCCINT	Entrada	Terminales a 1.8 V para el núcleo de lógica interno.
VCCO	Entrada	Terminales de alimentación para drivers de salida (1,5V, 1,8V, 2,5V, 3,3V). Deben estar todos conectados al mismo voltaje.
GND	Entrada	Tierra. Todos deben estar conectados.
GCK0, CK1, GCK2, GCK3	Entrada	Terminales para entrada de relojes. Se conectan a búferes de Reloj Globales o a entradas DLL. No son pines dedicados.
I/O	Bidireccional	Pueden configurarse como entradas y/o salidas.

Terminales del FPGA:

PIN	FUNCION
1	GND
2	TMS
3	I/O
4	I/O
5	I/O
6	I/O
7	I/O
8	I/O
9	I/O
10	I/O
11	I/O
12	GND
13	VCCO
14	VCCINT
15	I/O
16	I/O
17	I/O
18	I/O
19	GND
20	I/O
21	I/O
22	I/O
23	I/O
24	I/O
25	GND
26	VCCO
27	I/O
28	VCCINT
29	I/O
30	I/O
31	I/O
32	GND
33	I/O
34	I/O
35	I/O
36	I/O
37	VCCINT
38	VCCO
39	GND
40	I/O
41	I/O
42	I/O
43	I/O
44	I/O
45	I/O
46	I/O
47	I/O
48	I/O
49	I/O
50	M1
51	GND
52	M0

PIN	FUNCION
53	VCCO
54	M2
55	I/O
56	I/O
57	I/O
58	I/O
59	I/O
60	I/O
61	I/O
62	I/O
63	I/O
64	I/O
65	GND
66	VCCO
67	VCCINT
68	I/O
69	I/O
70	I/O
71	I/O
72	GND
73	I/O
74	I/O
75	I/O
76	VCCINT
77	GCK1
78	VCCO
79	GND
80	GCK0
81	I/O
82	I/O
83	I/O
84	I/O
85	GND
86	I/O
87	I/O
88	I/O
89	I/O
90	VCCINT
91	VCCO
92	GND
93	I/O
94	I/O
95	I/O
96	I/O
97	I/O
98	I/O
99	I/O
100	I/O
101	I/O
102	I/O
103	GND
104	DONE

PIN	FUNCION
105	VCCO
106	/PROGRAM
107	I/O
108	I/O
109	I/O
110	I/O
111	I/O
112	I/O
113	I/O
114	I/O
115	I/O
116	I/O
117	GND
118	VCCO
119	VCCINT
120	I/O
121	I/O
122	I/O
123	I/O
124	GND
125	I/O
126	I/O
127	I/O
128	VCCINT
129	I/O
130	VCCO
131	GND
132	I/O
133	I/O
134	I/O
135	I/O
136	I/O
137	GND
138	I/O
139	I/O
140	I/O
141	I/O
142	VCCINT
143	VCCO
144	GND
145	I/O
146	I/O
147	I/O
148	I/O
149	I/O
150	I/O
151	I/O
152	I/O
153	I/O
154	I/O
155	CCLK
156	VCCO

PIN	FUNCION
157	TDO
158	GND
159	TDI
160	I/O
161	I/O
162	I/O
163	I/O
164	I/O
165	I/O
166	I/O
167	I/O
168	I/O
169	I/O
170	GND
171	VCCO
172	VCCINT
173	I/O
174	I/O
175	I/O
176	I/O
177	GND
178	I/O
179	I/O
180	I/O
181	I/O
182	GCK2
183	GND
184	VCCO
185	GCK3
186	VCCINT
187	I/O
188	I/O
189	I/O
190	GND
191	I/O
192	I/O
193	I/O
194	I/O
195	VCCINT
196	VCCO
197	GND
198	I/O
199	I/O
200	I/O
201	I/O
202	I/O
203	I/O
204	I/O
205	I/O
206	I/O
207	TCK
208	VCCO

ANEXO B:

CARACTERÍSTICAS DE LA TARJETA DIGILAB 2E DE DIGILENT®

Este anexo presenta un resumen de las características de la tarjeta Digilab 2E que se encuentran consignadas en el Manual de Referencia de Digilent Inc. La tarjeta de desarrollo Digilab 2E (D2E) basada en FPGA es una excelente plataforma de desarrollo de prototipos de sistemas y circuitos digitales. La tarjeta está construida alrededor del FPGA Spartan 2E XC2S200E- PQ208 de Xilinx®.

Características Principales:

- Todas las señales de E/S disponibles se erutan a través de seis conectores de expansión DIP marcados de la letra A a la F, puertos paralelo y serie y otros dispositivos de la tarjeta. 122 señales se enrutan a través de los conectores de expansión y las demás a través de los puertos.
- Incluye un pulsador conectado al terminal 77 del FPGA (GCK1) y un LED (conectado al terminal 69) para pruebas rápidas del circuito y de la programación del FPGA en las cuales es útil una señal de entrada y una de salida externas(ejemplo: control de reset e indicador de encendido).
- Alimentación con 2 reguladores a 1.5A (3.3V y 2.5V), cuya entrada es una fuente DC externa de entre 5V y 10V.
- Oscilador de 50MHz conectado al terminal 80 (GCK0) del FPGA.
- Un socket para una SPROM de Xilinx®.
- Puerto paralelo DB- 25 para comunicación y transferencia de datos con el computador. Soporta configuración y otras operaciones mediante el estándar Boundary- Scan. Las señales Boundary- Scan se conectan del FPGA al puerto paralelo a través de un búfer octal 74HC244.
- Un header separado para Boundary- Scan para utilizar con los cables paralelos para programación 3 o 4 de Xilinx®.
- Puerto serie RS- 232

Diagrama de conexión del FPGA con el puerto paralelo:

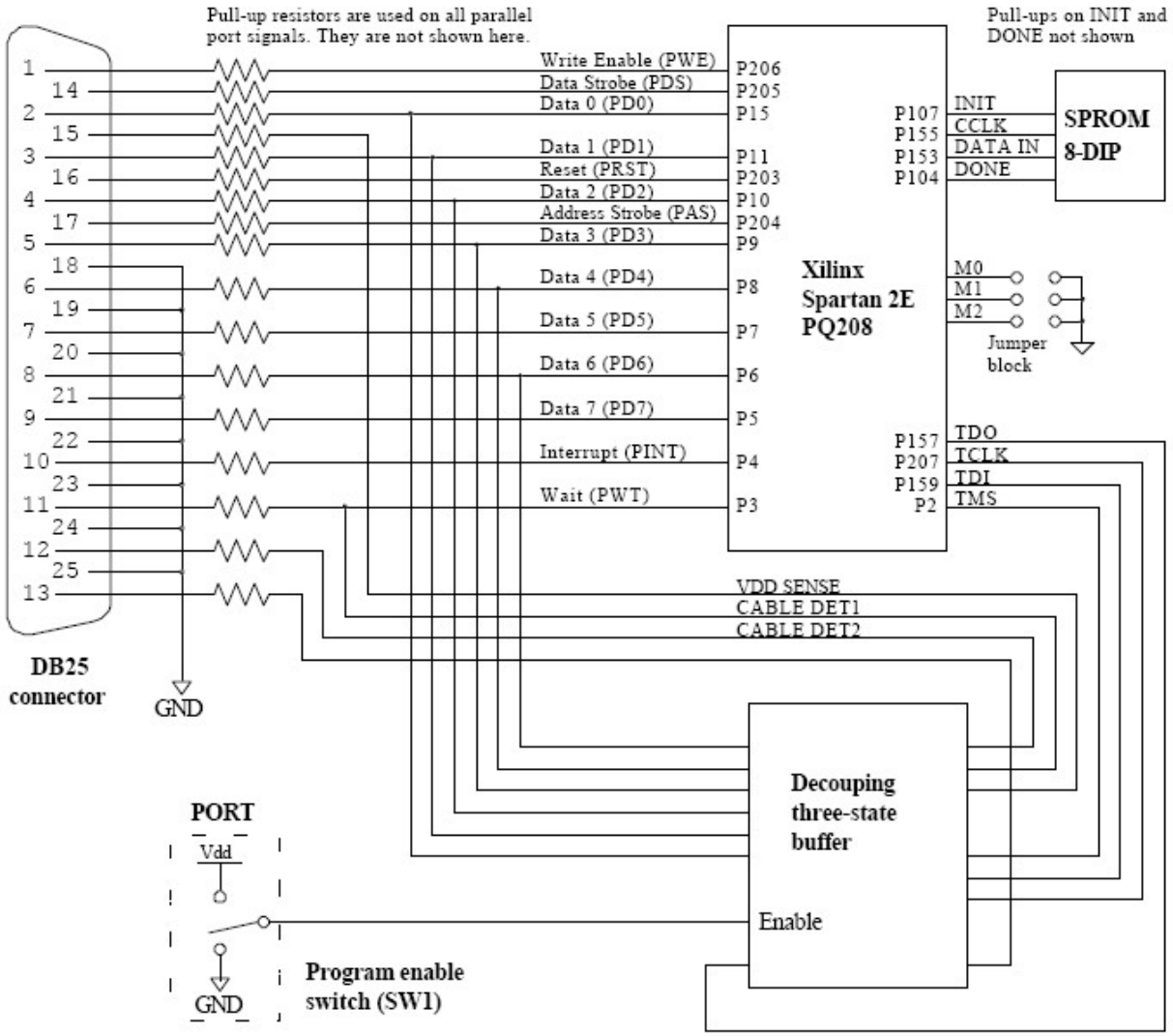


Diagrama del Puerto Paralelo:

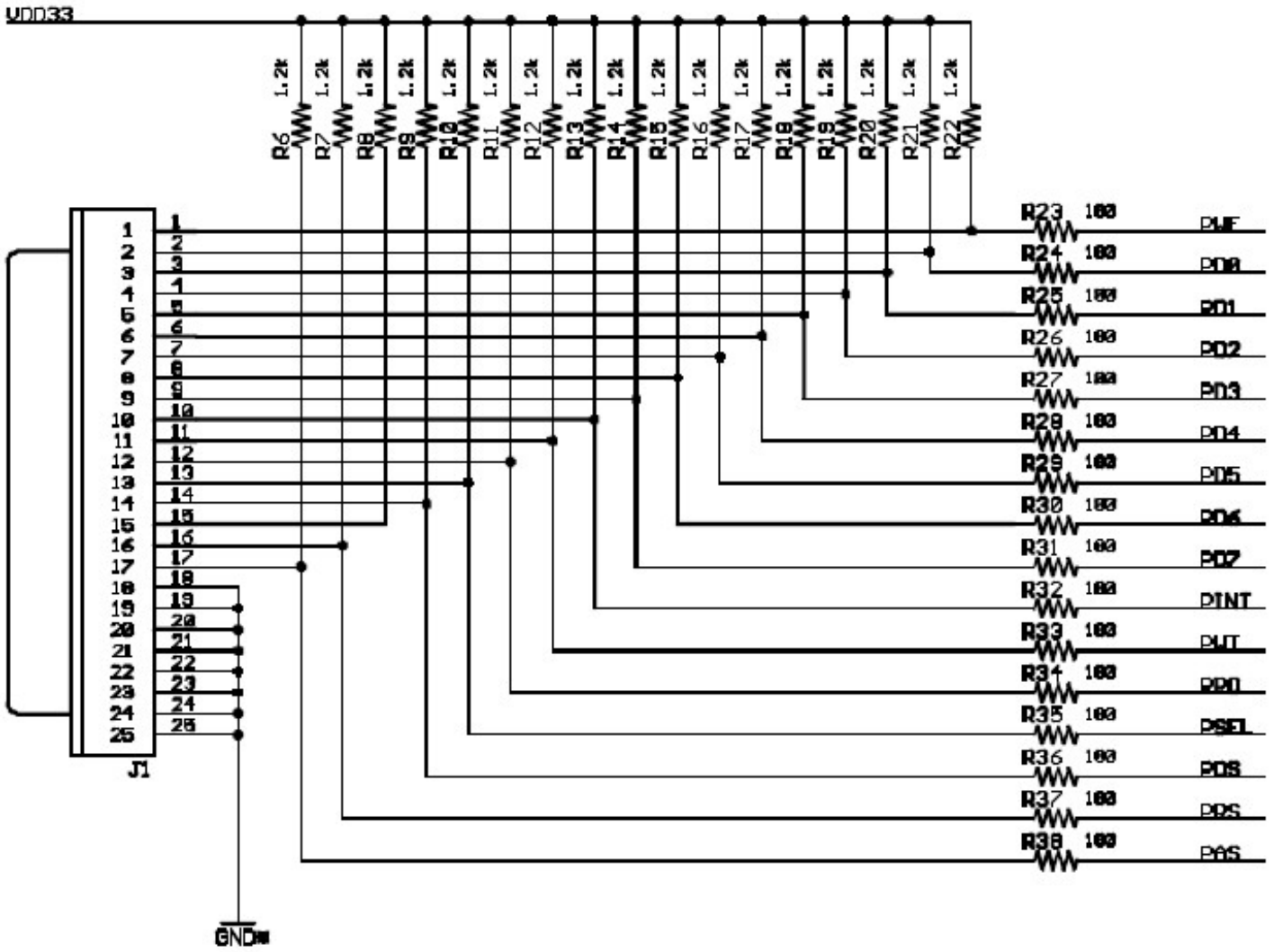


Diagrama de la interfaz de configuración Boundary Scan (Búfer octal):

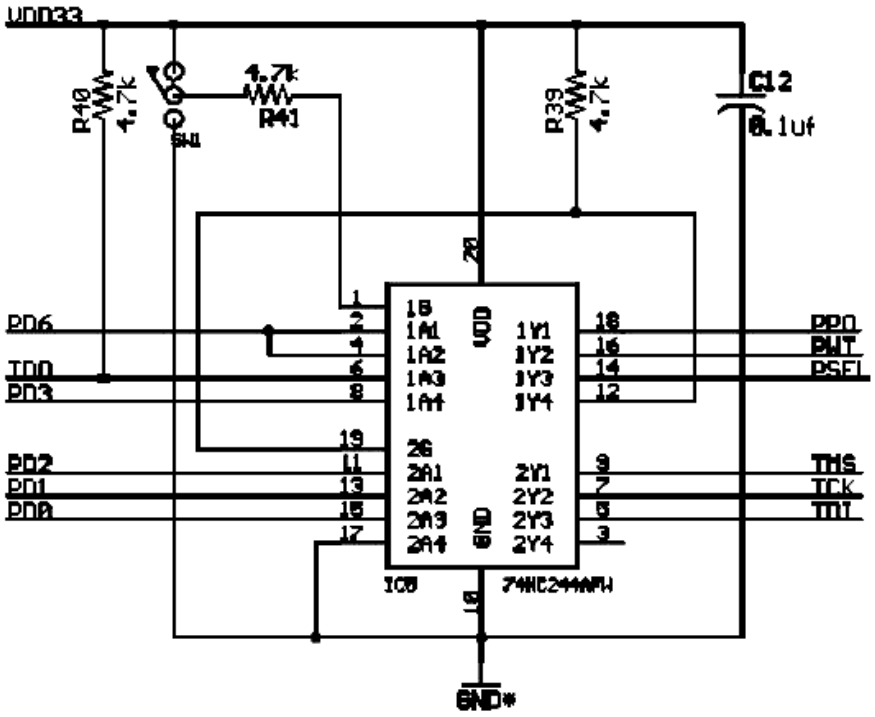
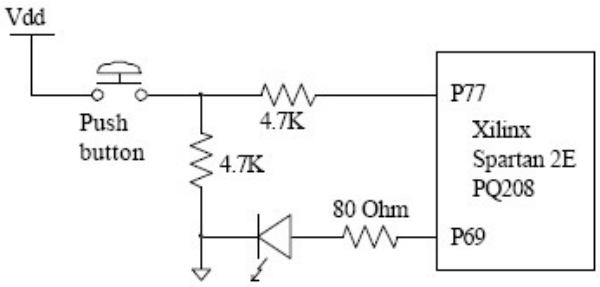


Diagrama de conexión del pulsador y del LED.



ANEXO C:
EL LENGUAJE XSVF DE XILINX®

Este anexo presenta un resumen de la Nota de Aplicación 503 de Xilinx® (APP503) que describe los formatos SVF y XSVF.

XSVF provee la funcionalidad y tiene la forma de un archivo SVF, pero en un formato binario más compacto. Un archivo XSVF se genera mediante el traductor de archivos svf2xsvf que puede ser descargado de Internet y que provee Xilinx junto con la nota de aplicación XAPP058 para los usuarios de sus dispositivos. Cada instrucción XSVF tiene 1 byte de longitud y va acompañada de un argumento de longitud variable. Hay una equivalencia entre las instrucciones SVF y las instrucciones XSVF: para casi todas las instrucciones SVF hay una XSVF equivalente.

La siguiente tabla que se encuentra en el documento APP503 contiene las instrucciones XSVF y los códigos de operación (binarios) que corresponden a cada una.

XSVF Instruction	Binary Encoding (hex)
XCOMPLETE	0x00
XTDOMASK	0x01
XSIR	0x02
XSDR	0x03
XRUNTEST	0x04
XREPEAT	0x07
XSDRSIZE	0x08
XSDRTDO	0x09
XSETSDRMASKS	0x0a
XSDRINC	0x0b
XSDRB	0x0c
XSDRC	0x0d
XSDRE	0x0e
XSDRTDOB	0x0f
XSDRTDOC	0x10
XSDRTDOE	0x11
XSTATE	0x12
XENDIR	0x13
XENDDR	0x14
XSIR2	0x15
XCOMMENT	0x16

ANEXO D:
ARCHIVO BSDL DEL FPGA XC2S200E- PQ208 DE XILINX®

El archivo BSDL correspondiente al FPGA XC2S200E- PQ208 de Xilinx® se encuentra en el CD adjunto, en la carpeta \Anexos\Anexo D.

ANEXO E:
CÓDIGOS FUENTE DEL ANALIZADOR DESARROLLADO

Los archivos correspondientes al código fuente de las aplicaciones desarrolladas se encuentran en el CD adjunto en \Anexos\Anexo E.

ANEXO F:
DIAGRAMAS DE LA INTERFAZ GRÁFICA DESARROLLADA

Este anexo presenta el diagrama correspondiente a la interfaz gráfica del LABS que fue descrita en el capítulo 6 y que se desarrolló en LabVIEW. El Instrumento y sub- instrumentos desarrollados en LabVIEW se encuentran en el CD en la carpeta \Anexos\Anexo F. Inicialmente se presenta el diagrama completo y luego las estructuras que aparecen escondidas pues tienen más de un cuadro o sub- diagrama (Las estructuras de este tipo que se utilizaron son Event, Case, Sequence, While y For). Aquí se describen brevemente cada una de sus partes o sub- diagramas.

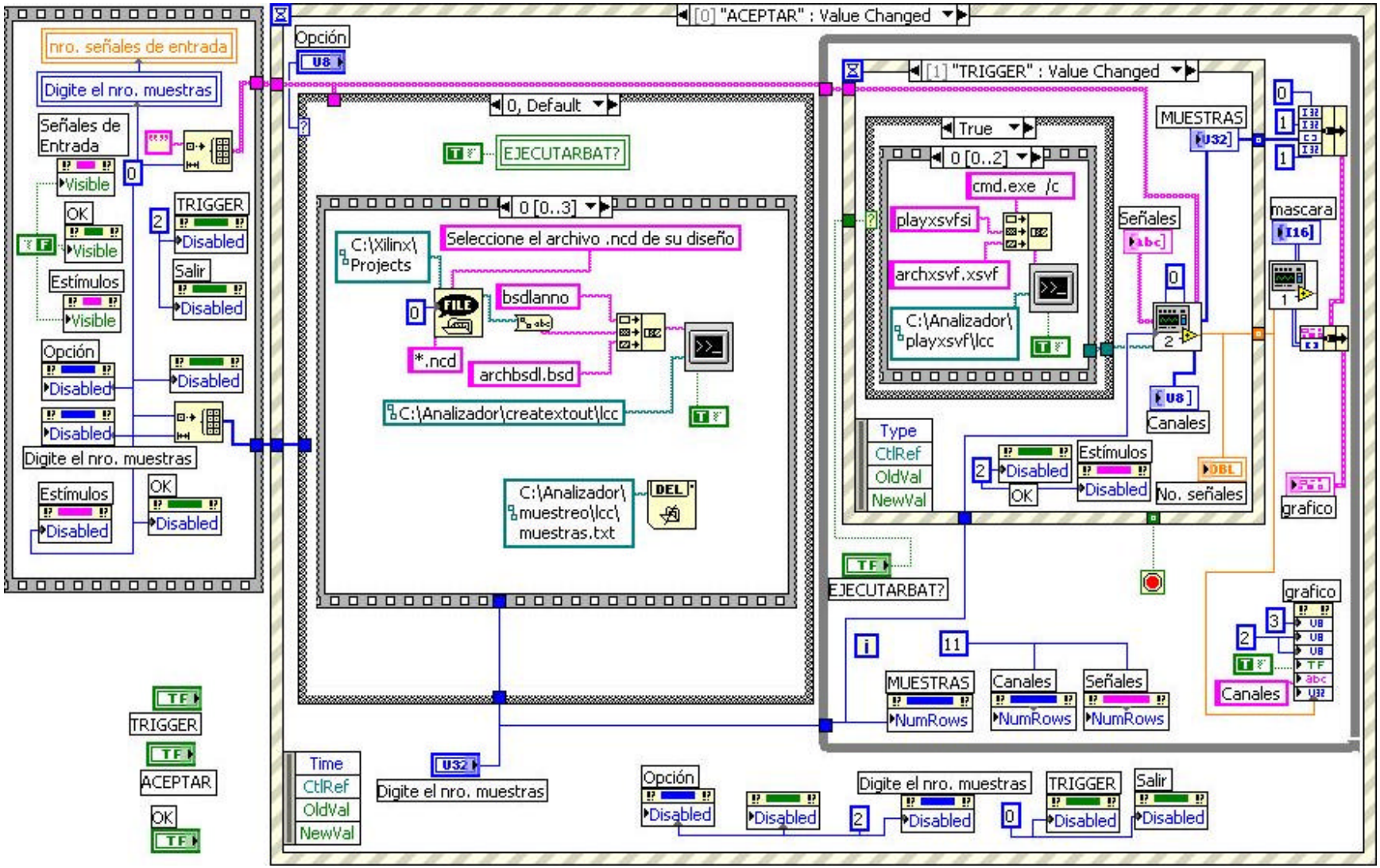
En el Diagrama General de Analizador desarrollado que se presenta en la página siguiente, se pueden observar de izquierda a derecha dos estructuras principales: La primera es una de tipo "Sequence" en la cual se inicializan variables y se establecen propiedades de los indicadores que se muestran en el panel de control. En la parte inferior izquierda se encuentran 3 señales de control de tipo booleano o lógico (es decir que pueden tomar sólo uno de dos valores: Falso o Verdadero): son "TRIGGER" que ejecuta el disparo, "Aceptar" que corresponde al botón "ACEPTAR" del panel frontal y "OK" que corresponde al botón "OK" del panel frontal.

La estructura de la derecha es de tipo "Event". Esta se ejecuta cuando hay un cambio en el botón "OK", es decir, cuando el usuario pulsa el botón "Aceptar" en el panel frontal. En esta estructura se capturan los valores de los dos controles que modifica el usuario inicialmente en el panel frontal; estos controles se pueden ver dentro de ella: "Opción" (en la esquina superior izquierda) y "Digite el nro. muestras" (parte inferior izquierda). Los otros valores dentro de esta estructura en la parte inferior corresponden a algunas propiedades de los controles del panel frontal.

En esta estructura "Event" hay a su vez dos sub- estructuras: a la izquierda una de tipo "Case" y a la derecha un lazo "While". En el "Case" se ejecuta sólo uno de los diagramas, según la opción seleccionada y se genera el archivo XSVF, es decir, la prueba. Por otro lado, el lazo "While" se ejecuta hasta que el control "Terminar" sea pulsado. En este lazo, se realiza la ejecución de la prueba generada en el "Case" tantas veces como haya un cambio en el control "TRIGGER".

En la estructura "Case" la opción puede ser "Analizador" que corresponde al caso 0, "Abrir" que es el caso 1 y "Aplicar" que es el caso 2. En el diagrama general se observa el caso 0, dentro del cual hay una estructura "Event" con 4 cuadros numerados del 0 al 3.

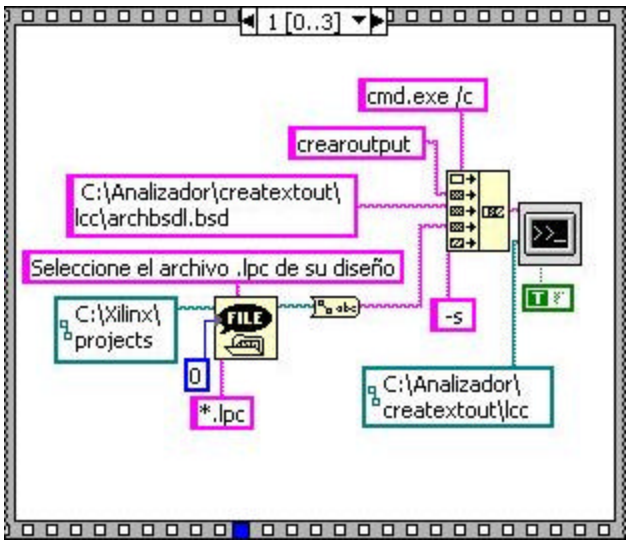
Diagrama General del Analizador.



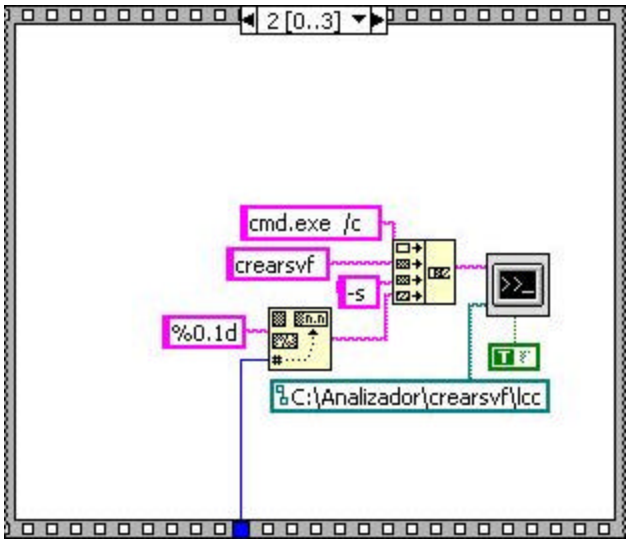
Los cuadros se ejecutan de manera secuencial. Para ejecutar cada una de las aplicaciones creadas externamente se utilizó un instrumento de las librerías de LabVIEW llamado **system exec.vi.** Esta función permitió invocar la ejecución de las aplicaciones creadas.

El cuadro 0 ejecuta la aplicación *BSDLAnno.* Además borra el archivo de muestras para eliminar lo que haya quedado de la ejecución anterior.

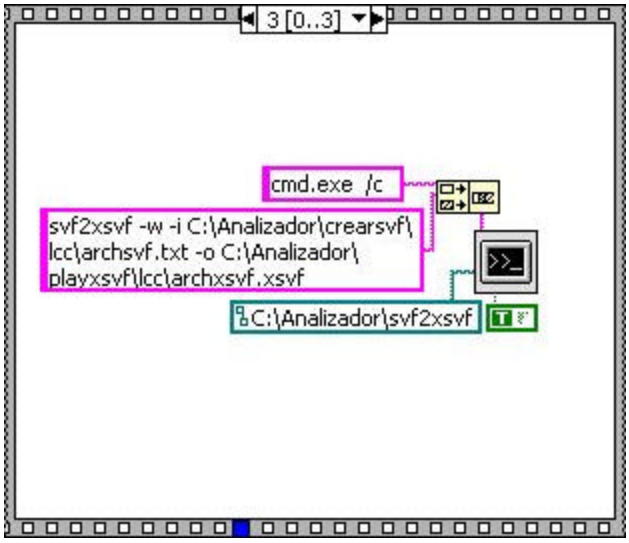
El cuadro 1 ejecuta la aplicación *crearoutput* como se puede ver en la siguiente figura.



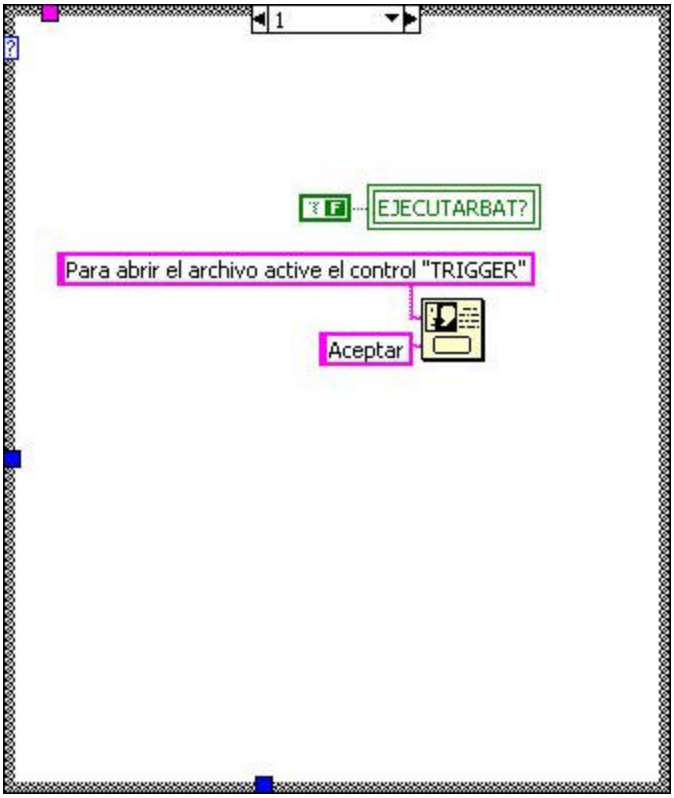
El cuadro 2 ejecuta la aplicación *crearsvf*. En este cuadro uno de los argumentos que se necesita es el número de muestras, por lo que aparece una conexión en color anaranjado que viene del exterior del cuadro.



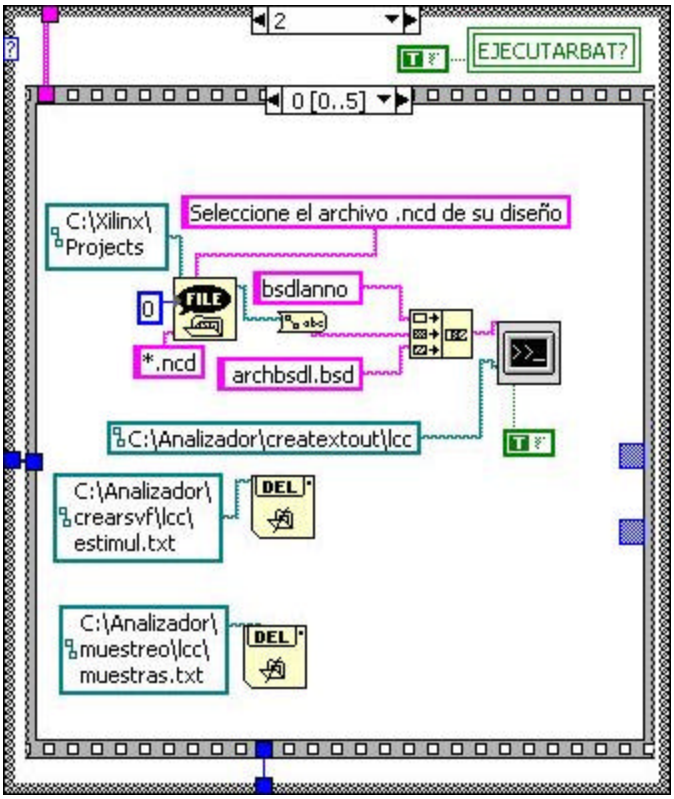
El cuadro 3 es el último de la secuencia y ejecuta la aplicación svf2xsvf. De esta manera queda generado el archivo XSVF para ejecutar la prueba.

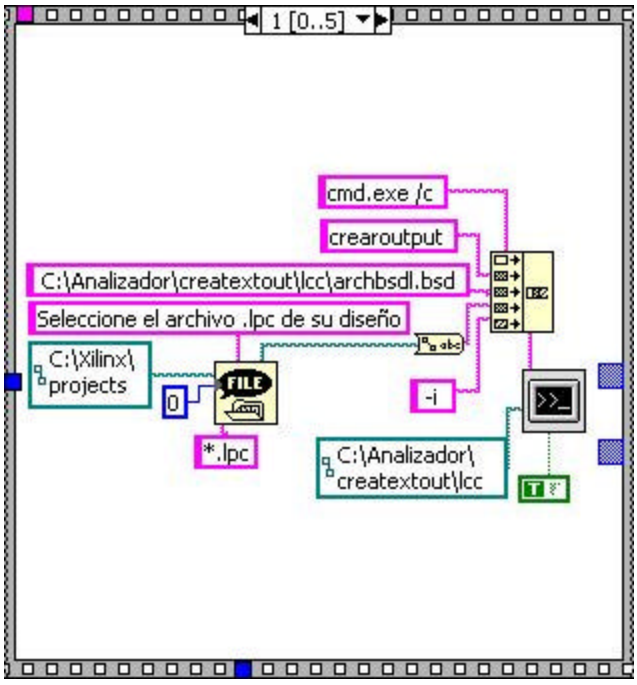


El siguiente caso es el 1 que corresponde a la opción “Abrir”. En este caso, no se genera archivo XSVF sino que aparece un cuadro de diálogo que indica al usuario que pulse en control “TRIGGER” del panel frontal.

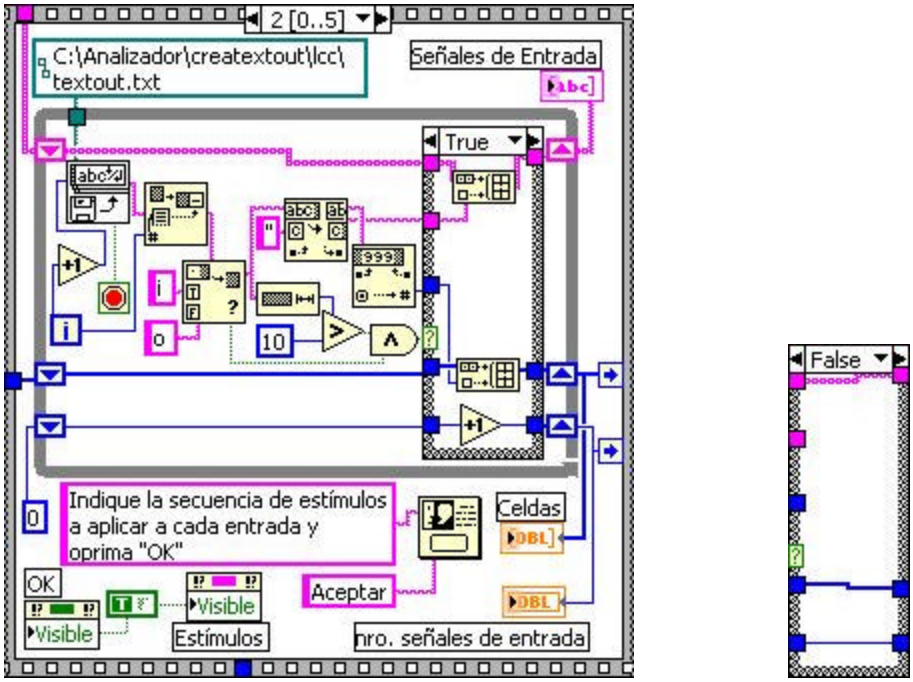


El último caso es el 2 que corresponde a la opción “Aplicar”. Dentro de este caso hay una estructura “Sequence” de 6 cuadros numerados del 0 al 5. Los cuadros 0 y 1 (las figuras a continuación) son equivalentes a los del caso 0; ejecutan de forma secuencial las aplicaciones *BSDLAnno* y *crearouput.*

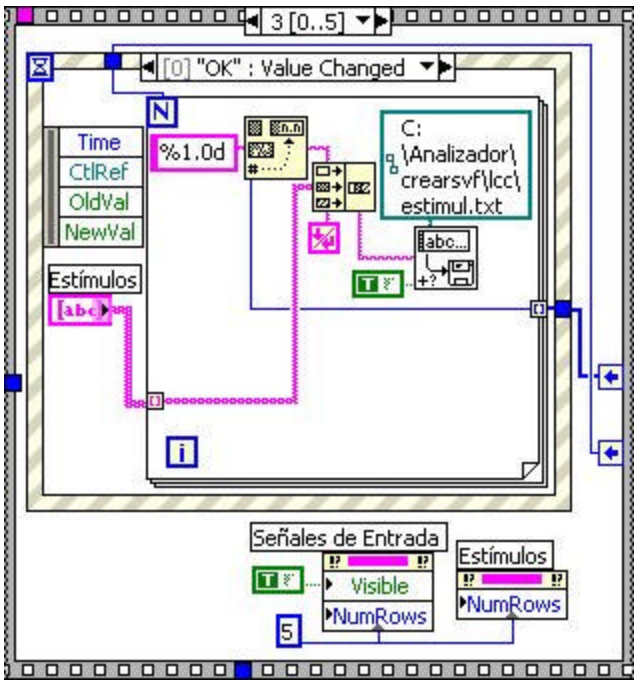




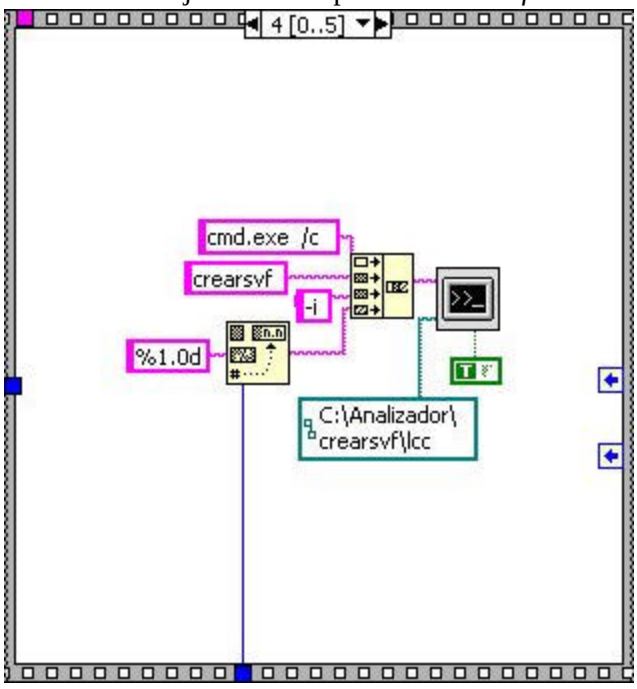
El siguiente cuadro de la secuencia (cuadro 2, en la figura a continuación) se encarga de crear un indicador con las señales de entrada (el arreglo “Señales de Entrada” que se ve en la figura).



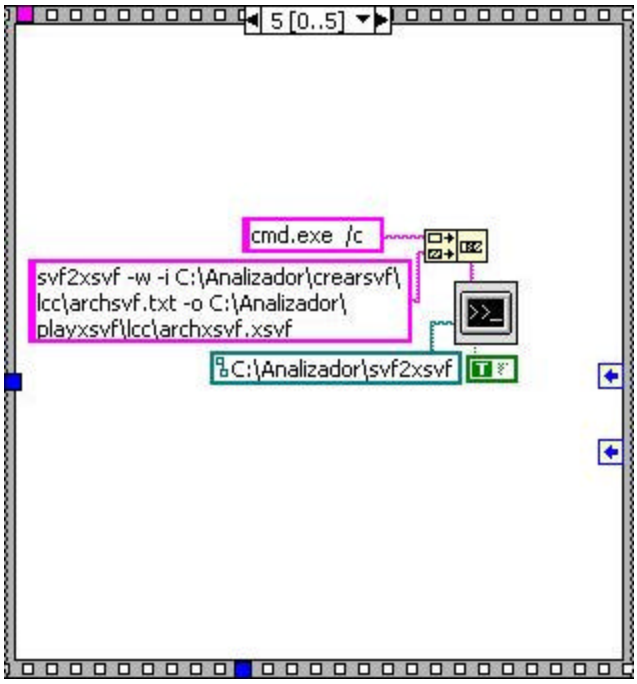
El cuadro 3 crea un arreglo de controles que el usuario puede modificar (el arreglo “Estímulos”) y contiene el proceso para crear al pulsar el botón OK (a partir de los estímulos que haya creado el usuario) el archivo “estimul.txt” necesario para la creación del archivo SVF de la prueba.



El cuadro 4 de la secuencia invoca la ejecución del proceso *crearsvf*.



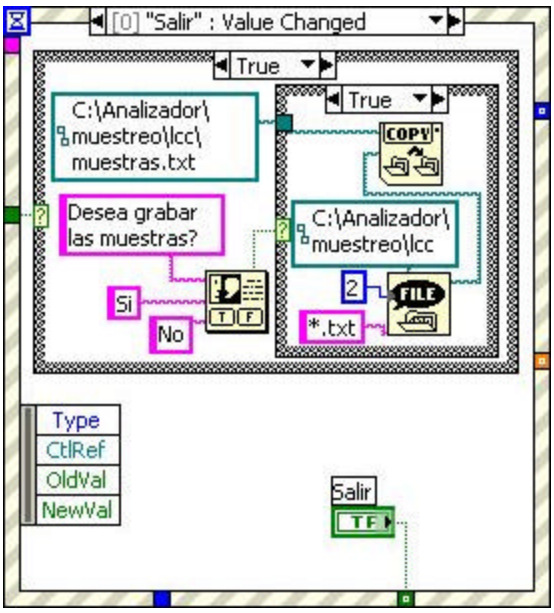
El último cuadro de la secuencia (cuadro 5), invoca la ejecución de la aplicación svf2xsvf, terminando así la generación del archivo XSVF para la prueba.



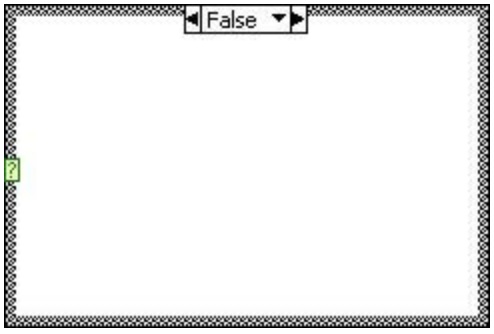
Aquí termina la descripción de los 3 casos de la estructura Case. A continuación, continúa la descripción del lazo "While" que ejecuta el archivo XSVF creado, es decir, ejecuta la prueba.

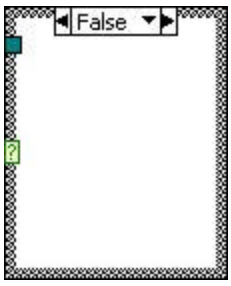
Dentro del lazo “While”, se encuentra otra estructura de tipo “Event”; en este caso, la estructura es sensible a 2 eventos: el evento [0] se ejecuta cuando hay un cambio en la señal “Salir”, es decir, cuando se pulsa el botón “Terminar” del panel frontal.

En la figura siguiente se observa el sub- diagrama del evento [0]; en este evento la señal “Salir” se conecta a la condición de salida del lazo While con lo que se termina la ejecución de la interfaz; dentro del diagrama del evento, en una estructura de tipo “Case”, se controla la opción que permite guardar las muestras tomadas: en el caso de que se haya ejecutado una prueba (caso “True”) aparece un cuadro de diálogo que abre al usuario la posibilidad de guardar las muestras.



En caso de que se haya abierto un archivo previamente almacenado (Caso “False”, figura inferior izquierda) no se hace nada y la aplicación termina directamente. La figura a la derecha corresponde a que el usuario no desea guardar las muestras.





En cuanto al evento [1] su diagrama se observa en el Diagrama General del analizador. El evento [1] es sensible a la señal “TRIGGER” y se ejecuta cada vez que hay un cambio en el botón “TRIGGER” del panel.

En el evento [1] se ejecuta una estructura de tipo “Case”; si el caso es “True” (que corresponde a las opciones “Analizador” y “Aplicar”) se ejecuta la prueba y la captura, mediante una estructura tipo “Sequence” o secuencia de 3 cuadros: el cuadro 0 corresponde a la ejecución de *playxsvfsi* (este cuadro se ve en el Diagrama General del Analizador); el cuadro 1 es la ejecución de la aplicación *muestreo* y el cuadro 2 es el archivo muestras.txt creado en el paso anterior que contiene los datos que se visualizan, como se ve en las dos figuras siguientes:

Si el caso es “False” (corresponde a la opción “Abrir”), no se ejecuta la prueba sino que se abre una ventana de diálogo para seleccionar el archivo de muestras que se va a visualizar.

Al finalizar la ejecución de esta estructura de tipo “Event”, se ha creado o seleccionado el archivo de las muestras que se van a visualizar, por lo que este archivo va a un sub- instrumento creado que se encarga de leer este archivo de muestras y generar los datos necesarios como entrada del indicador de tipo “Waveform Digital Graph”, que es el gráfico digital que se observa en el panel frontal con los estados de cada una de las señales, y que se llama “grafico” en el extremo derecho del lazo “While” en el Diagrama General del Analizador. De esta manera, se pueden visualizar los datos contenidos en el archivo de muestras seleccionado.

ANEXO G:

ARCHIVO SVF CREADO

TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET IDLE;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 5 TDI (07) ; -- EN ESTA LINEA EL ULTIMO PARÁMETRO DEPENDE DE LA FUNCIÓN A EJECUTAR: SI LA OPCIÓN ES “ANALIZADOR” ENTONCES LA INSTRUCCIÓN ES SAMPLE (CODIGO:01h) Y SI LA OPCIÓN ES “APLICAR” ENTONCES LA INSTRUCCIÓN ES INTEST (CODIGO: 07h)
SDR 1022 TDI (0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000010000008000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000) SMASK (000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000001048008009800000492000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000); -- ESTA LÍNEA SE REPITE TANTAS VECES COMO MUESTRAS SE VAYA A CAPTURAR. EL VALOR DE TDI Y DE SMASK DEPENDEN DE CADA DISEÑO PARTICULAR. SMASK ES IGUAL EN TODAS LAS LINEAS Y TDI ES TODO CEROS (CUANDO LA INSTRUCCIÓN SAMPLE) O DIFERENTE EN CADA MUESTRA SEGÚN LOS ESTÍMULOS APLICADOS (CUANDO LA INSTRUCCIÓN ES INTEST)
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET IDLE;

ANEXO H:

ARCHIVOS VHDL CORRESPONDIENTES A LAS PRUEBAS

Los archivos esquemáticos y en VHDL correspondientes a las pruebas realizadas se encuentran en la carpeta \Anexos\Anexo H del CD adjunto.

1. Inversor

*Archivo onoff.vhd

*Archivo negac.vhd

*Archivo bufer.sch

*Archivo onoff.lpc

2. Registro de 8 bits con carga paralela

*Archivo reg8bits.vhd

*Archivo contad.vhd

*Archivo bufer.sch

*Archivo reg8bits.lpc