INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

“DESARROLLO DE UNA HERRAMIENTA

PARA EL DISEÑO DE SISTEMAS RECONFIGURABLES”

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

ING. ISRAEL ROMAN PALACIOS

DIRECTOR DE TESIS: DR. LUIS ALFONSO VILLA VARGAS

CODIRECTOR DE TESIS: DR. MARCO ANTONIO RAMIREZ SALINAS

México D.F. Diciembre de 2009

SIP-14

INSTITUTO POLITECNICO NACIONAL

SECRETARIA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 10:00 horas del día 18 del mes de noviembre de 2009 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis de grado titulada:

"DESARROLLO DE UNA HERRAMIENTA PARA EL DISEÑO DE SISTEMAS RECONFIGURABLES"

ROMÁN	**PALACIOS**	**ISRAEL**
Apellido paterno	materno	nombre(s)

Con registro: **B 0 7 1 4 5 8**

aspirante al grado de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***SU APROBACIÓN DE LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Presidente

Dr. Luis Pastor Sánchez Fernández

Secretario

Dr. Marco Antonio Moreno Armendáriz

Primer vocal
(Director de tesis)

Dr. Luis Alfonso Villa Vargas

Segundo vocal
(Director de tesis)

Dr. Marco Antonio Ramírez Salinas

Tercer vocal

Dr. Miguel Ángel Alemán Arce

Suplente

Dr. Víctor Hugo Ponce Ponce

EL PRESIDENTE DEL COLEGIO

Dr. Jaime Álvarez Gallegos

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION
EN COMPUTACION
DIRECCION

INSTITUTO POLITECNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESION DE DERECHOS

En la Ciudad de **México Distrito Federal** el día **30** del mes **11** del año **2009**, el que suscribe **Israel Roman Palacios** alumno del Programa de **Maestría en Ciencias de la Computación** con número de registro **B071458**, adscrito al **Centro de Investigación en Computación**, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección del **Dr. Luis Alfonso Villa Vargas y el Dr. Marco Antonio Ramírez Salinas** y cede los derechos del trabajo intitulado **"Desarrollo de una Herramienta para el Diseño de Sistemas Reconfigurables"**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección **iroman@ipn.mx**, **lvilla@cic.ipn.mx**, o a la **mars@cic.ipn.mx**. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Israel Roman Palacios

RESUMEN

En el año de 1960 Gerald Estrin publicó un artículo que dio vida a un nuevo campo de investigación, el Cómputo Reconfigurable. Describió la arquitectura de una máquina conocida como "*Fixed-Plus*" la cual demostró tener un gran potencial y desempeño contra las arquitecturas basadas en el modelo de Von Neumann. Pero debido a la tecnología de la época no fue posible plantear una implementación completa de la máquina. Desde entonces, solo un pequeño grupo de investigadores siguieron trabajando sobre la idea original de Estrin. No fue sino hasta la aparición de los FPGA cuando la idea empezó a retomarse con mayor fuerza, y con el incremento de las capacidades de los FPGA un gran número de grupos de investigación trabajan sobre este campo hoy en día.

El principal uso que se le ha dado a la idea de Estrin ha sido el desarrollo de microprocesadores híbridos, un microprocesador de propósito general con lógica reconfigurable alrededor para generar coprocesadores en tiempo de ejecución y tratar de otorgar una nueva tendencia en el diseño de microprocesadores.

La presente tesis trata al cómputo reconfigurable de una manera más general, el objetivo de la tesis no es generar microprocesadores híbridos sino un nuevo esquema donde a partir de modelos generales basados en lógica reconfigurable diseñar modelos específicos para tareas específicas, estos modelos deberán de estar basados en módulos. Así mismo, se plantea el uso de lógica reconfigurable para la implementación de sistemas basados en esta tecnología.

Las principales aportaciones del presente trabajo se ven reflejadas en la reducción de tiempo necesario para la generación de sistemas basados en módulos, y la facilidad con que los módulos pueden ser manejados para ir construyendo nuevos sistema.

ABSTRACT

It was 1960 when Gerald Estrin published a paper which actually give birth to a new investigation field, Reconfigurable Computing. It describes a novel architecture known as “Fixed-Plus” machine which demonstrated a greater potential and performance than those machine based on the classical Von Newmann architecture. However this potential the technology available in those years make impossible a complete implementation of Estrin’s idea. Since then, just a few groups of investigation were working on Estrin’s idea. With FPGA this field of investigation recovers interest and a big number of investigation groups have been working developing new and better models.

The main usage for this technology have been the develop of hybrid microprocessors, a general purpose microprocessor tied to reconfigurable logic to produce coprocessors in run time execution in order to create a new way to design microprocessors.

This work treat the reconfigurable computing in a more general way, the main objective is not to generate hybrid microprocessors but a new scheme to design application specific circuits through reconfigurable models. The use of reconfigurable computing in order to implement systems based on this technology is studied.

The main contributions of this work are the reduction of the necessary time to generate module based systems and the simplicity that this modules are used to make new systems.

AGRADECIMIENTOS

A Dios, por darme la oportunidad de realizar una meta más en mi vida.

A mi familia, quienes siempre me han apoyado.

A mi novia, Teresa Amara, por su infinito apoyo, inspiración e impulso para lograr esta meta.

A mi Asesores, el Dr. Luis A. Villa y el Dr. Marco A. Ramírez, por su paciencia, mil gracias.

Al Instituto Politécnico Nacional, por darme los recursos y la oportunidad de realizar una más de mis metas.

Al Centro de Investigación en Computación, por proporcionarme el espacio y los medios para lograr finalizar mi tesis.

A mis amigos Miguel Angel, Julio, Eduardo, Adriel, Omar, Jaime, Cecilia y muy especialmente a Alan por su apoyo.

A todas esas personas que han hecho posible de una u otra manera el que hoy pueda llegar a finalizar una etapa más de mi vida.

Gracias al Instituto de Ciencia y Tecnología (ICyT DF) del Distrito federal ya que con su apoyo, a través del proyecto SIP/DF/2007/143, obtuvimos la infraestructura requerida para el desarrollo de esta tesis.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO 1

INTRODUCCIÓN

Desde su aparición, los Arreglos de Compuertas Programables por Campo (**FPGA**) a principios de los años 90 han sido el tema de numerosos trabajos de investigación; en muchos casos son usados como una forma fácil y rápida de hacer prototipos, además de ofrecer un mecanismo de bajo costo en aplicaciones de bajo volumen de producción. La principal razón de esto es la facilidad con la que un diseñador puede hacer un sistema digital. La flexibilidad de estos dispositivos viene acompañada de una reducción en el desempeño, por lo que un FPGA provee menor funcionalidad por área de silicio y un mayor tiempo de propagación que las aplicaciones en Circuitos Integrados de Aplicación Específica (**ASIC**). En un diseño tradicional, donde la configuración del FPGA no cambia, la flexibilidad es desperdiciada, mientras que los principales problemas continúan presentes, "*Un uso eficiente de los FPGA requiere un empleo directo de ésta flexibilidad; Una forma de explotarla es a través de la reconfiguración en tiempo de ejecución del dispositivo".*

Esta reconfiguración es un tema importante en el diseño de sistemas digitales debido al incremento en la demanda de silicio, actualizaciones del dispositivo y habilidad para corregir errores, esto es logrado adicionando lógica reconfigurable al sistema. El resultado es un sistema heterogéneo que tiene la ventaja de la lógica reconfigurable y los elementos de procesamiento típicos como los Procesadores de Propósito General (**GPP**) y los ASIC. Esta combinación permite a ciertas áreas del sistema ser reconfigurado en tiempo de ejecución mientras el resto sigue operando normalmente. Esta habilidad es conocida como Reconfiguración en Tiempo de Ejecución o Reconfiguración Parcial Dinámica (**RTR**).

Los sistemas digitales de hoy en día requieren mayores recursos de hardware, esto significa que necesitamos un mayor número de circuitos integrados o circuitos integrados de mayor tamaño para una aplicación específica, lo cual por ejemplo, no es una solución óptima en dispositivos móviles debido a sus dimensiones limitadas. Con la RTR, las tareas que no se traslapan en el dominio del tiempo y del espacio pueden ser mapeadas en la misma lógica reconfigurable, siendo la tarea inicial la primera en mapearse y ejecutarse, cuando ésta concluye, la siguiente tarea es mapeada para su ejecución, y así sucesivamente. Por ejemplo, en un teléfono celular capaz de soportar diferentes tecnologías de comunicación como GSM, WCDMA, WLAN y WiMax, sería posible una implementación en lógica reconfigurable y cada una de ellas ser mapeada cuando se necesite, lo que representa un ahorro en el área ocupada del dispositivo.

1.1 Planteamiento del Problema

Diseñar un sistema que facilité el diseño de sistemas digitales a partir de sistemas basados en lógica reconfigurable, pudiendo diseñar sistemas digitales que no varíen en el tiempo a partir de sistemas reconfigurables.

1.2 Justificación

El diseño de este tipo de sistemas representa un nuevo reto para la investigación en ésta área: *El poder controlar la aplicación de la reconfiguración parcial dinámica sobre un sistema compuesto por módulos de distinta naturaleza teniendo en cuenta que debe garantizar su funcionamiento mientras se modifica una única sección.*

1.3 Objetivo General

Diseñar un sistema de control de reconfiguración parcial dinámica sobre sistemas basados en componentes que permita una modificación segura y eficiente de los mismos, además de proveer un medio para diseñar sistemas digitales basados en esta tecnología.

1.4 Objetivos Particulares

- Diseñar circuitos de pruebas basados en la reconfiguración dinámica parcial.
- Construir un medio para controlar la configuración del FPGA.
- Analizar y diseñar las principales opciones que deberá de tener un sistema que permita la administración de proyectos basados en reconfiguración dinámica parcial.
- Desarrollar un sistema que permita la administración de los sistemas basados en reconfiguración dinámica parcial.

1.5 Propuesta de Solución

Se propone desarrollar una herramienta de que nos permita manipular los módulos de un sistema reconfigurable de una manera sencilla e intuitiva para poder armar esquemas que resuelvan tareas específicas.

1.6 Organización de la Tesis

La presente tesis se encuentra organizada de la siguiente manera; En el capítulo 1 se da una breve introducción al Cómputo Reconfigurable, estableciendo la idea general de los sistemas reconfigurables dinámicamente, por lo que se brinda la justificación del presente trabajo. También se describe el objetivo general de la tesis así como los objetivos particular que se persiguen.

En el capítulo 2 se da una explicación del Cómputo Reconfigurable, se brinda la historia de cómo se concibió la idea de la lógica reconfigurable, como fue evolucionando hasta llegar a los FPGAs actuales, se explican los métodos de programación de este tipo de sistemas así como el modelo sobre el que están basados.

En el capítulo 3 se realiza un análisis del hardware reconfigurable, las partes básicas del FPGA y de cómo estas interactúan. Se da una breve introducción a la reconfiguración en los FPGA de Xilinx, sobre los métodos existentes y la forma en la que estos son llevados a cabo.

En el capítulo 4 se explica cómo diseñar sistemas reconfigurables, a partir del flujo de diseño de Xilinx, se explican las técnicas de diseño y las fases que comprende este. Se da una breve explicación del uso de las herramientas empleadas para lograr el diseño de sistemas reconfigurables.

En el capítulo 5 primeramente se analiza el sistema capaz de satisfacer el objetivo de la presente tesis, con este diseño, se estudian las opciones para poder llevarlo a cabo, y se explica la forma en la que fue implementado el diseño propuesto. Se brindan ejemplos del uso del software desarrollado.

Finalmente en el capítulo 6 se dan las conclusiones y los trabajos a futuro derivados de la realización de este trabajo.

CAPITULO 2

CÓMPUTO RECONFIGURABLE

Los arquitectos de computadoras se han dado a la tarea de diseñar una arquitectura de propósito general que pueda ejecutar una amplia variedad de aplicaciones empleando paralelismo. Desafortunadamente, las arquitecturas de propósito general han demostrado tener un desempeño pobre al momento de lidiar con un gran número de aplicaciones, o cuando éstas implican un gran número de operaciones. El Cómputo Reconfigurable (**CR**) es un nuevo paradigma de computación en el cual se hace uso de lógica programable, también llamada lógica reconfigurable, para producir procesadores dedicados de tiempo limitado que reproduzcan la arquitectura requerida para la ejecución de una aplicación específica.

2.1 Antecedentes

Gerald Estrin, introdujo el concepto de CR (1), con el desarrollo de su máquina fix-plus, el paradigma del CR que definió fue el siguiente; "*La ganancia de la velocidad de cómputo en una gran variedad de tareas es incrementada cuando son ejecutadas en una configuración apropiada de la estructura de la computadora orientándola a un problema. Una forma de lograr esto es mediante un sistema basado en la utilización del mismo hardware en una gran variedad de estructuras de propósito específico. Esta capacidad es alcanzada mediante programación o reestructuración física de un segmento del hardware.*"

Para implementar su visión, Estrin diseñó un sistema computacional al cual nombró la máquina Fix-plus [40]. Al igual que muchos sistemas reconfigurables de hoy día, la máquina Fix-plus consta de tres secciones, como se muestra en la figura 2.1, los cuales son:

- Sección Fija (**F**). Puede ser implementado mediante un procesador de propósito general o un circuito de aplicación específica.

- Sección Variable (**V**). Consiste de varias subestructuras digitales de diferentes tamaños, las cuales pueden ser reorganizadas de diferentes formas dependiendo del problema.

- El Control Supervisor (**SC**). Coordina las operaciones entre la sección fija y las secciones variables.

La sección variable es formada mediante un conjunto de unidades funcionales orientadas a problemas específicos, como por ejemplo, funciones trigonométricas, logarítmicas, exponenciales, raíces, aritmética de números complejos, hiperbólicas, operaciones entre matrices, etc., esto se conseguía mediante módulos que eran insertados manualmente en una tarjeta madre, logrando así dar la funcionalidad deseada. Las interconexiones entre los módulos eran controladas mediante el cableado de la tarjeta madre.

La máquina Fix-plus fue desarrollada para acelerar el cálculo de eigenvalores en matrices, y mostró tener una ganancia en la velocidad de 2.5 a 1000 veces sobre una IBM7090. Sin embargo debido a la tecnología de la época, resulto difícil el uso de la misma. La reconfiguración debía ser manual y se tuvo que desarrollar software nuevo para poder emplearla.

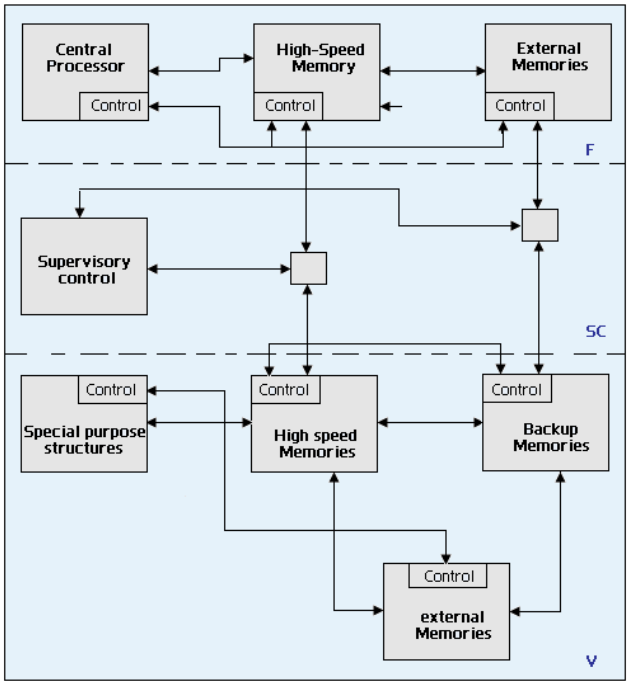


Figura 2.1. Estructura de la Máquina de Estrin.

En 1977, Franz J. Rammig, propuso el concepto de edición de hardware. El objetivo de su investigación era "*Desarrollar un sistema que sin interferencia mecánica o humana permitiera la construcción, cambios, procesamiento y destrucción de hardware digital real*".

Rammig materializó su concepto mediante el desarrollo de un editor de hardware similar a la arquitectura de los FPGA de hoy día. El editor fue construido sobre un conjunto de módulos y pines además de una función que mapeaba los pines. La circuitería de una función dada fue definida mediante una cadena de un alfabeto de dos letras, w = "wired/alambrado" y u = "unwired/no alambrado". Para construir el editor, se emplearon selectores que conectaban las entradas y las salidas de los módulos. El modelo de la máquina de Rammig se muestra en la figura 2.2.

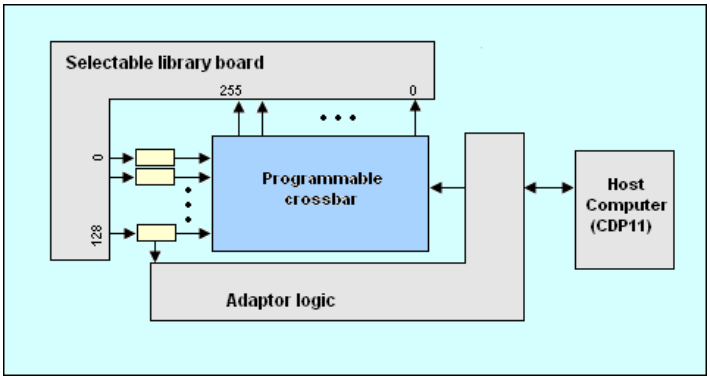


Figura 2.2. Estructura de la Máquina de Rammig.

2.2 Sistemas Reconfigurables

Los procesadores empleados en computadoras personales, estaciones de trabajo, servidores, etc., son flexibles debido a la versatilidad del conjunto de instrucciones empleado para la implementación de cualquier tarea. En general emplean una arquitectura rígida que les permite implementar las tareas mediante operaciones atómicas, las cuales, por ejemplo, son suministradas por una Unidad Aritmético Lógica (**ALU**), una unidad de punto flotante, etc. Los ASIC, por otro lado, son circuitos de hardware dedicados orientados a un número reducido de tareas o incluso dedicados a una sola tarea, las cuales son ejecutadas rápidamente, requieren menos componentes electrónicos y son más eficientes en el consumo de energía que los procesadores de propósito general. La principal desventaja en este tipo de componentes es la falta de flexibilidad ya que son trajes a la medida para las soluciones para las que están enfocadas.

En los últimos años una nueva clase de dispositivos ha emergido, los dispositivos de CR [39], los cuales prometen combinar la flexibilidad de los procesadores con la eficiencia de los ASIC. El hardware en estos dispositivos no es estático sino adaptable a cada aplicación individual; por lo que, se propone alcanzar la flexibilidad de los microprocesadores, haciendo uso de la eficiencia que brindan los ASIC. En la figura 2.3 se muestra cómo los sistemas digitales, basados en dispositivos reconfigurables, se sitúan en una zona

intermedia en la relación flexibilidad-desempeño. No son tan flexibles como un procesador de propósito general ni tan específicos, como un ASIC. Sin embargo se benefician de las características positivas de ambos.

Las diferencias más importantes entre la lógica reconfigurable y el procesamiento convencional se pueden resumir en los siguientes aspectos según K. Bondalapati y V. K. Prasanna[10]:

- **Computación espacial**. El procesamiento de los datos se realiza distribuyendo las computaciones de forma espacial, en contraste con el procesamiento secuencial.
- **Ruta de datos configurable (datapath)**. Empleando un mecanismo de configuración es posible cambiar la funcionalidad de las unidades de computación y de la red de interconexión.
- **Control distribuido**: Las unidades de computación procesan datos de forma local en vez de estar gobernados por una única instrucción.
- **Recursos distribuidos**: Los elementos requeridos para la computación se encuentran distribuidos por todo el dispositivo, en contraste con una única localización.

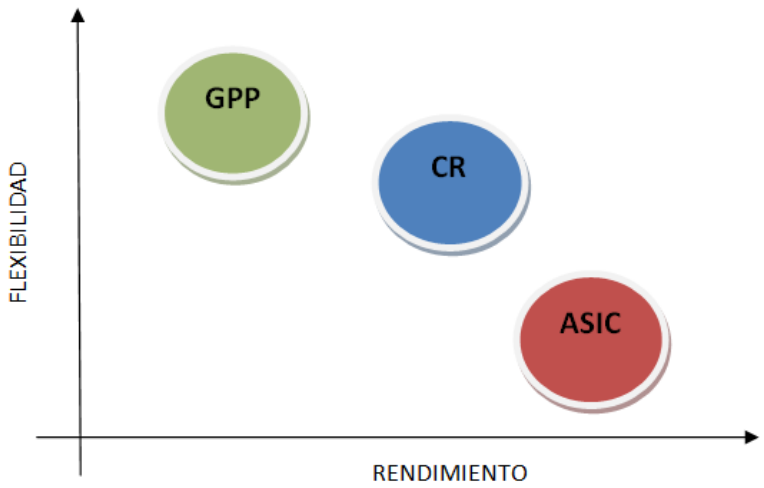


Figura 2.3. Sistemas de Computación según el Grado de Flexibilidad.

Otras características importantes de estos sistemas destacadas por A. DeHon [11] y R. Tessier [12] son la capacidad de adaptación, la posibilidad de configuración en tiempo de ejecución y la especialización. Beneficiándose de estas características específicas se han

desarrollado sistemas reconfigurables eficientes para aplicaciones como la programación genética [13], detección de patrones de texto [14, 15], criptografía [16, 17, 18], compresión de datos [19] o procesamiento de imágenes [20, 21] entre otras.

2.3 Modelo de un Sistema Reconfigurable

Una unidad reconfigurable puede ser considerado como un bloque funcional *F* que realiza diferentes tareas en tiempos diferentes. En cualquier punto del tiempo el comportamiento de este bloque está definido por un autómata finito de un conjunto dado $\{S_0, \ldots, S_r\}$ r Є N, un estado inicial S_0, un conjunto de estados finales A, un alfabeto de entrada y una función de transición, por lo que puede ser visto como un autómata finito determinista, en la figura 2.4 se puede ver un ejemplo del modelo.

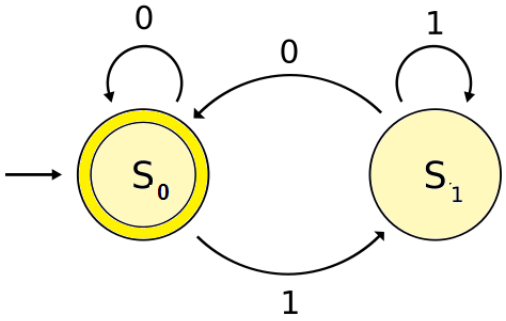


Figura 2.4. Modelo de un Sistema Reconfigurable.

2.4 Programación en Cómputo Reconfigurable

Existen básicamente dos formas de programar los dispositivos de CR: programación estático o en tiempo de compilación y programación dinámica o en tiempo de ejecución [28, 29].

2.4.1 Programación Estática o en Tiempo de Compilación (CTR)

Es la manera más simple y común de implementar aplicaciones con lógica reconfigurable. Su principal característica radica en que una vez que los dispositivos han sido configurados, se mantiene ésta configuración hasta que la tarea que se encuentra en ejecución termina. En la figura 2.5 podemos observar un diagrama de este tipo de configuración. El cual es muy similar a la de emplear ASIC's para la aceleración de aplicaciones. Desde el punto de vista de la aplicación, a esta no le importa si está siendo ejecutada en un ASIC o en un FPGA debido a que la configuración se mantiene a lo largo de la tarea.

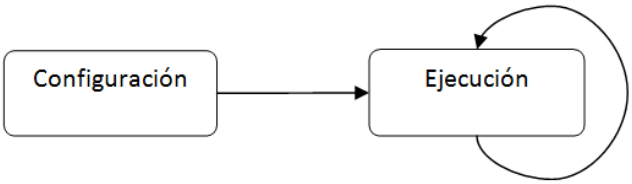


Figura 2.5. Compilación en tiempo de compilación.

2.4.2 Programación Dinámica o en Tiempo de Ejecución (RTR)

La configuración del dispositivo puede cambiar en cualquier momento, incluso si la tarea que se estaba ejecutando aún no termina. Un diagrama a bloques de éste tipo de configuración puede ser observado en la figura 2.6. Hace uso de esquemas de recolocación de los circuitos empleados por la tarea a lo largo del tiempo de ejecución. Las aplicaciones son configuradas múltiples veces durante la operación normal de la tarea.

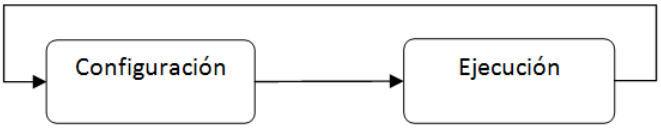


Figura 2.6. Compilación en tiempo de ejecución.

La naturaleza dinámica del hardware en RTR introduce dos nuevos problemas de diseño: El primero es seccionar la aplicación en segmentos que no necesiten ser ejecutados secuencialmente, esto se logra haciendo una división temporal en distintas fases de operación las cuales puedan ser ejecutadas independientemente de las otras fases, es decir, se logra concurrencia en la aplicación; El segundo problema de diseño se deriva del primero, y es acerca de la coordinación de los diferentes segmentos o fases de operación en la que fue dividida la aplicación, lo cual ocurre debido a que algunas fases de operación entregan datos que deben ser administrados a otras fases lo que implica mantener un canal de comunicación entre los segmentos.

La configuración que realiza una aplicación RTR debe ser organizada de tal forma que los módulos que la conforman permanezcan el tiempo suficiente para concluir su tarea. Además, esta tarea debe de ser relativamente independiente de los demás módulos. Debido a que las herramientas de diseño digital asumen modelos de hardware estático, la depuración de este tipo de circuitos se reduce a un proceso de prueba y error.

Un segundo problema de diseño es la coordinación de los diferentes módulos que conforman la aplicación, esto se debe a establecer los mecanismos de comunicación inter-módulos para poder pasar los datos intermedios sin perder consistencia en la información.

Esto representa un gran esfuerzo ya que se debe de garantizar la integridad de los datos aún cuando el módulo ya no se encuentre configurado en el dispositivo.

La reconfiguración dinámica tiene diversas variantes según el modo en el que ésta se aplique. Las tres más representativas son; Reconfiguración en contexto único, reconfiguración multi-contexto, reconfiguración parcial.

2.4.3 Reconfiguración en Contexto Único

Corresponde con el modo de configuración de los FPGAs que únicamente soportan un acceso secuencial a la memoria de configuración. En el caso de realizar una reconfiguración dinámica con estos dispositivos se sufren penalizaciones temporales importantes debido a que cada intercambio de funcionalidad requiere una reprogramación completa de los mismos. El modelo de este modo se representa en las figuras 2.7 (a). La configuración entrante sustituye completamente a la que estaba aplicada sobre la lógica configurable.

2.4.4 Reconfiguración Multi-Contexto

Los dispositivos que soportan este tipo de reconfiguración tienen varios bits de memoria de configuración para cada bit de los elementos configurables [33, 34]. En la figura 2.7 (b) se representa un modelo de estos dispositivos donde los bits de memoria pueden considerarse como múltiples planos de información de configuración. Cada plano debe configurarse totalmente, de igual forma que los de contexto único. Sin embargo, el cambio entre contextos se realiza de forma muy rápida, admitiéndose además la carga de una nueva configuración en un plano no activo mientras otro lo está.

2.4.5 Reconfiguración Parcial

Uno de los avances tecnológicos más importantes en el área de la reconfiguración consiste en la capacidad de algunos dispositivos para admitir la modificación de parte de la configuración mientras el resto del hardware sigue realizando la computación de forma ininterrumpida. La figuras 2.7(c) muestran este modelo de reconfiguración. En este caso el plano de configuración funciona como una memoria RAM. De este modo se pueden emplear las direcciones para especificar una determinada localización que de desea reconfigurar. La reconfiguración parcial dinámica también permite que se carguen configuraciones diferentes en áreas no usadas del dispositivo con el fin de reducir la latencia en el cambio de contexto, tal y como se propone en el trabajo de K. Danne [35].

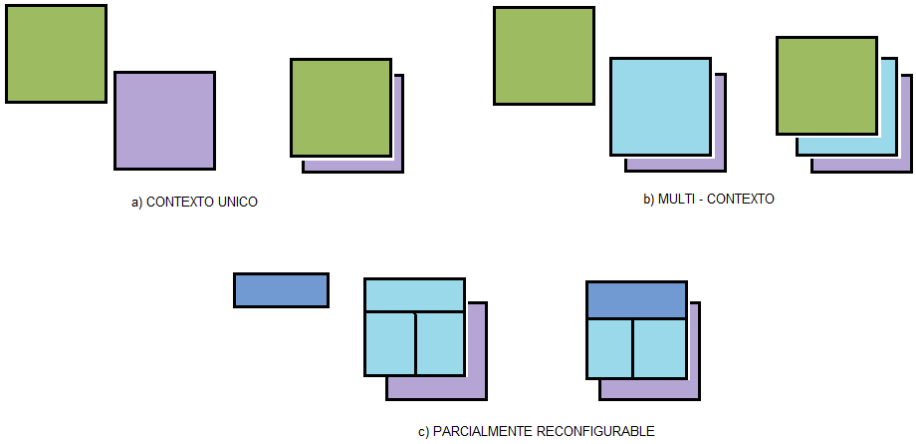


Figura 2.7. Modelos de Reconfiguración.

Una variante de la reconfiguración parcial realizada con el fin de reducir la penalización por el tiempo necesario para la carga de los *bitstreams* parciales es la *reconfiguración pipeline*. En este caso, la reconfiguración ocurre en incrementos de etapas de *pipeline* [36]. Este sistema está orientado a sistemas de estilo *datapath*, donde se emplean más etapas *pipeline* que las que caben simultáneamente en el hardware. En este caso la configuración de cada etapa se situaría delante del flujo de datos [11].

CAPITULO 3

HARDWARE RECONFIGURABLE

EL CR trata de llenar el espacio que existe entre el hardware y el software, alcanzando un mayor desempeño que el software, pero manteniendo un mayor nivel de flexibilidad que el hardware tradicional. Este tipo de cómputo se basa principalmente en FPGAs; los cuales contienen un arreglo de elementos básicos cuya funcionalidad es dictada por una configuración de bits en una Memoria RAM Estática (**SRAM**). Estos elementos básicos también conocidos como bloques lógicos, son conectados empleando recursos de ruteo que al igual que ellos son programables. De esta manera, los circuitos pueden ser implementados en el FPGA mapeando las funciones en los bloques lógicos y configurando la red de ruteo para conectar estos bloques hasta lograr formar el circuito deseado.

3.1 Antecedentes

Durante los años 80 y principios de los 90, los circuitos reconfigurables más utilizados fueron las Matrices de Lógica Programable (**PAL**) y los Dispositivos Lógicos Programables Complejos (**CPLD**), los cuales son dispositivos reconfigurables de baja capacidad. La funcionalidad del circuito se diseña y modifica desde un software en la PC del diseñador. La configuración resultante se descarga en el dispositivo mediante medios sencillos. Este mecanismo supuso una clara ventaja para el diseño de prototipos y pequeños volúmenes de producción respecto a la utilización de ASIC's.

La utilidad de estos dispositivos de baja capacidad no se centró únicamente en la elaboración de prototipos, sino que también se extendió a diseños industriales, básicamente aplicaciones de "*glue logic*"[1]; con ellos se permitía la agrupación de componentes con la consecuente reducción del área empleada en el circuito impreso; además, brindaban una cierta flexibilidad para corrección de errores y actualización del sistema sin necesidad de cambio en el circuito impreso, compensando de esta forma su sobrecoste respecto a la utilización de circuitos integrados.

A principio de los años 90 hacen su entrada en el mercado de los dispositivos reconfigurables los FPGAs con tecnología SRAM. Éstas disponían de una capacidad lógica

[1] El término de *glue logic* se denomina al conjunto de circuitos auxiliares necesarios en un sistema digital para adaptarlo a una configuración concreta.

mayor que los CPLD's y las PAL's y podían ser usados para procesamientos más complejos que un simple *glue logic*, por ejemplo: pre-procesamiento para comunicaciones digitales y algoritmos para visión artificial. Sin embargo, la evolución de los FPGA no acabó ahí. Al igual que ocurre con los ASIC, donde la capacidad de integración en éstos se incrementa de forma sorprendente año a año, lo que ha permitido que en la actualidad se integren sistemas digitales completos en un ASIC surgiendo los System-on-Chip (**SoC**), también afecta a los FPGA y otros dispositivos reconfigurables. Actualmente los FPGA disponen de capacidades de integración de millones de compuertas lógicas equivalentes, lo que permite, al igual que con los ASIC, integrar sistemas digitales completos en un único dispositivo FPGA, definiéndose este caso como System-on-Programmable-Chip (**SoPC**).

En comparación con los SoC basados en tecnología ASIC, los SoPC tienen desventajas en cuanto a precio y eficiencia puesto que no se tratan de circuitos diseñados específicamente para una determinada aplicación. Sin embargo, en muchos casos compensan esta situación con su flexibilidad y accesibilidad para los grupos de investigación, así como para las pequeñas y medianas empresas. El avance tecnológico que ha permitido la integración de sistemas en un único dispositivo ha ido acompañado de diversas terminologías para designar a éstos sistemas, sin embargo aún no se encuentra normalizada. A continuación se detallan algunos de estos términos:

- **System-on-Chip (SoC).** Circuito integrado formado por diversos módulos VLSI con distinta funcionalidad, que interconectados entre sí, ofrecen toda o casi toda la funcionalidad específica para una aplicación.

- **System-on-Programable-Chip (SoPC).** Se aplica este término específicamente cuando el dispositivo utilizado para realizar el sistema es un FPGA. En los SoPC no se utiliza la capacidad de reconfiguración dinámica que puedan disponer los FPGAs, sino únicamente las facilidades que ofrecen estos dispositivos en la fase de desarrollo y posteriores actualizaciones del sistema.

- **Configurable-System-On-Programable-Chip (CSoPC)**. Mediante éste término se definen los sistemas SoPC en los que se hace uso de la capacidad de reconfiguración de los mismos, para aplicaciones de CR. Pueden incluirse bajo la denominación CSoPC tanto los sistemas que admiten diferente configuración estática según ciertas condiciones, como los que utilizan la reconfiguración parcial dinámica para modificar en tiempo de ejecución una sección hardware.

- **Multiprocessor-Configurable-System-On-Programable-Chip (MCSoPC)**. Se aplica esta definición a los sistemas CSoPC que incluyen varios procesadores que ejecutan software, funcionando de forma simultánea.

La necesidad de controlar la complejidad generada por este nivel de integración ha incorporado nuevas metodologías basadas en el diseño a partir de módulos, como se muestra en la figura 3.1, con la reutilización de componentes prediseñados, denominados IPCores, cores o simplemente módulos, reutilizando incluso arquitecturas predefinidas. Aún así, la complejidad del diseño de un SoC, SoPC, CSoPC o MCSoPC es mayor respecto a la del diseño convencional de sistemas distribuidos en circuitos impresos y en distintos circuitos integrados, pero aporta numerosas ventajas como son:

- Reducción del costo del producto, ya que todo el sistema se encuentra alojado en un solo circuito integrado, en el caso de dispositivos con lógica reconfigurable integrada la reducción del área de silicio empleada también es factor de la reducción de costos.

- Incremento del rendimiento del sistema. Puesto que al integrar los buses en un dispositivo se disminuyen las líneas de interconexión existentes en el circuito impreso y además se reduce considerablemente la longitud de las mismas lo que permite velocidades mucho mayores de comunicación.

- Descenso significativo del consumo de energía.

- Reducción global de la superficie de silicio.

En cuanto a los módulos, se diferencian tres tipos en función de su flexibilidad para su implementación:

- **Soft cores**. Estos módulos están descritos en Lenguajes de Descripción Hardware (**HDL**), por ejemplo, VHDL o Verilog. El código representa entidades de hardware sintetizables. Estos módulos son flexibles y relativamente independientes de la tecnología, de manera tal que se pueden sintetizar para distintos dispositivos, incluso para emplearse en un ASIC o en diferentes dispositivos de lógica programable.

- **Firm cores**. La descripción de los mismos se realiza mediante información a nivel de compuertas optimizada en tamaño y rendimiento para un dispositivo o familias de dispositivos en concreto. Por tanto la asignación de recursos del dispositivo para realizar esas tareas ya está realizada antes de su utilización por parte del diseñador, restando únicamente la fase de interconexión que se realizará en la implementación. No son muy flexibles pero permiten la protección de la Propiedad Intelectual y en general son muy eficientes.

- **Hard cores**. En el caso del diseño de un ASIC este tipo de bloques son módulos preparados para ser integrados en el silicio del dispositivo. Disponen además de la información de los recursos necesarios a bajo nivel que hay que implementar por ejemplo: la información del ruteado exacto. Están optimizados en aspectos tales como rendimiento, área o consumo de energía. En el caso de dispositivos lógicos programables, son módulos ya integrados en el silicio del dispositivo.

Los Soft cores admiten mayor grado de flexibilidad mientras que los Hard cores resuelven las necesidades más complejas (procesadores avanzados, multiplicadores de alto rendimiento, etc.) y se encuentran mejor optimizados. El diseño de un SoC, SoPC, CSoPC o MCSoPC agrupa y mezcla disciplinas de diseño de hardware, software, telecomunicaciones, etc. éste nuevo concepto de diseño crea un entorno convergente que abre las puertas a innovadores productos, mercados, competidores, tecnologías y servicios [4]. Los participantes en éste entorno son los fabricantes de módulos y de circuitos integrados; los diseñadores y verificadores de sistemas; los desarrolladores de sistemas operativos y de aplicaciones, así como las empresas dedicadas a las herramientas de desarrollo que permitan esta evolución.

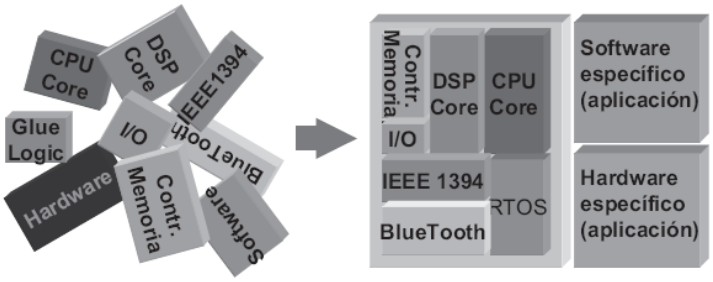


Figura 3.1. Diseño de Sistemas Basados en Bloques.

Con el fin de guiar al diseñador en la tarea de seleccionar una arquitectura para su sistema se ha evolucionado hacia un modo de diseño basado en módulos más restringido denominado diseño basado en plataformas [6], el cual consiste en que el fabricante de los dispositivos proponga al diseñador el uso de una especificación concreta para la interconexión de los módulos y determinadas topologías de bus. Generalmente, estas propuestas están condicionadas por el microprocesador que integre el fabricante en los mismos. Los fabricantes de lógica programable proponen sus plataformas para desarrollar sistemas integrados en un chip empleando sus FPGA de alta capacidad.

Estas plataformas SoPC están resultando una oportunidad dentro del área de los SoC [6]. En la actualidad, sólo un pequeño grupo de grandes multinacionales se aventura a diseñar SoC sobre ASIC. En contraste, con los SoPC's basados en plataformas han puesto al alcance de toda la comunidad diseñadora de sistemas electrónicos la posibilidad de realizar

diseños integrando sus desarrollos específicos y reutilizando los ya realizados por otras fuentes en un único dispositivo.

El CR implica un paso más en el aprovechamiento de las propias características de los dispositivos reconfigurables, es decir, la manipulación de la lógica configurada en el dispositivo durante el funcionamiento del sistema. Aunque el término se acuñó a finales de los 60 por investigadores de la Universidad de California [40], realmente la investigación en esta área se ha incrementado notablemente a finales de los años 90 y en la presente década. El nivel de integración de los dispositivos reconfigurables actuales más la posibilidad de modificar parcial y dinámicamente su configuración (característica disponible en algunos de ellos), han acelerado la investigación en este campo tanto a nivel teórico como en posibles aplicaciones.

A pesar de que estas plataformas son recientes ya existen diversos grupos de investigación que están proponiendo soluciones para controlar la reconfiguración parcial dinámica sobre este tipo de sistemas. La heterogeneidad y complejidad de los módulos actuales implican la integración de secciones de control dentro de los mismos. La forma más flexible de resolver dichas operaciones de control es utilizando microprocesadores. Muchos módulos integran pequeños microprocesadores para trabajar de forma conjunta con secciones dedicadas de hardware para procesamientos más intensos (también parte del mismo módulo). Por tanto, en un SoPC puede ser habitual disponer de diversos procesadores distribuidos en los diferentes módulos que conforman al sistema.

3.2 Tecnología Reconfigurable: FPGAs

Los FPGAs consisten básicamente de una matriz de Bloques Lógicos Configurables (**CLB**), una matriz de interconexión y Bloques de Entrada y Salida (**IOB**), en la figura 3.2 se puede ver la arquitectura de un FPGA estándar.

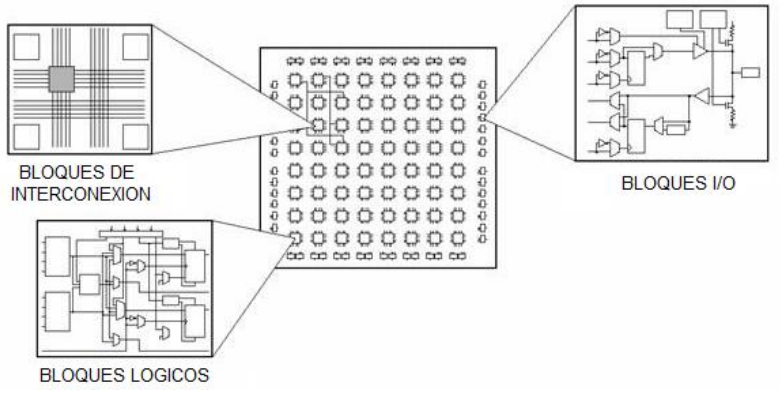


Figura 3.2. Modelo Genérico de un FPGA.

3.2.1 Bloques Lógicos Configurables

Los CLBs constan básicamente de una parte combinacional que permite implementar funciones lógicas más una parte secuencial formada por flip-flops, que permite sincronizar la salida con una señal de reloj externa, lo cual es útil para realizar circuitos secuenciales y la implementación de registros. La estructura de un bloque lógico varía de un fabricante a otro. Sin embargo, la lógica combinacional se basa principalmente en Tablas de Búsqueda o Look-Up Tables (**LUT**).

Una LUT es un componente de memoria que almacena una tabla de verdad. Las direcciones de la memoria son las entradas de la función lógica a implementar, y en cada localidad de dicha memoria se almacena el resultado de la combinación correspondiente de las entradas. En una LUT de n x 1 es posible implementar cualquier función lógica de n entradas. En la figura 3.3 se muestra la estructura básica de una LUT de n entradas. En la figura 3.4 podemos observar cómo es posible elegir entre un circuito combinacional y un circuito secuencial, esto mediante la configuración de un multiplexor para decidir hacia donde se dirigirá la salida por medio de un bit de configuración *BYPASS*.

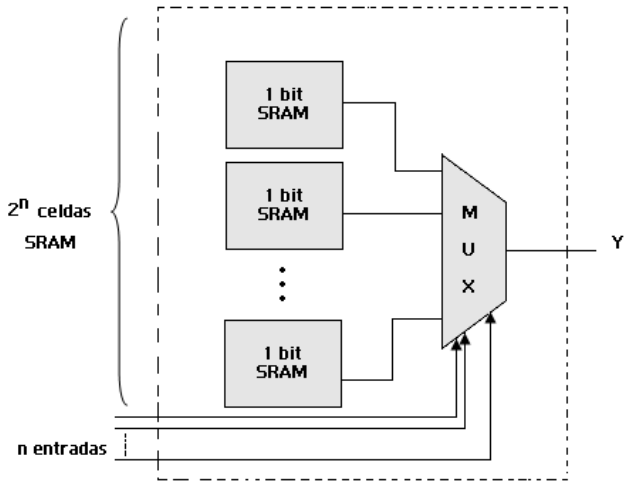


Figura 3.3. Estructura de una LUT de *n* Entradas.

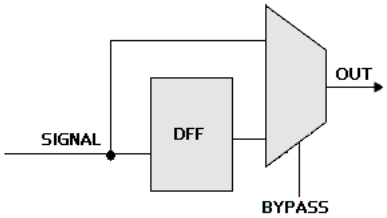


Figura 3.4. Flip – flop tipo D con Salida Opcional.

Por ejemplo, supóngase que se desea implementar la función lógica *Z = A AND B OR C*. Entonces, el contenido de una LUT de 3 entradas deberá ser el descrito en la tabla de verdad de la figura 3.5.

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figura 3.5. Tabla de Verdad de la Función Lógica *Z = A AND B OR C*.

Si se desea implementar funciones lógicas de más de n entradas se debe de interconectar más de una LUT para lograrlo. Resulta conveniente cuando se realizan circuitos combinacionales de varias entradas, separarlos en etapas por medio de registros, de forma tal que no se aumente el tiempo de propagación de las señales al utilizar varias LUT en cascada. A su vez, si se desea implementar registros o circuitos secuenciales, se deberá utilizar en general los flip-flops de más de un bloque lógico. La combinación de distintos bloques lógicos se realiza a través de los bloques de interconexión.

Generalmente la cantidad de entradas que posee una LUT no excede de 4 ó 5. Cuando se agrega una entrada, el número de celdas de memoria debe duplicarse con el consiguiente aumento de la lógica de selección (multiplexor). Esto produce que el tiempo de propagación de las señales dentro de la LUT aumente, disminuyendo la frecuencia máxima de operación. Actualmente la mayor parte de los dispositivos FPGA cuentan con LUTs de 4 entradas [43, 44].

Los distintos fabricantes de FPGAs llaman de diferentes formas a los componentes lógicos de un FPGA. El elemento básico de los FPGAs de Xilinx es la Celda Lógica (**LC**) [44]. Una celda lógica contiene una LUT, un multiplexor, lógica de acarreo, un registro y lógica de configuración. Este registro puede funcionar como flip-flop D o como un candado. La

polaridad del flanco de reloj puede ser configurada (ascendente o descendente) así como también la polaridad de las señales de habilitación del reloj, set y reset.

Altera, por su parte, llama a un elemento básico Elemento Lógico (**LE**) [43]. Un Bloque de Arreglos de Elementos Lógicos (**LAB**) se encuentra conformado por un conjunto de ocho a diez LE. Un LE difiere en su estructura interna de una LC; sin embargo, las funcionalidades de ambos bloques son similares.

3.2.2 Tecnología de Programación

Un punto a tener en cuenta es como se almacena la información correspondiente a las funciones lógicas que se desean implementar y las conexiones que deben hacerse entre bloques lógicos (tecnología de programación). A continuación se describen las tecnologías de programación más usadas en el mercado:

- **SRAM**. En éstas, la configuración de las LUT es cargado mediante un proceso de configuración en el momento de encendido del FPGA. Debido a que se basa en una SRAM el contenido de la memoria se pierde cuando se deja sin energía; la información de las celdas SRAM generalmente se almacena en memorias seriales EEPROM conocidas como memorias de configuración. En el momento de encendido del circuito, toda la información es transferida a los bloques e interconexiones del FPGA mediante el proceso de configuración el cual es generalmente automático, dado que el propio FPGA contiene un circuito interno que se encarga de efectuar la programación.

- **Antifusible** (Antifuse). Un FPGA que utiliza este tipo de tecnología sólo puede ser programado una vez. Se las llama antifusible porque a diferencia de un fusible, donde originalmente existe una conexión y mediante el paso de corriente dicha conexión es destruida, en este caso la conexión debe ser creada, es decir que no existe originalmente. Una vez que es programado no se puede recuperar el estado original de la conexión. Un antifusible consiste en dos líneas perpendiculares conductoras, normalmente de aluminio, separadas por una capa de dieléctrico, normalmente algún óxido. La programación se realiza estableciendo una tensión elevada entre dos líneas que se cruzan. Esta tensión es superior a la rigidez dieléctrica del óxido y el campo eléctrico lo rompe estableciendo un pequeño arco entre las dos pistas, que las funde parcialmente, soldándolas, creando así una conexión permanente.

- **Flash**. Los FPGA basados en celdas flash recogen las ventajas principales de las dos técnicas anteriores situándose en un punto intermedio. Su tamaño es bastante más reducido que el de una celda de SRAM, aunque sin llegar al tamaño reducido

de un antifusible; son reprogramables, aunque la velocidad de programación es bastante más lenta que en el caso de una SRAM; y son no volátiles, por lo que no necesitan un dispositivos auxiliar para guardar la configuración interna, como en el caso de la SRAM.

3.2.3 Bloques de Entrada/Salida

La función de un bloque de entrada/salida es permitir el paso de una señal hacia dentro o hacia el exterior del dispositivo. Por este motivo, dependiendo de la interfaz eléctrica, el mismo debe contar con recursos tales como:

- Salidas configurables como Tri-estado o de colector abierto.
- Entradas con posibilidad de pull-up o pull-down programables.
- Registros de salida.
- Registros de entrada.

Además, debido a que hoy en día los FPGA funcionan a altas velocidades de operación y que los mismos tienen desde cientos hasta miles de terminales de conexión, los bloques de entrada/salida deben ser capaces de proveer una adecuada terminación en cuanto a impedancia se refiere, y de tal forma evitar reflexiones de las señales. Esto que en el pasado se lograba conectando una resistencia cerca de la terminal del dispositivo, en la actualidad debe ser implementado dentro del propio FPGA debido a la gran cantidad de terminales que estos dispositivos poseen.

3.2.4 Recursos de Interconexión

Para utilizar las celdas lógicas, debe disponerse de una red de interconexión flexible. Sin embargo, debe considerarse que cuanto mayor flexibilidad tenga la red de interconexión mayor será el consumo de potencia y menor la velocidad de operación del dispositivo. En general, las conexiones internas entre elementos de un FPGA pueden realizarse básicamente de dos formas:

- Líneas directas. Tienen lugar entre bloques adyacentes.
- Conexiones de propósito general. Se realizan a través de una matriz de interconexión cuando se desean conectar dos bloques no adyacentes. Su misión es conectar canales verticales y horizontales que permiten llegar al punto final de la conexión. Cuanto más larga es la línea de ruteo, mayores serán los retrasos introducidos tanto por la propia longitud de la línea como por los elementos de interconexión utilizados a lo largo de la misma.

3.3 Configuración Dinámica en los FPGA de Xilinx

Los FPGA son dispositivos lógicos programables que permiten la implementación de sistemas digitales. Consisten en un conjunto de celdas lógicas que pueden ser configuradas para realizar una función específica mediante bitstreams de configuración. Algunos FPGA permiten la configuración parcial, donde un bitstream configura solamente un subconjunto de los componentes internos. La configuración parcial dinámica es lograda mientras el dispositivo esta activo, algunas áreas del dispositivo pueden ser configuradas mientras el resto no son afectadas y permanecen operacionales mientras se realiza la programación. Para las familias de FPGA de Xilinx Virtex 5, Virtex 4, Virtex II, Virtex II Pro, Spartan II y Spartan IIE existen dos flujos documentados para realizar la configuración dinámica: Basada en Módulos y Basada en Diferencias.

- ***Reconfiguración Basada en Diferencias.*** En este esquema de reconfiguración el diseñador debe de realizar cambios a bajo nivel del diseño, esto se logra mediante el "*FPGA Editor*", una herramienta de edición de bajo nivel, en la que el diseñador puede cambiar diferentes partes del FPGA como: las LUT, multiplexores, RAM interna, la inicialización de los flip-flops, y los valores de reset. Una vez editados estos cambios, un bitstream parcial es generado, conteniendo solamente las diferencias entre el antes y el después de los cambios. Para diseños complejos esta técnica no es recomendable debido a la gran cantidad de componentes a modificar. La presente tesis se basa en la reconfiguración basada en módulos, por lo que la reconfiguración basada en diferencias ya no será estudiada más en el presente trabajo.
- ***Reconfiguración Basada en Módulos.*** La Basada en Módulos permite al diseñador dividir el diseño completo en módulos. Para cada módulo el diseñador genera un bitstream de configuración a partir de un lenguaje HDL, ejecutando las etapas de síntesis y el mapeo y ruteo independientemente de los otros módulos. Un bitstream inicial completo debe de ser generado y después los bitstreams parciales para cada módulo. Para garantizar que cada vez que un módulo es reconfigurado los canales de comunicación entre módulos permanezcan sin cambios, unos módulos especiales llamados "*Bus Macro*" son implementados para mantener las conexiones entre los módulos correctas.

  El diseño completo debe de ser dividido en módulos y cada uno de estos debe de ser independiente. Si todos los módulos son completamente independientes, sin puertos de entrada salida comunes salvo las señales de reloj no hay necesidad de emplear Bus Macro ya que no hay comunicación entre los módulos. En otro caso, cuando existe la necesidad de comunicar dos o más módulos dentro del diseño, los Bus Macro deben ser empleados para que las señales puedan viajar a través de

las fronteras del área reconfigurable. El código HDL empleado debe de garantizar que cualquier señal empleada para comunicar dos módulos en el diseño, siendo al menos uno de ellos un módulo reconfigurable, debe de pasar mediante un Bus Macro. Cada vez que una reconfiguración parcial es hecha, los Bus Macro garantizan que los canales de comunicación entre los módulos funcionen correctamente.

Los módulos reconfigurables tienen las siguientes propiedades:

- Las áreas reconfigurables deben estar bien definidas dentro de la topología del FPGA.
- Una vez que se ha definido un región reconfigurable, ésta no puede ser alterada, es decir, no se pueden mover las fronteras de la región configurable.
- Los módulos reconfigurables se comunican con otros módulos, ya sean módulos fijos u otros módulos reconfigurables mediante los Bus Macro.
- La implementación debe de ser diseñada de tal forma que los otros módulos no quieran interactuar con un módulo en proceso de reconfiguración. La implementación debe de asegurar una operación correcta durante el proceso de reconfiguración.
- El número de *slices* que ocupan el área reconfigurable debe ser múltiplo de cuatro, con un mínimo de cuatro y un máximo del área total del dispositivo, en el caso del Virtex II Pro y FPGA anteriores a él, el alto de las áreas está definido por el alto del dispositivo; para los FPGA posteriores, el alto de las regiones está dado por el número de *slices* ocupados, por ejemplo, cuadrados de dos por dos slices, rectángulos de dos por cuatro, etc.
- Todos los recursos lógicos cubiertos por el área reconfigurable se considera como parte del área reconfigurable. Es decir, el área reconfigurable no se limita a los CLB sino también a las LUT, bloques de memoria RAM, multiplicadores, IOB, etc., así como todos los recursos de ruteo exceptuando las señales de reloj.

CAPITULO 4

DISEÑO DE SISTEMAS RECONFIGURABLES

La RTR es útil para aplicaciones que requieren cargar diferentes diseños en la misma área del dispositivo o la flexibilidad para cambiar parte de un diseño sin hacer un reset o reconfigurar completamente el dispositivo. Con esta capacidad, nuevas áreas de aplicación son posibles, en particular, actualizaciones de hardware y reconfiguración en tiempo de ejecución. Para las familias de FPGA de Xilinx Virtex 5, Virtex 4, Virtex II, Virtex II Pro, Spartan II y Spartan IIE, como se vio en el capítulo pasado existen dos flujos documentados para realizar la configuración dinámica: Basada en Diferencias y Basada en Módulos [41], a continuación se detalla el proceso de diseño de sistemas reconfigurables basados en módulos.

4.1 Herramientas de Desarrollo

Para la elaboración de esta tesis se emplearon las siguientes herramientas de desarrollo:

- **Xilinx ISEWebpack 9.2.04i_PR12**. Software de Xilinx para el diseño de circuitos digitales basados en lenguajes HDL. Se empleo para realizar la síntesis de los módulos del diseño de pruebas. Esta versión fue empleada debido a que es la última que soporta el diseño de sistemas reconfigurables con acceso al público en general previa autorización de Xilinx. Las versiones 10.X y 11.1 del ISE no soportan éste tipo de diseño. Para poder emplear este software es necesario instalar la versión base que es la 9.02i, disponible de la pagina de Xilinx, una vez instalado se deberá hacer la actualización a la versión 9.2.04i, hecho esto se procede a actualizar a la versión 9.2.04i_PR12 mediante un script de *Perl* que es proporcionado por Xilinx. En la figura 4.1 se muestra el flujo de instalación de los programas de diseño de Xilinx, el correspondiente a éste paquete es el ilustrado del lado izquierdo de la figura. En la figura 4.2 se muestra una pantalla de las variables de entorno del ISEWebpack, donde se muestra la versión del entorno, prueba de que la instalación fue correcta.

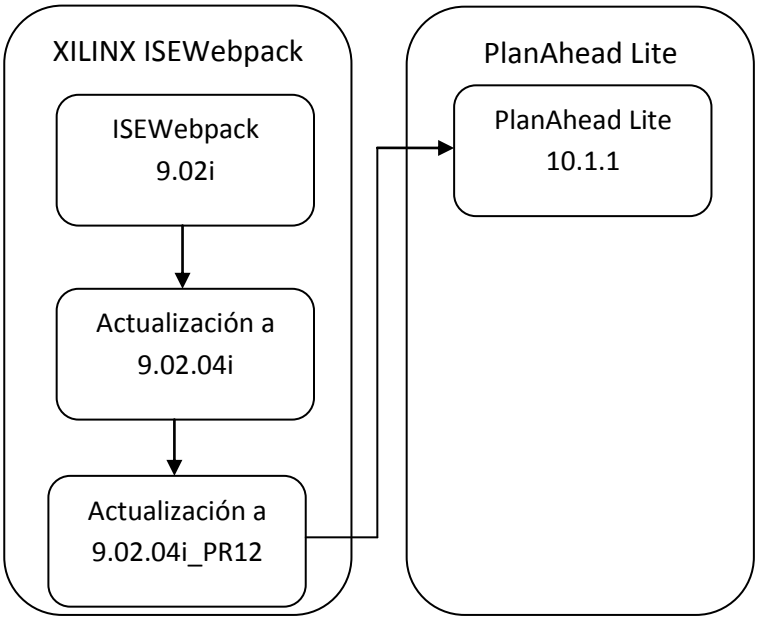


Figura 4.1. Flujo de Instalación de las Herramientas de Xilinx.

- **PlanAhead Lite 10.1.1.** Software de Xilinx para la administración del diseño de circuitos digitales basados en lenguajes HDL. Se empleo para realizar el mapeo y ruteo de los circuitos sintetizados por el ISEWebpack, además de hacer el armado de los archivos bitstream. Esta es la última versión del Plan Ahead como herramienta de diseño independiente dentro de la suite de diseño de Xilinx, a partir de la versión 11.1 son distribuidos juntos el Plan Ahead y el ISE aún en la versión libre (WebPack), de la misma forma que con el ISE la versión 11.1 no soporta este tipo de diseño. En este caso el software es simplemente instalado y no es necesario algún paso adicional para su uso. En la figura 4.1 se muestra el flujo de instalación de este paquete, el correspondiente al PlanAhead se muestra del lado derecho de la figura.

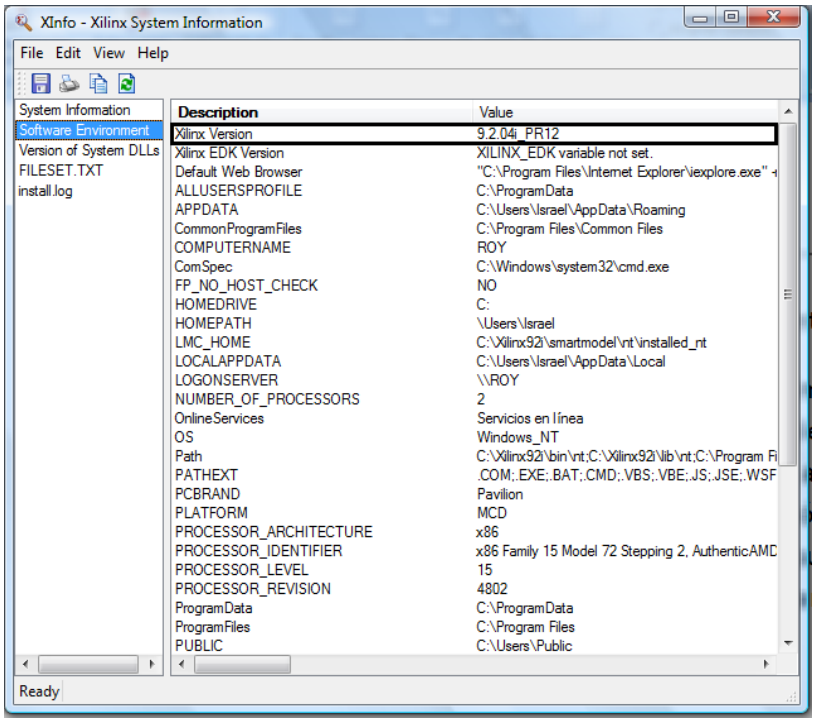


Figura 4.2. Variables de Xilinx una vez Completada la Instalación.

4.2 Flujo de Diseño para la Reconfiguración Basada en Módulos

La creación de un diseño parcialmente reconfigurable, donde los módulos se comunican entre sí, requiere la creación e implementación del diseño siguiendo un conjunto específico de reglas. En particular, el flujo de diseño para este tipo de reconfiguración se basa en el flujo de diseño modular [41], figura 4.3, con ciertas diferencias. La figura 4.4 muestra las etapas de diseño para sistemas reconfigurables. Para este flujo, el diseño de los módulos estáticos y dinámicos se debe implementar por separado, con un último paso de ensamble para generar los bitstreams completos y parciales. El proceso de generación de diseños para reconfiguración parcial basada en módulos se puede dividir en tres fases principales:

- ***Primer Fase (Initial Budgeting)*.** Se crea el archivo de mayor nivel, las restricciones del diseño, se realiza la simulación de los diferentes escenarios por separado.
- ***Segunda Fase (Active Module)*:** Se implementa cada módulo mediante el proceso de mapeo y ruteo.
- ***Tercer Fase (Final Assembly)*:** Se ensamblan los módulos individuales para completar el diseño, aquí se generan los bitstreams del sistema.

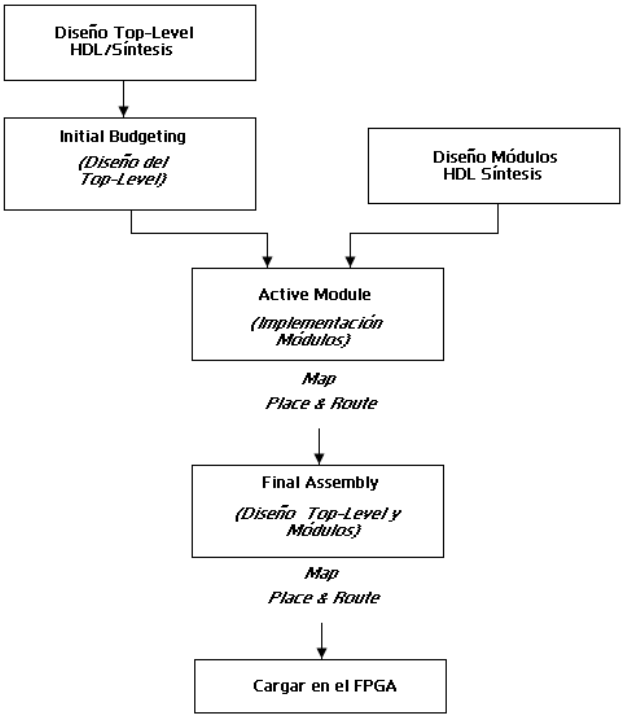


Figura 4.3. Flujo del Diseño Modular.

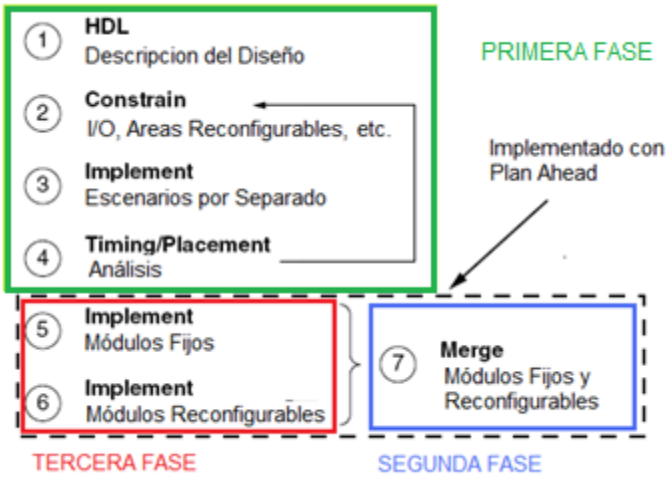


Figura 4.4. Etapas de Diseño para Sistemas Reconfigurables.

4.2.1 Primer Fase de Diseño

Durante esta fase se describe el modelo del sistema, número y tamaño de los módulos, pines de entrada y salida, posición de los módulos, etc, en la figura 4.5 se presenta un diagrama esquemático del archivo de mayor nivel del diseño, como ejemplo de lo que sucede en esta fase de diseño. Para esta fase se emplea la herramienta de ISE Webpack. Los objetivos de esta fase son los siguientes:

- Se establece la topología del diseño, es decir, se establecen los lugares y las áreas a ocupar por los módulos. Estas especificaciones deben ser incluidas en un archivo de restricciones .ucf.
- Se define la ubicación de los pines de entrada y salida.
- El resto de la lógica que no pertenece a los módulos debe tener restricciones para mapearse en un sitio fijo. No debe haber por tanto, ninguna lógica en el archivo de mayor nivel sin restricciones. (Esto solo es necesario para versiones ISEWebpack 8 y anteriores)
- Las restricciones para cada Bus Macro deben ser insertadas manualmente en el archivo de restricciones, de modo que se localicen entre los módulos actuando de puentes de comunicación.
- Se realizan las simulaciones del diseño para cada módulo, para comprobar que sea útil en comportamiento y en tiempos.

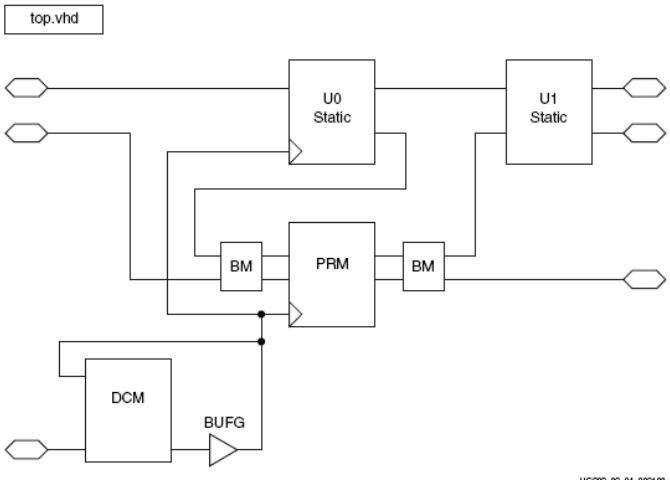


Figura 4.5 Diagrama de un Archivo de Mayor Nivel Jerárquico.

Todos los módulos reconfigurables que serán sintetizados en una sola área reconfigurable deben de tener los mismos nombre y tipos en los puertos de entrada / salida. Esta consistencia en los nombres garantiza que serán asociados con la descripción del archivo de mayor nivel, como se muestra en a figura 4.6.

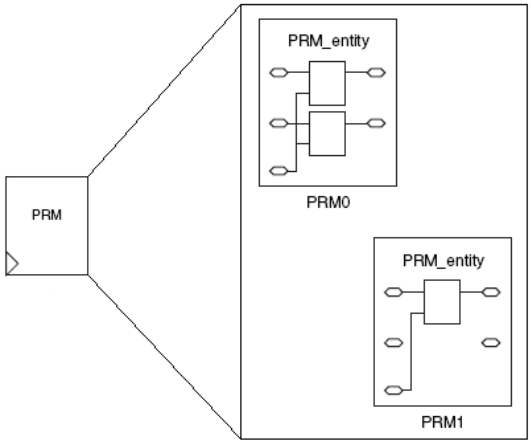


Figura 4.6. Esquema de Diferentes Módulos Reconfigurables.

Las restricciones del diseño son básicamente tres, restricciones para asignar pines de entrada / salida, para delimitar las regiones reconfigurables y estáticas y el empleado para colocar los Bus Macros. La primera restricción es empleada en todos los diseños con FPGA, por lo que no será analizado en el presente trabajo.

Para delimitar las regiones a emplear por las áreas reconfigurables empleamos las restricciones de área, éstas previenen que la lógica reconfigurable se mezcle con la lógica estática en las regiones delimitadas para los módulos estáticos y viceversa, que la lógica estática se mezcle con la reconfigurable dentro de las áreas designadas para ser ocupadas solamente por la lógica reconfigurable. Existen cuatro restricciones para manipular las áreas reconfigurables, *AREA_GROUP*, *AREA_GROUP_RANGE*, *MODE* y *LOC*.

- ***AREA_GROUP***. Esta restricción nos ayuda a relacionar elementos del diseño lógico con un área del FPGA. Debe de existir una declaración de esta restricción por cada región reconfigurable. No es necesario definir una restricción de este tipo para las regiones estáticas, aunque su uso no afecta el comportamiento del sistema. En la figura 4.7 se muestra como se asocian los módulos definidos en el archivo .vhd de más alto nivel, contra el archivo .ucf. El siguiente ejemplo muestra como asociar el módulo “modulorec_2” al área “REGION2”:

INST "modulorec_2" AREA_GROUP = "REGION2";

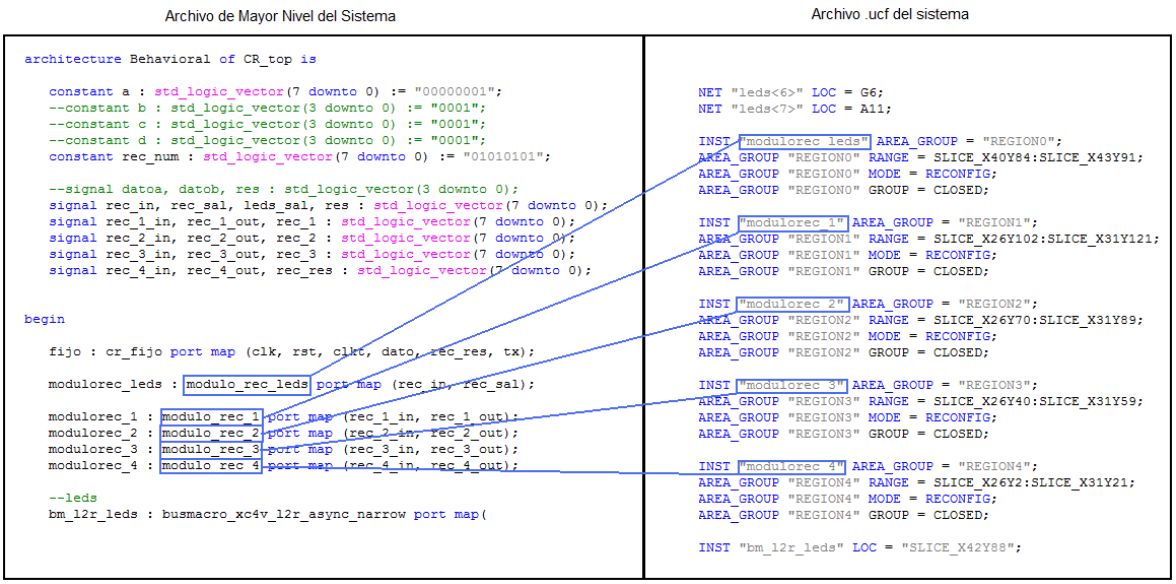


Figura 4.7. Relación de los Módulos Reconfigurables y la Restricción AREA_GROUP.

- ***AREA_GROUP_CONSTRAINTS***. Al menos una restricción de este tipo debe ser declarado por cada AREA_GROUP, la declaración básica de esta restricción consiste en crear conjuntos de slices, partiendo del mínimo de un slice. Además de los slices, si un diseño hace uso de otros elementos lógicos del FPGA como la memoria BRAM, multiplicadores, etc. se debe de declarar una restricción de este tipo por cada conjunto de elementos internos. Existen algunas reglas sencillas para el uso de la restricción:

    - Esta restricción no es opcional debido a que define el tamaño y la figura de las regiones asociadas con la restricción AREA_GROUP.
    - Se deben definir formas rectangulares.
    - Si son declaradas varias regiones reconfigurables, entonces éstas no deberán traslaparse unas con otras.
    - Todos los recursos del dispositivo como multiplicadores, memorias BRAM, etc. deben de ser asociados a una restricción de este tipo.

  Para definir regiones de slices se comienza describiendo de la esquina inferior izquierda, seguido de la esquina superior derecha, de esta forma el rectángulo formado por estas dos coordenadas será el área designada para el módulo reconfigurable, de igual forma se declaran los demás elementos lógicos del dispositivo que pertenecerán al área reconfigurable. En la figura 4.8 se muestra un ejemplo de declaración de esta restricción y la asociación contra el FPGA. En el

siguiente ejemplo se muestra un área reconfigurable asociada a la región "REGION2" la cual comienza en el slice (X26, Y70) y termina en el slice (X31, Y89):

AREA_GROUP "REGION2" RANGE = SLICE_X26Y70:SLICE_X31Y89;

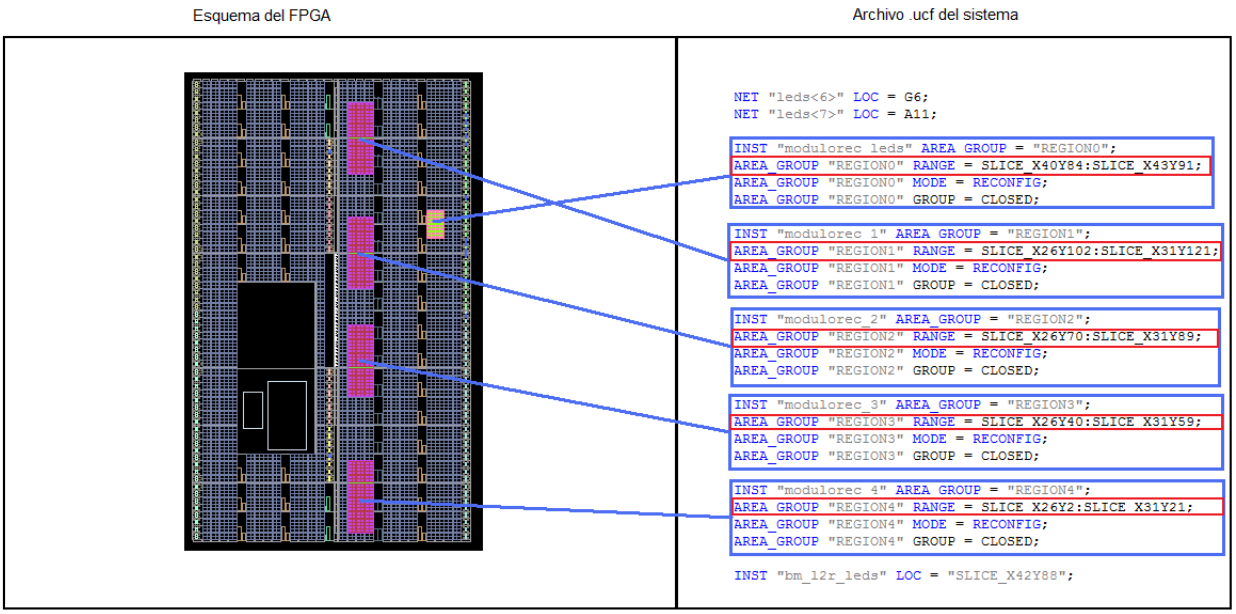


Figura 4.8. Áreas Generadas en el FPGA debido a la Restricción AREA_GROUP_RANGE.

- ***MODE***. Esta restricción nos ayuda a marcar las regiones que serán implementadas para hacer uso de la reconfiguración dinámica. La forma de indicarlo es la siguiente:

AREA_GROUP "REGION2" MODE = RECONFIG;

- ***LOC***. Esta restricción es útil para ubicar los Bus Macros del sistema, además de que nos permite controlar la ubicación de pines de entrada / salida y las primitivas del reloj. En la figura 4.9 se puede apreciar la declaración de esta restricción y como es reflejado en el FPGA. Para el caso de los Bus Macros debe de ponerse de tal forma que el Bus Macro quede en la frontera del área reconfigurable y el área estática, esto se puede ver en la figura 4.10. A continuación se da un ejemplo de cómo se declara la ubicación de un bus macro llamado *"bm_l2r_leds"* que se ubica en el slice X42Y88:

INST "bm_l2r_leds" LOC = "SLICE_X42Y88";

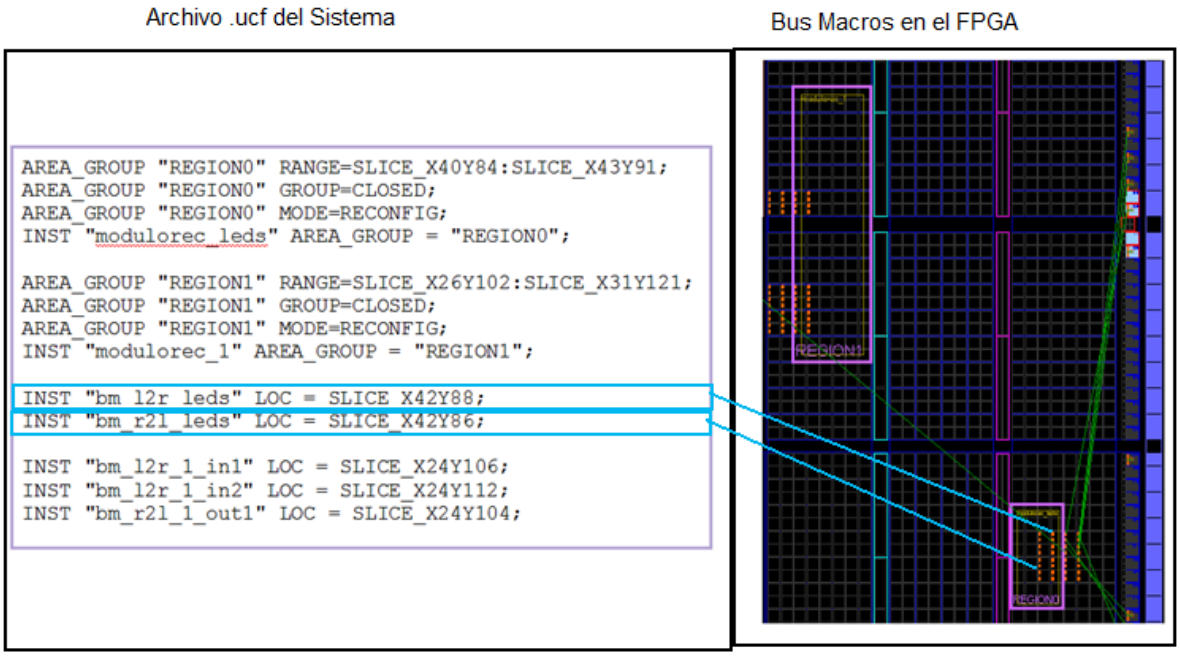


Figura 4.9. Restricciones de Áreas y Bus Macros.

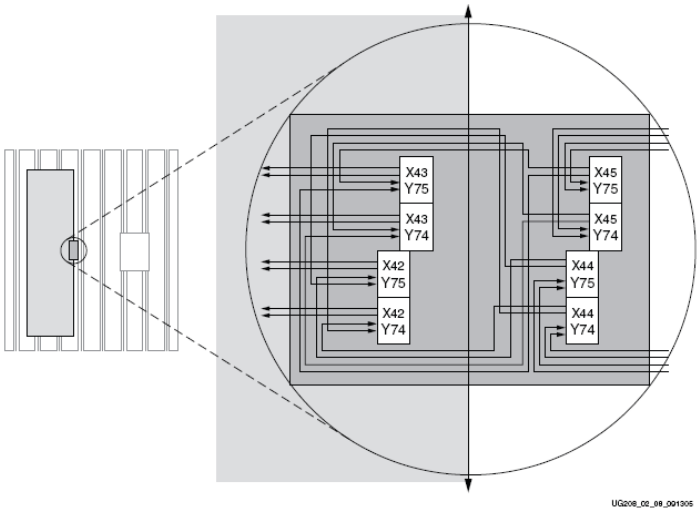


Figura 4.10. Ubicación Correcta de los Bus Macro.

4.2.2 Ejemplo de la Primer Fase de Diseño

El siguiente ejemplo ilustra un sistema sencillo, consta de una sola entrada, señal de reloj, y una salida, transmisor RS232 del FPGA a la PC para poder comprobar que el sistema está funcionando. Ambas entradas de la unidad reconfigurable se mantienen fijas en los valores 0x55 y 0x33, se llevaron a cabo dos funciones, una compuerta AND y un sumador. En la figura 4.11 se muestra un diagrama esquemático del sistema a crear.

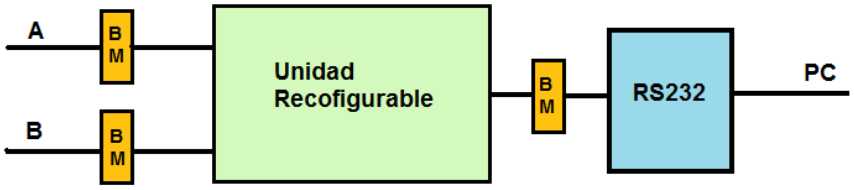


Figura 4.11. Diagrama Esquemático del Ejemplo.

Lo primero que se hace es diseñar el archivo de mayor nivel jerárquico, en el cuál se dará la arquitectura del sistema, en la tabla 4.1 se muestra el archivo de mayor nivel, con la explicación del archivo.

Archivo VHDL	Explicación
library IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.STD_LOGIC_UNSIGNED.ALL;	Librerías de propósito general
use work.busmacro_xc4v_pkg.all;	Paquete que contiene la definición de los Bus Macros.
entity CR_TOP_VHDL is Port (clk : in STD_LOGIC; tx : out STD_LOGIC); end CR_TOP_VHDL;	Declaración de la entidad.
architecture Behavioral of CR_TOP_VHDL is	Declaración de la arquitectura de la entidad.
constant a : std_logic_vector(7 downto 0) := "01010101"; constant b : std_logic_vector(7 downto 0) := "00110011";	Señales constantes que servirán como entradas del sistema.
signal UR1_a, UR1_b, UR1_z, dato : std_logic_vector(7 downto 0);	Señales auxiliares para implementar el sistema.
component CR_FIJO Port (clk : in STD_LOGIC; dato : in STD_LOGIC_VECTOR(7 downto 0); tx : out STD_LOGIC); end component;	Declaración de la unidad fija del sistema.
component CR_UR1 Port (a, b : in STD_LOGIC_VECTOR(7 downto 0); z : out STD_LOGIC_VECTOR(7 downto 0)); end component;	Declaración de la unidad reconfigurable del sistema.
begin	Aquí comienza la arquitectura de la entidad.
FIJO : CR_FIJO port map(clk, dato, tx);	Instanciación de la sección fija del sistema.
UR1 : CR_UR1 port map(UR1_a, UR1_b, UR1_z);	Instanciación de la sección variable del sistema.
bm_a_UR1 : busmacro_xc4v_l2r_async_narrow port map(a(0), a(1), a(2), a(3),	Instanciación de un Bus Macro del sistema. Este Bus Macro sirve de puente entre la señal “a” definida anteriormente y la entrada “a” de la

a(4), a(5), a(6), a(7), UR1_a(0), UR1_a(1), UR1_a(2), UR1_a(3), UR1_a(4), UR1_a(5), UR1_a(6), UR1_a(7));	unidad reconfigurable.
bm_b_UR1 : busmacro_xc4v_l2r_async_narrow port map(b(0), b(1), b(2), b(3), b(4), b(5), b(6), b(7), UR1_b(0), UR1_b(1), UR1_b(2), UR1_b(3), UR1_b(4), UR1_b(5), UR1_b(6), UR1_b(7));	Instanciación de un Bus Macro del sistema. Este Bus Macro sirve de puente entre la señal “b” definida anteriormente y la entrada “b” de la unidad reconfigurable.
bm_z_UR1 : busmacro_xc4v_r2l_async_narrow port map(UR1_z(0), UR1_z(1), UR1_z(2), UR1_z(3), UR1_z(4), UR1_z(5), UR1_z(6), UR1_z(7), dato(0), dato(1), dato(2), dato(3), dato(4), dato(5), dato(6), dato(7));	Instanciación de un Bus Macro del sistema. Este Bus Macro sirve de puente entre la señal “dato” definida anteriormente y la salida “z” de la unidad reconfigurable. La señal dato es la señal que será enviada a la PC por el RS232.
end Behavioral;	Fin de la descripción de la arquitectura.

Tabla 4.1. Explicación del Archivo VHDL que Define al Sistema.

Una vez teniendo la arquitectura del sistema, se procede a la implementación de los módulos fijos y variables, todos los archivos empleados para generar este ejemplo se encuentran en el Anexo A. En la figura 4.12 se muestra la definición de la arquitectura

para las dos funciones a integrar en la unidad reconfigurables, en la figura 4.12 (a) se muestra el ejemplo del sumador y en la figura 4.12 (b) el ejemplo de la compuerta lógica AND.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CR_UR1 is
    Port ( a, b : in  STD_LOGIC_VECTOR (7 downto 0);
           z : out  STD_LOGIC_VECTOR (7 downto 0));
end CR_UR1;

architecture Behavioral of CR_UR1 is

begin

   z <= a + b;

end Behavioral;
```

(a)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CR_UR1 is
    Port ( a, b : in  STD_LOGIC_VECTOR (7 downto 0);
           z : out  STD_LOGIC_VECTOR (7 downto 0));
end CR_UR1;

architecture Behavioral of CR_UR1 is

begin

   z <= a AND b;

end Behavioral;
```

(b)

Figura 4.12. Código Fuente de los Módulos Reconfigurables.

Ahora falta definir las restricciones del sistema, en la tabla 4.2 se explican las restricciones del sistema.

Archivo UCF	Explicación
NET "clk" LOC = AE14; NET "tx" LOC = W1;	Restricción para los pines del reloj y del transmisor.
INST "UR1" AREA_GROUP = "REGION1";	Asociación de la unidad reconfigurable “UR1” a la región reconfigurable “REGION1”
AREA_GROUP "REGION1" RANGE = SLICE_X26Y102:SLICE_X31Y121;	Establecer el tamaño del área reconfigurable “REGION1”, en este ejemplo, es el rectángulo generado del SLICE X26, Y102 al SLICE X31, Y121.
AREA_GROUP "REGION1" MODE = RECONFIG;	Indicación de que el área “REGION1” será reconfigurable.
AREA_GROUP "REGION1" GROUP = CLOSED;	Indicación de que el área “REGION1” será cerrada, es decir, nada ajeno a la unidad reconfigurable debe

	de ser mapeado dentro de esta área, así como que nada de la unidad debe ser mapeado fuera de esta área.
INST "bm_a_UR1" LOC = "SLICE_X24Y104";	Indicación de la ubicación física de los Bus Macro del sistema, notar que el Bus Macro empieza fuera del área reconfigurable, y termina dentro de ésta área. Los Bus Macro ocupan 4 slices, el área reconfigurable comienza en el SLICE 26, y todos los Bus Macro en el SLICE 24, el último SLICE ocupado por los Bus Macro es el SLICE 27 que se encuentra dentro de la unidad reconfigurable.
INST "bm_b_UR1" LOC = "SLICE_X24Y108";	
INST "bm_z_UR1" LOC = "SLICE_X24Y112";	

Tabla 4.2. Explicación del Archivo UCF que Define la Arquitectura del Sistema

Teniendo todos estos archivos se procede a realizar la síntesis del diseño en el ISE Webpack, teniendo que hacer este proceso dos veces, una para cada módulo reconfigurable, de esta forma obtenemos los archivo .ngc que se emplearán en la siguiente fase de diseño.

4.2.3 Segunda Fase de Diseño

En este momento el diseño ya esta sintetizado y se han definido las restricciones del mapeo y ruteo. Se puede iniciar por tanto la implementación de todos los módulos. Cada módulo puede ser implementado por separado pero siempre en el contexto del archivo de mayor nivel, así como las restricciones especificadas. Posteriormente se pueden generar los bitstreams para cada uno de los módulos reconfigurables. Para esta fase existen dos posibilidades, la primera es hacer todo el flujo desde línea de comandos, la segunda consiste en emplear el software de Plan Ahead, en el presente trabajo se muestra el flujo haciendo uso de la segunda opción. Los objetivos de esta fase de diseño son:

- Comprobar las reglas de diseño, señales, áreas, buses, relojes, etc.
- Generar los archivos de mapeo y ruteo de los diferentes módulos reconfigurables.
- Opcionalmente se pueden corregir problemas con la ubicación de los Bus Macro o de las áreas reconfigurables, ya que el editor del Plan Ahead nos da la posibilidad de ello.

Apoyados en el Plan Ahead, se importan los archivos .ngc generados en la fase anterior y el archivo de restricciones .ucf, un ejemplo se puede apreciar en la figura 4.13. Después se asocian los módulos reconfigurables a sus respectivas áreas reconfigurables y se procede a hacer el mapeo y ruteo de la aplicación. En esta fase se pueden detectar algunos problemas como el uso de elementos que no fueron declarados dentro del archivo de restricciones. Situaciones específicas como áreas reconfigurables que funcionen como nodos o como sumideros, es decir, áreas donde se pretendan generar señales o áreas donde entren señales pero no tenga salidas.

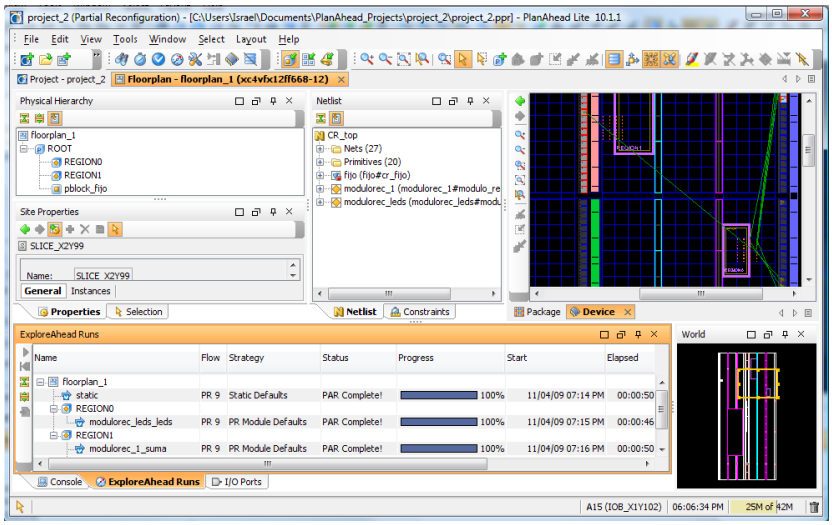


Figura 4.13. Ventana del Plan Ahead.

Una vez cargados los archivos iniciales, se procede a cargar cada uno de los archivos .ngc correspondiente a cada módulo reconfigurable anexando a cada región reconfigurable los módulos correspondientes y marcando cual será el módulo activo para el archivo de configuración inicial. Los pasos anteriores se repiten con cada uno de los módulos de diseño. En la figura 4.14 se da un ejemplo de cómo se anexan los módulos y como se muestran las unidades activas.

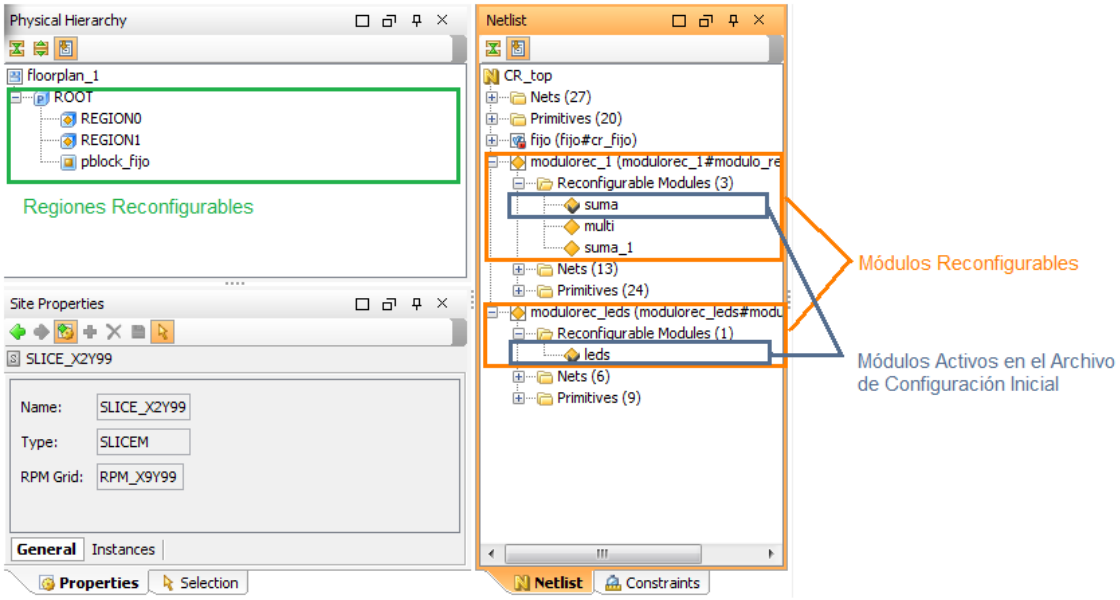


Figura 4.14. Módulos y Áreas Reconfigurables con Módulos Activos.

Una vez que todos los módulos han sido integrados, se procede a realizar el mapeo y ruteo del diseño, éste proceso se comienza por el módulo estático y después sobre todos y cada uno de los bloques dinámicos. En la figura 4.15 se puede ver como muestra el Plan

Ahead esta fase del diseño y en la figura 4.16 se muestra el flujo de información para esta etapa del diseño.

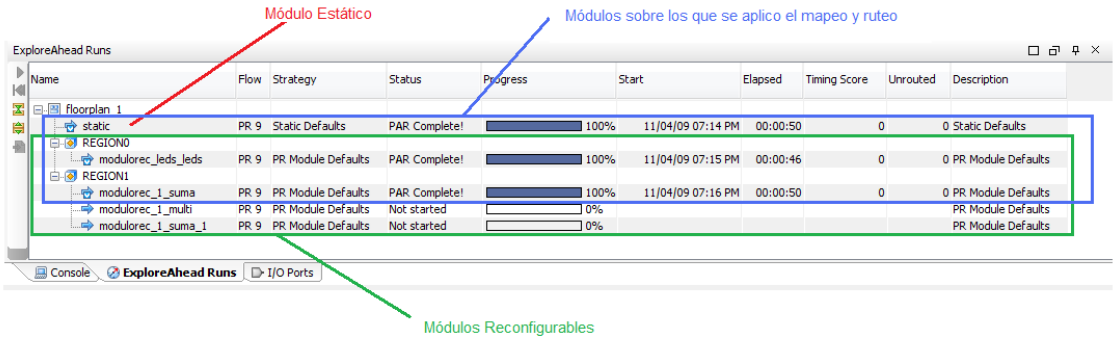


Figura 4.15. Monitor del Proceso de Mapeo y Ruteo.

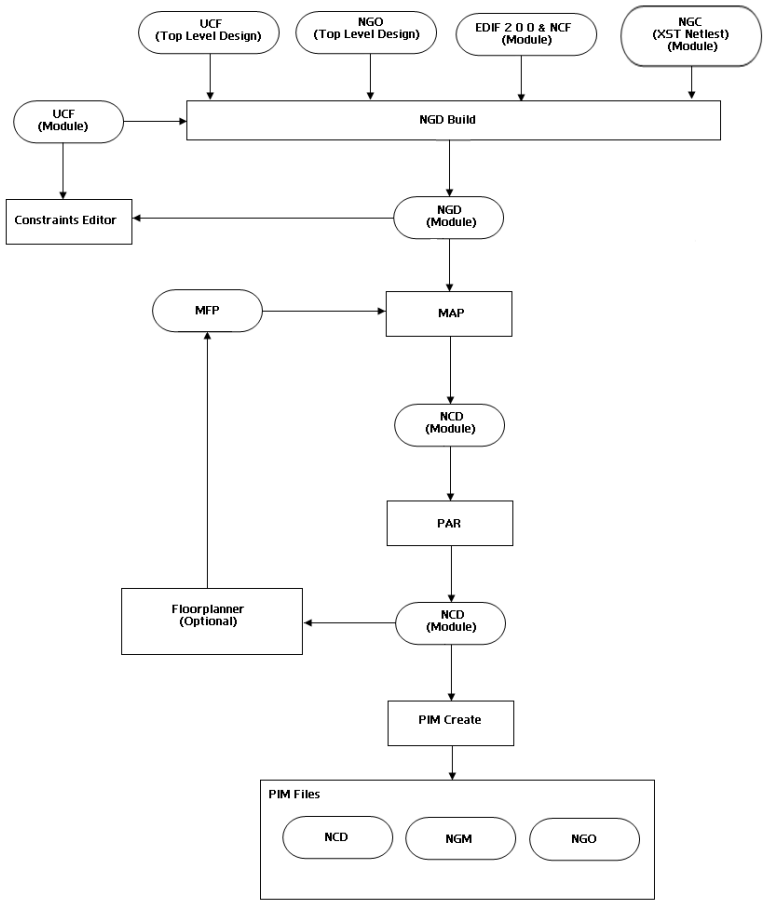


Figura 4.16. Flujo de Datos para la Fase de Active Module.

4.2.4 Tercer Fase de Diseño

En esta fase se realiza el proceso de combinación de los módulos individuales para obtener un diseño completo. El mapeo y ruteado llevado a cabo durante la fase de

implementación de cada módulo se preserva, manteniendo el rendimiento de cada módulo. El flujo de reconfiguración parcial requiere que los *bitstreams* inicialmente cargados en la FPGA correspondan a un diseño completo para que toda la lógica global y no reconfigurable esté mapeada y fijada, de modo que solo las partes reconfigurables del diseño cambien durante la reconfiguración parcial. Los objetivos de esta fase son:

- Generar el archivo .bit de carga inicial del sistema.
- Generar los archivos .bit de los módulos reconfigurables del sistema.
- Generar archivos .bit _blank para cada módulo reconfigurable, estos archivos son generados por la herramienta del Plan Ahead independientemente de los módulos cargados., y sirven para “apagar” cada módulo.

Esta se lleva a cabo con el ensamblador del Plan Ahead, el cual realiza todos los pasos necesarios para generar los archivos .bit finales, en la figura 4.17 se muestra la pantalla de este ensamblador trabajando y en la figura 4.18 el flujo de datos para esta etapa de diseño. Al finalizar esta etapa tenemos los bitstreams necesarios para la configuración y reconfiguración del dispositivo.

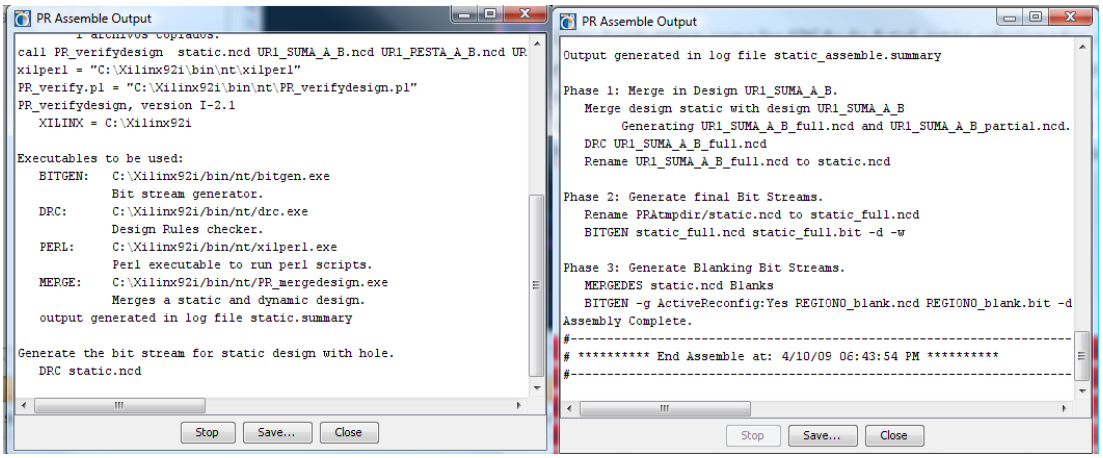


Figura 4.17. Ensamblador del Plan Ahead.

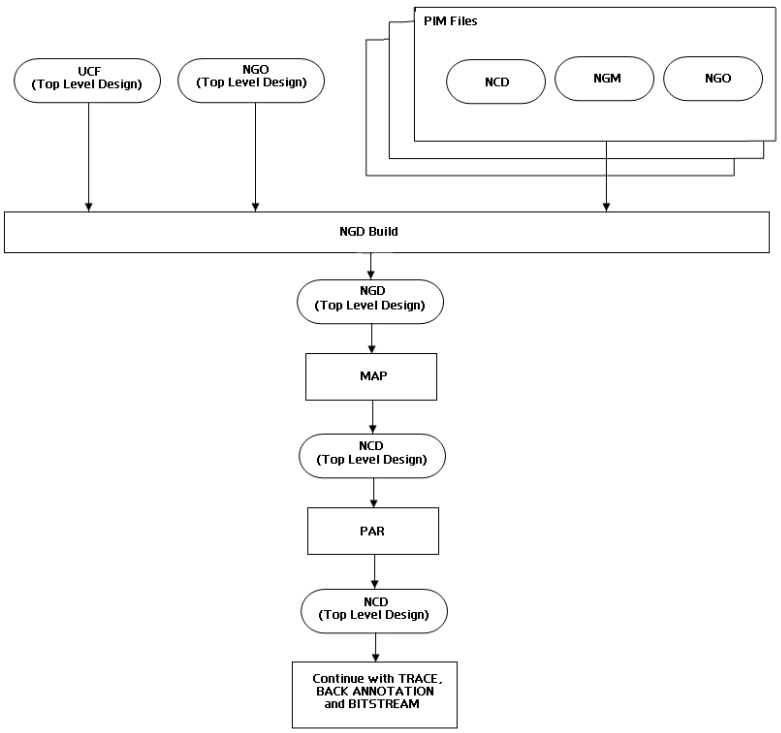


Figura 4.18. Flujo de datos del Final Assembly.

CAPITULO 5

SISTEMA PARA EL DISEÑO DE SISTEMAS RECONFIGURABLES

El CR implica un paso más en el aprovechamiento de las propias características de los dispositivos reconfigurables, es decir, la manipulación de la lógica configurada en el dispositivo durante el funcionamiento del sistema. La investigación en esta área se ha incrementado notablemente a finales de los años noventa y en la presente década. El nivel de integración de los dispositivos reconfigurables actuales más la posibilidad de modificar parcial y dinámicamente su configuración (característica disponible en algunos de ellos), han acelerado la investigación en este campo tanto a nivel teórico como en posibles aplicaciones. Esto supone un nuevo reto para la investigación en esta área: *El poder controlar la aplicación de la reconfiguración parcial dinámica sobre un sistema compuesto por módulos de distinta naturaleza teniendo en cuenta que deben mantener su funcionamiento mientras se modifica una única sección del sistema.*

5.1 Requerimientos del Sistema

Diseñar e implementar un sistema de software que facilite el diseño de sistemas digitales a partir de módulos de lógica reconfigurable. Este sistema deberá ser capaz de realizar las siguientes tareas:

- Administrar los Proyectos.
- Administrar las Bibliotecas de Funciones.
- Proveer un Medio para la Configuración del FPGA.
- Tomar la Configuración Actual del Sistema.

En la figura 5.1 se puede observar el diagrama de casos de uso general del sistema.

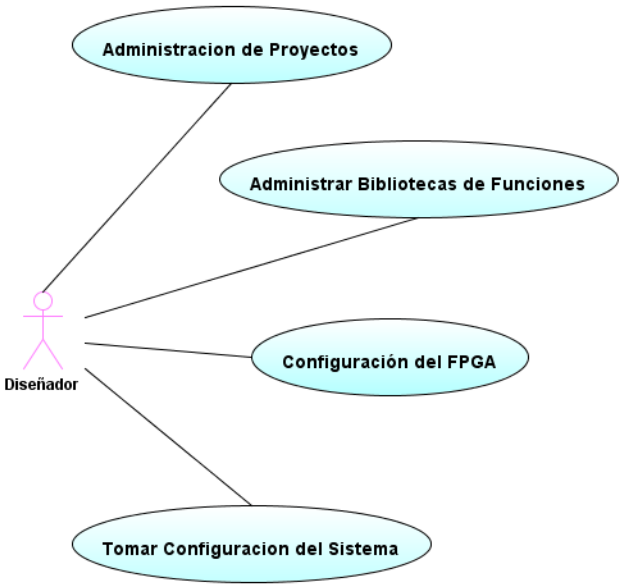


Figura 5.1. Casos de Uso Generales del Sistema.

5.1.1 Administración de Proyectos

El objetivo principal de este módulo es el de gestionar las características básicas del proyecto que se está trabajando. Será el responsable de generar la estructura del sistema de archivos, de las carpetas[2] y del archivo con las características del proyecto. Esta administración se centra básicamente en dos tareas:

- **Alta de Nuevos Proyectos**. Para esto se debe pedir el archivo de carga inicial .bit y el archivo de restricciones, además de la ubicación donde será guardado el proyecto. Opcionalmente pedirá el archivo en VHDL o Verilog correspondiente al archivo de mayor jerarquía del diseño.
- **Salvar Proyecto**. También será el encargado de guardar los cambios que vaya registrando el proyecto, como por ejemplo, altas de funciones, cambios en las funciones, descripción de las funciones, etc.

En la figura 5.2 se muestra el diagrama de casos de uso de este módulo del proyecto, y en la figura 5.3 (a) el diagrama de estados para dar de alta un nuevo proyecto, en la figura 5.3 (b) el diagrama de estados para salvar los cambios hechos a un proyecto.

[2] En el entorno gráfico de los sistemas operativos los directorios son denominados metafóricamente como carpetas, por lo que de aquí en adelante se empleará la palabra carpeta para referirse a los directorios que conforman el sistema de archivos.

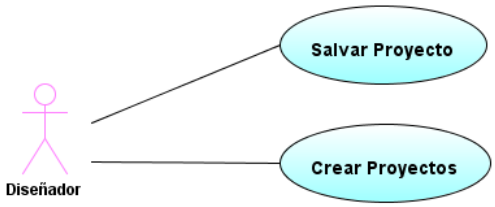


Figura 5.2. Diagrama de Casos de Uso de la Administración de Proyectos

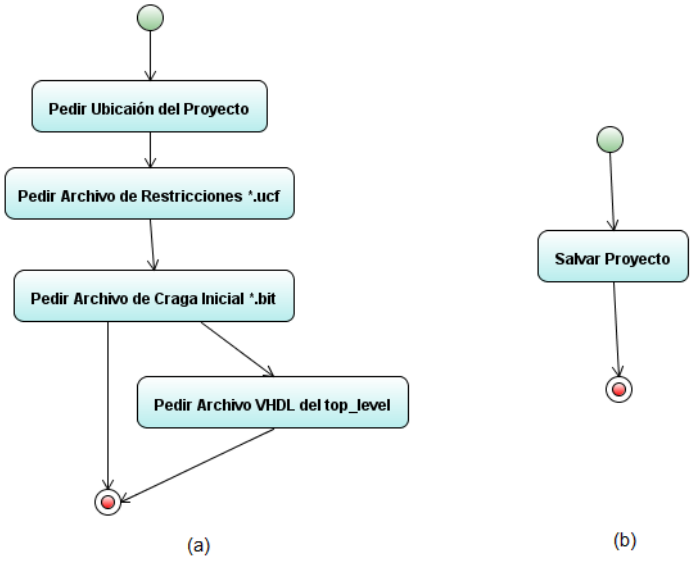


Figura 5.3. Diagrama de Estados para Crear un Proyecto.

5.1.2 Administrador de Bibliotecas de Funciones

Este módulo es el encargado de gestionar el sistema de las bibliotecas de componentes que se irán cargando al proyecto. Es el responsable del manejo de los archivos que se encargan de gestionar las características de las bibliotecas de funciones asociadas a los módulos reconfigurables. Las funciones de éste módulo se pueden agrupar en tres tareas básica:

- **Altas de Funciones**. Mediante esta tarea se pueden dar de alta nuevas funciones para los módulos reconfigurables. Durante la creación de funciones se podrá asignar un nombre a la función, será posible elegir un color para identificar a la función cuando este programada, además de la asociación de las funciones a los módulos reconfigurables.
- **Cambios en Funciones.** Se permite modificar los datos contenidos en las funciones, como lo es el color de reconfiguración, nombre, descripción, archivos de carga .bit,

.vhd y .v asociados a la función, agregar o quitar asociaciones de las funciones a los módulos reconfigurables.

- **Bajas de Funciones.** Esta opción se utiliza para dar de baja funciones del proyecto, lo cual implica que todas las asociaciones de la función con los módulos reconfigurables se perderán.

En la figura 5.4 se puede apreciar el diagrama de casos de uso para este módulo, en la figura 5.5 (a) el diagrama de estados para el alta de funciones, en la figura 5.5 (b) el diagrama para las bajas y en el diagrama 5.5 (c) los cambios a las funciones.

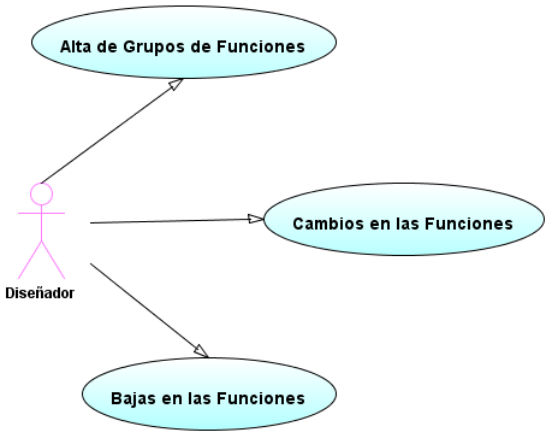


Figura 5.4. Casos de Uso del Administrador de Bibliotecas de Funciones.

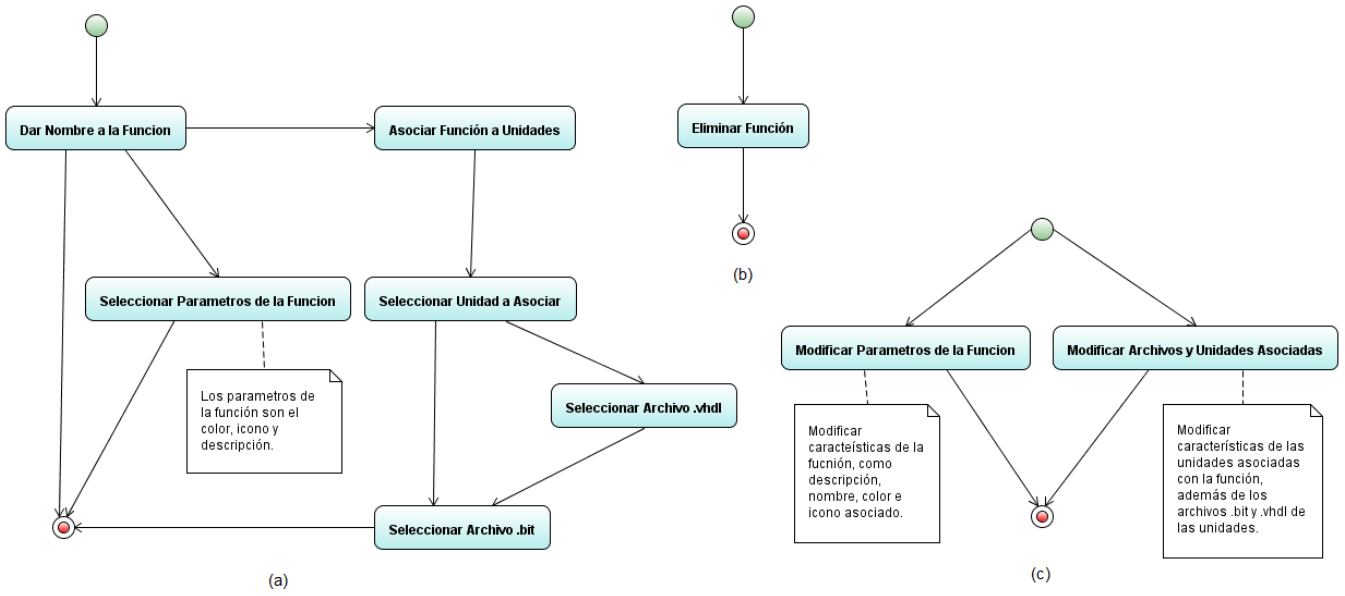


Figura 5.5. Diagramas de Estados del Administrador de Bibliotecas de Funciones.

5.1.3 Control de la Configuración del FPGA

Módulo encargado de la configuración del FPGA. Este módulo es el responsable de configurar al FPGA con la función asociada a ese módulo de entre el conjunto de opciones que existen en cada función, realiza básicamente dos tareas:

- **Configuración Total ó Configuración Inicial del Dispositivo**. Es la configuración necesaria para poder empezar a trabajar con el diseño, se cargará el archivo .bit que se eligió al momento de dar de alta el proyecto.
- **Reconfiguración Parcial de cada Módulo**. El sistema es capaz de decidir que archivo utilizará para configurar el FPGA dependiendo de la función elegida por el diseñador.

En la figura 5.6 se muestra los casos de uso de esta etapa del proyecto, en la figura 5.7 (a) se muestra el flujo de la configuración total del dispositivo, en la figura 5.7 (b) el flujo correspondiente a la reconfiguración dinámica.

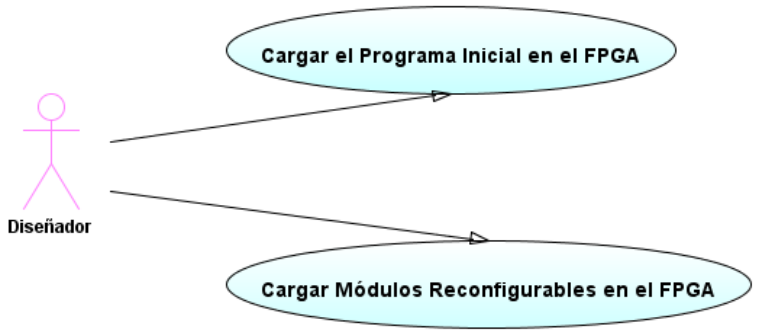


Figura 5.6. Casos de Uso del Control de Configuración del FPGA.

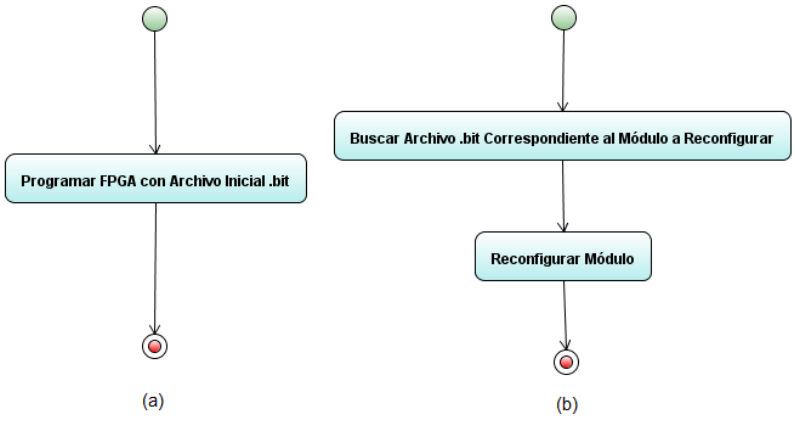


Figura 5.7. Diagramas de Flujo para los Procesos de Configuración.

5.1.4 Tomar Configuración Actual del Sistema

Cuando todos los bloques que conforman el diseño del FPGA tengan asociados un archivo .vhd o .v, será posible tomar la configuración actual del sistema. Esto con la finalidad de poder contar con un sistema completo a partir del diseño que se está probando. En la figura 5.8 se muestra el caso de uso de esta sección del proyecto, y en la figura 5.9 el flujo para poder tomar la configuración del sistema.

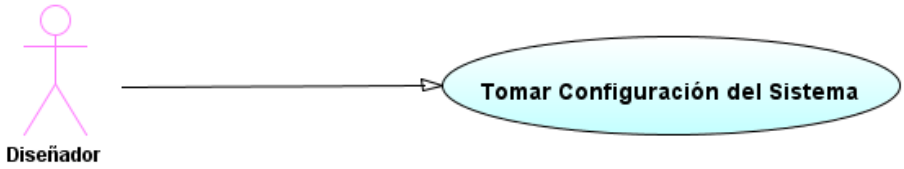


Figura 5.8. Casos de Uso para la Toma de Configuración del Sistema.

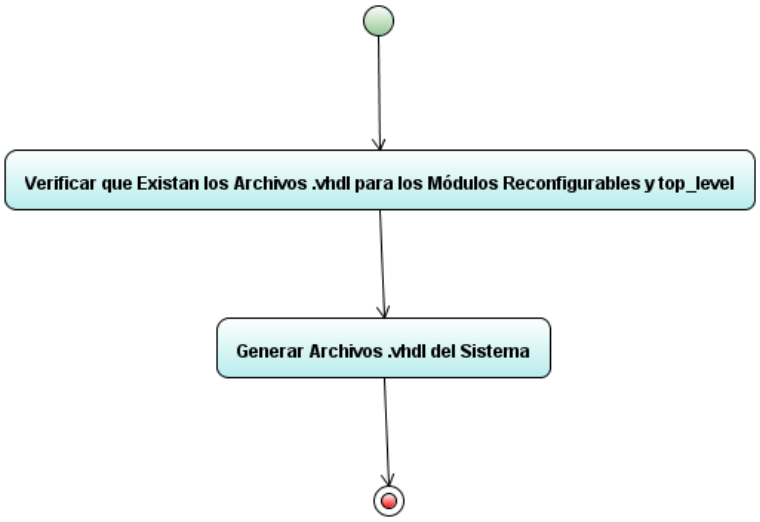


Figura 5.9. Diagrama de Flujo para la Toma de Estatus.

5.2 Herramientas de Desarrollo

Para el desarrollo del sistema se emplearon las siguientes herramientas.

- **NetBeans IDE 6.5.1**. Software de Sun Microsystems para el desarrollo de sistemas de información, principalmente aquellos diseñados para lenguaje Java. Es una herramienta pensada para el desarrollo de sistemas en lenguaje Java, provista por Sun Microsystems, la misma empresa que fomenta el uso de Java.
- **Java 1.6.0_13**. Fue elegido el lenguaje de programación Java debido principalmente a su independencia de la plataforma, lo que permitiría desarrollar posibles extensiones de la herramienta en ambientes tipo Unix, Linux, Mac OS, etc.

además de que la herramienta propuesta en esta tesis podría ser empleada en cualquiera de estos sistemas operativos con un mínimo de cambios, y el enfoque de desarrollo modular, lo que facilitará que otras personas continúen desarrollando sobre la herramienta.

- **Swing**. Es una plataforma independiente, Model-View-Controller Gui Framework para Java, la cual por ser implementada en Java brinda independencia en la plataforma sobre la que ese está ejecutando, modularidad, y su extensibilidad para desarrollar módulos a la medida.

Para probar que los circuitos se configuraran correctamente en el FPGA se empleo una tarjeta de desarrollo ML403 con un FGA Virtex 4FX12. La familia Virtex 4 fue la última familia completa de FPGAs de Xilinx que soportaba la reconfiguración dinámica, de la familia Virtex 5 solo algunos modelos la soportan y en la familia Virtex 6 el flujo de diseño cambia pero las herramientas actuales aún no lo soportan. En la figura 5.10 podemos ver una imagen de la tarjeta de desarrollo.

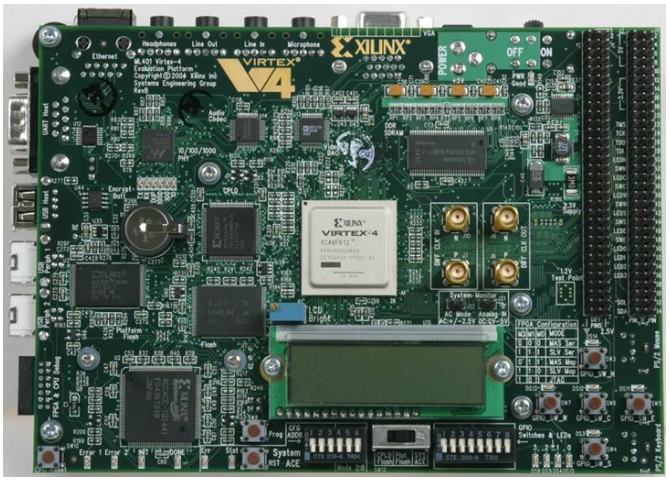


Figura 5.10. Tarjeta de Desarrollo ML403 de Xilinx.

5.3 Desarrollo del Sistema

El sistema fue desarrollado en Java, con el propósito de hacerlo multi-plataforma y facilitar de esta forma complementos futuros y / o rediseños. El ambiente fue creado en Swing. En la figura 5.11 se muestra un diagrama esquemático de las entradas y las salidas del sistema.

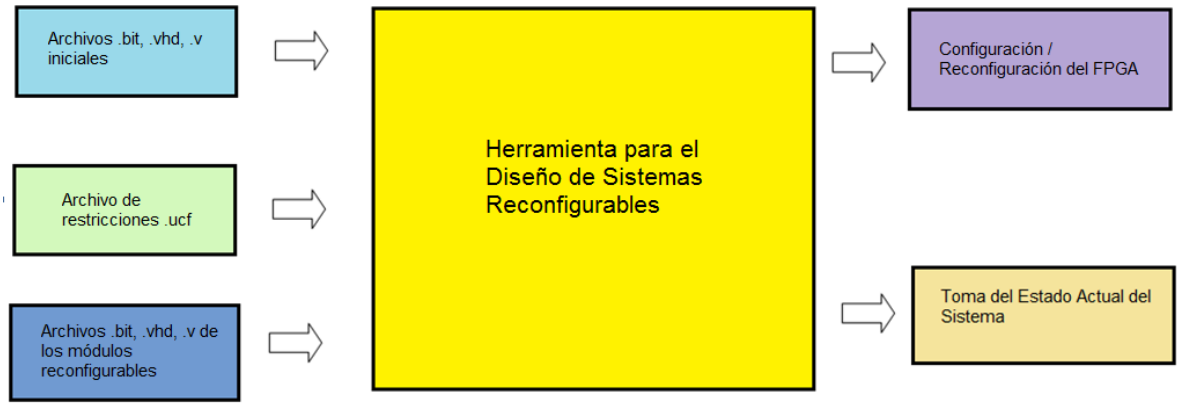


Figura 5.11. Principales Entradas y Salidas del Sistema.

La clase principal del proyecto se llama *MainControl.java* que se encuentra dentro del Source Package *cr.controller,* toda la funcionalidad del proyecto gira alrededor de esta clase. Dentro de ésta se encuentra el procedimiento principal del proyecto, el cual manda a llamar a la interfaz gráfica cuando es instanciado, los eventos que ocurren en la interfaz gráfica son tratados en ésta clase, y de aquí son dirigidos hacia las clases auxiliares que realizan el procesado de los eventos. En la figura 5.12 (a) se puede ver la instanciación de la clase, en ella se manda a llamar al método iniciar el cual inicializa la interfaz gráfica, esto se puede ver en la figura 5.12 (b).

```
public static void main(String[] args){
    MainControl main = MainControl.getMainControl();

    main.iniciar();
}
```

(a)

```
public void iniciar(){
    //vistaPrincipal.setVisible(true);

    //se establece el objeto como una tarea, de lo contrario por
    //alguna extraña razon no reescala los elementos internos.
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            vistaPrincipal.setVisible(true);
        }
    });

    //modelo.getListaFunciones().addElement("Suma");
    modelo.insertarLineaDetalle("Aplicacion iniciada");
    //modelo.insertarLineaDetalleError("Esto es un error");
}
```

(b)

Figura 5.12. Métodos para la Instanciación de la Clase MainControl.

5.3.1 Áreas de Trabajo

El sistema se organizó en cinco áreas de trabajo principales, estas se pueden ver en la figura 5.13.

- **Barra de Menús**. En esta área se encuentran los menús de opciones con las acciones disponibles para el sistema. Actualmente se encuentran habilitados cuatro menús de uso general.
    - **Proyecto**. Aquí tenemos las opciones para la administración de los proyectos.
    - **Funciones**. En este menú podemos administrar las características de las Bibliotecas de Funciones.
    - **Sistema**. En esta opción podemos cambiar el archivo .bit inicial del proyecto, también podemos configurar al FPGA con este archivo inicial.
    - **Agregados**. En esta opción se agregan las nuevas funcionalidades del sistema.
- **Barra de Iconos**. Aquí se encuentran los iconos con las funciones mas empleadas.
- **Bibliotecas de Funciones**. En esta área se despliegan las funciones con las que cuenta el sistema.
- **Área de Trabajo**. Aquí se puede ver representado el diseño del FPGA, además de contar con un visor de los archivos .vhd.
- **Consola de Mensajes**. Es el lugar donde tiene la salida el sistema para indicarnos las acciones que se han ido tomando.

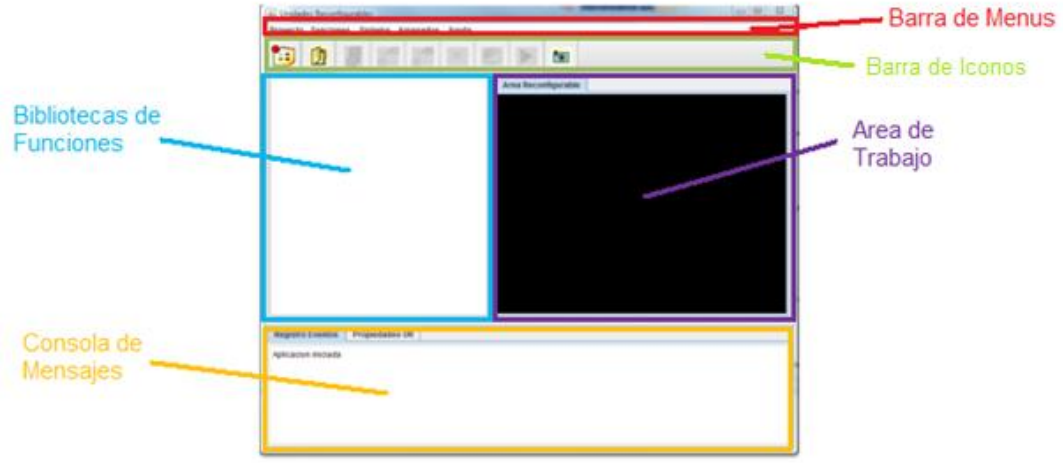


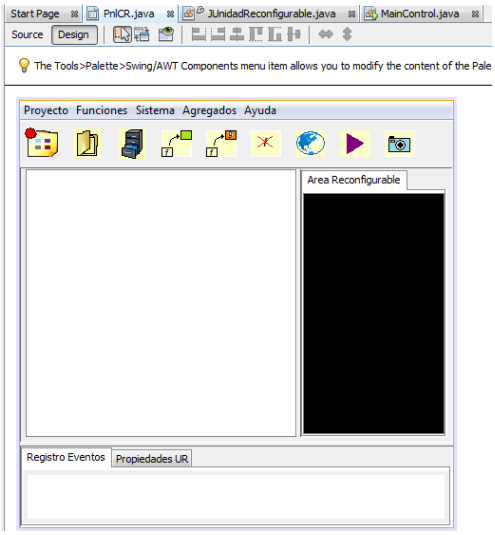
Figura 5.13. Distribución General de las Áreas de Trabajo.

La vista principal fue construida a partir de la interfaz gráfica con la que cuenta la IDE de desarrollo, esta es llamada como se muestra 5.12 (b), instancia la clase *PnlCR* que se encuentra dentro del package cr.vista.main, en la figura 5.14 (a) se puede observar la declaración del objeto en la clase MainControl, en la figura 5.14 (b) podemos ver la estructura de la ventana dentro de la IDE.

```
public class MainControl {

    private static MainControl singletonMainControl = null;

    private cr.vista.main.PnlCR vistaPrincipal = null;
    private cr.modelo.MdlCR modelo = null;
```



(a) (b)

Figura 5.14. Instanciación de la Clase PnlCR

A continuación se detallaran los puntos más importantes para las cuatro secciones principales del sistema analizadas en los requerimientos del mismo.

5.3.2 Administración de Proyectos

Los proyectos recién creados constituyen la unidad mínima funcional de un diseño. Los elementos necesarios para constituir esta unidad mínima son:

- **Nombre del Proyecto**. Nombre otorgado por el usuario del sistema al dar de alta un nuevo proyecto.
- **Archivo .ucf**. Archivo que modela en base a restricciones al sistema reconfigurable, en base a este archivo se conocen el número y tamaño de las unidades reconfigurables. Este archivo es generado desde la primera fase del diseño descrita en el punto 4.3.1.
- **Archivo .bit inicial**. Este es al archivo de carga inicial. Este archivo es generado en el Plan Ahead, en la tercera fase del diseño descrita en el punto 4.3.3.

Opcionalmente se puede tener el archivo .vhd que describe al sistema, este archivo es generado en la primera fase de diseño explicada en el inciso 4.3.1. Una vez creado un proyecto el sistema puede ser configurado en un FPGA debido a que se cuenta con el archivo de carga inicial.

Para la administración de proyectos de hace uso de un sistema de archivos preestablecido, la estructura de este sistema se detalla a continuación:

- **Carpeta Raíz**. La cual lleva el nombre del proyecto. Esta es la carpeta principal y la de más alto nivel. La ubicación de esta carpeta la da el usuario del sistema.
- **Carpeta vhdl**. En esta carpeta se alojan los archivos .vhd y .v que están asociados a las funciones de las unidades reconfigurables, y se encuentra dentro de la Carpeta Raíz. Dentro de esta carpeta se crearán subcarpetas una por cada unidad reconfigurable que tiene asociado al menos un archivo .vhd o .v, con excepción del archivo de mayor nivel jerárquico del diseño, el cual es copiado al mismo nivel que las carpetas de las funciones. En la figura 5.16, en la etiqueta de *<vhdl>* se puede apreciar la ruta relativa de este archivo con respecto a la carpeta raíz.
- **Carpeta bit**. En esta carpeta se alojan los archivos .bit asociados a las funciones de las unidades reconfigurables, se encuentra dentro de la Carpeta Raíz al mismo nivel que la Carpeta vhdl, al igual que en esta última, se crea una carpeta por cada una de las unidades reconfigurables que tienen asociado al menos un archivo .bit, de la misma forma que la carpeta anterior, el archivo .bit que contiene el bitstream de configuración inicial es copiado al mismo nivel que las carpetas de las funciones. En la figura 5.16, en la etiqueta de *<inicial>* se puede apreciar la ruta relativa de este archivo con respecto a la carpeta raíz.
- **Archivo de Restricciones**. Se realiza una copia del archivo de restricciones del diseño, el cuál es copiado dentro de la Carpeta Raíz.
- **Archivo .xml del Proyecto**. Al igual que la carpeta raíz este archivo tiene el mismo nombre que el proyecto, este archivo sirve principalmente para describir las relaciones entre los archivos y las unidades reconfigurables.

En la figura 5.15 se muestra un ejemplo de esta estructura de archivos, el proyecto lleva por nombre *"Ejemplo"*, dentro de esta carpeta se puede apreciar el archivo *Ejemplo.xml* que tiene la información del proyecto, un archivo de restricciones *CR.*ucf y las carpetas *bit* y *vhdl.*

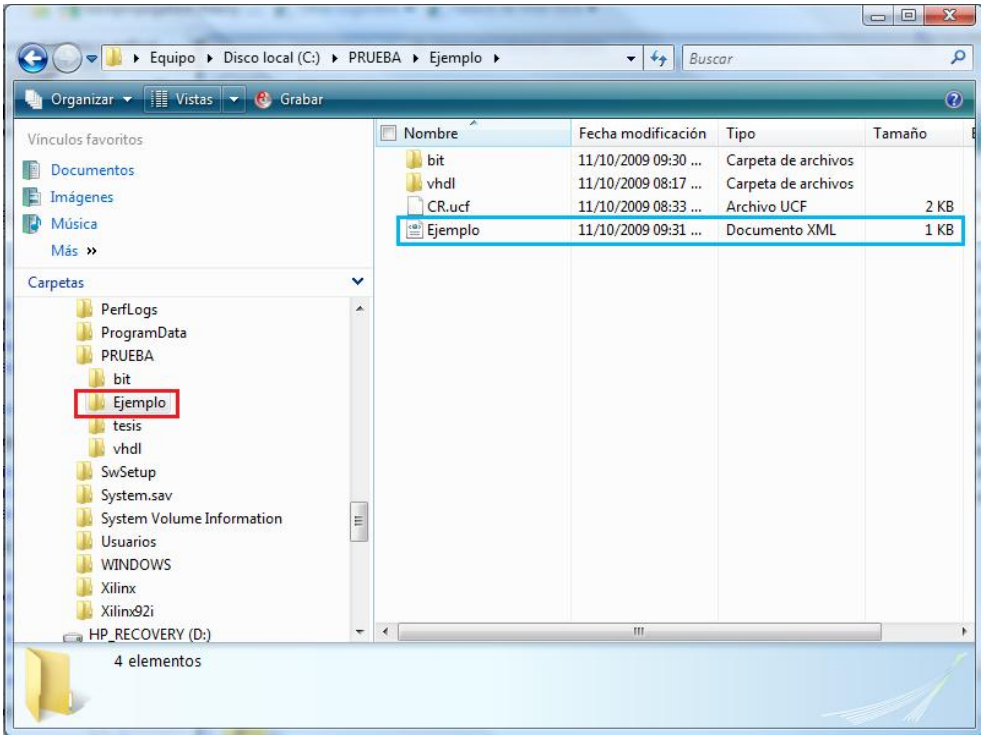


Figura 5.15. Ejemplo de la Estructura de Archivos del Sistema.

El archivo XML del proyecto define todas las características del proyecto. Al crear un proyecto en este archivo se guarda la información de cual y donde se encuentra el archivo de restricciones, la descripción del proyecto, el archivo de programación inicial, el cual se guarda dentro de la carpeta bit, el archivo .vhd que modela el top level del proyecto, el cual es guardado dentro de la carpeta vhdl, y finalmente la lista de funciones con sus relaciones hacia las unidades reconfigurables. En la figura 5.16 se muestra un ejemplo de este archivo al momento de ser creado el proyecto, la relación de las funciones será descrito más adelante.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <proyecto>
    <archivoUCF>.\CR.ucf</archivoUCF>
    <descripcion />
    <inicial>.\bit\static_full.bit</inicial>
    <vhdl>.\vhdl\CR_top.vhd</vhdl>
    <funciones />
  </proyecto>
```

Figura 5.16. Ejemplo de una Estructura Inicial del Archivo XML del Proyecto.

Cuando se crea un proyecto una vez dado el evento en la interfaz gráfica, este es atrapado por el método *nuevoProyecto* perteneciente a la clase *MainControl* en la figura 5.17 (a) se muestra parte del método donde se muestra la forma en la que es llamado el árbol de

directorios donde se ubicará al proyecto, al comienzo de la figura 5.17 (b) se pueden apreciar las funciones *abrirArchivoUCF*, para ubicar al archivo de restricciones del proyecto, *modelo.getUnidadesReconfigurables,* para analizar el archivo anterior y ubicar las unidades reconfigurables del proyecto, y *getArchivoFuncionInicial*, que se emplea para cargar el archivo .vhd o .v inicial. Al final de la función *nuevoProyecto* se habilitan las funciones que se podrán emplear una vez creado el proyecto.

```
private boolean nuevoProyecto(){
    modelo.insertarLineaDetalle("Proyecto Nuevo...");

    limpiarEntorno();

    //---------------------------------------------------------------------
    JArbolDirectorios directorioProyecto = new JArbolDirectorios(vistaPrincipal, true);
    directorioProyecto.setTitle("Ubicacion del Proyecto");
    WindowUtils.centrarDentroDePantalla(vistaPrincipal, directorioProyecto);
    directorioProyecto.setVisible(true);
    if(!directorioProyecto.isSelectionDone()){
        JOptionPane.showMessageDialog(vistaPrincipal, "Debe seleccionar un directorio para el proyecto",
        return false;
    }

    String rutaDirectorio = directorioProyecto.getRutaDirectorio();
    modelo.setRutaProyecto(rutaDirectorio);
```

(a)

```
if(!abrirArchivoUCF(file, modelo)) return false;

if(modelo.getUnidadesReconfigurables()!=null){

    if(!getArchivoFuncionInicial()){
        return false;
    }

    this.vistaPrincipal.enableAgregarFuncion(true);
    this.vistaPrincipal.enableGuardar(true);
    this.vistaPrincipal.enableInicializarModulo(true);

    return true;
}
```

(b)

Figura 5.17. Partes Críticas de la Función *nuevoProyecto.*

5.3.3 Administración de Bibliotecas de Funciones

Las funciones son conjuntos de elementos que le agregan funcionalidad al proyecto. Existe un elemento el cual es obligatorio para crear la función, el **Nombre de la Función**, el cual es otorgado por el usuario al momento de dar de crear la función. Existen otros parámetros opcionales que se detallan a continuación.

- **Descripción.** Comentario que da el usuario sobre la función creada.
- **Color.** Color que será empleado para diferenciar las funciones cuando son configuradas en el dispositivo, el color es elegido por el usuario mediante una paleta de colores.

- **Icono.** Imagen representativa de la función, el usuario da una ruta para encontrar la imagen.
- **Relación de las Unidades Reconfigurables.** Se eligen las unidades reconfigurables que serán afectados por la función, no necesariamente deben de estar todas, incluso como se dijo anteriormente puede ser un conjunto vacío. Para lograr esta relación se piden dos archivos al usuario:
    - **Archivo .bit para la Unidad Reconfigurable.** Estos archivos son generados en la tercera fase de diseño descrita en el punto 4.3.3, son elegidos por el usuario y queda bajo la responsabilidad del usuario asociar correctamente estos archivos hacia las unidades reconfigurables.
    - **Archivo .vhd para la Unidad Reconfigurable.** Son los archivos que definen el comportamiento de la función. Estos archivos son generados en la primera fase del diseño descrita en el punto 4.3.1.

Cada vez que se asocia una función a una unidad, los archivos asociados (.bit y .vhd o .v) son copiados en sus respectivas carpetas dentro de la estructura de archivos del proyecto, formándose dentro de estas una carpeta por cada unidad que tenga una función asociada. En la figura 5.18 se muestra la estructura interna de la carpeta bit.

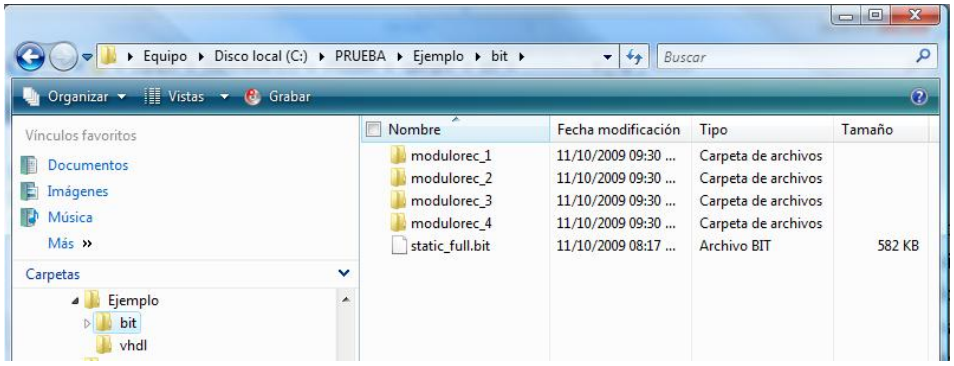


Figura 5.18. Ejemplo de la Estructura de Archivos para las Funciones.

La administración de las funciones se hace alrededor el archivo XML del proyecto, cada vez que se da de alta una función, se le cambian parámetros o se da de baja, los cambios se ven reflejados en este archivo. En la figura 5.19 la estructura del archivo XML con la función "*SUMA1*". El elemento *<funcion>* define cada una de las funciones que se irán creando dentro del proyecto, por lo tanto, existirán tantos elementos de este tipo como funciones sean creadas, acepta los siguientes atributos:

- **nombre**. El cual contiene el nombre de la función, este es el nombre que le da el usuario al crear la función.

- **descripcion**. Este atributo contiene la descripción de la función, al igual que el nombre es otorgada por el usuario.
- **color**. Número decimal que identifica el color elegido por el usuario como identificador visual cuando la unidad sea configurada con la función.
- **icono**. Ruta del icono que será empleado para mostrar la función.

Cada función, como ya se explico puede tener asociado un conjunto de unidades reconfigurables, estas unidades son mapeadas dentro del archivo XML con el elemento *<unidad>* el cual contiene al atributo **nombre** que indica el nombre de la unidad reconfigurable al cuál se le asociará la función, y dentro de la etiqueta la ruta relativa de la ubicación del archivo .bit de configuración.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <proyecto>
    <archivoUCF>.\CR.ucf</archivoUCF>
    <descripcion />
    <inicial>.\bit\static_full.bit</inicial>
    <vhdl>.\vhdl\CR_top.vhd</vhdl>
  - <funciones>
    - <funcion nombre="SUMA1" descripcion="Este es un sumador + 1" color="-205" icono="">
      - <unidades>
          <unidad nombre="modulorec_1">.\bit\modulorec_1\modulorec_1_module_1_1_partial.bit</unidad>
          <unidad nombre="modulorec_2">.\bit\modulorec_2\modulorec_2_module_2_1_partial.bit</unidad>
          <unidad nombre="modulorec_3">.\bit\modulorec_3\modulorec_3_module_3_1_partial.bit</unidad>
          <unidad nombre="modulorec_4">.\bit\modulorec_4\modulorec_4_module_4_1_partial.bit</unidad>
        </unidades>
      </funcion>
    </funciones>
</proyecto>
```

Figura 5.19. Ejemplo de un Archivo XML con la Función “SUMA1” Agregada.

Al igual que las características ya analizadas del proyecto, para dar de lata una función, la acción es llevada a cabo por el método *agregarFuncion* que se encuentra en la clase *MainControl*, en la figura 5.20 se pueden ver las primeras instrucciones de la función las cuales arman el árbol de las posibles unidades reconfigurables sobre las cuales se pueden crear funciones. La manipulación del archivo XML del proyecto se lleva a cabo desde la función *setToXML*, figura 5.21 (a), que también pertenece a la clase *MainControl*, el recuperado de la información lo llevamos a cabo desde la función *getFromXML*, figura 5.21 (b), que también se encuentra en la clase *MainControl*.

```
private void agregarFuncion(){
    cr.modelo.MdlPropiedadesFuncion modeloFuncion = new cr.modelo.MdlPropiedadesFuncion();
    modeloFuncion.setUnidadesReconfigurablesExistentes(modelo.getListaUnidadesReconfigurables());

    cr.controller.CtlPropiedadesFuncion controlPropiedades = new cr.controller.CtlPropiedadesFuncion(modelc
    controlPropiedades.mostrar();
```

Figura 5.20. Función agregarFuncion.

```
private void setToXML(String nombreArchivo){
    System.out.println("Ruta: " + nombreArchivo);
    configuracion.ProyectoXML archivo = new configuracion.ProyectoXML(nombreArchivo);

    cr.estructura.ArchivoProyecto_File archivoProyecto = new cr.estructura.ArchivoProyecto_File();

    if(modelo!=null){
        archivoProyecto.archivoConstraint = getRelativeFileName((String) modelo.getArchivoConstraint()
```

(a)

```
private void getFromXML(String nombreArchivo, String rutaProyecto){
    configuracion.ProyectoXML archivo = new configuracion.ProyectoXML(nombreArchivo);
    archivo.leer(true);

    if(archivo.getArchivoProyecto()==null) return;

    cr.modelo.MdlCR nuevoModelo = new cr.modelo.MdlCR();

    if(nuevoModelo!=null){
        vistaPrincipal.setModelo(nuevoModelo);
    }
```

(b)

Figura 5.21. Funciones para la Manipulación del Archivo XML

5.3.4 Configuración del FPGA

La configuración del FPGA se hace mediante la información contenida en el archivo XML del proyecto. En este archivo se tienen guardados los archivos .bit que van relacionados con las unidades reconfigurables. Cuando existe un evento para configurar una unidad se busca el archivo .bit correspondiente a la unidad para la función que quiera ser configurada. Si el archivo es encontrado la unidad se configura, si no se encuentra o no existe la relación el sistema manda un mensaje de error a consola dando la causa del por qué no se pudo realizar la configuración. Este proceso se lleva a cabo mediante la utilería de Xilinx del Impact llamado desde modo batch, en la figura 5.22 se ve un ejemplo de código para llamar al Impact en batch.

```
setmode -bs
setcable -p auto
identify
assignfile -p 3 -file C:\Users\Israel\Documents\Investigacion\...
program -p 3
setmode -bs
quit
```

Figura 5.22. Ejemplo de Código de Configuración Mediante Impact

A continuación se explicará el significado de la secuencia de comandos:

1. setmode –bs. Le indica al Impact que el modo de operar será en *Boundary-Scan* (JTAG).
2. setcable –p auto. Sirve para detectar automáticamente el cable, Impact lo busca en todos los puertos posibles.

3. identify. Identifica automáticamente todos los dispositivos dentro de la cadena de programación.
4. assignfile –p 3 –file C:\... Asigna un archivo de configuración al tercer dispositivo de la cadena identificada anteriormente. Esta ubicación es propia de la tarjeta de desarrollo usada y puede variar entre tarjetas de desarrollo distintas, en el caso de esta tesis la tarjeta de desarrollo empleada en la tercera posición tiene el Virtex 4.
5. program –p 3. Esta es la instrucción para programar el tercer dispositivo de la cadena identificada, se programara con el último archivo asignado. Al igual que en punto anterior ya se conoce de antemano la posición donde se encuentra el dispositivo a programar.
6. setmode –bs. Se emplea para limpiar el buffer de programación. En el desarrollo de esta tesis se observó que en algunos casos cuando no se indica esta instrucción de alguna forma se queda información en el buffer de programación, lo cual causa problemas al momento de realizar la siguiente configuración.
7. quit. Termina el proceso del Impact.

El proceso de programación se lleva a cabo desde la función programarFuncion en la clase MainControl, esta función lo que hace es verificar que exista una relación de la función con la unidad reconfigurable que se quiere programar, después si esta existiere, comprueba que está disponible el archivo del bitstream. Una vez que se han cumplido estas dos condiciones se procede al armado del batch para el impact, en la figura 5.23 se muestra como es que se arma ésta secuencia de comandos.

```
FileWriter instrucciones = null;
File f = new File("instrucciones.txt");
try {
    instrucciones = new FileWriter(f);
    PrintWriter pw = new PrintWriter(instrucciones);
    pw.println("setmode -bs");
    pw.println("setcable -p auto");
    pw.println("identify");
    pw.println("assignfile -p 3 -file " + rutaArchivo);
    pw.println("program -p 3");
    pw.println("setmode -bs");
    pw.println("quit");
    instrucciones.close();
} catch (IOException ex) {
    Logger.getLogger(MainControl.class.getName()).log(Level.SEVERE, "Error al preparar el ar
    modelo.insertarLineaDetalleError("Error al preparar el archivo instrucciones.txt " + ex.
}

String cadenaEjecucion = "impact -batch " + f.getPath();
```

Figura 5.23. Código Fuente del Armado de la Secuencia de Programación.

5.4 Ejemplo de Uso

A continuación se dará un ejemplo de cómo funciona la herramienta. El ejemplo está basado en un sumador en cascada como se muestra en la figura 5.24. Tiene una entrada y

cuatro sumadores los cuales funcionan como los módulos reconfigurables. La parte fija del sistema es implementada por un transmisor RS232 que permite ver la salida.

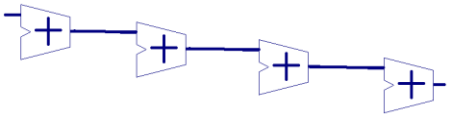


Figura 5.24. Circuito de Prueba de Sumadores en Cascada

5.4.1 Crear un Proyecto

En el menú Proyecto en la opción de Nuevo o con el icono de Nuevo Proyecto se comienza a crear un nuevo proyecto. El sistema pedirá una ubicación del proyecto, la cual le dará el nombre al proyecto, además pedirá un archivo .ucf, para el nombre del archivo es el de CR.ucf, en una siguiente ventana, figura 5.25 (a) daremos de alta los archivos .bit y .vhd. Al finalizar de cargar estos archivos tendremos en el área de trabajo representados a los módulos que conforman el sistema según lo indicado en el archivo CR.ucf, figura 5.24 (b). En la figura 5.25 (c) podemos ver la salida en consola cuando se da de alta un nuevo proyecto.

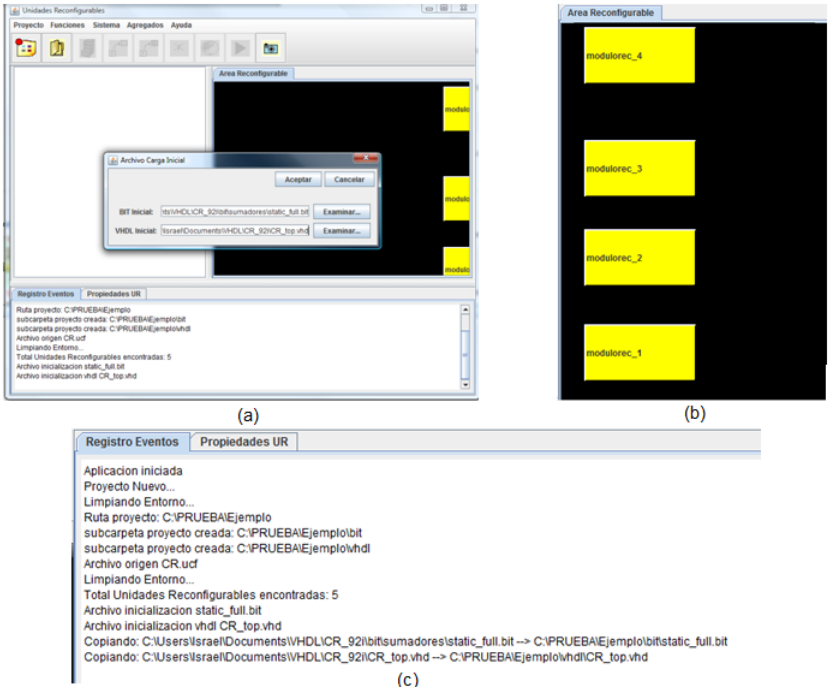


Figura 5.25. Pantallas para la Opción de Crear un Nuevo Proyecto.

5.4.2 Altas de Funciones

En esta sección se describirá el proceso para crear funciones. En la figura 5.26 Se encuentra las pantallas para crear funciones y la salida en la pantalla de las Bibliotecas de

Funciones. En estas pantallas se da el nombre a la función, una descripción de lo que hace la función, un color para identificarla cuando el módulo ha sido programado con esta función, y algún icono característico de la función, figura 5.26 (a). En esta misma pantalla se pueden hacer las asociaciones de las unidades a las funciones, se elige una unidad de entre las unidades posibles, se le indica el archivo .bit y .vhd asociados figura 5.26 (b). La salida la podemos observar en la figura 5.26 (c) donde vemos como se van reflejando las funciones en el área de la Biblioteca de Funciones.

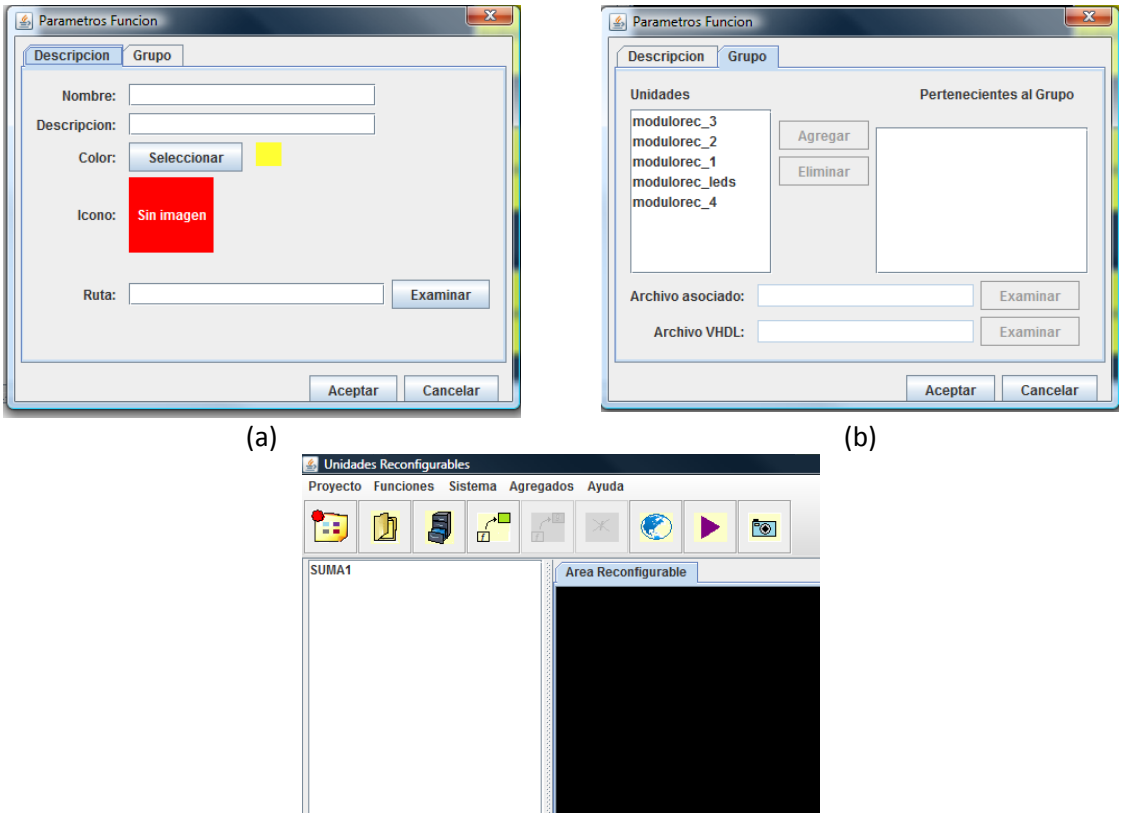
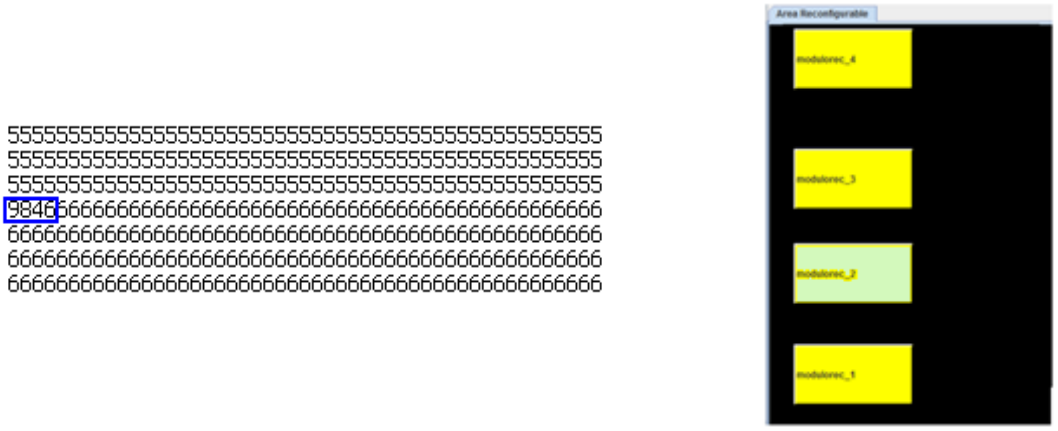


Figura 5.26. Pantallas para la Alta de Funciones

5.4.3 Configuración del FPGA

Para reconfigurar cualquiera de las unidades reconfigurables primero se debe de llevar a cabo la programación inicial del dispositivo, y en un segundo paso la programación de las unidades reconfigurables. Esto se lleva a cabo arrastrando la función deseada hasta la unidad reconfigurable que se quiere configurar, el sistema decidirá si existe un archivo .bit para la función que corresponda a la unidad reconfigurable. En la figura 5.27 (a) se puede ver la reconfiguración de una unidad, y en la figura 5.27 (b) la salida de la reconfiguración, nótese en el recuadro azul un transitorio debido a que las unidades reconfigurables están en cascada.



(a) (b)

Figura 5.27. Pantallas de Reconfiguración del FPGA

CAPITULO 6

CONCLUSIONES Y TRABAJOS A FUTURO

6.1 Conclusiones

Con la aparición de la familia de Virtex II Pro Xilinx dotó a sus dispositivos la habilidad de ser reconfigurados dinámicamente, esta reconfiguración, aunque un poco pobre, debido a que el dispositivo debía ser seccionado en columnas para formar las áreas reconfigurables por lo que se perdía el concepto de utilizar al máximo el área útil del dispositivo, mostro una interesante tendencia en el diseño de sistemas digitales. Tiempo después con la aparición de las familias Virtex 4 y Virtex 5, Xilinx da mayor flexibilidad hacia este tipo de características en sus dispositivos, lo que ha motivado al surgimiento de técnicas y herramientas que ayuden al diseño de sistemas reconfigurables.

La principal aportación de la presente tesis fue desarrollar una herramienta que facilite el diseño de sistemas a partir de sistemas reconfigurables, por ejemplo, si se tuviera un diseño de 10 regiones reconfigurables, y estas a su vez tuvieran asociadas 10 funciones, al final de la metodología de diseño descrita en el capítulo 4 se tendrían 100 archivos .bit para la reconfiguración, 10 archivos _blank generados adicionalmente y el full.bit, lo que arroja un total de 111 archivos para el control del sistema. El controlar tal cantidad de archivos puede ser una labor tediosa, esta labor es controlada con la herramienta desarrollada, una vez cargado el proyecto las funciones pueden ser usadas sin necesidad de recordar que archivo genera x función.

Adicionalmente, debido al control que se lleva de los proyecto, estos pueden ser compartidos por varias personas o grupos de trabajo, lo que facilitaría el intercambio de modelos o sistemas así como sus funciones.

6.2 Trabajos a Futuro

Los trabajos a futuro se podrían centrar en el desarrollo de extensiones a la herramienta para resolver problemas específicos, por ejemplo, para el diseño de redes neuronales artificiales, sistemas difusos, sistemas de control, tolerancia a fallos, sistemas con microprocesadores acoplados, etc. Además de trabajar en los aspectos de usabilidad de la herramienta, elementos gráficos para visualizar las conexiones entre unidades reconfigurables, funciones de exportación e importación de proyectos, métodos para mostrar información más detallada de las unidades.

Se podría trabajar en modelos de pruebas más robusto, esto junto con elementos gráficos más descriptivos podrían dar información sobre el estado de las señales entre unidades, a la entrada del sistema, a la salida del sistema, tiempos de reconfiguración, etc. Finalmente sería posible trabajar con los archivos de configuración .bit directamente, proponiendo un sistema de validación de las funciones, esto con el fin de detectar que el archivo .bit realmente está diseñado para una unidad reconfigurable *x* y que no hay error del diseñador al tratar de asociarla a alguna unidad.

BIBLIOGRAFÍA

[1] Y.W. Chan. "Physical design for SoC". Tutorial Soc. http://cc.ee.ntu.edu.tw/~ywchang, 2002.

[2] R. Rajsuman. "System-on-a-Chip. Design and Test". Artech House Publishers, 2000.

[3] J. Borel. "SoC Design Challenges: the EDA MEDEA Roadmap". Servicio de Publicaciones de la Universidad de Cantabria, 2001.

[4] A. Chao. "Platform Design System SoC. Faraday Technology Corporation". http://www.faraday-tech.com, 2002.

[5] Virtual Socket Interface Aliance. "Specifications, Standars, Technical Documents". http://www.vsia.org, 2003.

[6] G. Martin, H. Chang. "Winning the SoC Revolution: Experiences in Real Design". Kluwer Academic Publishers, Massachusetts, USA, 2003.

[7] M. Barr. "A Reconfigurable Computing Primer". http://www.netrino.com/Articles/RCPrimer/, 1998.

[8] K. Compton, S. Hauck. "Reconfigurable Computing: A Survey of Systems and Software". ACM Computing Surveys, 34(2):171-210, 2002.

[9] J. Tuley. "Soft Computing Reconfigures Designer Options". Embedded Systems, pp76, 1997.

[10] K. Bondalapati, V.K. Prasanna. "Reconfigurable Computing Systems". Proceedings of the IEEE, 90(7):1201 – 1217, Julio 2002.

[11] A. DeHon. "Reconfigurable Architectures for General Purpose Computing". A.I. Technical report No. 1586. Artificial Intelligence Laboratory. Massachusetts Institute of Technology, 1996.

[12] R. Tessier, W. Burleson. "Reconfigurable Computing for Digital Signal Processing: a survey". Journal of VLSI Signal Processing, (28):7 – 27, Mayo 2001.

[13] R. Sidhu, A. Mei, V.K. Prassana. "Genetic Programming using Self-Reconfigurable FPGAs". Lecture Notes in Computer Science, 1673:301 – 312, 1999.

[14] R. Sidhu, A. Mei, V.K. Prasanna. "String Matching on Multicontext FPGAs using Self-Reconfiguration". En Proceedigns of the ACM International Symposium in Field
Programmable Gate Arrays (FPGA'99), 217 – 226, Febrero 1999.

[15] R. Sidhu, V.K. Prasanna. "Fast Regular Expression Matching using FPGAs". En Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01), 698 – 709, Abril 2001.

[16] J. Hauser, J. Wawrzynek. "Garp: A MIPS Processor with a Reconfigurable Coprocessor". En Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97), 12 – 21, Abril 1997.

[17] K.H. Leung, K.W. Wong, P.H.W. Leong. "FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor". En Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00), 68 – 76, Abril 2000.

[18] J. Lázaro, A. Astarloa, U. Bidarte, *et al*. "High Throughput Serpent Encryption Implementation". Lecture Notes in Computer Science, 3203: 996 – 1000, 2004.

[19] W.J. Huang, N. Saxena, E.J. McCluskey. "A reliable LZ data compressor on reconfigurable coprocesors". En Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00), 249 – 258, Abril 2000.

[20] J. Burns, A. Donlin, J. Hogg, *et al*. "A Dynamic Reconfiguration Run-Time System". En Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97), 66 – 76, Abril 1997.

[21] J. Gause, P.Y.K. Cheung, W. Luk. "Reconfigurable Shape-Adaptive Template Matching Architectures". En Proceedigns of the IEEE FPGA Custom Computing Machine Conference 2002, 98 – 110, 2002.

[22] M. Ahrens, A. El Gamal, D. Galbraith, J. Greene, *et* al. "An FPGA family optimized for high densities and reduced routing delay". En Proceedings of the IEEE Custom Integrated Circuits Conference, 31.5.1 – 31.5.4, 1990.

[23] H. Hsieh, W. Carter, J. Y. Ja, E. Cheung, *et al.* "Third-generation Architecture Boosts Speed and Density of Field-programmable Gate arrays". En Proceedings of the IEEE Custom Integrated Circuits Conference, 31.2.1 – 31.2.7, 1990.

[24] Actel Corporation. *Accelerator Series FPGAs: ACT3 Family*.

[25] Actel Corporation. *SX Family of High Performance FPGAs*.

[26] S. Hauck, T.W. Fry, *et al*. "*The Chimaera Reconfigurable Functional UnitK*". IEEE Transactions on VLSI Systems, 12(2):206 – 217, Febrero 2004.

[27] C.R. Rupp, M. Landguth, T. Garverick, *et al*. "*The NAPA adaptive processing architecture*". En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, 28 – 37, Abril 1998.

[28] B. L. Hutchings, M. J. Wirthlin, "*Implementation Approaches for Reconfigurable logic Applications*". In Proc. 5th Int. Workshop on Field Programmable Logic and Applications (FPL), Vol. 975 of Lecture Notes in Computer Science, pp. 419–428. 1995.

[29] E.Sanchez, M. Sipper, O. Aenni, et al, "Static and Dinamic Configurable Systems", IEEE Transactions On Computer, pp. 556 – 564. 1997.

[30] G. McGregor, P. Lysaght, "*Self Controlling Dynamic Reconfiguration: A Case Study*".

Lecture Notes in Computer Science, 1673 : 144 – 154, 1999.

[31] P. Lysaght, "*Aspects of Dynamically Reconfigurable Logic*". IEE Coloquium on Reconfigurable Systems, Febrero 1999.

[32] R. Sidhu, V.K. Prasanna, "*Efficient Metacomputation using Self-Reconfiguration*". Lecture Notes in Computer Science, 2438 : 698 – 709, 2002.

[33] S. Trimberger, D. Carberry, *et al,* "*A time-multiplexed FPGA*". En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97)*, 22 – 28, Abril 1997.

[34] A. DeHon, "*DPGA Utilization and Applications*". En *Proceedigns of the ACM International Symposium in Field Programmable Gate Arrays (FPGA'96)*, 115 – 121, Febrero 1996.

[35] K. Danne, C. Bobda, H. Kalte, "*Run-Time Exchange of Mechatronic Controllers Using Partial Hardware Reconfiguration*". Lecture Notes in Computer Science, 2778 : 272 – 281, 2003.

[36] H. Schmit, "*Incremental Reconfiguration for Pipelined Applications*". En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines(FCCM'97)*, 16 – 18, Abril 1997.

[37] N. Shirazi, W. L. Peter, Y. K. Cheung, "*Automating Production of Run-Time Reconfigurable Designs*". *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, 147 – 156, Abril 1998.

[38] W. Luk, N. Shirazi, P.Y.K. Cheung, "*Compilation tools for run-time reconfigurable designs*". En *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'97)*, 56 - 65, Abril 1997.

[39] Enzler, Rolf, "*Architectural Trade-offs in Dynamically Reconfigurable Processors*", Swiss Federal Institute Of Technology Zurich, Tésis Doctoral, Suiza, 2004. 2004.

[40] G. Estrin and R. Turn, "Automatic assignment of computations in a variable structure computer system," *IEEE Transactions on Electronic Computers*, vol. 12, no. 5, pp. 755 – 773, 1963.

[41] Xilinx Inc., Application Note XAPP290.

[42] *The Programmable Logic Data Book*, San Jose, CA: Xilinx, Inc., 1994.

[43] *"Ciclone III Device Handbook". Altera Corp., 2008.*

[44] *"Spartan 3 FPGA Family: Complete Data Sheet". Xilinx Inc., 2008.*

[45] *"Virtex-4 FPGA User Guide". Xilinx Inc., 2009.*

[46] *"Virtex-4 FPGA Configuration User Guide". Xilinx Inc., 2009.*

ANEXO A

CÓDIGO HDL DEL EJEMPLO SENCILLO

Archivo CR_TOP_VHDL.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.busmacro_xc4v_pkg.all;

entity CR_TOP_VHDL is
   Port ( clk : in  STD_LOGIC;
          tx : out  STD_LOGIC);
end CR_TOP_VHDL;

architecture Behavioral of CR_TOP_VHDL is

          constant a : std_logic_vector(7 downto 0) := "01010101";
          constant b : std_logic_vector(7 downto 0) := "00110011";

          signal UR1_a, UR1_b, UR1_z, dato : std_logic_vector(7 downto 0);

          component CR_FIJO
  Port ( clk  : in  STD_LOGIC;
                          dato : in  STD_LOGIC_VECTOR(7 downto 0);
               tx   : out STD_LOGIC);
  end component;

          component CR_UR1
  Port ( a, b : in  STD_LOGIC_VECTOR(7 downto 0);
                          z    : out STD_LOGIC_VECTOR(7 downto 0));
  end component;

begin

          FIJO : CR_FIJO port map(clk, dato, tx);
          UR1 : CR_UR1 port map(UR1_a, UR1_b, UR1_z);
```

```
bm_a_UR1 : busmacro_xc4v_l2r_async_narrow port map(
                                        a(0),
                                        a(1),
                                        a(2),
                                        a(3),
                                        a(4),
                                        a(5),
                                        a(6),
                                        a(7),
                                        UR1_a(0),
                                        UR1_a(1),
                                        UR1_a(2),
                                        UR1_a(3),
                                        UR1_a(4),
                                        UR1_a(5),
                                        UR1_a(6),
                                        UR1_a(7));

bm_b_UR1 : busmacro_xc4v_l2r_async_narrow port map(
                                        b(0),
                                        b(1),
                                        b(2),
                                        b(3),
                                        b(4),
                                        b(5),
                                        b(6),
                                        b(7),
                                        UR1_b(0),
                                        UR1_b(1),
                                        UR1_b(2),
                                        UR1_b(3),
                                        UR1_b(4),
                                        UR1_b(5),
                                        UR1_b(6),
                                        UR1_b(7));

bm_z_UR1 : busmacro_xc4v_r2l_async_narrow port map(
                                        UR1_z(0),
                                        UR1_z(1),
                                        UR1_z(2),
                                        UR1_z(3),
                                        UR1_z(4),
                                        UR1_z(5),
                                        UR1_z(6),
                                        UR1_z(7),
                                        dato(0),
                                        dato(1),
                                        dato(2),
```

```
                                    dato(3),
                                    dato(4),
                                    dato(5),
                                    dato(6),
                                    dato(7));


end Behavioral;
```

Archivo de Restricciones CR.ucf

```
NET "clk" LOC = AE14;
NET "tx" LOC = W1;

INST "UR1" AREA_GROUP = "REGION1";
AREA_GROUP "REGION1" RANGE = SLICE_X26Y102:SLICE_X31Y121;
AREA_GROUP "REGION1" MODE = RECONFIG;
AREA_GROUP "REGION1" GROUP = CLOSED;

INST "bm_a_UR1" LOC = "SLICE_X24Y104";
INST "bm_b_UR1" LOC = "SLICE_X24Y108";
INST "bm_z_UR1" LOC = "SLICE_X24Y112";
```

Archivo CR_FIJO.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity CR_FIJO is
   Port ( clk : in  STD_LOGIC;
          dato : in  STD_LOGIC_VECTOR (7 downto 0);
          tx : out  STD_LOGIC);
end CR_FIJO;

architecture Behavioral of CR_FIJO is

          signal baud_count : integer range 0 to 650 := 0;
          signal flag1, flag2, en_16_x_baud : std_logic;

          component uart_tx is
          Port (         data_in : in std_logic_vector(7 downto 0);
                                    write_buffer : in std_logic;
                                    reset_buffer : in std_logic;
                                    en_16_x_baud : in std_logic;
                                              serial_out : out std_logic;
```

```
                                        buffer_full : out std_logic;
                        buffer_half_full : out std_logic;
                                                        clk : in std_logic);
    end component;

begin

    --reloj de transmision
    baud_timer: process(clk)
    begin
        if clk'event and clk = '1' then
            if baud_count = 650 then
                baud_count <= 0;
                en_16_x_baud <= '1';
            else
                baud_count <= baud_count + 1;
                en_16_x_baud <= '0';
            end if;
        end if;
    end process baud_timer;

    --transmisor RS232
    RS232_tx: uart_tx
    port map (    data_in => dato,
                  write_buffer => '1',
                  reset_buffer => '0',
                  en_16_x_baud => en_16_x_baud,
                        serial_out => tx,
                   buffer_full => flag1,
            buffer_half_full => flag2,
                                                clk => clk );

end Behavioral;
```

Archivo CR_UR1.vhd (sumador)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CR_UR1 is
   Port ( a, b : in  STD_LOGIC_VECTOR (7 downto 0);
          z : out  STD_LOGIC_VECTOR (7 downto 0));
end CR_UR1;

architecture Behavioral of CR_UR1 is
```

```
begin

        z <= a + b;

end Behavioral;
```

Archivo CR_UR1.vhd (compuerta AND)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CR_UR1 is
   Port ( a, b : in  STD_LOGIC_VECTOR (7 downto 0);
          z : out  STD_LOGIC_VECTOR (7 downto 0));
end CR_UR1;

architecture Behavioral of CR_UR1 is

begin

        z <= a AND b;

end Behavioral;
```

ANEXO B

CÓDIGO HDL DEL SUMADOR

Archivo CR_top.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.top_pac.all;
use work.busmacro_xc4v_pkg.all;

entity CR_top is
   Port (  clk, rst, clkt, dato : in  STD_LOGIC;
           tx : out STD_LOGIC;
           leds : out std_logic_vector(7 downto 0));
end CR_top;

architecture Behavioral of CR_top is

           constant a : std_logic_vector(7 downto 0) := "00000001";
           constant rec_num : std_logic_vector(7 downto 0) := "01010101";

           signal rec_in, rec_sal, leds_sal, res : std_logic_vector(7 downto 0);
           signal rec_1_in, rec_1_out, rec_1 : std_logic_vector(7 downto 0);
           signal rec_2_in, rec_2_out, rec_2 : std_logic_vector(7 downto 0);
           signal rec_3_in, rec_3_out, rec_3 : std_logic_vector(7 downto 0);
           signal rec_4_in, rec_4_out, rec_res : std_logic_vector(7 downto 0);

begin

           fijo : cr_fijo port map (clk, rst, clkt, dato, rec_res, tx);

           modulorec_leds : modulo_rec_leds port map (rec_in, rec_sal);

           modulorec_1 : modulo_rec_1 port map (rec_1_in, rec_1_out);
           modulorec_2 : modulo_rec_2 port map (rec_2_in, rec_2_out);
           modulorec_3 : modulo_rec_3 port map (rec_3_in, rec_3_out);
           modulorec_4 : modulo_rec_4 port map (rec_4_in, rec_4_out);

           --leds
           bm_l2r_leds : busmacro_xc4v_l2r_async_narrow port map(
                                        rec_sal(0),
                                        rec_sal(1),
                                        rec_sal(2),
                                        rec_sal(3),
                                        rec_sal(4),
```

```
                                rec_sal(5),
                                rec_sal(6),
                                rec_sal(7),
                                leds_sal(0),
                                leds_sal(1),
                                leds_sal(2),
                                leds_sal(3),
                                leds_sal(4),
                                leds_sal(5),
                                leds_sal(6),
                                leds_sal(7));

bm_r2l_leds : busmacro_xc4v_r2l_async_narrow port map (
                                rec_num(0),
                                rec_num(1),
                                rec_num(2),
                                rec_num(3),
                                rec_num(4),
                                rec_num(5),
                                rec_num(6),
                                rec_num(7),
                                rec_in(0),
                                rec_in(1),
                                rec_in(2),
                                rec_in(3),
                                rec_in(4),
                                rec_in(5),
                                rec_in(6),
                                rec_in(7));

--modulo 1
bm_l2r_1 : busmacro_xc4v_l2r_async_narrow port map(
                                a(0),
                                a(1),
                                a(2),
                                a(3),
                                a(4),
                                a(5),
                                a(6),
                                a(7),
                                rec_1_in(0),
                                rec_1_in(1),
                                rec_1_in(2),
                                rec_1_in(3),
                                rec_1_in(4),
                                rec_1_in(5),
                                rec_1_in(6),
                                rec_1_in(7));

bm_r2l_1 : busmacro_xc4v_r2l_async_narrow port map (
                                rec_1_out(0),
                                rec_1_out(1),
                                rec_1_out(2),
                                rec_1_out(3),
                                rec_1_out(4),
```

```
                                rec_1_out(5),
                                rec_1_out(6),
                                rec_1_out(7),
                                rec_1(0),
                                rec_1(1),
                                rec_1(2),
                                rec_1(3),
                                rec_1(4),
                                rec_1(5),
                                rec_1(6),
                                rec_1(7));

--modulo 2
bm_l2r_2 : busmacro_xc4v_l2r_async_narrow port map(
                                rec_1(0),
                                rec_1(1),
                                rec_1(2),
                                rec_1(3),
                                rec_1(4),
                                rec_1(5),
                                rec_1(6),
                                rec_1(7),
                                rec_2_in(0),
                                rec_2_in(1),
                                rec_2_in(2),
                                rec_2_in(3),
                                rec_2_in(4),
                                rec_2_in(5),
                                rec_2_in(6),
                                rec_2_in(7));

bm_r2l_2 : busmacro_xc4v_r2l_async_narrow port map (
                                rec_2_out(0),
                                rec_2_out(1),
                                rec_2_out(2),
                                rec_2_out(3),
                                rec_2_out(4),
                                rec_2_out(5),
                                rec_2_out(6),
                                rec_2_out(7),
                                rec_2(0),
                                rec_2(1),
                                rec_2(2),
                                rec_2(3),
                                rec_2(4),
                                rec_2(5),
                                rec_2(6),
                                rec_2(7));

--modulo 3
bm_l2r_3 : busmacro_xc4v_l2r_async_narrow port map(
                                rec_2(0),
                                rec_2(1),
                                rec_2(2),
                                rec_2(3),
```

```
                                rec_2(4),
                                rec_2(5),
                                rec_2(6),
                                rec_2(7),
                                rec_3_in(0),
                                rec_3_in(1),
                                rec_3_in(2),
                                rec_3_in(3),
                                rec_3_in(4),
                                rec_3_in(5),
                                rec_3_in(6),
                                rec_3_in(7));

bm_r2l_3 : busmacro_xc4v_r2l_async_narrow port map (
                                rec_3_out(0),
                                rec_3_out(1),
                                rec_3_out(2),
                                rec_3_out(3),
                                rec_3_out(4),
                                rec_3_out(5),
                                rec_3_out(6),
                                rec_3_out(7),
                                rec_3(0),
                                rec_3(1),
                                rec_3(2),
                                rec_3(3),
                                rec_3(4),
                                rec_3(5),
                                rec_3(6),
                                rec_3(7));

--modulo 4
bm_l2r_4 : busmacro_xc4v_l2r_async_narrow port map(
                                rec_3(0),
                                rec_3(1),
                                rec_3(2),
                                rec_3(3),
                                rec_3(4),
                                rec_3(5),
                                rec_3(6),
                                rec_3(7),
                                rec_4_in(0),
                                rec_4_in(1),
                                rec_4_in(2),
                                rec_4_in(3),
                                rec_4_in(4),
                                rec_4_in(5),
                                rec_4_in(6),
                                rec_4_in(7));

bm_r2l_4 : busmacro_xc4v_r2l_async_narrow port map (
                                rec_4_out(0),
                                rec_4_out(1),
                                rec_4_out(2),
                                rec_4_out(3),
```

```
                                    rec_4_out(4),
                                    rec_4_out(5),
                                    rec_4_out(6),
                                    rec_4_out(7),
                                    rec_res(0),
                                    rec_res(1),
                                    rec_res(2),
                                    rec_res(3),
                                    rec_res(4),
                                    rec_res(5),
                                    rec_res(6),
                                    rec_res(7));

        --salida de los leds
        leds <= leds_sal;

end Behavioral;
```

Archivo .ucf de Restricciones

```
NET "clk" LOC = AE14;
NET "rst" LOC = E7;
NET "tx" LOC = W1;
NET "clkt" LOC = D2;
NET "dato" LOC = G9;

NET "leds<0>" LOC = E2;
NET "leds<1>" LOC = E10;
NET "leds<2>" LOC = A5;
NET "leds<3>" LOC = F9;
NET "leds<4>" LOC = C6;
NET "leds<5>" LOC = G5;
NET "leds<6>" LOC = G6;
NET "leds<7>" LOC = A11;

INST "modulorec_leds" AREA_GROUP = "REGION0";
AREA_GROUP "REGION0" RANGE = SLICE_X40Y84:SLICE_X43Y91;
AREA_GROUP "REGION0" MODE = RECONFIG;
AREA_GROUP "REGION0" GROUP = CLOSED;

INST "modulorec_1" AREA_GROUP = "REGION1";
AREA_GROUP "REGION1" RANGE = SLICE_X26Y102:SLICE_X31Y121;
AREA_GROUP "REGION1" MODE = RECONFIG;
AREA_GROUP "REGION1" GROUP = CLOSED;

INST "modulorec_2" AREA_GROUP = "REGION2";
AREA_GROUP "REGION2" RANGE = SLICE_X26Y70:SLICE_X31Y89;
AREA_GROUP "REGION2" MODE = RECONFIG;
AREA_GROUP "REGION2" GROUP = CLOSED;
```

```
INST "modulorec_3" AREA_GROUP = "REGION3";
AREA_GROUP "REGION3" RANGE = SLICE_X26Y40:SLICE_X31Y59;
AREA_GROUP "REGION3" MODE = RECONFIG;
AREA_GROUP "REGION3" GROUP = CLOSED;

INST "modulorec_4" AREA_GROUP = "REGION4";
AREA_GROUP "REGION4" RANGE = SLICE_X26Y2:SLICE_X31Y21;
AREA_GROUP "REGION4" MODE = RECONFIG;
AREA_GROUP "REGION4" GROUP = CLOSED;

INST "bm_l2r_leds" LOC = "SLICE_X42Y88";
INST "bm_r2l_leds" LOC = "SLICE_X42Y86";

INST "bm_l2r_1" LOC = "SLICE_X24Y106";
INST "bm_r2l_1" LOC = "SLICE_X24Y104";

INST "bm_l2r_2" LOC = "SLICE_X24Y74";
INST "bm_r2l_2" LOC = "SLICE_X24Y72";

INST "bm_l2r_3" LOC = "SLICE_X24Y44";
INST "bm_r2l_3" LOC = "SLICE_X24Y42";

INST "bm_l2r_4" LOC = "SLICE_X24Y6";
INST "bm_r2l_4" LOC = "SLICE_X24Y4";
```

Archivo CR_Fijo.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.fijo_pac.ALL;

entity CR_fijo is
   Port (          clk, rst, clkt, dato : in  STD_LOGIC;
                                   dato_a : in std_logic_vector(7 downto 0);
                                   tx : out STD_LOGIC);
end CR_fijo;

architecture Behavioral of CR_fijo is

        constant clken : std_logic := '1';
        constant dato1 : std_logic_vector(7 downto 0) := X"36";

        signal bin_ascii1, bin_ascii2, datos : std_logic_vector(7 downto 0);
```

```
        signal res : std_logic_vector(3 downto 0);
        signal clk_uart : std_logic;

begin
        res <= dato_a(7 downto 4) or dato_a(3 downto 0);

        ctr_reloj : reloj port map (clk, rst, clken, clk_uart);
        mux : mux8_2 port map (bin_ascii2, bin_ascii1, clken, datos);
        cod1 : tec2ascii port map (dato1, bin_ascii1);
        cod2 : bin2ascii port map (res, bin_ascii2);
        transmisor : RS232 port map (clk_uart, rst, datos, tx);
end Behavioral;
```

Archivo modulorec_leds.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modulo_rec_leds is
   Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
          b : out  STD_LOGIC_VECTOR (7 downto 0));
end modulo_rec_leds;

architecture Behavioral of modulo_rec_leds is
begin

        b <= a + "01010100";

end Behavioral;
```

Archivo modulo_rec_1.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modulo_rec_1 is
   Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
          b : out  STD_LOGIC_VECTOR (7 downto 0));
end modulo_rec_1;

architecture Behavioral of modulo_rec_1 is
begin
```

```
        b <= a + "00000010";

end Behavioral;
```

Archivo modulo_rec_2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modulo_rec_2 is
   Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
          b : out  STD_LOGIC_VECTOR (7 downto 0));
end modulo_rec_2;

architecture Behavioral of modulo_rec_2 is
begin

        b <= a + "00000010";

end Behavioral;
```

Archivo modulo_rec_3.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modulo_rec_3 is
   Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
          b : out  STD_LOGIC_VECTOR (7 downto 0));
end modulo_rec_3;

architecture Behavioral of modulo_rec_3 is
begin

        b <= a + "00000010";

end Behavioral;
```

Archivo modulo_rec_4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modulo_rec_4 is
   Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
          b : out  STD_LOGIC_VECTOR (7 downto 0));
end modulo_rec_4;

architecture Behavioral of modulo_rec_4 is
begin

              b <= a + "00000010";

end Behavioral;
```

GLOSARIO

ANTIFUSIBLE	Medio de programación de FPGA en el cual las conexiones entre los elementos que conforman al FPGA son creados mediante aumentos en la corriente del sistema.
ARQUITECTURA DE COMPUTADORA	La arquitectura de una computadora explica la situación de sus componentes y permite determinar las posibilidades de que un sistema informático, con una determinada configuración, pueda realizar las operaciones para las que se va a utilizar.
BITSTREAMS	Cadena de bits que se emplean para configurar un FPGA.
BUS MACRO	Puente de comunicación entre las unidades reconfigurables y el resto del sistema, son macros pre-ruteadas y pre-mapeadas.
CORES	Núcleos o módulos que desempeñan una tarea en específico.
DISEÑO MODULAR	Diseño que consiste en dividir el problema principal en subproblemas, cada uno de ellos enfocándose a una tarea concreta.
FLASH	Tipo de memoria no volátil
HARDWARE ESTÁTICO	Hardware que no es variable en el tiempo, es decir, que la función que desempeña siempre es la misma en cualquier punto del tiempo.
HARDWARE RECONFIGURABLE	Es aquél que viene descrito mediante un lenguaje de descripción de hardware. Su naturaleza es completamente diferente a la del hardware estático. Se desarrolla de una manera muy similar a como se hace con el software.
LÓGICA PROGRAMABLE	Lógica que es empleada por dispositivos programables.
LÓGICA RECONFIGURABLE	Lógica que puede variar su función en el tiempo, a diferencia de la lógica programable que solo se puede emplear una vez.
MAPEAR (MAPEADA, MAPEO)	Acción por la cual un diseño digital sintetizado es ubicado dentro de un dispositivo reconfigurable.
MULTIPLEXOR	Dispositivo que puede recibir varias entradas y transmitirlas por un medio de transmisión compartido.

PARALELISMO	Forma en la cual varios cálculos o procesos pueden realizarse simultáneamente.
PROGRAMACIÓN DINÁMICA	Programación de dispositivos donde éste es programado mientras está funcionando, no es necesario detenerlo para realizar la programación.
PROGRAMACIÓN ESTÁTICA	Programación de dispositivos donde éste debe de ser parado completamente antes de poder ser programado.
RECONFIGURACIÓN	Característica de algunos dispositivos digitales como FPGA o CPLD que les permite se configurados totalmente un gran número de veces.
RECONFIGURACIÓN DINÁMICA PARCIAL	Característica que presentan algunos FPGA que les permite modificar secciones sin necesidad de detener todo el dispositivo para realizar la reconfiguración.
RUTEO	Dirigir la información que se transmite a través de una red desde su origen hasta su destino, eligiendo el mejor camino posible a través de la/s red/es que los separan.
SLICES	Unidad mínima de un FPGA de Xilinx.
SOFTWARE	Equipamiento lógico o soporte lógico de una computadora que comprende el conjunto de los componentes lógicos necesarios para hacer posible la realización de una tarea específica

SIGLAS

ALU	Unidad Aritmético Lógica
ASIC	Circuitos Integrados de Aplicación Específica
BRAM	Bloque de memoria RAM
CLB	Bloques Lógicos Configurables
CPLD	Dispositivos Lógicos Programables Complejos
CR	Cómputo Reconfigurable
CSoPC	Configurable-System-On-Programable-Chip
EEPROM	ROM Programable y Borrable Eléctricamente
GPP	Procesadores de Propósito General
FPGA	Arreglos de Compuertas Programables por Campo
GSM	Sistema Global para las Comunicaciones Móviles
HDL	Lenguajes de Descripción Hardware
IOB	Bloques de Entrada y Salida
LAB	Bloque de Arreglos de Elementos Lógicos
LC	Celda Lógica
LE	Elemento Lógico
LUT	Tablas de Búsqueda
MCSoPC	Multiprocessor-Configurable-System-On-Programable-Chip
PAL	Matrices de Lógica Programable
PC	Computadora Personal
RAM	Memoria de Acceso Aleatorio
RTR	Reconfiguración en Tiempo de Ejecución
SoC	System-on-Chip
SoPC	System-on-Programmable-Chip
SRAM	RAM Estática
VLSI	Circuito Integrado con Alta Escala de Integración
WCDMA	Acceso Múltiple por División de Código de Banda Ancha
WiMax	Interoperabilidad Mundial para Acceso por Microondas

WLAN	Red de Área Local Inalámbrica