Instituto Politécnico Nacional

Centro de Investigación en Computación

Secretaría de Investigación y Posgrado

ADMINISTRACIÓN DE RECURSOS HARDWARE EN ARQUITECTURAS RECONFIGURABLES

T E S I S

QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS EN INGENIERÍA DE
CÓMPUTO CON OPCIÓN EN SISTEMAS DIGITALES

P R E S E N T A

ING. JOSÉ DE JESÚS MATA VILLANUEVA

DIRECTOR (ES) DE TESIS:
Dr. Marco Antonio Ramírez Salinas
M. en C. Osvaldo Espinosa Sosa

MÉXICO, D.F. 2012

SIP-14 bis

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 10:00 horas del día 7 del mes de diciembre de 2012 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Administración de recursos hardware en arquitecturas reconfigurables"

Presentada por el alumno:

MATA	**VILLANUEVA**	**JOSÉ DE JESÚS**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

B	1	0	1	8	8	0

aspirante de: **MAESTRÍA EN CIENCIAS EN INGENIERÍA DE CÓMPUTO CON OPCIÓN EN SISTEMAS DIGITALES**

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

LA COMISIÓN REVISORA

Directores de Tesis

Dr. Marco Antonio Ramírez Salinas

M. en C. Osvaldo Espinosa Sosa

Dr. Adolfo Guzmán Arenas

Dr. Amadeo José Argüelles Cruz

Dr. José Luis Oropeza Rodríguez

PRESIDENTE DEL COLEGIO DE PROFESORES

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION
DIRECCION

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 10 del mes Diciembre del año 2012, el que suscribe José de Jesús Mata Villanueva alumno del Programa de Maestría en Ciencias en Ingeniería de Cómputo con Opción en Sistemas Digitales con número de registro B101880, adscrito a Centro de Investigación en Computación, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección de Dr. Marco Antonio Ramírez Salinas y M. en C. Osvaldo Espinosa Sosa y cede los derechos del trabajo intitulado "Administración de recursos hardware en arquitecturas 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 jjmata@ieee.org. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

JJMV

José de Jesús Mata Villanueva
Nombre y firma

RESUMEN

En esta tesis se describe un método de administración de recursos de hardware reconfigurable para Sistemas inmersos en Chips Programables, también conocidos como **SoPC** por sus siglas en inglés, que hace uso de técnicas de reconfiguración parcial en tiempo de ejecución. Este método se ha implementado en un SoPC basado en un dispositivo FPGA Virtex 5 de la empresa Xilinx como plataforma para la prueba de concepto.

El método maneja exitosamente los conceptos de *slot* reconfigurable, tarea de hardware relocalizable de tamaño y cantidad variable de puertos, área reconfigurable heterogenea, división del área en intervalos libres y multitarea en software y hardware.

Como una fase preliminar al desarrollo del método de administración, en este trabajo se presenta un mecanismo para implementar tareas de hardware con las características previamente mencionadas, el cual se basa en el uso de interfaces intermedias de localización conocida y un mecanismo adicional para el enrutamiento de las señales, con lo que se generan rutas dedicadas y compatibles entre las interfaces de los *slots* reconfigurables. Para el desarrollo de estos mecanismos se han desarrollado un par de herramientas de software. Esta forma de implementación asegura la compatibilidad entre todas las particiones reconfigurables del SoPC y las tareas de hardware que en él puedan ser configuradas.

Estos dos mecanismos abren caminos de desarrollo en investigación en arquitecturas reconfigurables que van desde el desarrollo de nuevas herramientas de diseño asistido por computadora (CAD) hasta el diseño y desarrollo de sistemas tolerantes a fallos.

ABSTRACT

This thesis describes the development of a method for hardware resources administration in a System on a Programmable Chip (SoPC) able to perform Partial Run Time Reconfiguration. This method is implemented on a SoPC based in Virtex 5 FPGA by the Xilinx manufacturer as a proof of concept.

The administration method successfully manages concepts like reconfigurable slot, variable size and ports relocatable hardware tasks, heterogeneous reconfigurable area, free intervals division of the area and hardware and software multitasking.

As a preliminary step to the development of the administration method, this work introduces an implementation mechanism for tasks with the previously mentioned characteristics. This mechanism is based on the use of known location intermediate interfaces and on two routing tools able to generate dedicated and compatible routes among the interfaces of the reconfigurable slots. This implementation method guarantees the compatibility of all the reconfigurable partitions available in the SoPC and all the hardware tasks implemented on it.

This two methods open up development paths on reconfigurable architecture subjects ranging from Computer Assisted Design tools, to the design and development of fault tolerant systems.

Agradecimientos

You owe it to all of us to get on with what you're good at
– W. H. Auden

A mi familia: Roge, Fili, Adri y Elsa.

A mis amigos de todos los tiempos (sin orden particular, salvo mujeres primero): Aidee, Alhelí, Paty, Tonatiuh, Mick, Bernardo, Rubén, Cesar, Ernesto, Adrian, J. Adrian y Enrique.

A la generación B10 de MCIC en particular a Alejandro, Rodolfo y Sergio a quienes quiero contar entre mis amigos.

A todos los profesores de quienes recibí clases en el CIC, en particular a mis asesores en este trabajo Dr. Marco Ramírez, M. en C. Osvaldo Espinosa y al Dr. José Luis Oropeza.

A todos ustedes mi agradecimiento, la mejor versión de mi mismo y estas páginas.

Sinceramente,

José de Jesús Mata Villanueva

Índice de contenido

Índice de ilustraciones

Índice de tablas

Índice de Listados

GLOSARIO

FPGA

Del inglés *Field Programmable Gate Array* es un dispositivo electrónico que contiene recursos lógicos y de interconexión que pueden ser programados para implementar funciones lógicas de complejidad variable.

Reconfiguración parcial

La reconfiguración parcial es la modificación de la lógica de una región de recursos de un dispositivo FPGA en tiempo de ejecución mediante la descarga de un archivo de configuración parcial.

Partición

Una partición es una sección lógica de un diseño que el diseñador define. Cada partición es implementada o preservada de una implementación previa. Una partición preservada conserva su funcionamiento y su implementación idénticas.

Partición reconfigurable

El atributo de particion reconfigurable es asignado a una instancia de un diseño, con lo que se indica que esa instancia es reconfigurable y que las herramientas de diseño deben tratarla como tal.

Pines de partición

Los pines de partición son la interfaz lógica y física entre entre la lógica estática y la lógica reconfigurable. Son instanciados automáticamente durante el flujo de diseño.

Frame

Mínima sección direccionable de la memoria de configuración del FPGA.

Configuración

Dentro del flujo de diseño para reconfiguración parcial, una configuración es un diseño

completo con un módulo reconfigurable por cada partición reconfigurable. Un diseño de reconfiguración parcial puede constar de varias configuraciones que generan un *bitstream* completo y un bitstream parcial por cada partición reconfigurable.

CAPÍTULO 1. INTRODUCCIÓN

1.1 ANTECEDENTES

La llegada de los Arreglos de Compuertas Programables en Campo (FPGA) al ámbito de la computación estableció un área de investigación ampliamente abordada en años recientes, conocida en general como cómputo reconfigurable (RC). Este interés está fundado en la probada capacidad de los sistemas de hardware dedicado para acelerar aplicaciones de cómputo intensivo, tradicionalmente implementadas en software, como: algoritmos de procesamiento digital de señales, criptografía, reconocimiento de patrones; entre otros. Los sistemas RC ofrecen, en comparación con sistemas basados en Circuitos Integrados de Aplicación Específica (ASIC), flexibilidad y bajo costo para volúmenes de producción pequeños. Un listado no exhaustivo de las líneas de investigación que involucran el RC incluye: Co-diseño de Hardware + Software (Hw + Sw), compilación de aplicaciones híbridas, técnicas eficientes de *enrutamiento,* mapeo de algoritmos de cómputo intensivo a tecnologías de cómputo reconfigurable, reconfiguración en tiempo de ejecución (RTR), también conocida como reconfiguración parcial dinámica y con ella, la planificación y administración de tareas reconfigurables, sistemas operativos para RC, etc.

Los dispositivos FPGA permiten implementar funcionalidad computacional mediante una serie de recursos programables. En un sistema RC convencional, la información de configuración del dispositivo, obtenida tras el proceso de implementación, es almacenada en su memoria de programación antes de la puesta en marcha del sistema y no se modifica salvo por corrección de errores o actualizaciones. Esta modificación requiere el paro del sistema. La reconfiguración parcial dinámica, en dispositivos que la soportan, ofrece una forma de flexibilizar la funcionalidad de un sistema RC al permitir que determinadas regiones del dispositivo sean modificadas con nueva información de configuración mientras el sistema está en funcionamiento sin interferir en el resto de la configuración. Esta flexibilidad trae consigo una serie de ventajas entre las que se encuentran:

- Posibilidad de modificar la funcionalidad del hardware en tiempo de ejecución.
- Desarrollo de aplicaciones novedosas como Sistemas de Radio de Táctica

Conjunta (JTRS).

- Aumento del rendimiento y disminución de tiempos muertos– Si bien ciertas partes del sistema son reconfiguradas en tiempo de ejecución, el resto (la parte estática) no sufre tiempos de paro. Además para determinadas aplicaciones, puede existir una configuración específica hecha a medida para cada caso posible con optimizaciones de rendimiento.
- Tiempos más cortos de reconfiguración – Si el dispositivo no debe ser configurado en su totalidad, el tiempo de configuración se reduce.
- Mayor flexibilidad para realizar actualizaciones.

De estas características se desprende la posibilidad de desarrollar sistemas modulares basados en FPGA donde determinados módulos sean cargados y descargados del dispositivo de forma dinámica. Mas aún, al poder utilizar estos módulos como co-procesadores que trabajen en conjunto con procesadores de propósito general (GPP), como es habitual en un sistema embebido convencional basado en FPGA, se ha generado la inquietud de desarrollar sistemas híbridos que ocupen *tareas de hardware* (HWTask) como si fuesen tareas de software convencionales. La posibilidad de explotar el dispositivo FPGA de esta forma fue abordada quizá por primera vez por Brebner en [7] con la introducción del concepto de Unidad Lógica Intercambiable (SLU), definida como un circuito lógico capaz de realizar alguna función que recibe datos de entrada y devuelve datos de salida, con las características particulares de que la implementación física de dicho circuito se realiza en un área fija dentro del FPGA y que sus interfaces de entrada y salida están, también, fijas como parte del diseño. El autor presenta a la SLU como el equivalente en sistemas de hardware virtual de las páginas o segmentos de memoria en los sistemas de memoria virtual. Este concepto fue probado entonces con la implementación de un *Sistema Operativo de Hardware Virtual* para el FPGA XC6200 de Xilinx. A partir de este trabajo se han desarrollado diversos esfuerzos enfocados al desarrollo de sistemas de cómputo reconfigurable que implementen tareas de hardware reconfiguración dinámica. El diseño de estos sistemas entraña desafíos particulares pero, como en el caso de las SLU y las páginas de memoria, tienen un análogo en problemas de administración de recursos computacionales convencionales.

En un sistema de cómputo tradicional que implemente multiprogramación, debe haber mecanismos y políticas que aseguren la correcta administración de los recursos del sistema entre las diversas aplicaciones, evitando con ello conflictos que puedan dar lugar a fallos diversos como pérdida de información y largos tiempos de ejecución. En general los recursos que se deben administrar se pueden dividir en recursos de almacenamiento y de entrada/salida (obviando el tiempo de procesador). Esta administración ha sido ampliamente estudiada en textos como Tannenbaum [6]. Como parte de la administración, debe existir un cierto nivel de abstracción que le oculte al programador de aplicaciones los detalles complicados del hardware y sólo le entregue interfaces con las cuales interactuar, que sean fáciles de utilizar para las tareas específicas que desee resolver. El uso de reconfiguración parcial en tiempo de ejecución en una aplicación de cómputo reconfigurable presenta nuevos e importantes retos de planificación, administración y abstracción que han sido estudiados en diversos trabajos de investigación, como en [5], en los que Sabeghi y Bertels listan las actividades de administración más importantes a tener en cuenta en las aplicaciones de cómputo reconfigurable, tales como: ubicación y reubicación de lógica en el FPGA, planificación de la ubicación, enrutamiento, redes de FPGA, abstracciones de tiempo real, Interfaces de Programación de Aplicaciones API y servicios varios.

Dejando de lado entre otros temas, las redes de FPGA y las abstracciones de tiempo real, nos enfocaremos en los problemas de ubicación/reubicación, planificación y enrutamiento, desde un enfoque similar al de un sistema operativo tradicional. En resumen, los problemas relacionados con nuestra propuesta son:

- Ubicación/reubicación de lógica reconfigurable: Implica encontrar el lugar idóneo dentro del dispositivo FPGA, por la disposición y disponibilidad de recursos, para colocar una tarea de hardware que la aplicación requiere en ese momento, de forma tal que los recursos sean adecuadamente aprovechados, evitando traslapes con otros circuitos que ocasionarían como mínimo un mal funcionamiento y en última instancia, daños irreparables al dispositivo. En un ambiente multitarea de reconfiguración parcial, la resolución de este problema se realiza en tiempo de ejecución.

- Planificación: De la mano con el tema de ubicación/reubicación de tareas de hardware, la planificación debe manejar el tiempo de creación y destrucción de las tareas de hardware procurando mantener un balance entre todas las aplicaciones concurrentes. Resalta la importancia del tiempo que una aplicación tarda en ser cargada en hardware, el cual, de ser muy elevado, eclipsa los beneficios que se pudieran obtener de ella en términos de rendimiento.
- Enrutamiento y comunicación: La interconexión de tareas de hardware entre si y, especialmente para el trabajo que nos compete, con el GPP a través de los recursos de interconexión del FPGA es un aspecto importante en general en sistemas reconfigurables y en particular en los parcialmente reconfigurables, pues dependiendo de la estrategia que se siga, contribuye de forma significativa al área total utilizada por las tareas de hardware reconfigurable. Al mismo tiempo, el hecho de aumentar el número de tareas de hardware además de la flexibilidad de su ubicación, vuelve complejo al enrutamiento, más aún si este debe resolverse en tiempo de ejecución.

La investigación en este campo ha sufrido cambios muy diversos y cada vez mas importantes a medida que la tecnología de FPGA ha presentado variaciones. En particular los temas de ubicación y reubicación se han visto muy afectados por estas variaciones. A mediados de la década de 1990, en una propuesta publicada por Bazargan y otros [21], se plantearon una serie de métodos para encontrar el espacio óptimo para la carga de tareas de hardware en sistemas reconfigurables. Desde su aparición este trabajo ha sido tomado como referencia para trabajos posteriores de diversos grupos de investigación. No obstante, es importante hacer notar que el modelo de dispositivo reconfigurable utilizado es un modelo idealizado y que no corresponde con las arquitecturas actuales de dispositivos. Así mismo, no considera el problema de la comunicación de las tareas de hardware con un GPP maestro. Existen una serie de modelos y alternativas para resolver estas dificultades, mismas que se tratarán en el capítulo siguiente, sólo queda hacer hincapié en que estas opciones siempre estarán atadas a las prestaciones que ofrezcan los dispositivos en cuestión.

Es así como los sistemas que soporten reconfiguración parcial dinámica demandan,

según sus características, técnicas equivalentes a las utilizadas en la planificación y administración de recursos computacionales clásicos como son memoria, sistemas de archivos, periféricos y tareas. Se requiere entonces, administrar de forma eficiente los recursos propios de los dispositivos reconfigurables, con base en información proporcionada por las aplicaciones Hw+Sw, acerca de los recursos de hardware necesarios para su ejecución como son: cantidad y tipo de recursos y en su caso latencia y potencia disipada; esta información será utilizada por algún método que garantice dentro de los alcances y necesidades de la aplicación, el aprovechamiento de los recursos reconfigurables.

La tesis que se presenta se enmarca dentro de este contexto, en las líneas de investigación de planificación, administración de tareas y sistemas operativos para arquitecturas reconfigurables en tiempo de ejecución basadas en FPGA y procesadores de propósito general. Con estos antecedentes es posible que el lector pueda introducirse en el problema que se resolverá con esta investigación.

1.2 PLANTEAMIENTO DEL PROBLEMA

Muchos de los métodos desarrollados a la fecha para la búsqueda de recursos libres y ubicación de tareas de hardware en un FPGA en tiempo de ejecución pueden ser considerados incompletos y sin correspondencia con el estado actual de la tecnología de FPGA. Es común encontrar trabajos que incurren en omisiones en sus modelos de FPGA. Omisiones tales como:

- Considerar al dispositivo como un conjunto de recursos del mismo tipo distribuidos en dos dimensiones de forma homogénea.
- No tomar en cuenta la comunicación con un procesador maestro o entre módulos, según el objetivo del sistema.
- Ignorar la granularidad de la reconfiguración de los dispositivos.

Estas omisiones vuelven a los métodos desarrollados en tales trabajos inaplicables, o cuando menos difícilmente adaptables, a las arquitecturas de FPGA actualmente en uso.

Tomando en cuenta lo anteriormente expuesto, en este trabajo se desarrolla un método

para la administración de recursos de hardware en una arquitectura reconfigurable. Este método considera el problema de ubicación de tareas de hardware reconfigurable en un modelo realista, considerando la granularidad de la reconfiguración y la comunicación de las tareas con un procesador de propósito general. El problema se ataca desde dos frentes: la implementación del hardware necesario para soportar este modelo y la generación de un modulo de software de sistema operativo con capacidad de administrar los recursos de hardware de este modelo, partiendo de las limitaciones y alcances de la tecnología de reconfiguración parcial y basándose en trabajos previos de diversos grupos de investigación y los aportes realizados en diversas tesis de grado, por alumnos graduados del Laboratorio de Micro-tecnología y Sistemas Embebidos del CIC-IPN en los campos de sistemas operativos para arquitecturas reconfigurables y reconfiguración parcial dinámica, así como la literatura reciente sobre el tema.

La plataforma de desarrollo para la prueba de concepto de esta investigación es modelo ML507 con un FPGA Virtex 5 de la empresa Xilinx, el dispositivo cuenta con procesador embebido PowerPC y capacidades de reconfiguración del área FPGA en tiempo de ejecución a través del puerto de acceso a configuración interna conocido como ICAP por sus siglas en inglés. Una arquitectura genérica para este tipo de dispositivos puede verse en la Ilustración 1-1.

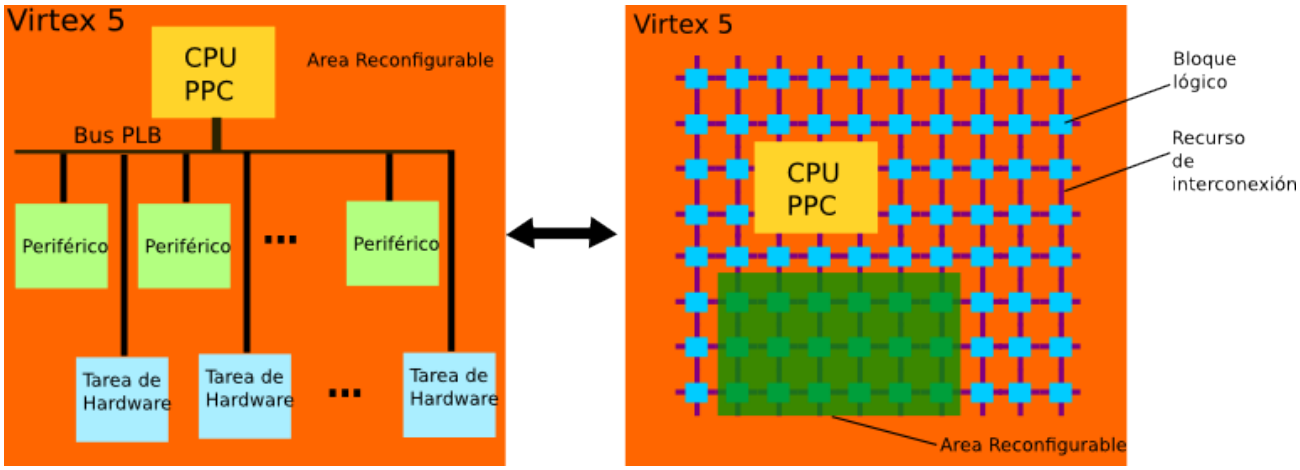

Ilustración 1-1: Modelo de arquitectura de cómputo reconfigurable

El desarrollo de la investigación hace uso de la arquitectura antes descrita, en donde se busca integrar capacidades de reconfiguración y administración del área de hardware

reconfigurable (en la Ilustración 1-1, representada por el área sombreada). Se requiere desarrollar un modelo para signar tareas hardware reconfigurable, es necesario un método para soportar la re-ubicación de las tareas de hardware, y como parte central de la tesis, se plantea el desarrollo de un administrador de recursos reconfigurables que a solicitud de programas de aplicación, ubique una tarea reconfigurable en el dispositivo FPGA si las condiciones necesarias se cumplen. Se considera la disponibilidad y correspondencia de recursos, así como la no interferencia con otras tareas de hardware. Es decir la cantidad de recursos, heterogeneidad del dispositivo y granularidad de la configuración.

El administrador propuesto se muestra en el diagrama de bloques de la ilustración 1-2 y su fucionamiento será explicado en el capítulo 4.

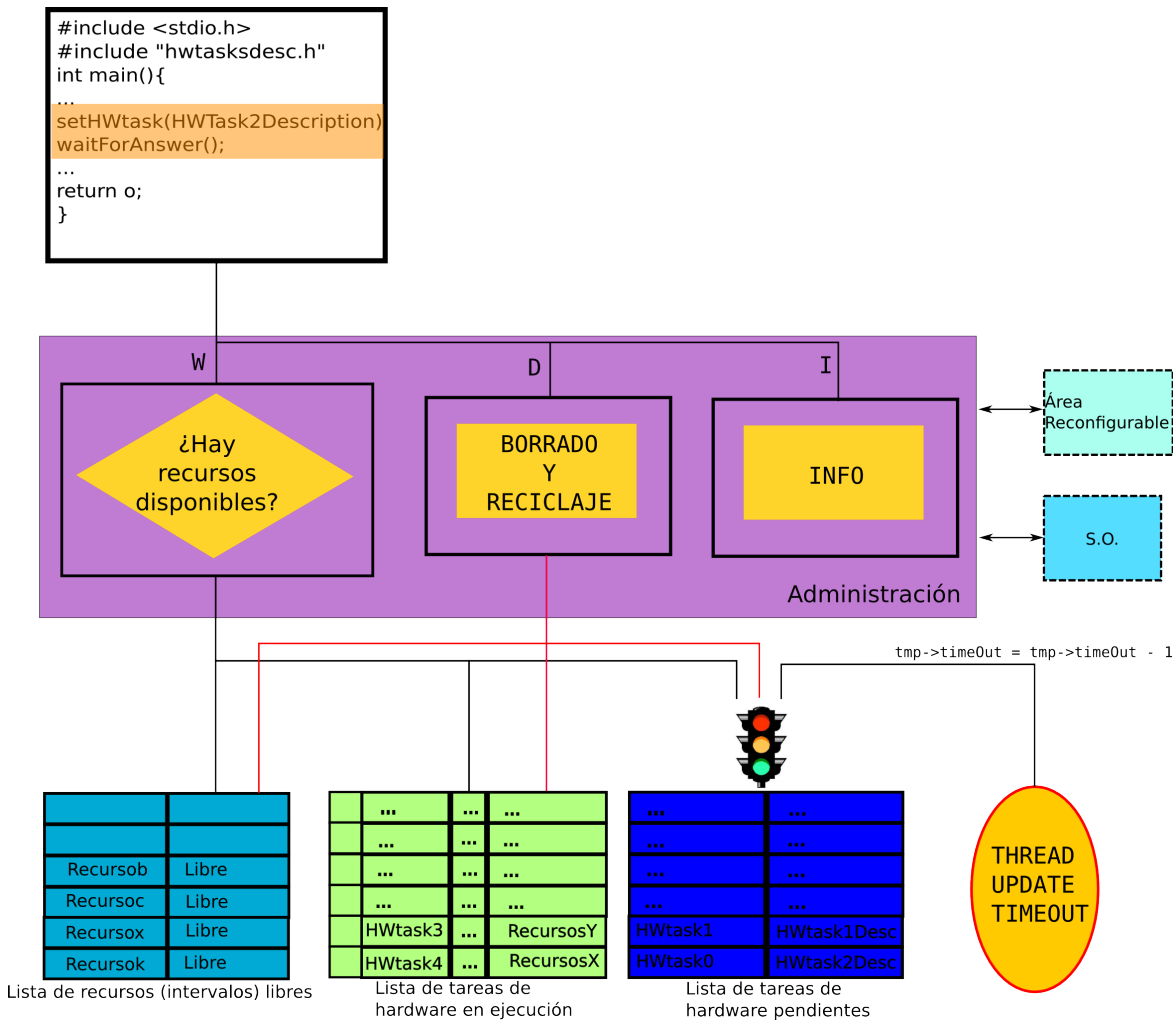

Ilustración 1-2: Diagrama de bloques del administrador de recursos

Así mismo, en el marco del tema de tesis, se requiere desarrollar un conjunto de herramientas útiles para la interpretación de archivos de configuración de los dispositivos FPGA de la empresa Xilinx.

1.3 JUSTIFICACIÓN

Los problemas de administración de recursos de hardware en arquitecturas reconfigurables en tiempo de ejecución tienen que ser resueltos para facilitar el uso de esta tecnología en aplicaciones de cómputo tradicional, diversificar su campo de acción, popularizarla y permitir que sus bondades sean aprovechadas en mayor cantidad de aplicaciones. Esto debe hacerse desde un enfoque realista anclado en el estado actual de la tecnología de dispositivos FPGA que permita el desarrollo de pruebas de concepto que sustenten el enfoque. La industria informática apunta hacia estas arquitecturas para resolver problemas de rendimiento en aplicaciones de cómputo de alto desempeño y sistemas embebidos, se puede observar esta tendencia con la introducción de nuevos sistemas basados en microprocesador y FPGA por parte de empresas como Intel. Así mismo, el uso de esta tecnología ha sido planteado en años recientes como una buena alternativa para el desarrollo de aplicaciones tolerantes a fallos físicos del dispositivo, como se requiere en aplicaciones para la industria aeroespacial, en las que los dispositivos están sujetos a radiaciones. Es en este marco que esta investigación adquiere relevancia.

Aunado a las razones expuestas, la creación de herramientas de análisis para arquitecturas reconfigurables permitirá seguir avanzando en la investigación en este campo y diversificar los temas que se aborden en los proyectos de investigación de nuestro laboratorio.

1.4 OBJETIVOS DEL TRABAJO

1.4.1 OBJETIVO GENERAL

Desarrollar un método de administración de recursos de hardware en arquitecturas dinámicamente reconfigurables.

1.4.2 OBJETIVOS PARTICULARES

- Estudiar la estructura de los archivos de configuración de hardware *bitstream* para el dispositivo FPGA Virtex 5.
    - Desarrollar herramientas de análisis de archivos de configuración de hardware *bitstream* para la arquitectura Virtex 5.
    - Desarrollar un módulo de software para llevar a cabo tareas de relocalización de hardware a partir de archivos de configuración *bitstream* parciales.
- Implementar una plataforma de hardware que soporte reconfiguración y relocalización de tareas de hardware.
    - Definir un modelo de ejecución de tareas, área de reconfiguración y una interfaz de comunicación entre la tarea reconfigurable y procesador de propósito general.
- Desarrollar mecanismo de administración de recursos basado en los modelos de área y tarea reconfigurable definidos previamente.
- Implementar reconfiguración parcial dinámica de tareas de hardware ordenada por el procesador embebido con el método y modelos desarrollados.
    - Generar una aplicación híbrida de prueba.
- Integrar y evaluar los elementos mencionados en los objetivos previos.

1.5 ALCANCES

Esta investigación toma en cuenta las limitaciones citadas en el planteamiento del problema y se enfoca en la ubicación de tareas en espacios (*slots*) de tamaño fijo y localización base conocida. La limitación a este tipo de arquitectura obedece a la necesidad de comunicar las tareas de hardware con el procesador embebido por medio del bus del sistema. Un esquema diferente obligaría a generar soluciones de comunicación más intrincadas y no necesariamente más convenientes como el desarrollo de técnicas de enrutamiento dinámico o soluciones no ortodoxas como el uso de la *capa de configuración* del dispositivo como puente entre el procesador embebido y tareas no conectadas al mismo.

1.6 CONTRIBUCIONES

Se espera contribuir al campo de las arquitecturas reconfigurables con un método de implementación de plataformas de hardware que soporten tareas reconfigurables con capacidad de relocalización y un método novedoso de administración de recursos para aplicaciones dinámicamente reconfigurables basado en un modelo realista así como colaborar en el afianzamiento en nuestro Centro de Investigación, de las bases ya establecidas de la investigación en arquitecturas reconfigurables, especialmente en sistemas parcialmente reconfigurables, sistemas operativos orientados a cómputo reconfigurable y sistemas embebidos. Esto no sólo mediante el método de administración propuesto, sino con el desarrollo de las herramientas de análisis para arquitecturas reconfigurables que en el transcurso del trabajo se hicieron necesarias.

1.7 ESTRUCTURA DE LA TESIS

El resto de este trabajo está estructurado como sigue:

El capítulo 2 presenta el estado del arte de las áreas mas afines a este trabajo, especialmente los sistemas operativos reconfigurables, la administración de recursos y las técnicas de implementación.

El capítulo 3 presenta el marco teórico, enfocándose a los flujos de implementación de sistemas reconfigurables y sus limitaciones.

El capítulo 4 presenta el desarrollo de hardware y software de esta propuesta, partiendo del análisis de las limitaciones que la tecnología de reconfiguración parcial de FPGA impone. Se presentan las propuestas de solución a estas limitaciones y su implementación.

El capítulo 5 presenta las pruebas realizadas a este desarrollo y sus resultados.

El capítulo 6 presenta una serie de conclusiones y de trabajos propuestos a futuro.

CAPÍTULO 2. ESTADO DEL ARTE

2.1 INTRODUCCIÓN

El cómputo reconfigurable es un área con mucha actividad de investigación. Temas como el diseño de celdas básicas eficientes, localización de circuitos, *particionado* en módulos reconfigurables, enrutamiento eficiente, implementación de algoritmos de diversas ramas de computación en dispositivos reconfigurables, co-diseño de hardware y software, etc., son sólo algunas de las líneas de investigación derivadas de este campo. En este marco, los sistemas reconfigurables en tiempo de ejecución y en particular la implementación de sistemas multitarea ha traído consigo nuevos temas alrededor de los cuales se está realizando investigación. Un tema de interés para este trabajo es el desarrollo de sistemas operativos para este tipo de plataformas. Esta línea de investigación es relativamente reciente y aún no hay nada definitivo a pesar de que se han planteado varias propuestas y se han desarrollado diferentes prototipos. Otra línea de investigación relacionada, son los métodos y herramientas para la implementación de sistemas que soporten reconfiguración en tiempo de ejecución. Este trabajo se encuentra dentro de estas dos líneas de investigación y para su desarrollo se deben considerar los siguientes aspectos del problema.

- Abstracciones
    - Tareas hardware
- Administración de recursos
    - Localización y re-localización de tareas de hardware en la lógica del dispositivo.
    - Planificación de tareas
    - Enrutamiento
- Servicios.
    - Carga y descarga de tareas
    - Comunicación

- Detención y reinicio de tareas

- Implementación de sistemas reconfigurables

A continuación se presenta un resumen de trabajos previos, desarrollados en los temas mencionados en la lista precedente, comenzando con algunas propuestas de sistemas operativos.

2.2. SISTEMAS OPERATIVOS PARA CÓMPUTO RECONFIGURABLE

En [7] y otros artículos relacionados, Brebner introduce el enfoque de sistemas operativos aplicado al hardware parcialmente reconfigurable. Propone el uso de *swappable logic units* (SLU) como tareas independientes que pueden ser intercambiadas por el sistema operativo. Las SLU son segmentos lógicos rectangulares del mismo tamaño en los que la aplicación puede ser ubicada. El concepto fue extendido en [8] para permitir distintas formas geométricas.

De acuerdo a Kearney y Warren [13], ReconfigME es el primer sistema operativo para sistemas reconfigurables, si se considera al sistema operativo no sólo como un cargador de aplicaciones, sino también un administrador de recursos. De acuerdo con los autores, en investigaciones previas se ha trabajado sobre carga de tareas, interfaz con el sistema anfitrión y en algunos casos en tiempo de ejecución, pero en su opinión, no reunían los requerimientos para poder ser considerados sistemas operativos. La definición de sistema operativo, sin embargo, es bastante difusa como hace notar Tanenbaum en [6]. ReconfigME, no obstante, no trabaja dentro de un sistema embebido, el sistema operativo se ejecuta sobre una PC que controla un FPGA como un periférico.

Jean y otros en [16] discuten un sistema en tiempo de ejecución en el que un administrador de recursos planifica tareas a un conjunto de FPGA. Sin embargo cada tarea ocupa un solo FPGA, por lo que el sistema no enfrenta los problemas de reconfiguración parcial y localización de tareas.

Merino y otros en [17] dividen la superficie reconfigurable en un arreglo de sub-áreas

predefinidas, a las que llaman *slots*. Este sistema planifica tareas a estos *slots* basándose en una tabla de asignación de *slots* a tareas. Cada tarea ocupa un solo *slot*, por lo que el tipo de tareas que se pueden asignar a los *slots* está limitado por el tamaño del *slot*.

Simmler y otros en [18] tratan el problema de la conmutación de tareas en un entorno *preventivo* (*preemptive*). Llegando incluso a proponer los periodos en los que las tareas no pueden ser retiradas. Sin embargo, tampoco trabajan con reconfiguración parcial.

Burns y otros en [8] describen funciones de sistema operativo capaces de trasladar y rotar tareas de hardware para lograr un ajuste óptimo de la tarea al dispositivo.

Autores como Shirazi en [19] han trabajado en los compromisos entre el tiempo de reconfiguración y la calidad de los circuitos en relación con el método usado para la configuración y la información acerca de la secuencia de configuración disponible en tiempo de compilación.

Un esfuerzo importante en sistemas operativos para dispositivos reconfigurables es el de Kwok-Hay So con el proyecto BORPH (Berkeley Os for ReProgrammable Hardware), detallado en [14] y artículos relacionados. Con BORPH se trata el tema desde una perspectiva dedicada a mejorar la usabilidad de los sistemas reconfigurables. Los sistemas reconfigurables no han alcanzado gran popularidad debido a la dificultad que representa para programadores de aplicaciones el cambio de paradigma hacia la combinación de hardware con software. El proyecto BORPH plantea el cambio de la idea de “tareas hardware”, por “procesos hardware”, asiéndose a la semántica de los sistemas Unix, familiar para un buen número de programadores. A grandes rasgos el proyecto consiste en la adaptación de un kernel Linux para incluir abstracciones de hardware reconfigurable, darle los mismos servicios a un proceso hardware que a uno software (acceso a memoria, entrada/salida, acceso al sistema de archivos, etc.) y en esencia, no reconocer diferencia entre uno y otro. Crea además una interfaz de llamadas al sistema orientadas a hardware reconfigurable, ofreciendo como menciona Tanenbaum en [6] abstracciones manejables para una tarea complicada. Al ser una plataforma Linux, es transportable; actualmente trabaja sobre una plataforma BEE2, que utiliza una red de 4 FPGA. Usa el concepto de región hardware, en cada una de las cuales puede cargarse una configuración como proceso hardware. Para el caso de la BEE2, cada FPGA es una

región hardware y en palabras del autor, el portar el sistema a otra plataforma implica la definición de la región hardware que se utilizará. Otro punto importante es la definición de un tipo de archivo ejecutable híbrido (.BOF), que contiene tanto la información del programa software, como las configuraciones hardware que el ejecutable necesitará.

Steiger y otros en [20] plantean atacar un sistema con reconfiguración parcial modelando el área del dispositivo como un sistema de una sola dimensión. Su enfoque es similar al uso de *slots*, en el sentido de restringir la forma en que el dispositivo es reconfigurado. Parten el área reconfigurable en columnas que pueden ser repartidas en dirección horizontal entre diferentes tareas que, por supuesto, medirán a lo ancho, alguna cantidad de columnas. Su trabajo es relevante pues a diferencia de los trabajos anteriores, trabaja con reconfiguración parcial dinámica y plantean todos los módulos necesarios para tal fin. No obstante, un tema que no explican es como se realiza la implementación del sistema que soporta estas características, ni trabajan con la heterogeneidad del dispositivo. A nivel de simulación también trabajan con un sistema en dos dimensiones. No obstante, para su implementación en prototipo, el uso de un sistema donde las tareas solo pueden ser ubicadas sobre un eje horizontal, tiene una razón de ser y se comentará mas adelante.

2.3 TAREAS EN HARDWARE

Una tarea en hardware es un circuito digital sintetizado, pre-localizado y pre-enrutado. La tarea se guarda en un formato independiente de la posición y puede ser re-localizado a diferentes posiciones dentro del dispositivo reconfigurable por un sistema operativo. Estas tareas tienen varias características, su funcionamiento, por ejemplo, el cual no es relevante para el sistema operativo. Por otro lado, sus características estructurales y de tiempo de ejecución (si la aplicación depende de los tiempos de ejecución) son necesarios para que el sistema operativo planifique su ejecución y localización.

Las características estructurales más importantes son el tamaño y la forma. Cada tarea de hardware requiere una cierta área para poder ser implementada, esta área está dada en recursos lógicos reconfigurables (típicamente, los Bloques Lógicos Configurables, CLB). La forma en general se modela como un rectángulo que incluye los recursos reconfigurables y los recursos de enrutamiento usados por la tarea. No obstante existen otros modelos más complejos descritos en [10]. El modelo rectangular simplifica la

localización de las tareas, sin embargo provoca fragmentación del dispositivo, es decir desperdicio de área no utilizada. Se han hecho propuestas para atacar el problema de la fragmentación mediante reubicación de recursos lógicos en [10] y [11], pero estas técnicas dependen de las posibilidades que ofrezca el dispositivo para reconfigurar los recursos. En general, el uso de estas técnicas depende de la granularidad de la reconfiguración. Término que se explicará en el capítulo 3.

2.4. ADMINISTRACIÓN DE RECURSOS

La complejidad del mapeo de tareas a dispositivos depende fuertemente del modelo de área utilizado. Los dos modelos principales son los modelos 1D y 2D mostrados en la Ilustración 2-1.

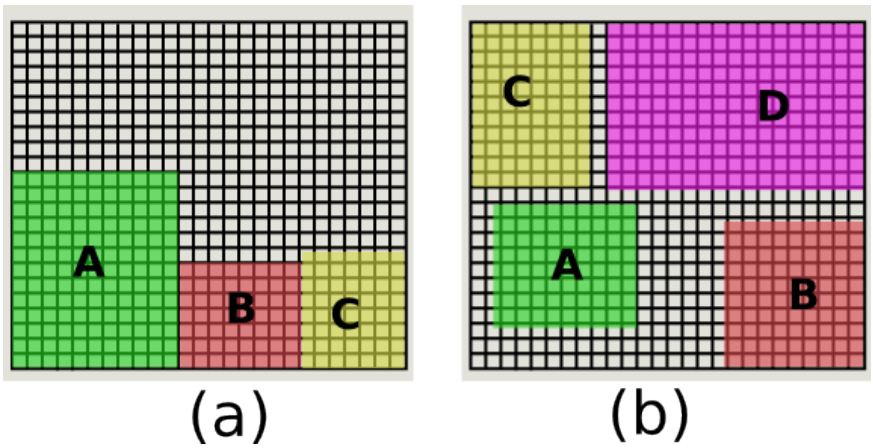

Ilustración 2-1: Modelos de área 1D y 2D

En ambos modelos el dispositivo reconfigurable es representado por un área rectangular de recursos reconfigurables. En el modelo 1D las tareas pueden ser asignadas en cualquier punto a lo largo de la dimensión horizontal. La dimensión vertical es fija y abarca la altura total del área que haya sido definida como reconfigurable. Este modelo simplifica la planificación y localización de tareas pero sufre de fragmentación.

Barzagan y otros en [21] tratan el modelo 2D y presentan estructuras de datos y algoritmos eficientes para localización rápida en tiempo de ejecución. Steiger y otros en [20] parten del trabajo de Barzagan para mejorarlo y experimentar con tareas rectangulares compuestas de sub-tareas. Es importante mencionar que el modelo 2D no

es soportado por la mayoría de los dispositivos reconfigurables modernos. No obstante en el modelado de sistemas reconfigurables, sigue siendo muy popular y utilizado, como se puede corroborar en la literatura. Por su cuenta, se ha demostrado que es mucho mas factible la implementación del modelo 1D que del modelo 2D.

Estos modelos y la relocalización en general, plantean otras dificultades por las consideraciones que hacen. En particular, suponen un área reconfigurable homogénea, es decir que todos los recursos reconfigurables son del mismo tipo. Esto contrasta con los FPGA modernos que cuentan con hardware dedicado como bloques de memoria RAM **BRAM** o bloques de operaciones aritméticas eficientes. Estos recursos no se encuentran mas que en zonas específicas de los dispositivos reconfigurables, por lo que la relocalización de tareas de hardware que utilicen estos módulos se encuentra limitada.

Dependiendo de la complejidad que el sistema requiera, las tareas requieren utilizar algún tipo de interfaz avanzada con el sistema operativo, en general una capa de abstracción estándar alrededor de ellas que permita manejar a todas las tareas de la misma forma. Huang y otros en [22] de manera particular plantean una interfaz interesante que permite, incluso, manejar la tarea como un proceso Unix. Las tareas pueden interrumpirse y es posible guardar y restituir su contexto.

La tarea con su interfaz, aún necesita de una forma de interconexión con el bus del sistema para interactuar con el procesador. El problema del enrutamiento consiste en encontrar la manera de generar estas conexiones. El enrutamiento consume recursos que ocupan área. Algunos trabajos han considerado que en general los recursos de comunicación están disponibles en cantidades suficientes en todo momento o que la comunicación se realiza por los puertos de configuración mediante una técnica conocida como *readback*, o proponen dejar espacio entre las tareas para realizar las conexiones [10]. En un modelo 1D es posible realizar las conexiones con el estado actual de las tecnologías reconfigurables. Sin embargo en modelos 2D el problema aún no está resuelto. Así pues, es importante la investigación en ambos modelos y es factible realizar prototipos para modelos 1D, mientras para modelos 2D, realizar simulaciones al menos en este punto, como lo muestra la literatura, es lo más común.

2.5 IMPLEMENTACIÓN DE SISTEMAS RECONFIGURABLES

Como se mencionó al discutir la importancia del trabajo de Steiger y otros en [20], hay un punto que no clarifican: la implementación específica de un sistema como el que proponen en un dispositivo. No se pone en duda que lo hayan realizado y probablemente para el estado de las herramientas de implementación del fabricante del dispositivo que utilizaron (Virtex-II de Xilinx) era una implementación quizá obvia o cuando menos soportada. Sin embargo, conforme han ido cambiando las arquitecturas de dispositivos FPGA, el soporte para este tipo de implementaciones se ha eliminado si es que alguna vez existió. Aunque las herramientas soportan el flujo de implementación para reconfiguración parcial, este flujo tiene varias limitaciones: no soporta traslape de particiones reconfigurables ni soporta compatibilidad entre particiones. En la literatura reciente no hay muchos esfuerzos reportados en este sentido. Y aunque en principio el desarrollo de una herramienta completa que soporte un flujo de implementación con las características que este trabajo necesita no es el objetivo primario, si es un problema que se debe considerar si es que se desea llegar a implementar un método de administración de recursos con tareas re-localizables. En [27] Sohanghperwala propone una herramienta (OpenPR) de asistencia para *floorplanning* e implementación de sistemas basados en particiones reconfigurables. Describe la técnica ya conocida en la literatura, de los *anti-núcleos* para evitar el uso de las particiones reconfigurables por rutas de la parte estática del diseño. Esta técnica debería ser capaz de permitir la estandarización del enrutamiento de interfaz de las particiones, que como se explicará en el capítulo 4, es lo que finalmente permite la re-localización de tareas en un modelo en 1D. Sin embargo, dejando de lado la herramienta que se ofrece, la reproducción de la técnica que se utiliza resulta complicada.

Un proyecto en la misma línea que OpenPR es Recobus Builder [31], que mediante la generación de funciones macros para enrutamiento provee otra forma de implementar un sistema con particiones compatibles. Desafortunadamente, esta herramienta en particular solo trabaja con arquitecturas de FPGA hasta Virtex-II. Sería entonces conveniente buscar alguna otra técnica que permita la implementación de este tipo de sistemas con facilidad.

2.6 RESUMEN

Este capítulo presentó las principales ideas que hay alrededor de la implementación de un sistema como el que se intenta realizar en este trabajo. Se mostró que en el campo de los sistemas operativos para sistemas parcialmente reconfigurables dinámicamente, no todo está dicho, es un campo joven aún. Se presentaron los diversos modelos de tareas de hardware y de área reconfigurable y sus diferentes problemas. De acuerdo a este estado del arte, prototipos de sistemas que implementan el modelo 1D son factibles y ya han sido realizados. El modelo 2D presenta mayores complicaciones y aún falta trabajo para llegar a la implementación de prototipos. También se presentaron un par de herramientas orientadas a la habilitación del desarrollo de sistemas que soporten reconfiguración parcial dinámica de tareas re-localizables y compatibles entre sí.

CAPÍTULO 3. MARCO TEÓRICO

El uso de la tecnología de reconfiguración parcial en un dispositivo FPGA requiere un conocimiento detallado del flujo de diseño para estos dispositivos, el formato de los archivos de configuración *bitstream* y su relación con la arquitectura del dispositivo objetivo y el proceso de configuración del mismo. Así también es necesario el conocimiento de herramientas especializadas de manipulación de archivos de diseño. Este conocimiento permite establecer los alcances y limitaciones de las técnicas de reconfiguración dinámica y desarrollar software capaz de manipular esta tecnología tanto en tiempo de diseño como en tiempo de ejecución, propósito de este trabajo.

En el presente capítulo se describe inicialmente el flujo de diseño de un circuito lógico en un dispositivo FPGA para posteriormente detallar el flujo de implementación y archivos de configuración para un diseño parcialmente reconfigurable en la arquitectura de los dispositivos Virtex 5 de la compañía Xilinx. Se describen brevemente las limitaciones que dicho flujo y arquitectura imponen a los módulos reconfigurables (tareas reconfigurables en el desarrollo de esta tesis) y se introducen las herramientas especializadas e ideas que ayudarán a sortear estas dificultades en la implementación presentada en este trabajo.

3.1. FLUJOS DE DISEÑO PARA DISPOSITIVOS FPGA DE XILINX

3.1.1. EL FLUJO GENERAL DE DISEÑO

La implementación de un diseño en FPGA es un proceso de varios pasos que llevan al diseño desde su descripción en alguna de las formas soportadas por el fabricante (lenguajes de descripción de hardware, captura de esquemático, etc) hasta la generación de un archivo de configuración *bitstream* descargable en el dispositivo por alguna de las interfaces soportadas. En general, este proceso, conocido como flujo de diseño, comprende tres etapas: Captura de diseño y síntesis, implementación del diseño y verificación. La ilustración 3-1 muestra una vista general del proceso. Estas estapas son descritas a continuación.

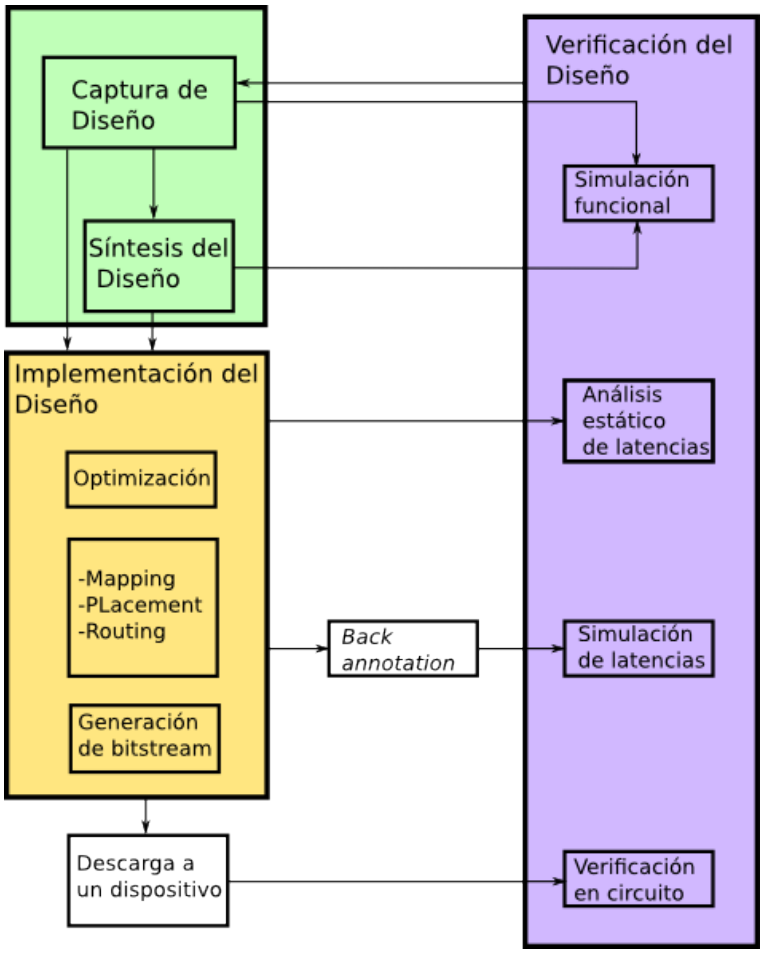

Ilustración 3-1: Flujo de diseño para dispositivos Xilinx

1. Captura de diseño y síntesis – El diseño se captura por medio de un lenguaje de descripción de hardware, un diagrama esquemático o una combinación de ambos métodos. El producto del proceso de captura es una descripción abstracta del comportamiento deseado del diseño. Esta descripción pasa por el proceso de síntesis lógica del cual se obtiene un archivo *netlist*, una descripción del diseño en términos de elementos lógicos básicos – combinatorios y secuenciales – y las interconexiones entre tales elementos.
2. Implementación – Conversión de la descripción lógica del diseño –el archivo *netlist*– en un formato de archivo físico para una arquitectura de FPGA específica, es decir la traducción de los elementos lógicos básicos e interconexiones definidas en el archivo *netlist* en recursos físicos específicos de un tipo de dispositivo FPGA particular. Esta nueva forma de descripción es finalmente transformada en un archivo binario de configuración conocido como *bitstream.* Este archivo es

descargable en el dispositivo FPGA objetivo. Inicializará los recursos del FPGA a los valores necesarios para que cumpla con la funcionalidad requerida.

3. Verificación del diseño – Mediante el uso de simuladores o pruebas *in situ*, se verifica que el diseño cumpla con la funcionalidad deseada y que cumpla con los requerimientos de tiempo deseados.

El flujo de diseño completo es un proceso iterativo entre las tres etapas previamente mencionadas hasta conseguir el funcionamiento deseado del diseño.

Para los propósitos de este trabajo las etapas de verificación no resultan relevantes en primera instancia, salvo la verificación en circuito, como se verá en el capítulo 4. Por otro lado las etapas de síntesis e implementación resultan de particular interés. Es en estas etapas donde se define la creación de particiones reconfigurables y su interconexión con la parte estática del diseño. Mas aún, la flexibilidad del sistema reconfigurable y sus posibilidades dependerá del comportamiento de la etapa de implementación y a partir de este comportamiento, si es el caso, el diseñador deberá buscar soluciones a los impedimentos que esta etapa imponga al tipo de sistema que desee implementar. Cada etapa ocupa una serie de herramientas de software que el fabricante pone a disposición del diseñador para lograr un objetivo particular. Cada herramienta recibe una serie de archivos de entrada y entrega un conjunto de archivos particular al terminar su procesamiento. Tales archivos de salida son utilizados por alguna herramienta posterior. Así se forma una cadena de herramientas como se puede ver en la Ilustración 3-2.

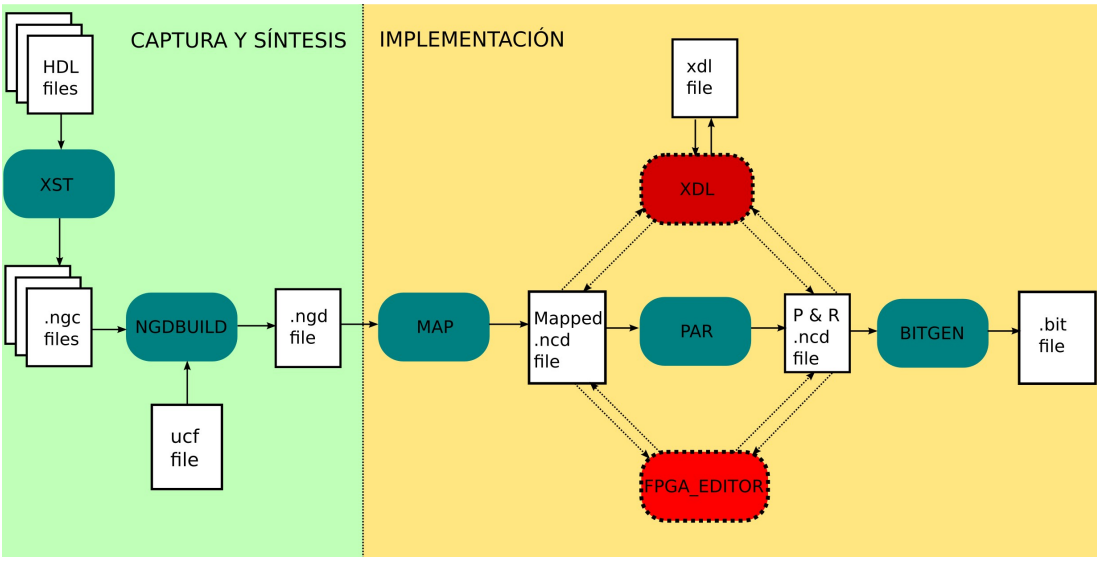

Ilustración 3-2: Cadena de herramientas del flujo de diseño para dispositivos Xilinx.

La cadena de herramientas trabaja como se describe a continuación para un flujo de diseño que parte de captura por medio de un lenguaje de descripción de hardware y utiliza sólo formatos de archivo propietarios de Xilinx:

1. XST (Xilinx Synthesis Technology) recibe un un conjunto de archivos HDL (y en su caso archivo de restriccione XCF) y entrega un *netlist* del diseño con restricciones en un archivo tipo NGC .
2. NGDBUILD - Recibe archivos *netlist* NGC y archivos *User Constraints File* (UCF) y genera un archivo *Native Generic Database* (NGD). Este tipo de archivo contiene una descripción del diseño en términos de elementos lógicos y de elementos primitivos de Xilinx y una descripción de la jerarquía original del diseño expresada en el archivo *netlist* de entrada. En esta etapa, se resuelven los componentes de las bibliotecas del sistema, bibliotecas de usuario y macros.
3. MAP – Con MAP inicia el proceso de implementación del diseño. Tomando la descripción lógica y jerárquica contenida en un archivo NGD, MAP mapea o traduce esa descripción a una representación física en términos de los componentes con los que la arquitectura del FPGA objetivo cuenta, es decir *Configurable Logic Blocks* (CLB), bloques de memoria RAM, componentes de manejo de señales de reloj, bloques de entrada y salida, etc. El archivo de salida de esta etapa es del tipo *Native Circuit Description* (NCD).
4. PAR (*Place and Route*)- Dependiendo de las opciones con las que se ejecute MAP, y de la arquitectura objetivo, el archivo de salida puede contener una representación sólo mapeada o también ubicada del diseño. PAR se encarga de cumplir con tal proceso de ubicación y enrutamiento (*place and route*). El proceso de ubicación (*placement*) consiste en distribuir los componentes que en la etapa previa se determinó que serían necesarios para la implementación del diseño entre los componentes reales del dispositivo FPGA tomando en cuenta su ubicación física y buscando cumplir con restricciones de ubicación y área, y favorecer el cumplimiento de restricciones de latencias. Tras el proceso de ubicación, los componentes se conectan entre sí mediante el proceso de enrutamiento (*route*). Este proceso buscará resolver todas las conexiones mientras optimiza la latencia

máxima del circuito. Al finalizar estos dos procesos, PAR entrega un nuevo archivo NCD, esta vez con el diseño completamente ubicado y enrutado.

5. BITGEN – Una vez que el diseño ha sido completamente mapeado, ubicado y enrutado en un archivo NCD, BITGEN genera un archivo binario de configuración *bitstream.* Este archivo contiene la información de configuración que deberá descargarse a la memoria de programación del dipositivo para que implemente la funcionalidad deseada.

Cada herramienta tiene un comportamiento muy específico y ofrece un cierto grado de control al diseñador sobre la implementación final del diseño. En general, las herramientas buscarán optimizar el diseño dentro de ciertos parámetros y para el diseñador esto será suficiente, no necesitará saber con mayor detalle como es que el diseño se implementó físicamente en el dispositivo mientras funcione de acuerdo a sus requerimientos. Este comportamiento de las herramientas y la forma en que un diseñador asume los resultados es adecuado para la mayoría de los diseños de propósito general, pero para aplicaciones con requerimientos específicos, como en nuestro caso, se vuelve necesario buscar caminos alternos que permitan llegar a un nivel más bajo en el conocimiento y manipulación de la implementación física de un diseño, como se verá más adelante.

3.1.2 CAMINOS ALTERNOS Y EXTENSIONES AL FLUJO ESTÁNDAR

La mayoría de los archivos generados en este proceso son de tipo propietario, es decir, su formato no es conocido salvo por el fabricante y aquellas entidades con las cuales tenga convenios de compartición de información. Hay, sin embargo, algunas excepciones con las que el fabricante nos abre una ventana a la arquitectura de sus dispositivos. Tal es el caso del formato de los archivos *bitstream* el cual es un formato semi-cerrado. Su estructura general está documentada en las guías de configuración de algunos dispositivos [24], aunque la función de la mayor parte de los bits de configuración no es información públicamente disponible. No obstante la información disponible ha resultado suficiente para los propósitos de este trabajo. En este capítulo se describirá más adelante la información pertinente de este formato de archivo. Otros casos importantes son dos herramientas que funcionan como pasos extra para la manipulación de detalles muy finos de implementación no accesibles en el flujo general mostrado en la Ilustración 2-2:

FPGA_EDITOR y XDL.

FPGA_EDITOR es una herramienta gráfica de edición a muy bajo nivel de la configuración interna de un dispositivo FPGA. Mediante la lectura y modificación de archivos NCD completa o parcialmente ubicados y enrutados, permite realizar modificaciones en la configuración de los elementos más básicos del dispositivo FPGA como las LUT, flip-flops, multiplexores y recursos de enrutamiento incluso al nivel de los cables de interconexión. Es posible incluso – pero muy poco recomendable – diseñar un circuito completo editando un archivo NCD desde FPGA_EDITOR teniendo control sobre toda la configuración interna del dispositivo. Ofrece una vista gráfica de la distribución de los componentes básicos del dispositivo objetivo y con ello un revelador vistazo a la arquitectura del mismo. También permite la generación de Macros y restricciones especiales como las restricciones de enrutamiento directo, estas últimas de particular interés para este trabajo.

XDL es una herramienta igualmente poderosa pero poco utilizada y documentada y de difícil manejo. Se trata de un lenguaje de descripción de hardware alternativo específicamente para dispositivos de Xilinx. La suite de herramientas de Xilinx cuenta con el comando *xdl* con el cual es posible convertir un archivo NCD en un archivo XDL (y viceversa). Un archivo XDL puede representar dos cosas: la descripción de la arquitectura interna de un dispositivo específico o la representación en forma de texto de un diseño mediante dos elementos básicos: instancias e interconexiones – conocidas como *nets* – entre instancias. Para el segundo caso, las instancias describen la configuración de un componente físico específico usado en el diseño mientras los *nets* describen una conexión entre una terminal de salida de una instancia (*source*) y una serie de terminales de entrada (*sinks*) en una o más instancias en términos de *Programmable interconnect points* (PIP) distribuidos en una serie de matrices de enrutamiento ubicadas en alguna posición dentro del dispositivo. Es posible entonces mediante archivos XDL modificar o construir un diseño con el mayor nivel de detalle posible, al tener acceso incluso a los PIP del dispositivo y todos los parámetros de configuración que una instancia puede tener. Es sin embargo una labor complicada pues es un lenguaje poco documentado y Xilinx no

provee herramientas que faciliten la programación mediante este lenguaje. No obstante, existen herramientas desarrolladas en el medio académico que permiten utilizar XDL desde un lenguaje de alto nivel como Java. Tal es el caso del proyecto **RapidSmith** [25], una API de Java capaz de manipular todos los elementos de XDL como objetos Java, abriendo la posibilidad de desarrollar herramientas CAD experimentales para la exploración de técnicas de implementación de diseños en FPGA: algoritmos de mapeo, ubicación y enrutamiento, herramientas para casos especiales de implementación, etc. El proyecto RapidSmith basa su potencial en la adquisición de la información completa de las arquitecturas de los dispositivos de Xilinx mediante el parseo de archivos XDL que los describen y que pueden ser obtenidos con algunas opciones de la herramienta *xdl.* Esta información abarca tanto la ubicación de todos los componentes internos del dispositivo y sus parámetros como la arquitectura de enrutamiento, es decir, la ubicación y distribución interna de las matrices de enrutamiento y todas las posibles conexiones que un cable en particular puede realizar otros mediante PIPS. Así, al conocer esta información, el API es capaz de crear instancias, configurarlas y ubicarlas en la posición que se desee, así como crear *nets* que ocupen los cables y PIPS que se requieran según la aplicación.

En el capítulo 4 se describirá el uso de estas herramientas para lograr la implementación de sistemas reconfigurables con tareas de hardware relocalizables. Su uso es necesario por las limitantes impuestas por el flujo general de implementación de diseños reconfigurables que a continuación se describirá.

3.1.3. EL FLUJO DE DISEÑO DE RECONFIGURACIÓN PARCIAL

La flexibilidad particular de los dispositivos FPGA de ser programados y reprogramados “en sitio”, sin tener que pasar por un proceso de refabricación, ha sido llevada un paso mas allá con la introducción de la tecnología de reconfiguración dinámica. Esta tecnología permite modificar un diseño activo en un dispositivo FPGA mediante la carga de un archivo de configuración parcial. Luego de la carga de un archivo *bitstream* para el dispositivo completo, es posible descargar *bitstreams* parciales que modifiquen regiones particulares en el FPGA sin interferir en el funcionamiento del resto de aplicaciones que se ejecuten en regiones del dispositivo que no sean reconfiguradas. La Ilustración 3-3 ilustra la idea básica.

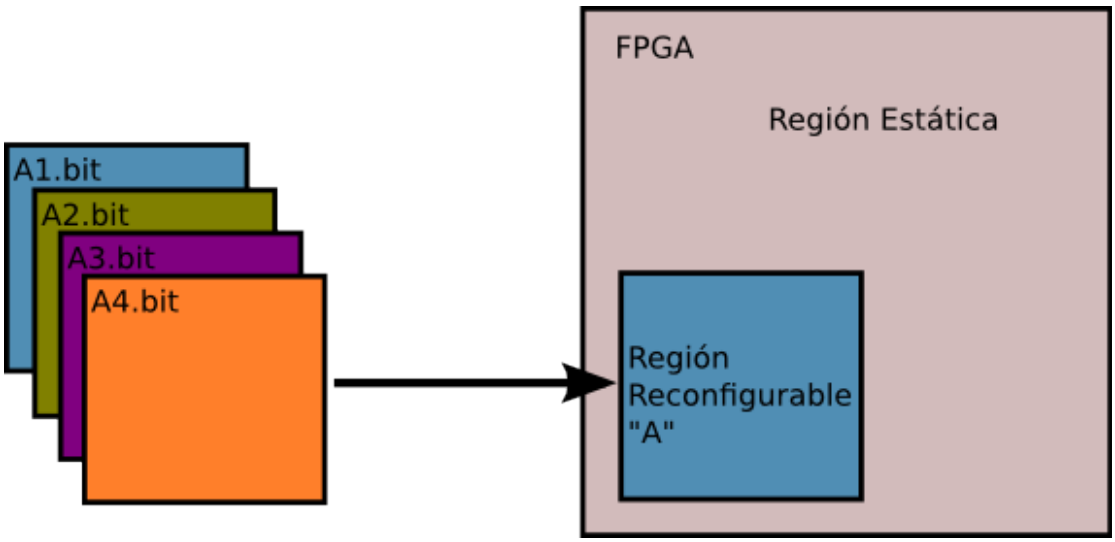

Ilustración 3-3: Idea básica de reconfiguración parcial

En un diseño reconfigurable como el mostrado en la Ilustración 3-3 la lógica está organizada en dos partes: la lógica estática y la lógica reconfigurable, distribuida en regiones. Las regiones reconfigurables podrán ser modificadas mediante la carga de *bitstreams* parciales (en la Ilustración 3-3: A1.bit, A2.bit, A3.bit y A4.bit) mientras la región estática sigue funcionando sin ser afectada por la carga de los bitstreams parciales. Así pues, el flujo de diseño particular para este tipo de diseños tendrá que entregarnos al final un bitstream general, para el FPGA completo y un bitstream parcial por cada módulo que deseemos implementar en cada partición reconfigurable existente.

La implementación de un diseño con reconfiguración parcial es muy similar a implementar muchos diseños sin esta característica pero que comparten lógica en común. Se utiliza el concepto de **partición** para separar la lógica reconfigurable de la estática y asegurar que la lógica común sea idéntica entre los múltiples diseños que se implementan. Las particiones son herramientas utilizadas en flujos de diseño jerárquicos como la reconfiguación parcial. Su finalidad básica es definir barreras jerarquicas alrededor de los módulos de un diseño de forma que puedan ser aislados del resto. Una vez que una partición ha sido implementada y exportada, puede ser insertada en otro diseño en una operación tipo “corta y pega”, preservando la ubicación y enrutamiento del módulo implementado en la partición exportada.

Así, el flujo de diseño de reconfiguración parcial es un proceso iterativo donde el flujo

general se repite tantas veces como sea necesario para implementar todos los *bitstream* parciales requeridos para los módulos reconfigurables que se desea implementar en las particiones reconfigurables presentes. En cada nueva iteración, la lógica implementada en la partición estática es importada de la primera implementación realizada. La ilustración 3-4 muestra una vista general de este proceso.

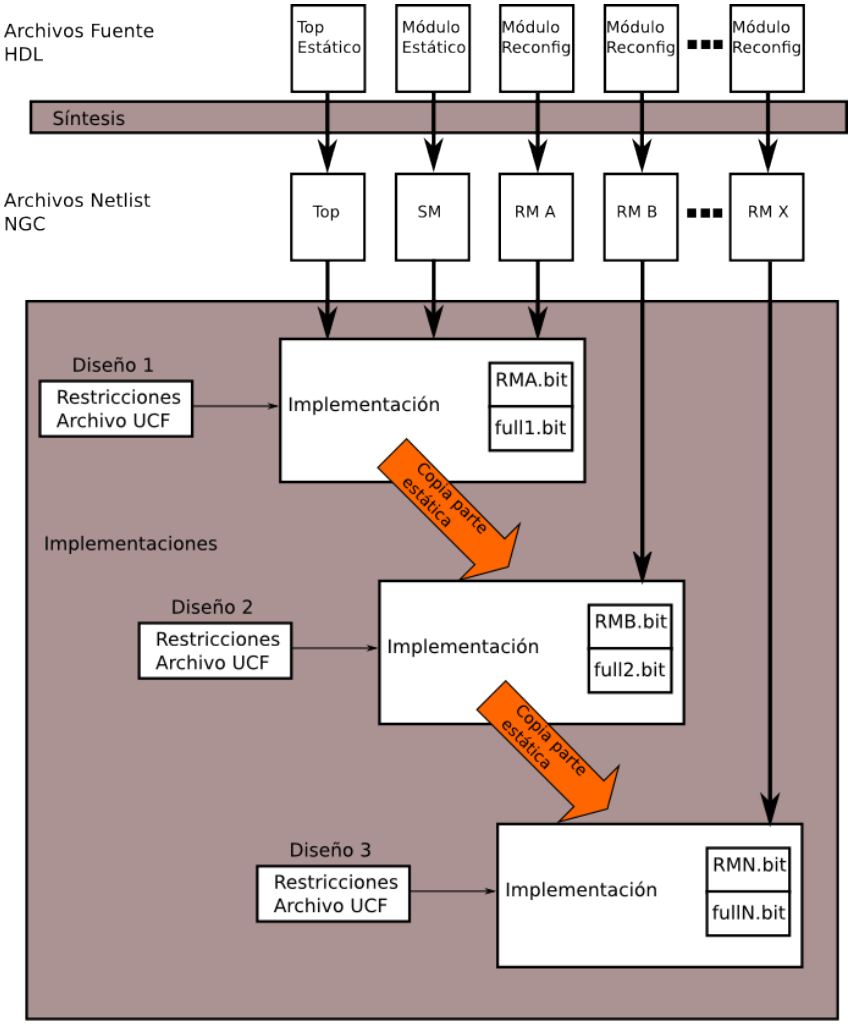

Ilustración 3-4: Visión general del flujo de implementación de un diseño con reconfiguración parcial

Como se ha dicho anteriormente, la implementación de un diseño de este tipo es similar a implementar varios diseños sin reconfiguración parcial, no obstante se deben tomar algunas consideraciones por cada etapa, partiendo de la síntesis, como a continuación se describe:

Síntesis

Cada módulo reconfigurable debe sintetizarse por separado. Debe deshabilitarse la inserción de terminales de entrada/salida al exterior si es que el módulo sólo ha de conectarse a la lógica estática y no hacia terminales externas del FPGA. Los módulos estáticos pueden sintetizarse en un solo *netlist* o por separado; al final, la herramienta NGDBUILD unirá todos los *netlist* sean de módulos reconfigurables o estáticos y la definición de particiones declarará las interfaces entre las partes estáticas y reconfigurables del diseño.

Implementación

Implementación – Para implementar la totalidad de los módulos reconfigurables, deben escogerse subconjuntos de todas las combinaciones posibles de módulos para las particiones reconfigurables existentes en el diseño e implementar cada combinación como un diseño individual. A cada implementación, se le conocerá como **configuración**. En un diseño con *n* particiones reconfigurables y *m* módulos posibles por particion existirán m^n configuraciones potenciales, pero solo es necesario implementar las suficientes para generar todos los *bitstreams* parciales requeridos.

Restricciones AREA_GROUP

De la mano con el concepto de particiones, deben declararse restricciones de agrupamiento de área en el archivo de restricciones UCF del diseño completo. Las restricciones AREA_GROUP asocian una etiqueta a un elemento de diseño o instancia de forma que toda la lógica contenida en ese elemento puede ser identificada como un grupo de elementos durante todo el proceso. En la definición de estas restricciones se deben tomar las siguientes consideraciones:

- Para cada restricción AREA_GROUP habrá que asociar cuando menos una restricción RANGE, indicando la forma y tamaño del grupo, es decir de la partición, por medio de las coordenadas de los recursos físicos de la partición ubicados en las esquinas inferior izquierda y superior derecha.
- La forma que estas particiones adoptan sólo puede ser rectangular y no puede haber particiones traslapadas.

- Se utilizarán más de una restricción RANGE cuando la partición sea heterogénea, es decir contenga recursos físicos de más de un tipo (por ejemplo CLB y BRAM). Habrá una restricción RANGE por cada conjunto continuo de elementos del mismo tipo.
- Los recursos CLB están conformados por dos elementos llamados SLICE, en este caso la restricción RANGE debe ser declarada para CLB completos, no SLICES individuales.
- Existen recursos físicos del dispositivo que no pueden ser reconfigurados y por tanto no deben ser considerados en una restricción AREA_GROUP.

La Ilustración 3-5 muestra un ejemplo de la definición de restricciones AREA_GROUP y RANGE para una instancia particular. El AREA_GROUP contiene tanto recursos tipo CLB (o SLICE) como recursos tipo BRAM.

```
INST "slot2_0/slot2_0/USER_LOGIC_I/rp_instance" AREA_GROUP = "pblock_slot2_0_USER_LOGIC_I_rp_instance";
AREA_GROUP "pblock_slot2_0_USER_LOGIC_I_rp_instance" RANGE=SLICE_X16Y0:SLICE_X19Y19;
AREA_GROUP "pblock_slot2_0_USER_LOGIC_I_rp_instance" RANGE=RAMB36_X1Y0:RAMB36_X1Y3;
```

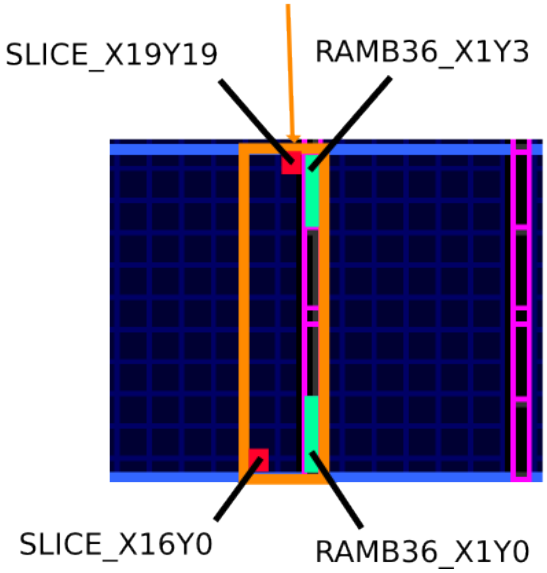

Ilustración 3-5: Ejemplo de definición de una restricción AREA_GROUP

Particiones

Además de generar las restricciones AREA_GROUP, deben generarse los atributos de partición. La información de las particiones se almacena en el archivo especial *xpartition.xml*. Este archivo XML es generado de forma automática por herramientas como planAhead y el script *gen_xp.tcl* y es utilizado como entrada por las herramientas de

implementación MAP y PAR. Deberá existir un archivo *xpartition.xml* para cada configuración a implementar.

El atributo partición tiene a su vez atributos como RECONFIGURABLE y STATE. El atributo RECONFIGURABLE determina si una partición se implementa de forma tal que al final del proceso, se genere un bitstream parcial para tal partición. El atributo STATE indica, para la configuración particular, si la partición será implementada o importada desde una configuración previamente implementada. Mediante este atributo se indica que la partición estática del diseño se implementa sólo en la primera configuración y se importa en el resto. Como se ha dicho previamente, al ser importada, su ubicación y enrutamiento son persistentes entre las configuraciones que la utilicen. No así con las particiones reconfigurables que en general, se implementarán en todas las configuraciones.

Terminales de Partición

Un detalle importante de la implementación de particiones reconfigurables es su interfaz con la parte estática del diseño. Es necesario tener puertos para las señales que entren y salgan de las particiones reconfigurables y éstas deben ser implementadas siempre en el mismo lugar para las diferentes configuraciones. Hasta antes de la versión 13.2 del juego de herramientas de Xilinx, la herramienta utilizada para lograr estas interfaces era el *bus macro.* Esencialmente un *bus macro* es un componente de hardware que se instancia de forma que las señales que pasen de la parte estática del diseño a la parte reconfigurable crucen por el *busmacro,* la señal en cuestión pasará a su receptor de forma transparente. A partir de la versión 13.2 del juego de herramientas, el uso de los *bus macro* ha sido descontinuado en favor de las terminales de partición. Las terminales de partición son ahora la conexión física y lógica entre particiones estáticas y reconfigurables. Físicamente cada pin de partición es una LUT (en el flujo de diseño se le conoce como lógica *proxy*) configurada en modo *route through*, es decir, la función lógica de la LUT es OUT = IN. El uso de estos componentes es automático dentro del flujo de reconfiguración parcial, es decir, el usuario no necesita hacer nada mas allá de declarar particiones reconfigurables en su diseño para que las herramientas de diseño instancíen por si mismas las terminales de partición necesarias. La Ilustración 3-6 muestra el uso de terminales de partición de

entrada y salida en una partición reconfigurable.

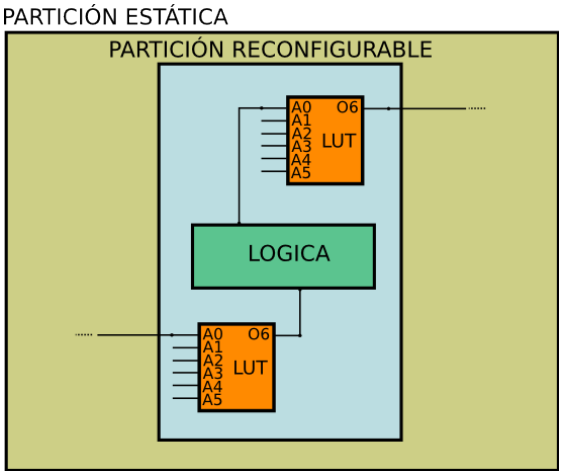

Ilustración 3-6: Pines de partición

Como se muestra en la Ilustración 3-6, las terminales de partición se encontrarán físicamente en su correspondiente partición reconfigurable, pero hay que aclarar que jerárquicamente se encuentran se encuentran al nivel de la partición estática. Esto permite que tales componentes, al ser parte de la lógica estática, se implementen en la misma posición dentro de la partición reconfigurable para todas las configuraciones. Lo anterior no quiere decir que los pines de partición se encontrarán en las mismas posiciones relativas para todas las particiones existentes. Otra consideración es que las señales que entren a las LUT proxy no utilizarán de manera predeterminada las mismas entradas para cada LUT, sino las que PAR considere como las óptimas para sus propósitos. Mas aún, incluso si las posiciones relativas de los pines de partición fueran consistentes en todas las particiones reconfigurables y las LUT proxy utilizaran las mismas entradas consistentemente, no hay ninguna garantía de que el enrutamiento desde la partición estática hasta estos pines sea compatible. En el capítulo 4, se analizará el impacto que estos detalles tendrán en la compatibilidad entre particiones reconfigurables y las tareas de hardware y se presentará el método desarrollado para salvar los inconvenientes que estas consideraciones introducen en el flujo de diseño.

Las consideraciones anteriores son las más importantes dentro del flujo de diseño para reconfiguración parcial. En cuanto al uso de las herramientas, no existen opciones especiales para el uso de reconfiguración parcial salvo lo anteriormente mencionado. Así, al definir restricciones AREA_GROUP y particiones con atributos de reconfiguración, al

final del flujo de herramientas, BITGEN generará automáticamente los *bitstream* parciales requeridos para cada configuración sin necesidad de mayor intervención del usuario.

3.2. ARCHIVOS DE CONFIGURACIÓN BITSTREAM PARA DISPOSITIVOS VIRTEX 5

Los dispositivos FPGA son configurados mediante la carga de archivos *bitstream* en la memoria interna del dispositivo. Estos archivos contienen información de configuración específica para la arquitectura específica del dispositivo.

Un archivo *bitstream* se compone de tres secciones:

- Autodetección de ancho de bus
- Palabra de sincronización
- Configuración del FPGA

La autodetección de ancho de bus es un patrón de bits insertado al inicio del *bitstream* que se usa en modos de configuración en paralelo para detectar el ancho del bus de configuración con el que se trabajará.

La palabra de sincronización se usa para permitir que la lógica de configuración se alinee al procesamiento de palabras de 32 bits.

La información de configuración comprende comandos para la lógica de configuración del dispositivo y datos de configuración que se escriben en la memoria interna.

3.2.1. Distribución de la memoria de configuración

La información de configuración contenida en el *bitstream* se vacía en la memoria de configuración interna. Cada localidad de esta memoria guarda la configuración de un aspecto particular de un grupo de recursos físicos del FPGA.

La memoria de configuración de los FPGA Virtex 5 está dividida en *frames* distribuidos en todo el dispositivo. Estos *frames* son la unidad mínima de direccionamiento de la memoria y por tanto cualquier operación de lectura o escritura actúa sobre *frames* completos. Un *frame* consiste en 41 palabras de 32 bits de información de configuración que abarca una fila.

La fila es una cantidad determinada de bloques básicos (20 CLB, 40 IOBs, 4 BRAM, etc) apilados uno sobre otro, con una linea de bloques de reloj que pasa por la mitad de ellos. De las 41 palabras que contiene un *frame*, la información de las primeras 20 corresponde a los bloques básicos por encima de la línea de bloques de reloj, las últimas 20 a los bloques por abajo de la línea de reloj y la palabra central corresponde a la línea de bloques de reloj. La ilustración 3-7 muestra esta disposición de palabras en un *frame* correspondiente a recursos CLB.

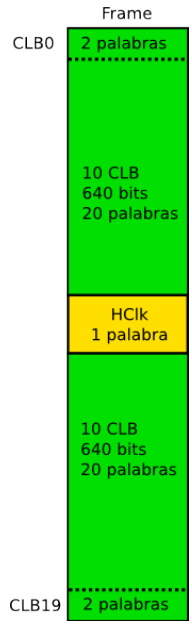

Ilustración 3-7: Distribución de palabras de un frame de fila de CLB

Direccionamiento de Frames

Cada *frame* de configuración tiene una dirección única de 32 bits dividida en 5 campos como se muestra en la Ilustración 3-8.

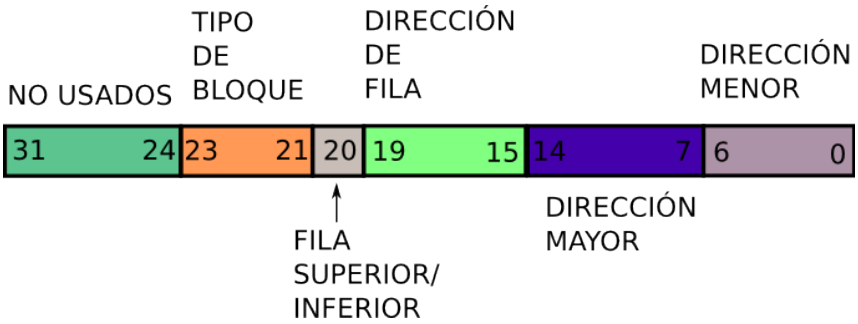

Ilustración 3-8: Dirección de frame y sus 5 campos

- **Dirección de fila e indicador superior/inferior** – El FPGA se divide en una mitad superior y una inferior, indicado en la dirección de un frame con el bit 20. Las direcciones de fila van del 0 al 9 para ambas mitades del dispositivo y la cuenta inicia con las dos filas más cercanas al centro con la dirección 0.
- **Dirección mayor –** Cada fila está dividida en una misma cantidad de columnas por fila. Cada columna corresponde a un tipo de recurso de acuerdo a la distribución del dispositivo. Las columnas se numeran de izquierda a derecha partiendo del 0. Hay dos secuencias de direcciones mayores por fila, la primera con una dirección para cada columna, es usada para acceder a la configuración básica de cada tipo de recurso. La segunda asigna direcciones particulares a cada bloque de columnas RAM. Esta secuencia permite acceder al contenido escrito o por escribir de los bloques RAM. Herramientas como DATA2MEM y XMD utilizan esta segunda secuencia para inicializar bloques RAM con programas de usuario o *bootloops* para lanzar programas que se ejecuten en memoria externa.
- **Dirección menor y tipo de bloque** – La dirección menor permite acceder a cada uno de los frames que conforman una columna. La cantidad de frames por columna depende del tipo de bloque y de la columna. Los tipos de bloque para Virtex 5 son:
    - Interconexión y configuración – Esta sección contiene la información de configuración normal para bloques e interconexión. La tabla 3-1 enlista el número de frames o direcciones menores por columna para este tipo de bloque. Para todos los tipos de bloque, salvo los bloques CLK, los frames 0 a 25 contienen la información de interconexión para la columna.

Bloque	*Número de frames*
CLB	36
DSP	28
BRAM	30
IOB	54
CLK	4

Tabla 3-1: Frames por columna

    - Contenido de BRAM – Como se ha mencionado , hay información de configuración básica y de contenido para BRAM, representada esta última por

este tipo de bloque.

- Interconexión y bloque de *frame* especial – Para proyectos que utilizan reconfiguación dinámica hay un *frame* especial por columna que contiene información especial para este tipo de proyecto. En este proyecto no ha sido necesaria su manipulación, pero no puede ser completamente descartada.

La Ilustracion 3-9 muestra los conceptos de fila, dirección mayor, *frame* y dirección menor en la arquitectura del dispositivo Virtex 5 utilizado para el desarrollo de este trabajo.

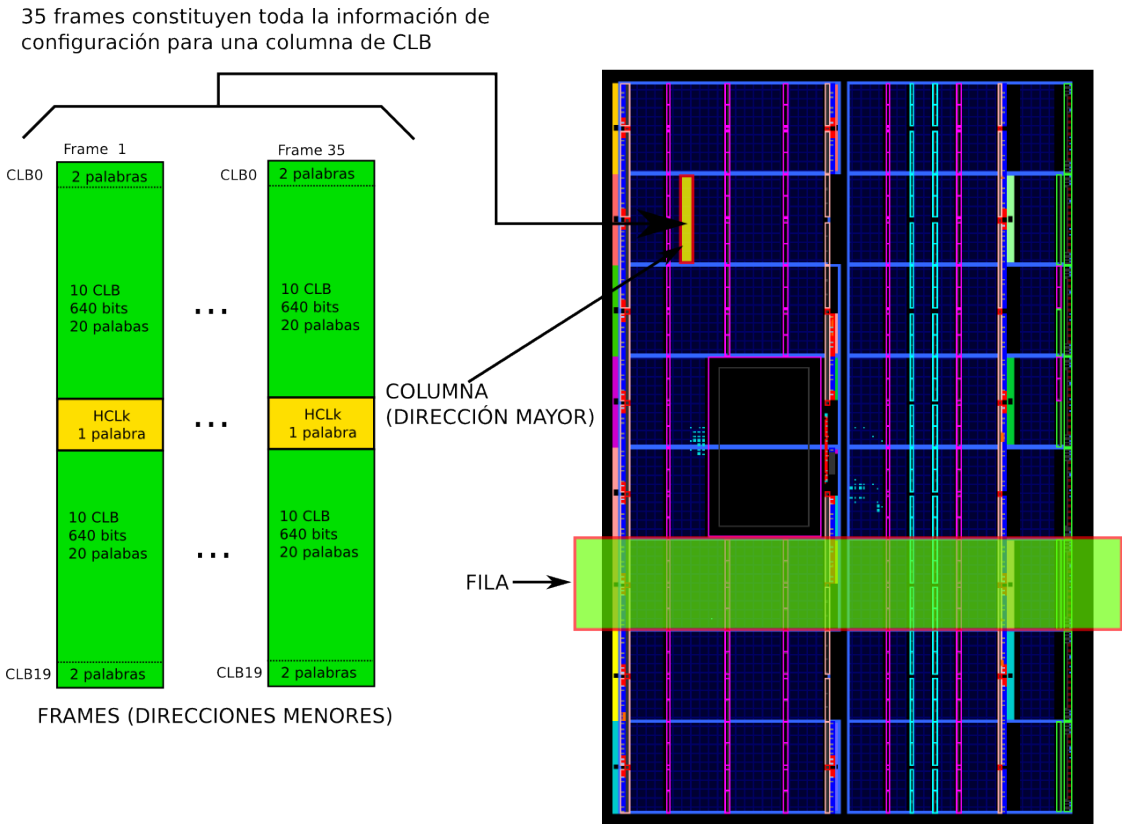

Ilustración 3-9: Fila, columna y frames en un dispositivo Virtex 5

De la información presentada hasta el momento sobre el direccionamiento de *frames* se desprende un concepto particular del campo de investigación en reconfiguración parcial: La **granularidad de reconfiguración** en un FPGA en modo de reconfiguración parcial está definida por la mínima unidad de reconfiguración, es decir la mínima cantidad de recursos que pueden ser configurados en una sola operación. Para la arquitectura de los dispositivos Virtex-5 esta variable está definida como la columna.

3.2.2 La información de configuración del bitstream

Después de la autodetección de ancho de bus y la palabra de sincronización, el bitstream contiene la información de configuración, consistente en dos tipos de paquetes de comandos y datos.

Paquete tipo 1 – Este paquete se usa para leer o escribir a los registros de configuración del FPGA. Se utilizan 5 bits para identificar al registro. Y su encabezado es una palabra de 32 bits como se muestra en la Tabla 3-2. Después del encabezado sigue la sección de datos del paquete, que contiene la cantidad de palabras de 32 bits especificados por el campo *word count* del encabezado.

Tipo de encabezado	*Opcode*	*Dirección del registro*	*Reservado*	*Word count*
[31:29]	[28:27]	[26:13]	[12:11]	[10:0]
001	xx	RRRRRRRRRxxxxx	RR	xxxxxxxxxxx

Tabla 3-2: Formato del encabezado para el paquete tipo 1

Existen 20 registros de configuración a los cuales se escribe o lee, en la tabla 3-4 se describen los mas importantes para este trabajo.

Registro	*Lectura/Escritura*	*Dirección*	*Descripción*
CRC	Lectura/Escritura	00000	Registro de Revisión de Redundancia Cíclica (CRC). Las escrituras a este registro provocan la ejecución de un CRC contra la información del *bitstream*. Si el valor escrito corresponde al calculado, se permite la inicialización del dispositivo.
FAR	Lectura/Escritura	00001	Registro de Dirección de Frame. Este registro contiene la dirección del frame al que se escribirá información de configuración.
FDRI	Escritura	00010	Registro de Entrada de Datos de Frame. La información escrita a este registro se escribe al frame apuntado por FAR.
FDRO	Lectura	00011	Registro de Salida de Datos de Frame. La información leida de este registro corresponde a la información del frame apuntado por FAR.
CMD	Lectura/Escritura	00100	Registro de comandos. Indica a la lógica de configuración cambie de estado ciertas señales globales o realizar otras funciones de configuración

Tabla 3-3: Registros de configuración mas importantes

Existen 4 códigos de operación como se muestran en la Tabla 3-4:

Opcode	*Función*
00	NOP
01	Lectura
10	Escritura
11	Reservado

Tabla 3-4: Códigos de operación

Paquete tipo 2 – Este tipo de paquete se usa para escribir grandes cantidades de datos (en formato de *frame*). Debe ser precedido por un paquete tipo 1. No contiene una dirección pues se utiliza la del paquete tipo 1 precedente. La tabla 3-5 muestra el formato del encabezado de este paquete. La sección de datos contendrá la cantidad de palabras indicadas por *word count.*

Tipo de encabezado	*Opcode*	*Word count*
[31:29]	[28:27]	[26:0]
010	RR	xxxxxxxxxxxxxxxxxxxxxxxxxx

Tabla 3-5: Formato de encabezado del paquete tipo 2

La secuencia completa de comandos presentes en el bitstream para cargar la información de configuración e inicializar el dispositivo se encuentra documentada en la guía de usuario de configuración para el FPGA Virtex-5 [24] y no será explicada en este trabajo. No obstante es importante hacer notar la relación entre los registros FAR y FDRI. En general, un comando de escritura al registro FDRI estará precedido de una escritura al registro FAR. En un bistream generado en modo normal, sólo se escribirá la primera dirección de frame a la que se cargará información y la lógica de configuración actualizará los valores del registro FAR automáticamente. En un bitstream en modo de depuración se escribirán todas las direcciones de frame al registro FAR. El bitstream en modo de depuración será utilizado como se verá en el capítulo 4 para un análisis detallado de la secuencia de configuración del dispositivo. La función de los registros FAR y FDRI y la relación entre ellos serán utilizadas en el desarrollo presentado en el capítulo 4.

3.3. RESUMEN

En este capítulo se ha presentado una descripción del flujo básico de diseño en dispositivos FPGA y del flujo para reconfiguración parcial. Se describieron brevemente las

limitaciones más relevantes de estos flujos para el desarrollo de este proyecto y se introdujeron las herramientas especializadas que se utilizarán para sortear estas limitaciones. El capítulo termina con una descripción básica del formato de los archivos de configuración *bitstream* que resultará importante en el capítulo 4. Se presentó el concepto de granularidad de la reconfiguración, otra de las limitantes a tener en cuenta en este desarrollo, y que se utilizará para definir las posibilidades que las tareas de hardware tendrán en este proyecto.

CAPÍTULO 4. DESARROLLO

La descripción de los flujos de diseño y archivos de configuración para los dispositivos FPGA Virtex 5 de Xilinx en el capítulo precedente, ofreció un primer vistazo a los alcances y limitaciones de esta tecnología para la implementación de un sistema reconfigurable en tiempo de ejecución. Retomando esa información y con un análisis detallado, este capítulo presenta el proceso seguido para implementar un Sistema en Chip Configurable (SoPC) capaz de ofrecer el concepto de tareas de hardware relocalizables de tamaño y cantidad de puertos de entrada/salida variable. Estas tareas se conectarán al bus del sistema para comunicarse con un procesador embebido que las utilizará como co-procesadores. El sistema desde el punto de vista software podrá ver un área reconfigurable como un conjunto continuo de recursos asignables a las tareas de hardware y será capaz de administrar dichos recursos.

En la sección 4.1 se delinea brevemente el proceso de desarrollo. En la sección 4.2 se retoman las limitaciones de los flujos de implementación, la arquitectura y los archivos de configuración, descritas en el capítulo 3, se analizan con detalle y se presentan las propuestas de solución utilizadas en este trabajo comparándolas con el estado del arte. Esta etapa de análisis muestra que este proyecto y este tipo de aplicaciones en general son completamente dependientes de las prestaciones del dispositivo FPGA elegido y los procesos de implementación de un SoPC en el mismo. La sección 4.3 presenta la fase de desarrollo de hardware del proyecto. La sección 4.4 presenta la fase de desarrollo de software embebido y sus bases en el desarrollo de herramientas de interpretación de archivos *bitstream*. Cabe hacer hincapié en la “naturaleza embebida” del software de la sección 4.4 pues aunque la sección 4.3 trata del desarrollo del hardware, éste es un producto final desarrollado a partir de herramientas de software, la mayoría desarrolladas por el fabricante pero también con herramientas creadas especialmente para esta aplicación que son las que finalmente permiten dar soporte para tareas con las características requeridas y que serán comentadas en la sección 4.3.

4.1. PROCESO DE DESARROLLO

El proceso de desarrollo toma como base el objetivo general del trabajo (como se detalló en el capítulo 1) y las ideas discutidas en el estado del arte, para en conjunción con el análisis de los alcances y limitaciones de la tecnología utilizada plantear propuestas de solución a los problemas que se presentan en implementaciones como ésta.

Una vez planteadas las propuestas de solución se diseñan e implementan. En primera instancia se resuelve la implementación del hardware del sistema y posteriormente sobre esta base se desarrollan los componentes necesarios de software embebido. El diseño del software embebido, en particular de la función que manipula las coordenadas del *bitstream* tiene como base las herramientas desarrolladas para la interpretación de este tipo de archivos, por lo que también se presentará el desarrollo de las mismas.

Así, el proceso de desarrollo será como sigue:

1. Análisis de la tecnología y presentación de propuestas de solución.
2. Desarrollo de hardware.
3. Desarrollo de software embebido

4.2. ANÁLISIS DE LA TECNOLOGÍA

El presente trabajo busca dotar de un método de administración de recursos de hardware a arquitecturas reconfigurables en tiempo de ejecución. En particular a un SoPC desarrollado en un FPGA Virtex-5 integrado en una plataforma ML507 de la compañía Xilinx. Este sistema es capaz de ejecutar un sistema operativo en el procesador embebido en el FPGA y sobre este sistema operativo, tareas de software que se ejecuten de manera simultánea y que realicen peticiones de configuración de tareas de hardware a un programa que actúa como administrador de recursos. La Ilustración 4-1 muestra una vista de alto nivel del planteamiento básico.

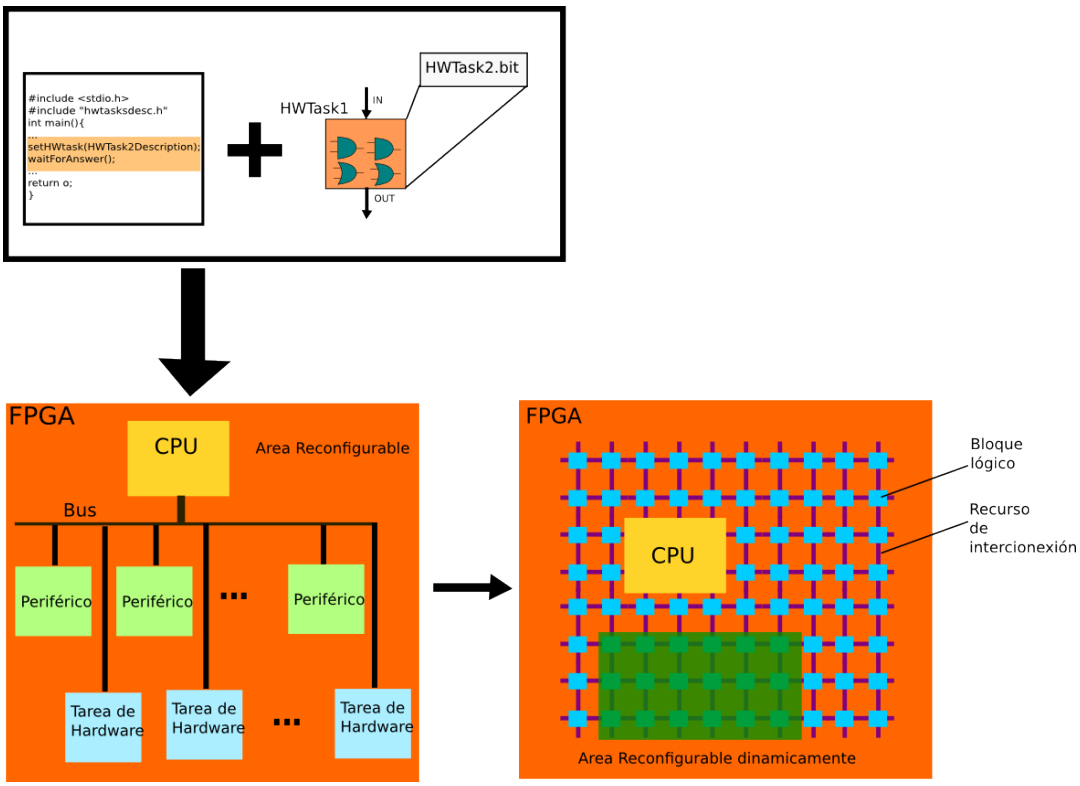

Ilustración 4-1: Vista de alto nivel del planteamiento básico del problema

Este desarrollo tiene dos vertientes, por un lado el desarrollo de una plataforma de hardware capaz de soportar relocalización de tareas de hardware reconfigurable y por otro lado, el software embebido que controle la asignación de recursos reconfigurables en tiempo de ejecución.

La posibilidad de soportar tareas relocalizables de hardware reconfigurable depende enteramente de las prestaciones del dispositivo FPGA y el proceso de implementación de un SoPC en tal dispositivo. Partiendo del hecho de que el FPGA soporta reconfiguración parcial en tiempo de ejecución, los aspectos más importantes a considerar en este sentido son los siguientes:

- Granularidad de la reconfiguración
- Distribución de recursos en el dispositivo (Heterogeneidad).
- Comunicación de las tareas de hardware con el procesador embebido y enrutamiento.

El estudio de estos aspectos arrojará el establecimiento de un *modelo de área* y un

modelo de tarea reconfigurable. Estos modelos definirán las características del área administrada y las tareas que podrán ser programadas en ella. En específico los modelos definirán la forma que puedan adoptar las tareas, la flexibilidad de la relocalización y las interfaces de comunicación con el procesador.

4.2.1. Granularidad de la reconfiguración

La granularidad de la reconfiguración de un dispositivo FPGA que soporta reconfiguración parcial en tiempo de ejecución define la mínima cantidad de recursos que pueden ser reconfigurados en una operación. Esta medida está relacionada con la forma en que las interfaces de programación del dispositivo pueden acceder a la memoria de programación para escribir información de configuración y la relación de esta memoria con los recursos reconfigurables del dispositivo. La granularidad de reconfiguración impacta directamente en la flexibilidad de la relocalización de tareas de hardware pues también define la minima cantidad y disposición de recursos que pueden ser relocalizados.

Como ya ha sido tratado en la sección 3.2.1 de este trabajo, la granularidad de la reconfiguración para los dispositivos Virtex 5 es de una columna de 20 CLB, o 4 BRAM o 2 elementos DSP48. Aunque la unidad de direccionamiento mínimo de la memoria de configuración del dispositivo es el *frame*, la información de configuración para cada uno de los elementos que forman la columna está distribuido en una cantidad particular de *frames* para cada tipo de columna y cada *frame* contiene una parte de la configuración de cada elemento que conforma la columna. Así la escritura de un solo *frame* afecta una cierta parte de la configuración de todos los elementos pero es necesaria la escritura de todos los *frames* de esa columna para que todos y cada uno de los elementos sean completamente configurados.

Cabe la posibilidad de poder acceder a la información de configuración de cada elemento que forme parte de una columna por separado mediante un tratamiento particular de los archivos de configuración al identificar que partes del *frame* pertenecen a cada elemento, pero dada la naturaleza propietaria de la información concerniente a la estructura interna de los *frames* de configuración, este manejo sería muy limitado e insuficiente para aumentar la granularidad de reconfiguración de la arquitectura. Los mayores inconvenientes para la implementación de un manejo de este tipo son la falta de

información sobre la configuración del enrutamiento y sobre las señales de reloj. Bien podría modificarse la configuración lógica de cada elemento individual, pero tal elemento está relacionado con otros por medio del enrutamiento existente entre ellos y más aún, cada elemento está relacionado con el resto del sistema por medio de las señales de reloj. Estas señales tienen rutas dedicadas dentro de la arquitectura. Al no conocer la información específica respecto a estos dos aspectos de la configuración del dispositivo, no es posible asegurar que un elemento pueda ser modificado de forma individual manteniendo su relación original con el resto de los elementos con los que tiene conexiones, ni su correcta conexión a las redes de señales de reloj.

De la granularidad de la reconfiguración del dispositivo se deduce que la opción mas viable para el modelo de tarea es que el tamaño de cada tarea esté dado en columnas, independientemente de si al implementar las tareas no se utiliza toda la lógica existente en el número de columnas asignadas. Este fenómeno es inevitable pues no existe forma de indicar a las herramientas de implementación cuántos recursos debe utilizar para la implementación de una tarea mas allá de indicar el AREA_GROUP donde ésta debe ser implementada. Esto quiere decir que la lógica del área reconfigurable no será utilizada de forma óptima, sino solamente de forma funcional y que existirá fragmentación, es decir, dentro del área reconfigurable habrá recursos no utilizados.

4.2.2. Distribución de recursos en el dispositivo (heterogeneidad)

La evolución de los dispositivos FPGA ha traído consigo el concepto de heterogeneidad, es decir la existencia de más de un solo tipo de recursos en el dispositivo. Las primeras familias de FPGA contaban con sólo un tipo de recursos para implementar lógica, los CLB. En este tipo de dispositivo, en teoría, una tarea podía ser implementada en cualquier parte del dispositivo y si éste soportaba reconfiguración parcial, relocalizada en prácticamente cualquier posición libre pues todos los recursos que la tarea necesitara, serían del mismo tipo. Sin embargo, por consideraciones de desempeño y flexibilidad, los diseñadores de FPGA han ido agregando recursos de diversos tipos a las arquitecturas reconfigurables, intercalados con los recursos básicos CLB, provocando la generación de áreas reconfigurables heterogéneas. La Ilustración 4-2 muestra los dos tipos de dispositivos, un sistema homogéneo y un sistema heterogéneo con una columna de

recursos BRAM y una columna de recursos DSP.

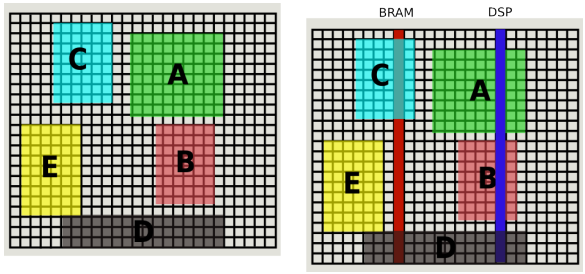

Ilustración 4-2: Comparativa entre un dispositivo homogéneo (izquierda) y un dispositivo heterogéneo (derecha).

Para aplicaciones de propósito general, este tipo de arquitecturas ofrecen mucha flexibilidad y mejor desempeño. Por ejemplo, un elemento tipo DSP48 permite hacer operaciones aritméticas complejas y optimizadas sustituyendo una gran cantidad de recursos básicos (cantidades diferentes según la aplicación y la arquitectura). De igual forma, un bloque BRAM permite implementar memorias de diferentes características dentro del dispositivo. Desafortunadamente para aplicaciones de reconfiguración parcial, esta heterogeneidad limita las opciones de ubicación de las tareas de hardware. En un sistema heterogéneo las tareas de hardware sólo pueden ser relocalizadas a posiciones libres cuyos recursos disponibles sean compatibles con los que la tarea necesita. La ilustración 4-3 ejemplifica esta idea con una vista de la arquitectura de un dispositivo Virtex 4.

El dispositivo con el que cuenta nuestra plataforma de desarrollo, es un dispositivo heterogéneo. Así que la relocalización de las tareas que se implementen estará limitada a áreas que cuenten con la misma distribución de recursos. Existe la posibilidad de utilizar el concepto de *sandbox* [26] para limitar la implementación de tareas a áreas homogéneas del dispositivo, pero este tiene dos inconvenientes: se limita el tamaño de las tareas al tamaño del *sandbox* y se limita el tipo de recursos que pueden conformar las tareas. La decisión sobre el aprovechamiento o no de los recursos heterogéneos del dispositivo formará parte de la definición del modelo de área utilizado. Impactará en la flexibilidad y generalidad del sistema que se implemente y en la etapa de desarrollo de software.

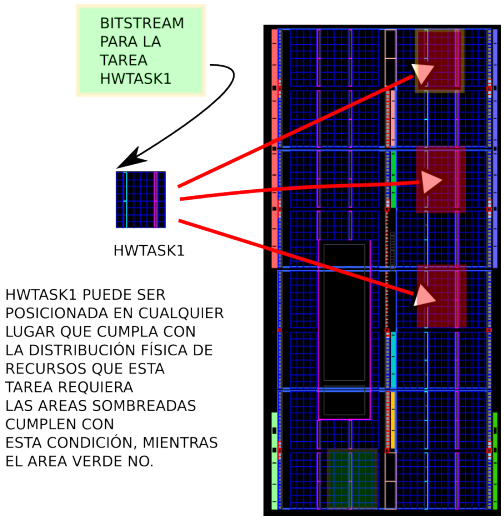

Ilustración 4-3: Flexibilidad de la ubicación de tareas de hardware reducida

4.2.3. Comunicación de las tareas de hardware con el procesador embebido y enrutamiento.

En un sistema reconfigurable en tiempo de ejecución con un procesador embebido como coordinador y usuario de tareas de hardware debe existir un mecanismo que permita comunicar al procesador con las tareas. El procesador debe ser capaz de enviar y recibir datos de cada tarea de hardware. El método más ortodoxo y único completamente soportado para conseguir esta comunicación es la conexión de las tareas de hardware al bus del sistema. Durante el proceso de implementación, en una partición reconfigurable se generan la cantidad de puertos de entrada y salida necesarios según la tarea de hardware que ocupe esa partición. Los puertos son enrutados y conectados con el bus utilizando el tipo de interfaz que se haya definido en la descripción del sistema. Como se describió en la sección 3.1.3 de este trabajo, las herramientas de implementación introducen interfaces entre la parte estática del diseño y las particiones reconfigurables, conocidas como terminales de partición. Las señales de los puertos de entrada y salida de las tareas implementadas en las particiones reconfigurables pasan por estas interfaces. El uso de las terminales de partición es la técnica que Xilinx considera adecuada para instruir a sus herramientas a ser consistentes en el enrutamiento estático entre las diferentes configuraciones de una misma partición reconfigurable. Sin embargo,

las herramientas no mantienen consistencia en el enrutamiento entre diferentes particiones. Esta falta de consistencia se debe a factores de ubicación de las terminales de partición y de enrutamiento de las señales de la parte estática del diseño que entran o salen de estas terminales y a la posibilidad de tener particiones reconfigurables de distinto tamaño.

El tamaño de las particiones es un problema menor. Si se desea tener tareas relocalizables es posible estandarizar el tamaño de las particiones reconfigurables y como se describió en el apartado anterior, la compatibilidad de recursos. Es incluso posible, como se verá más adelante, que una tarea sea compatible con una partición de tamaño mayor si existe un subconjunto de recursos compatibles entre la partición y la tarea.

Como se ha mencionado, las terminales de partición se implementan con LUT configurados en modo de enrutamiento a través, es decir, la única señal que entra al LUT sale de forma transparente por su terminal de salida. La Ilustración 4-4 muestra una vista simplificada (no se muestran las matrices de enrutamiento ni los multiplexores de entrada de la arquitectura) de un ejemplo sencillo de dos particiones reconfigurables donde se han implementado tareas que no son compatibles entre particiones. Esta incompatibilidad es la esperada al término del flujo normal de implementación para reconfiguración parcial.

Si las particiones son del mismo tamaño, hay tres condiciones que vuelven incompatibles a estas particiones:

- Ubicación relativa del pin de partición PP1 en cada partición.
- Pin de entrada de la señal_a proveniente de la parte estática del diseño.
- Enrutamiento estático.

Las tres condiciones provocan que las tareas implementadas para una partición no puedan ser relocalizadas en la otra y funcionar adecuadamente porque cada bitstream parcial genera una configuración particular en la partición donde es escrito, con ubicaciones y enrutamiento incluido. Un bitstream para la partición 1 que se escriba en la partición 2 provocaría la misma configuración relativa que provoca en la partición 1 mientras la configuración del enrutamiento estático para la partición 2 seguiría siendo la misma. Esto produciría conexiones incompletas, situación mostrada en la Ilustración 4-5.

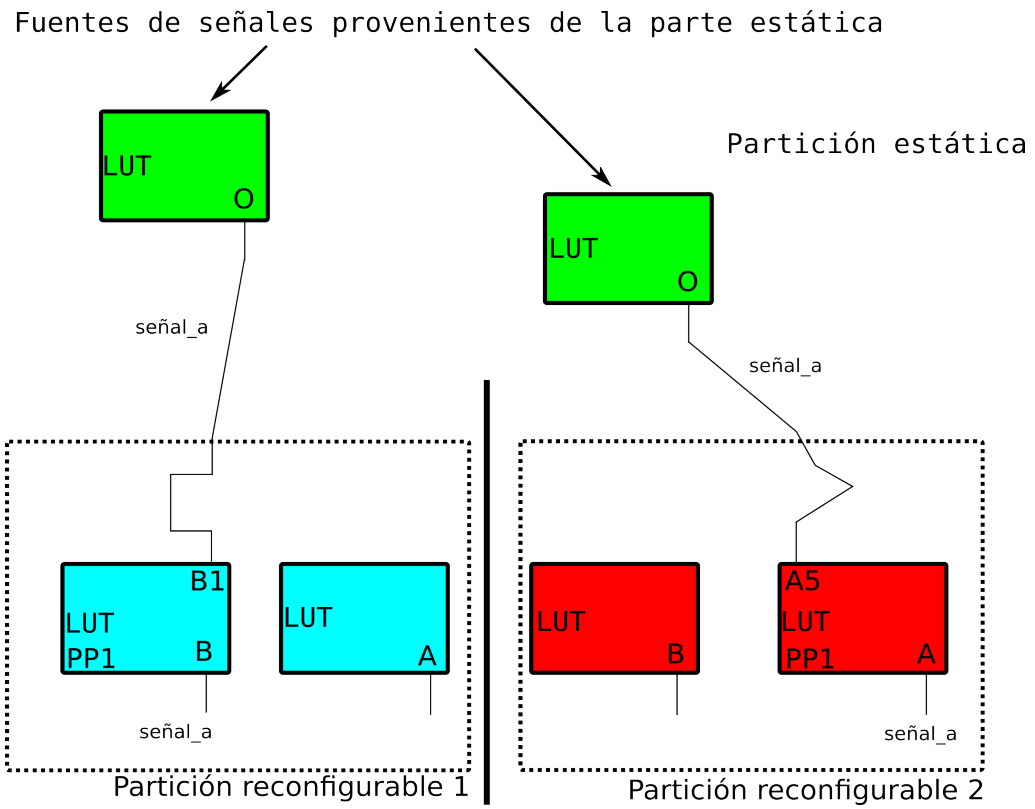

Ilustración 4-4: Incompatibilidad entre particiones reconfigurables

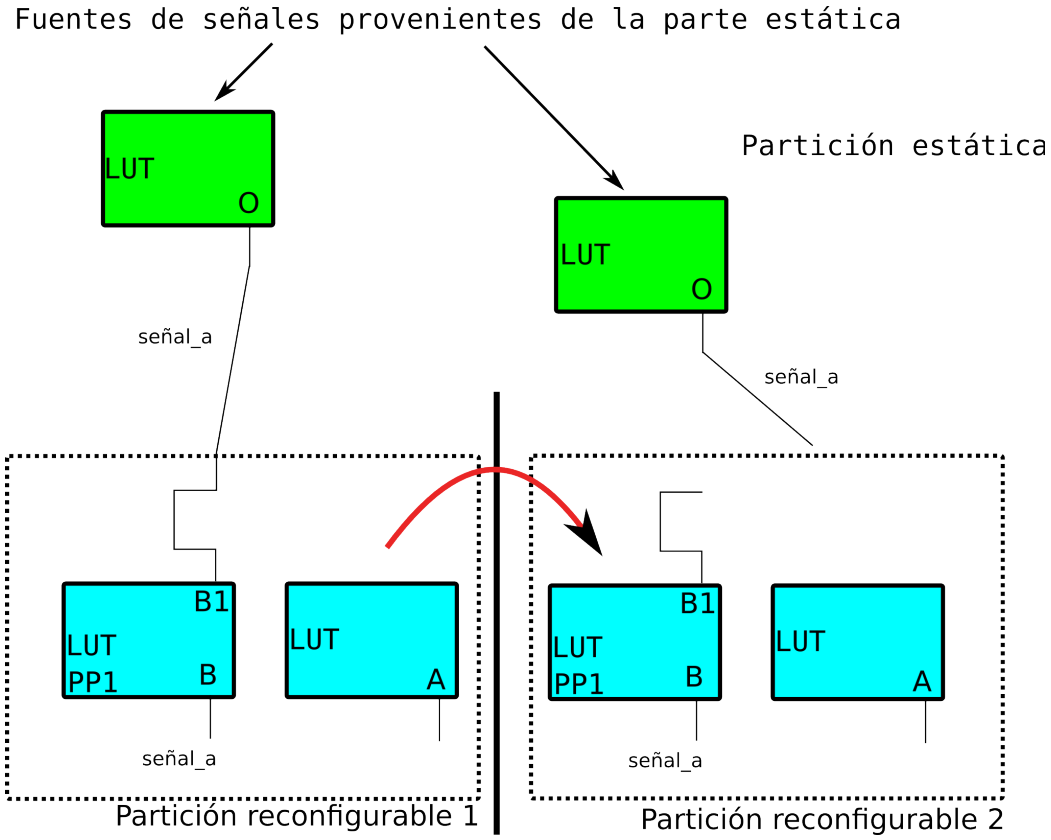

Ilustración 4-5: Conexiones incompletas por incompatibilidad de particiones

En la situación mostrada en la Ilustración 4-4, la partición 2 espera la llegada de la señal_a, en PP1 en una posición que no es relativamente la misma en la partición 1. Mas aún, la señal entra al PIN de partición por una entrada diferente de la LUT. Cualquiera de estas dos condiciones por si solas evitarían que la conexión entre la fuente de la señal estática para la partición 2 y el pin de partición PP1 se completara, pues se esperan condiciones de conexión completamente diferentes. Estas dos condiciones pueden ser corregidas con herramientas provistas por el flujo de diseño del fabricante. Con restricciones de ubicación y de bloqueo de terminales, como se verá en la sección 4.3, es posible controlar la ubicación y la entrada de las LUT de las terminales de partición. Pero aún existe otro problema: la resolución del enrutamiento entre la parte estática y la entrada de los pines de partición. Como se muestra en las dos figuras anteriores, la posición de las LUT de la parte estática del diseño que fungen como fuente (también podría tratarse de los sumideros de las señales) de las dos señales “senal_a”, con respecto a las particiones reconfigurables, no es la misma. Esto provoca que el enrutamiento de las señales tampoco sea el mismo relativo a la partición, ni en la parte estática ni potencialmente en la parte reconfigurable (esta última, la de interés), es decir, que el enrutamiento no utilice los mismos recursos de enrutamiento (matrices de enrutamiento y puntos de interconexión programables) según su posición relativa a las particiones. Esta diferencia se debe a que *PAR* (la herramienta de enrutamiento de Xilinx) buscará resolver las rutas de la forma mas conveniente para los criterios de optimización que utiliza. No hay manera de instruirlo a generar rutas compatibles bajo condiciones normales y ni siquiera existe tal concepto dentro de las herramientas. Hasta este momento, Xilinx no considera el concepto de relocalización de tareas de hardware entre particiones y no existe soporte para esta capacidad. El problema entonces, va más allá de la ubicación y las entradas elegidas en las terminales de partición, e incluso de las posiciones de las fuentes o sumideros de las señales en la parte estática pues aún con éstas variables controladas, no es posible instruir directamente a *PAR* a generar rutas compatibles. No obstante, es un comportamiento que puede ser forzado generando restricciones de enrutamiento dirigido compatibles.

La generación de estas restricciones junto con la generación de una capa intermedia de LUT de interfaz estática entre las fuentes o sumideros de señal y las terminales de

partición son parte de la propuesta de solución que presentamos para la implementación de tareas relocalizables.

Se ha propuesto la introducción de una capa intermedia de LUT de interfaz por un motivo importante: la procedencia de las fuentes o sumideros de señal. La representación mostrada en las dos Ilustraciones previas es una representación simplificada. En un diseño, estos componentes, de acuerdo a la arquitectura propuesta, serán instancias pertenecientes a la interfaz de la partición con el bus del sistema. Estas instancias tienen además de la señal que terminará en la partición reconfigurable, un conjunto de señales de control procedentes a su vez de otras instancias dentro de la interfaz con el bus. Aunque es posible controlar la ubicación de las instancias fuente/sumidero, agruparlas en las cercanías de las particiones reconfigurables y producir ubicaciones compatibles entre ellas, la cantidad de señales que utilizan y la procedencia de estas señales vuelve poco práctica esta alternativa. El mayor problema es que generar agrupaciones de instancias con una cantidad importante de señales entrando y saliendo de ellas tiende a generar congestión de señales y con ello problemas de enrutamiento. Es decir, es muy factible que un diseño sometido a estas restricciones se vuelva imposible de enrutar.

Por otro lado, la introducción de una capa intermedia de LUT interfaz consistente en el mismo tipo de instancias que las terminales de partición, es decir LUT en configuración de enrutamiento a través, entre las instancias fuente/sumidero y los pines de partición permite a las herramientas de mapeo y enrutamiento tomar las decisiones necesarias para resolver las señales de control de la interfaz con el bus. Con esta propuesta las restricciones de enrutamiento dirigido se generarán para las señales que conectan a esta interfaz con las terminales de partición.

La técnica que se propone para la generación de las restricciones de enrutamiento dirigido es el desarrollo basado en la biblioteca rapidSmith [25] de un *router* de propósito específico para esta aplicación. Agregaremos de esta forma un paso al flujo de diseño para reconfiguración parcial que asegure la compatibilidad entre particiones del enrutamiento de interfaz entre la parte estática del diseño y las particiones reconfigurables.

Por supuesto, la implementación de estas técnicas trae consigo algunas desventajas. La

más significativa, el uso de una LUT extra por cada señal de interfaz entre la parte estática del diseño y la parte reconfigurable. Aún así, las opciones para lograr compatibilidad entre particiones son muy reducidas. Se podría utilizar una técnica de generación de *anti-nucleos* [27] para forzar a *PAR* a utilizar recursos de enrutamiento específicos para las señales de interfaz y restringir estos recursos a un conjunto compatible para todas las particiones. Esta técnica no provocaría el uso de la LUT extra por cada señal, pero tiene dos inconvenientes. El primero es que su implementación es más compleja. Un *anti-nucleo* es una colección de *nets* enrutadas que utilizan la mayor parte de los recursos de enrutamiento de la partición reconfigurable en cuestión, salvo aquellos cuyo uso se quiere forzar, que se añade al netlist del diseño a enrutar y provoca que todos los recursos usados en el *anti-nucleo* sean ignorados en el proceso de enrutamiento de la partición, forzando el uso de aquellos recursos no presentes en el *anti-nucleo*. Su uso implicaría modificaciones más drásticas al flujo de reconfiguración parcial que incluirían un paso para generar los *anti-nucleos* en función de las características de la partición y las tareas que se quieren implementar y como está documentado en [27], un paso de enrutamiento extra del *router* integrado en la herramienta FPGA_EDITOR para generar el enrutamiento deseado, todo esto adicional al enrutamiento del sistema completo. Esto tiene un inconveniente importante: el algoritmo de enrutamiento utilizado por dicha herramienta es diferente del usado por PAR y sus resultados no tienen la misma calidad. Por supuesto, cabe la posibilidad de que la degradación del enrutamiento provocado por la introducción de una LUT extra, como en la técnica propuesta en este trabajo sea comparable con la degradación generada por el uso del *router* de FPGA_EDITOR, pero no existe documentación suficiente al respecto y su evaluación requeriría la implementación de ambas técnicas.

Otra posibilidad es la generación de macros de enrutamiento mediante FPGA_EDITOR o XDL. El problema de esta técnica es que se vuelve un proceso bastante tedioso. El uso de FPGA_EDITOR implica la edición manual en un IDE de todas las rutas que se deseen tener controladas. El uso de XDL implica un proceso similar pero en modo texto. No se sabe además si el uso de este tipo de macros como interfaces es respetado por PAR para proyectos con reconfiguración parcial en el flujo actual de diseño.

Dados los inconvenientes presentados por estas dos técnicas, en este trabajo han sido

descartadas en favor de la generación de un *router* de propósito específico en conjunto con un nivel extra de interfaz. La implementación de esta técnica, además de servir a los propósitos inmediatos de este trabajo, abre mayores posibilidades a futuro para la generación de herramientas CAD específicas para reconfiguración parcial como se discutirá más adelante.

Hay otro factor relevante en cuanto al enrutamiento que hasta ahora no se ha mencionado. Por defecto, el flujo de herramientas y en particular la herramienta PAR, puede introducir señales estáticas en las particiones reconfigurables. La introducción de estas señales provoca un comportamiento similar al descrito para las señales de interfaz: la generación de conexiones incompletas. Este comportamiento está permitido por lo expuesto previamente: la compatibildad entre particiones no es una característica que esté considerada en el flujo de diseño. Como tal, no está documentado en las guías de usuario de Xilinx como evitar este comportamiento y técnicas como los *anti-core* han sido usadas para forzar la exclusión de los recursos de enrutamiento de las particiones reconfigurables en el enrutamiento de las señales de la lógica estática. Afortunadamente, se puede solicitar asistencia de expertos de Xilinx en sus foros de ayuda. Existen una series de características ocultas en las herramientas de diseño de las cuales se puede obtener información a través de estos foros y una de esas características es la restricción de enrutamiento privado, que justamente cumple el fin que buscamos, evitar la introducción de señales estáticas en la lógica reconfigurable.

Finalmente, la Ilustración 4-6 muestra un ejemplo de dos particiones reconfigurables que cumplen con los criterios necesarios para ser compatibles entre sí. Se muestra la interfaz intermedia entre las fuentes/sumideros y las terminales de partición y de forma simplificada, la existencia de rutas controladas entre esta interfaz y las terminales de partición.

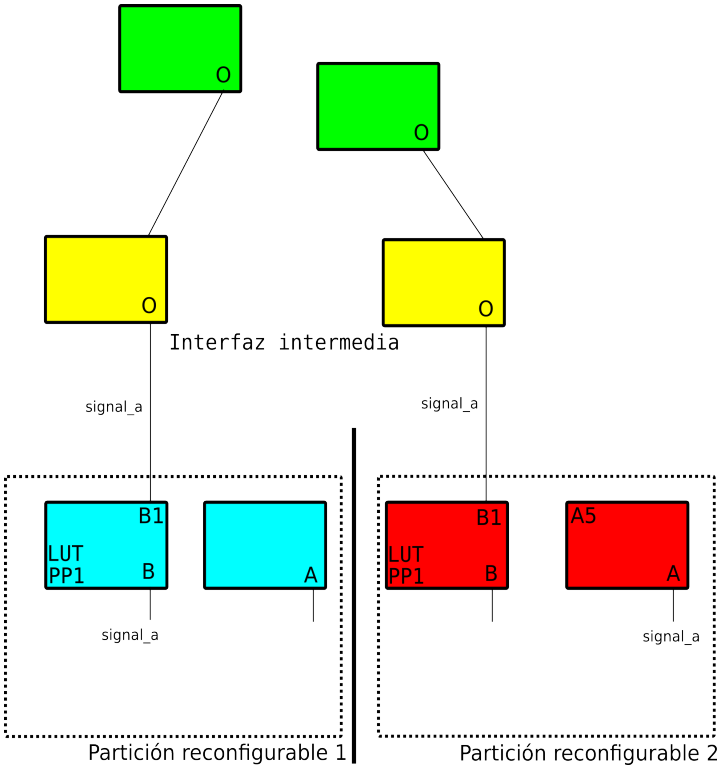

Ilustración 4-6: Particiones reconfigurables compatibles entre si.

4.2.4. Modelos de área y tarea reconfigurable

Fruto del análisis presentado en los puntos 4.2.1 a 4.2.3, es posible proponer un modelo de área reconfigurable y un modelo de tarea reconfigurable cuya implementación sea realmente factible.

El modelo propuesto se basa en el uso del concepto de *slot*, encontrado en propuestas similares reportadas en la literatura [28], pero adaptado a las limitaciones encontradas en la arquitectura Virtex-5, limitaciones que resultan ser mayores que en arquitecturas más antiguas por la falta de información libre sobre el formato de los archivos de configuración de los dispositivos, como se expuso previamente.

Un *slot* es una partición reconfigurable con interfaces de comunicación definidas, estandarizadas y compatibles para todas las instancias presentes en un diseño. Pueden existir, justo como con las particiones, *slots* heterogéneos y homogéneos conviviendo en un mismo diseño. Su existencia se definirá en tiempo de diseño dependiendo de las necesidades del sistema y de la flexibilidad que se desee tener en la implementación y

uso de las tareas de hardware.

El modelo de área propuesto está constituido por un conjunto de *slots* contiguos que pueden ser tratados como recursos individuales, como un solo recurso continuo, o como subconjuntos de recursos cuando una tarea de hardware necesite por su tamaño o número de puertos el uso de mas de un slot. La Ilustración 4-7 muestra un ejemplo de la implementación de este modelo de área con 5 *slots* homogéneos. Una tarea de hardware podrá ocupar cualquier cantidad de *slots* en su implementación y el área podrá ser compartida por hasta 5 tareas de hardware de un *slot* de tamaño.

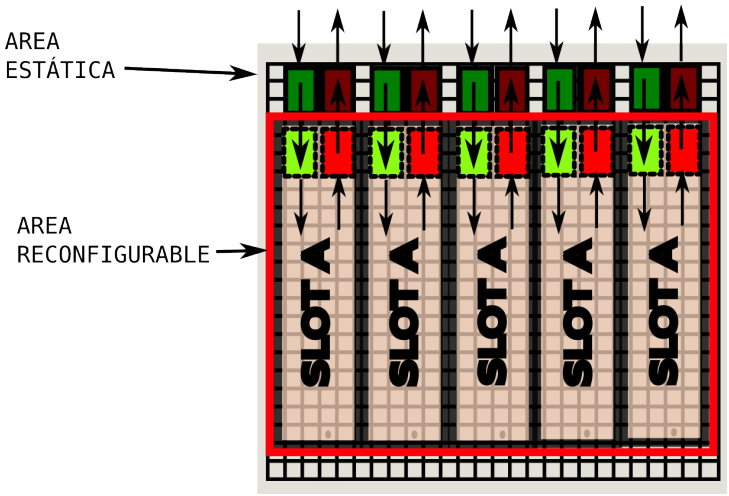

Ilustración 4-7: Implementación del modelo de área propuesto con 5 slots homogeneos.

Un slot estará formado por una cantidad determinada de columnas de recursos, o unidades mínimas de reconfiguración (UMR), como se definió al describir la granularidad de la reconfiguración del dispositivo. Cada *slot* contará con un puerto de entrada y uno de salida del tamaño en bits de la palabra del bus y tendrá asociada una interfaz estática intermedia necesaria para la generación de enrutamiento controlado, como se explicará más adelante.

De esta definición del modelo de área reconfigurable surge la definición de la tarea de hardware reconfigurable. Una tarea de hardware será una función implementada en una cantidad determinada de *slots* en función de la cantidad, tipo de recursos y puertos de entrada/salida que la tarea requiera. La Ilustración 4-8 muestra la relación entre una tarea de hardware de un *slot* de ancho y el *slot* reconfigurable. Se muestran las interfaces

estáticas y dinámicas y el módulo Intellectual Property Interface (IPIF) mediante el cual el *slot* se comunica con el bus del sistema.

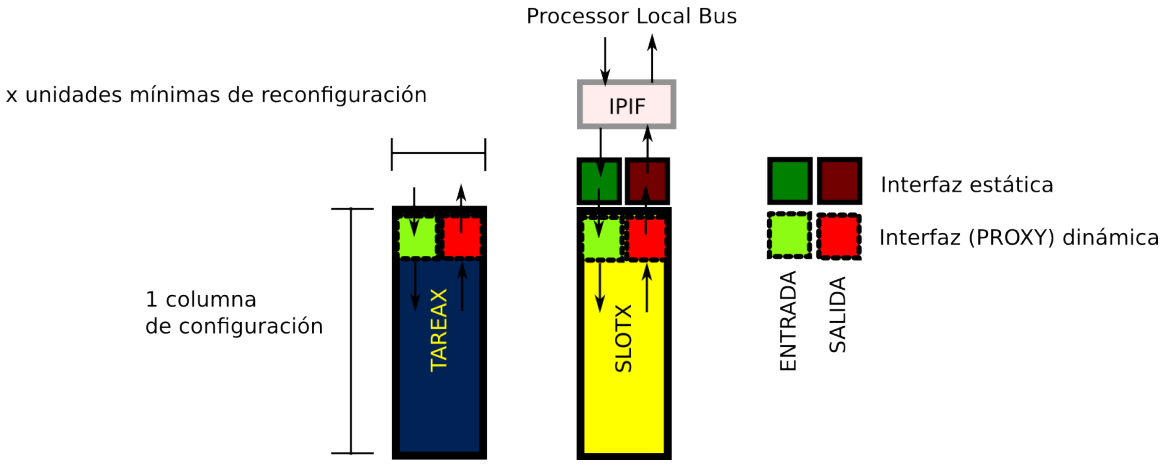

Ilustración 4-8: Relación entre el slot y la tarea de hardware

Estos modelos de área y de tarea ofrecen la flexibilidad de implementar tareas de distinto tamaño y cantidad de puertos. La Ilustración 4-9 muestra el caso más sencillo: una tarea que usa un solo slot y sus puertos I/O.

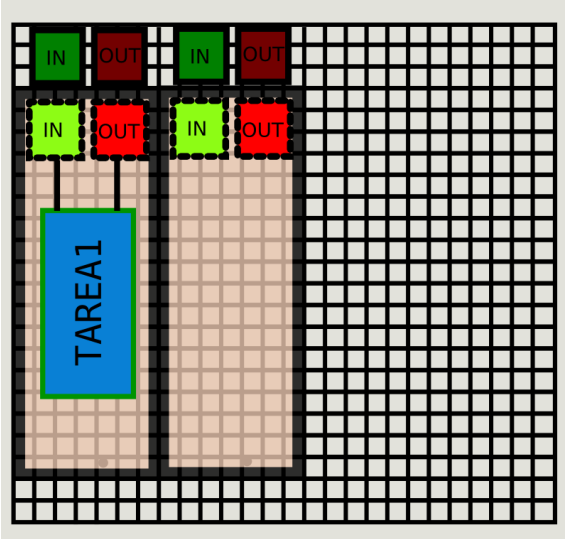

Ilustración 4-9: La tarea 1 solo utiliza un slot y sus puertos I/O

La ilustración 4-10 muestra el caso de una tarea que solo ocupa la lógica de un *slot* pero requiere dos puertos de entrada. El segundo puerto lo tomará del *slot* contiguo al primero. Los dos *slots* se considerarán ocupados.

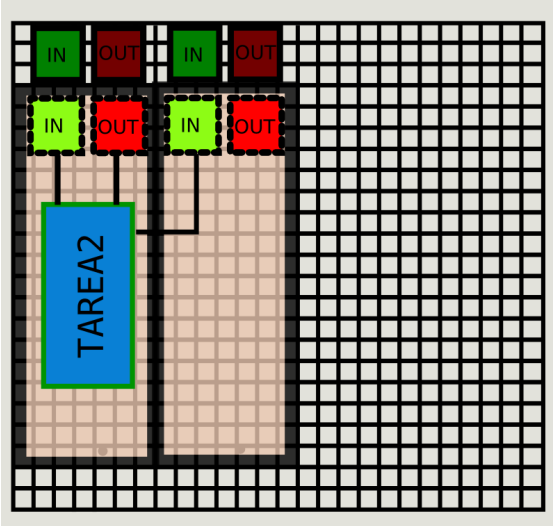

Ilustración 4-10: La tarea 2 ocupará los dos slots, al ocupar el puerto de entrada del segundo.

La ilustración 4-11 muestra el caso de una tarea que sólo requiere un puerto de entrada y uno de salida, pero necesita la lógica de dos *slots* para poder ser implementada. Los dos *slots* se considerarán ocupados.

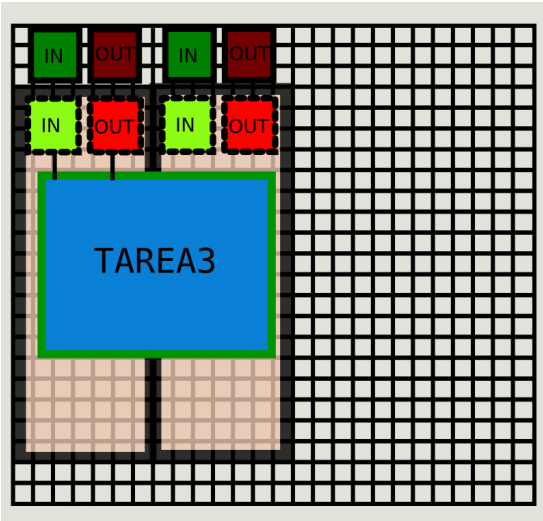

Ilustración 4-11: La tarea 3 ocupará la lógica de dos slots.

Hasta este punto no se ha mencionado nada sobre las señales de reloj en estos modelos. Hasta el avance máximo que ha tenido esta investigación no se ha requerido algún tratamiento especial para estas señales. Las herramientas de desarrollo las resuelven independientemente de las señales de interfaz de las particiones reconfigurables. En tanto un *bitstream* parcial se escriba en una región de reloj donde existan señales de reloj enrutadas, la tarea de hardware representada por ese *bitstream* parcial funcionará sin problemas.

Otra consideración a tomar en cuenta, en particular para el FPGA utilizado en este desarrollo es que no se pueden definir particiones que incluyan columnas de dos mitades (izquierda y derecha) del dispositivo. Esta limitante obedece a que justo por la mitad del dispositivo cruzan columnas de configuración de reloj y éstas no deben ser modificadas.

4.2.5. Recursos de hardware representados en software

Los modelos de área y tarea reconfigurable definen que el mínimo recurso reconfigurable será el *slot* y que el tamaño de cada tarea de hardware estará dado en una cantidad de *slots* contiguos.

De la definición del problema se infiere que el administrador de recursos reconfigurables trabajará con *slots* como su unidad mínima de recursos. Dado nuestro modelo de área, es posible representar los recursos libres como intervalos de *slots* disponibles. En la sección 4.4 se ahondará en este tema.

Con esta breve sección termina nuestro análisis de las posibilidades de la tecnología utilizada y la definición general de nuestras propuestas de solución. Como se podrá observar, por la cantidad de análisis presentado para la parte hardware de este trabajo, ésta es la parte del proceso que define los alcances de un sistema de este tipo, aunque el sistema operativo que se escoja también influirá, las características físicas de las tareas de hardware dependen del FPGA.

Las siguientes secciones describen el desarrollo del hardware y el software embebido basados en este análisis y las propuestas presentadas.

4.3. DESARROLLO DE HARDWARE

La etapa de desarrollo de hardware comprende el desarrollo de un SoPC en un dispositivo FPGA Virtex 5 (XC5VFX70TFFG1136) capaz de soportar el modelo de área y de tarea reconfigurable planteados en la sección 4.2 y la ejecución de un sistema operativo sobre el cual se desarrollará un administrador de recursos reconfigurables y aplicaciones de prueba que se ejecuten simultáneamente y soliciten la programación de tareas de hardware, tomen control sobre ellas y las utilicen. Como parte del desarrollo del SoPC también se genera el conjunto de *bitstreams* parciales de las tareas de hardware que se ejecutarán en el sistema.

El dispositivo FPGA es parte de la tarjeta de desarrollo ML507 de Xilinx y cuenta con un procesador embebido PPC. Esta tarjeta es capaz de soportar la ejecución de software de distinta naturaleza en el procesador PPC. Puede soportar la ejecución de programas en modo *standalone* (un solo programa en ejecución que inicia con el proceso de *bootstrap* del procesador) o una variedad de sistemas operativos que implementen multiprogramación.

Entre los sistemas operativos que la tarjeta de desarrollo puede soportar se encuentran Linux y Xilkernel. En este desarrollo se utiliza Xilkernel, un kernel POSIX desarrollado por Xilinx que se entrega de forma gratuita con el juego de herramientas de desarrollo de sistemas embebidos (EDK). Xilkernel ofrece una API que permite la creación de aplicaciones multihilo con servicios de sincronización como semáforos y cerrojos mutex, mecanismos de comunicación IPC, manejo de memoria dinámica temporizadores en software y manejo de interrupciones. Está fuertemente integrado al flujo de desarrollo de EDK y es altamente configurable dentro del mismo. Una desventaja es la carencia de la gran cantidad de aplicaciones, caracteristicas especiales y herramientas disponibles para sistemas Linux, que ciertamente facilitarían ciertas partes de este desarrollo. Por ejemplo, Xilkernel no ofrece un modo de ejecución en *background*, un shell moderno ni una biblioteca estándar robusta. Por otro lado, la cantidad de periféricos que deben implementarse dentro del FPGA para soportar Linux en esta tarjeta de desarrollo, según las herramientas con las que disponemos para tal fin, tienden a crear congestión en el dispositivo, es decir, utilizar muchos recursos y por ende limitar las posibilidades de

implementación de un área reconfigurable de tamaño significativo. En contraste, Xilkernel necesita una menor cantidad de periféricos y por lo tanto menos recursos del dispositivo. Es esencialmente por este motivo que se utilizará Xilkernel en este desarrollo y por lo tanto, el hardware que se desarrolla deberá soportar su ejecución.

El proceso de desarrollo de hardware se delinea a continuación:

- La base estructural del SoPC se desarrolla dentro de EDK. En esta herramienta se seleccionarán, según las necesidades descritas, los periféricos que el procesador PPC requiera que se implementen en el dispositivo FPGA para soportar las funciones que se desean. EDK ayudará en el proceso de selección de periféricos, selección del tipo de bus, interconexión y generación del mapa de direcciones del procesador. Para este trabajo de investigación en particular, el desarrollo en EDK llegará hasta la generación del *netlist* representativo de la parte estática del diseño reconfigurable. Es decir, dentro del flujo estándar de diseño, cubrirá hasta el proceso de síntesis lógica. Este *netlist* será utilizado en los siguientes pasos de implementación del SoPC y por otra parte en la generación del software embebido mediante el juego de herramientas de desarrollo embebido de Xilinx (SDK).
- Como parte del desarrollo en EDK, el concepto de *slot* reconfigurable se desarrolla como un periférico bajo las convenciones necesarias para ser utilizado dentro de EDK y añadido al SoPC en desarrollo. Es decir, una vez seleccionado el tipo de bus del SoPC, se desarrolla el periférico *slot* como un módulo que puede conectarse al tipo de bus elegido para el sistema en desarrollo. Este proceso utiliza la herramienta para creación de periféricos disponible en EDK, mediante la cual el código VHDL de la interfaz de comunicación con el bus se genera automáticamente y el diseñador sólo requiere desarrollar la lógica interna de trabajo del periférico, también como código VHDL. Para este trabajo, eso sólo implica las conexiones de la partición reconfigurable del *slot* con la interfaz estática intermedia (descrita de forma general en la sección 4.2.3.) y la conexión de ésta interfaz con la interfaz del bus. Se añadirán al SoPC la cantidad de *slots* que la disponibilidad de recursos permita y toda su lógica de funcionamiento formará parte del *netlist* general de la parte estática del diseño reconfigurable.

- Las tareas de hardware reconfigurable se desarrollan independientemente como código VHDL que se sintetiza en un *netlist* sin elementos de conexión al exterior del dispositivo. Como parte del proceso de implementación, cada uno de los *netlist* representativos de las tareas de hardware serán instanciados como parte del netlist general en la partición reconfigurable correspondiente.
- Previo al arranque del proceso de implementación, se definirán las siguientes restricciones:
    - Restricciones AREA_GROUP, de ubicación y de enrutamiento privado para cada partición reconfigurable.
    - Restricciones de localización y de bloqueo de entradas para las LUT de interfaz y los pines de partición.
- Basado en la cantidad y nombres de instancia de las particiones reconfigurables, la cantidad de configuraciones a implementar (configuración como fue definido en el capítulo 3) y las características del sistema se configura el archivo data_impl_custom.tcl y data_impl.tcl. Estos archivos definen las características de la parte reconfigurable del diseño y son utilizados por los *scripts* xpartition_custom.tcl y xpartition.tcl respectivamente para crear la definición de las particiones reconfigurables y ejecutar el flujo de implementación con base en tales definiciones.
- Se ejecuta la primera etapa de implementación con el *script* xpartition_custom.tcl. Este script ejecuta un ciclo de implementación para la primera configuración definida en data_impl_custom.tcl solo hasta el proceso de mapeo (ubicación de elementos lógicos en el dispositivo FPGA). A partir del diseño mapeado entran en acción dos herramientas desarrolladas específicamente para esta aplicación y aplicaciones similares. SlotRouter genera el enrutamiento de las señales existentes entre las terminales de partición y las LUT de interfaz estática de un solo *slot* del diseño utilizando solamente los recursos de enrutamiento de una región particular definida en un archivo de configuración. Esta forma de generar el enrutamiento permite que todas las señales de interés estén localizadas en una región particular y no interfieran con el resto de las particiones reconfigurables, comportamiento que

PAR no prevé. la herramienta MimicRouter propaga el enrutamiento generado para un *slot* al resto de los *slots* del sistema. Es decir, implementa la misma estructura relativa de enrutamiento (la misma relación de coordenadas entre los recursos de enrutamiento utilizados) desplazada horizontalmente a la posición correspondiente a las interfaces de cada *slot*. Al final de este primer proceso de implementación se obtiene el archivo .ncd del enrutamiento e interfaces para todos los *slots*. Este enrutamiento genera las condiciones necesarias para que las tareas implementadas en cada *slot* sean compatibles con el resto y por tanto sean relocalizables.

- Tras esta primera etapa de implementación, se generan restricciones de enrutamiento directo para las interfaces según el archivo .ncd generado por SlotRouter y MimicRouter mediante FPGA_EDITOR. Estas restricciones se añadirán al archivo general de restricciones de implementación. Su uso asegura que el enrutamiento de estas interfaces sea el definido por ellas y no el que PAR generaría en condiciones normales. La secuencia de ejecución de esta primera etapa del flujo de diseño se muestra en la Ilustración 4-12.
- Una vez generadas las restricciones de enrutamiento directo se ejecuta el proceso de implementación del diseño reconfigurable como se describió en el capítulo 3, usando el *script* xpartition.tcl. Al término del proceso se obtendrán el *bitstream* de configuración general representativo del SoPC y un *bitstream* parcial por cada tarea de hardware implementada. Con este paso concluye el proceso de implementación de hardware.

En las siguientes sub-secciones se detallarán los pasos más relevantes de los enlistados previamente para la implementación de un SoPC con 6 *slots* reconfigurables en un área heterogenea. Este sistema soportará la ejecución de Xilkernel y utilizará como salida estándar una terminal serial. Se describirá el desarrollo de las herramientas SlotRouter y MimicRouter como parte esencial de este nuevo flujo de diseño.

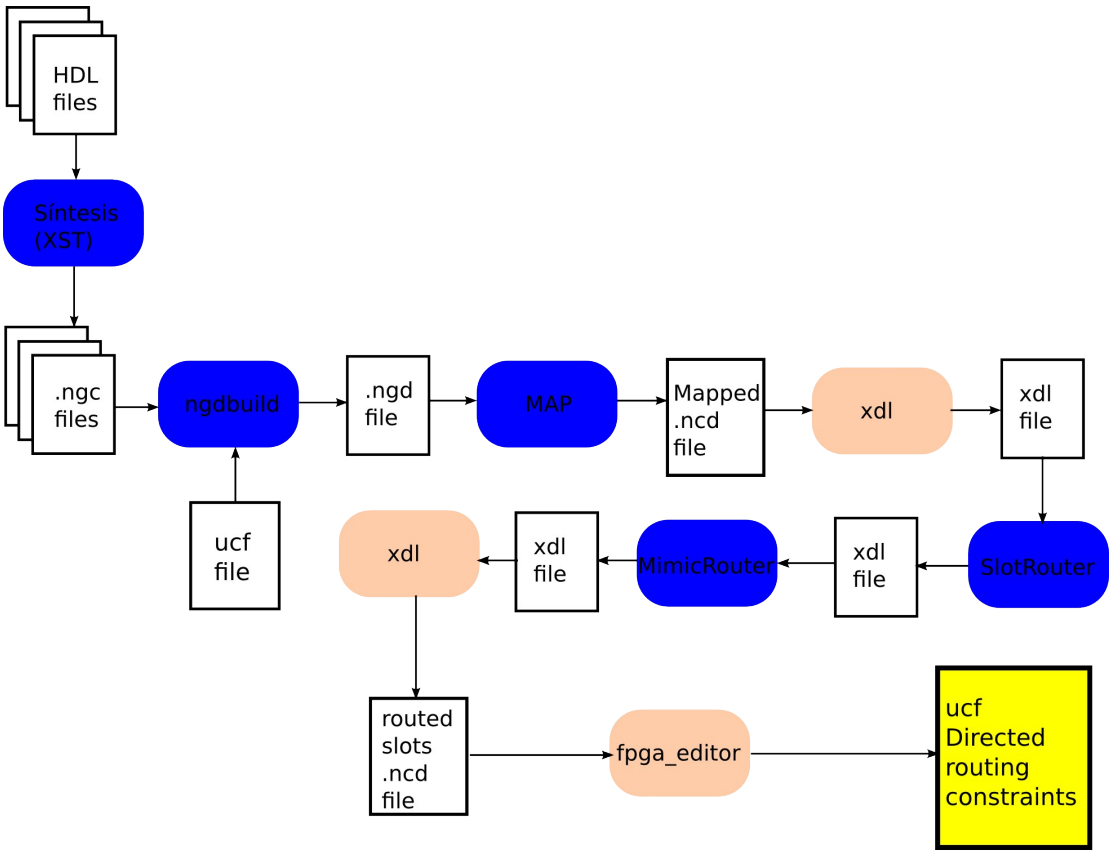

Ilustración 4-12: Primera etapa del flujo de diseño. Obtención de las restricciones de enrutamiento directo para las interfaces.

4.3.1. Diseño de la plataforma base del SoPC

Como se ha mencionado previamente, el SoPC implementado en el dispositivo FPGA deberá soportar dos características: ejecución de Xilkernel y reconfiguración parcial. El sistema trabajará con un bus tipo PLB a una frecuencia de 125MHz. Incluirá la memoria RAM externa de la tarjeta de desarrollo. Dispondrá del módulo de acceso a la memoria Compact Flash. Esta memoria funcionará como repositorio de *bitstreams* parciales. Un módulo UART configurado con un *baudrate* de 9600 se define como entrada y salida estándar. Como se mencionó previamente, el sistema contará con seis instancias del módulo *slot*, cuyo funcionamiento se describirá en la siguiente sección.

De acuerdo a [27], Xilkernel necesita un temporizador para controlar la planificación de procesos y un controlador de interrupciones. Para sistemas con procesador PPC, Xilkernel utiliza el temporizador interno del procesador. El módulo xps_intc se añadirá al sistema como controlador de interrupciones . Mas allá de estas dos necesidades, un sistema que

soporte Xilkernel puede ser tan sencillo o complejo como se desee.

El soporte de reconfiguración parcial necesita una interfaz de programación con el FPGA. Dado que este sistema necesita ser auto-reconfigurable la opción es la inclusión del módulo xps_hwicap. Este módulo permite el acceso al puerto de acceso a la configuración interna (ICAP) del FPGA. Es importante considerar que este módulo tiene restricciones de frecuencia de trabajo. Aunque puede trabajar a un máximo de 100MHz, se recomienda utilizarlo a frecuencias sobre 50MHz pero menores a los 100MHz que teóricamente puede alcanzar pues su correcto funcionamiento a esta frecuencia no es un objetivo alcanzable en el 100% de las implementaciones. En este sistema el módulo xps_hwicap trabajará a 62.5MHz.

El armado de esta estructura básica se realiza completamente en EDK. La tabla 4-1 muestra la lista de componentes incluidos en el sistema y la Ilustración 4-13 muestra el diagrama de bloques del mismo.

Función	***Componente***
Procesador	ppc440
Controlador de interrupciones	xps_intc
Bus	plb_v46
Memoria	• xps_bram_if_cntrl_1_bram • DDR2 SDRAM (externa)
Controladores de memoria	• xps_bram_if_cntrl_1 • DDR2_SDRAM
Periféricos	• RS232_UART • SysACE_CompactFlash • jtagppc_cntrl_inst • reset • 6 slots • xps_hwicap • xps_timer
IP	clock_generator

Tabla 4-1: Componentes del SoPC

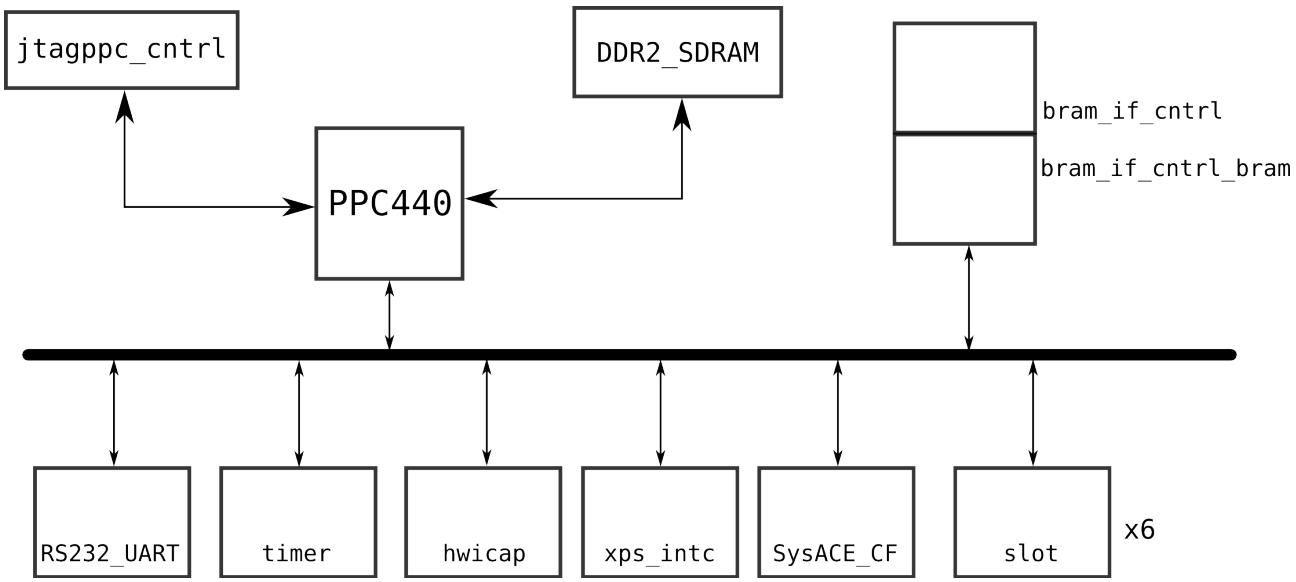

Ilustración 4-13: Diagrama de Bloques del SoPC

4.3.2. Diseño del *slot* reconfigurable y las interfaces estáticas

El *slot* reconfigurable se diseña como un periférico para el bus PLB. Esto implica el uso de la interfaz IPIF para su conexión con el bus y la creación de su lógica interna. Como todos los periféricos de usuario creados mediante la herramienta de creación o importación de periféricos, el periférico *slot* consistirá de al menos dos archivos VHDL: el archivo slot.vhdl y el archivo user_logic.vhdl. El segundo contiene la implementación del componente user_logic, que contiene la lógica del funcionamiento específico que se desea que implemente el periférico. El archivo slot.vhdl instancía el componente user_logic e implementa toda la lógica de la interfaz de comunicación con el bus. Este archivo es completamente generado por la herramienta de creación o importación de periféricos de acuerdo a la configuración que se haya indicado en esta herramienta. Esta configuración incluye entre otras cosas la generación de interrupciones, el uso del reset de software, etc. Para el caso de este periférico en particular, se configura el uso de reset de software, pero no el uso de interrupciones. Se configura además el uso de un registro interno mediante el cual se escribirán datos al periférico.

El archivo user_logic.vhdl instancia la lógica de la partición reconfigurable a través del componente rpX, donde X es el índice del *slot*. Es importante aclarar que se deberán crear tantos periféricos como *slots* se deseen, aunque entre todos la única diferencia será este índice. Este es un problema de las herramientas de implementación que no permiten

tener componentes del mismo nombre en distintas particiones. Durante el proceso de implementación, rpX será ocupado por el netlist de la tarea de hardware que se desee implementar en la partición reconfigurable en cuestión.

Además de la lógica de la partición reconfigurable, user_logic instancia la interfaz estática intermedia, formada por dos componentes: *ininterface* e *outinterface*. Estos componentes se definen en archivos VHDL del mismo nombre. Como se puede inferir, *ininterface* implementa la interfaz de entrada y *outinterface* la interfaz de salida. La Ilustración 4-14 muestra el diagrama de bloques del periférico *slot*.

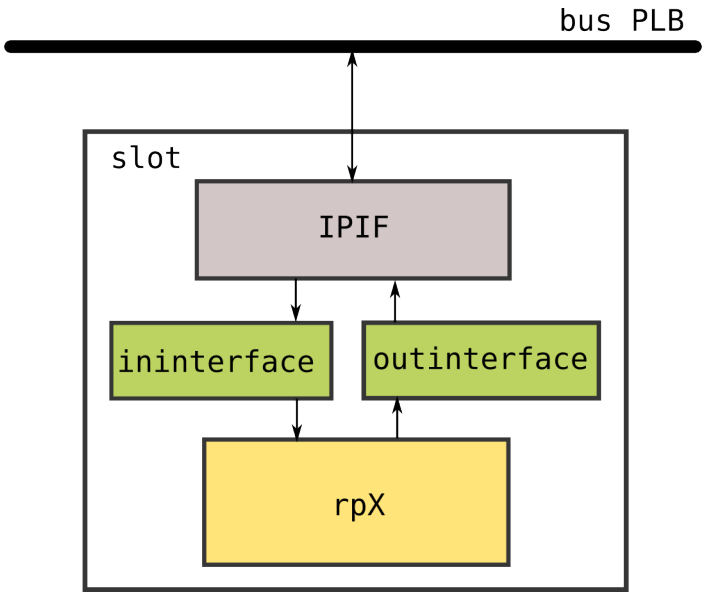

Ilustración 4-14: Diagrama de bloques del periférico slot

A pesar de ser sólo puentes de señales, las interfaces se definen como componentes diferentes porque manejan una cantidad diferente de señales. El componente *ininterface* recibe y entrega los 32 bits del puerto de entrada de la partición reconfigurable mas la señal de reset, mientras que el componente *outinterface* solo recibe y entrega los 32 bits del puerto de salida de la partición reconfigurable. Por cada señal que manejan las interfaces se debe instanciar una LUT en modo de enrutamiento a través. Estas LUT se instancian directamente como elementos primitivos del dispositivo Virtex-5. El parámetro INIT indica la función lógica que implementará la LUT. El siguiente listado muestra la definición de las entidades para ambas interfaces y la instanciación de una LUT de interfaz.

```
entity ininterface is
    Port ( iport : in  STD_LOGIC_VECTOR (0 to 31);
           resi  : in  STD_LOGIC;
           oport : out  STD_LOGIC_VECTOR (0 to 31);
           reso  : out STD_LOGIC
    );
end ininterface;
...

entity outinterface is
    Port ( iport : in  STD_LOGIC_VECTOR (0 to 31);
           oport : out  STD_LOGIC_VECTOR (0 to 31));
end outinterface;
...
-----------------------------
-- Instanciacion directa de un LUT de interfaz
-----------------------------
LUT6_inst0: LUT6 generic map (
                       INIT => X"AAAAAAAAAAAAAAAA"
              )
              port map(
                       O => sigoport(0),
                       I0 => sigiport(0),
                       I1 => '0',
                       I2 => '0',
                       I3 => '0',
                       I4 => '0',
                       I5 => '0'
              );
```

Listado 4-1: Entidades de interfaz de entrada/salida e instanciación de un LUT de interfaz.

Las LUT de interfaz irán acompañadas por restricciones de ubicación que aseguren que para cada *slot* se encuentren ubicadas en las mismas posiciones relativas. De la misma las terminales de partición serán acompañados por restricciones del mismo tipo. Los dos juegos de restricciones juntos aseguran una implementación como la que se observa en la Ilustración 4-8. La forma de definir estas restricciones se tratará en el apartado 4.3.4.

El listado 4-2 muestra la instanciación de la partición reconfigurable y las dos interfaces estáticas en el módulo *user_logic*. La señal *slv_reg0* viene del registro de entrada de la interfaz IPIF y alimenta a la interfaz de entrada. La señal *result* entregará la salida de la partición reconfigurable a la lógica de la interfaz IPIF que permite que el procesador lea el

resultado del módulo instanciado en la partición reconfigurable.

```
--USER logic implementation added here
      -- INTERFAZ DE ENTRADA
      interfaz_in_instance: ininterface port map(
                        iport => slv_reg0,
                        resi => Bus2IP_Reset,
                        oport => rp_in,
                        reso => sigres
      );
      -- MODULO RECONFIGURABLE
      rp_instance: rp0 port map (
                        inputport => rp_in,
                        clk => Bus2IP_Clk,
                        reset => sigres,
                        outputport => rp_out
      );
      -- INTERFAZ DE SALIDA
      interfaz_out_instance: outinterface port map(
                        iport => rp_out,
                        oport => result
      );
```

Listado 4-2: Instanciación de interfaces y partición reconfigurable

El resto de los detalles de implementación son similares a los de cualquier módulo generado mediante la herramienta de creación o importación de periféricos.

4.3.3. Diseño de las tareas de hardware

Las tareas de hardware que se implementen deberán seguir la convención del tamaño de palabra de los puertos de entrada y salida y el uso de una señal de reset. Las primeras tareas que se implementan para este trabajo son muy sencillas pues no se busca la implementación de un sistema muy complejo sino la demostración de la validez de las técnicas utilizadas para implementar un sistema como el que se ha propuesto. Así se ha decidido implementar las funciones más sencillas posibles: un inversor de 32 bits y un buffer de 32 bits.

Se escribe el código de las funciones en VHDL y se sintetizan en archivos *netlist* con la particularidad de que se debe evitar la inserción de componentes de entrada/salida al exterior del dispositivo.

4.3.4. Definición de restricciones

Por cada partición reconfigurable se definen restricciones AREA_GROUP, RANGE y de enrutamiento privado. Estas restricciones definen la ubicación física en el dispositivo de cada partición reconfigurable y prohiben el acceso de rutas de la parte estática a la partición reconfigrable Además, para las interfaces de cada partición y las terminales de partición se definen restricciones de ubicación (LOC y BEL) y de bloqueo de terminales (LOCK_PINS). Estas restricciones aseguran una ubicación de estas interfaces que es compatible entre todas las particiones y el uso de la misma entrada (A0) para todas las LUT.

Para esta implementación en particular se ha elegido tener *slots* de 2 CLB de ancho con dos *slots* heterogéneos que contienen bloques de BRAM. El ancho de 2 CLB se debe a la heterogeneidad del dispositivo que provoca agrupamientos de columnas de CLB que son múltiplos de dos como muestra la Ilustración 4-15. Un ancho mayor a 2 CLB reduciría el número de *slots* que se pueden implementar, que de por si resulta pequeño, y el sistema resultaría menos demostrativo en cuanto al manejo de los recursos en el administrador software que tendría que lidiar con menos recursos. No obstante, este es un parámetro que aún tiene que ser estudiado, pues *slots* tan pequeños no permiten la implementación de tareas de hardware complejas. La inclusión de *slots* heterogéneos obedece a la necesidad de implementar un sistema que demuestre varios conceptos, en este caso el manejo de heterogeneidad, y que pueda lidiar con áreas continuas de recursos. No incluir *slots* heterogéneos forzaría la generación de áreas reconfigurables discontinuas y limitaría aún más el tamaño de las tareas de hardware que podrían albergar.

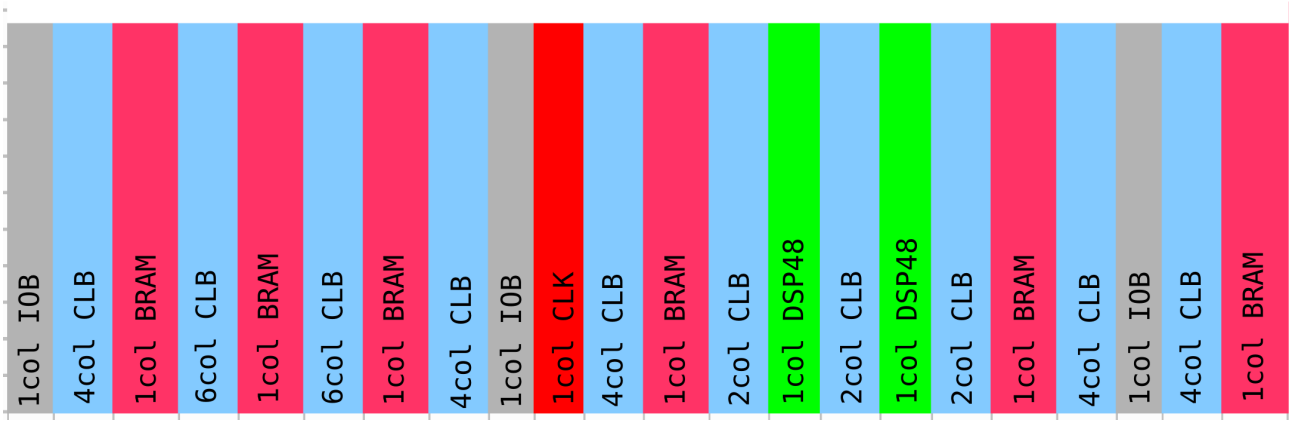

Ilustración 4-15: Distribución de columnas de recursos en el dispositivo

Se ha elegido además colocar el área reconfigurable en la posición más lejana posible de las zonas de mayor congestión. Esto es, en la fila más baja del dispositivo y en la mitad izquierda. Esta zona resulta la de menor congestión por la falta de periféricos externos que necesiten conexiones de entrada/salida con posiciones específicas en el dispositivo debido al alambrado externo al FPGA para la tarjeta de desarrollo. La Ilustración 4-16 muestra la ubicación del área reconfigurable en el dispositivo.

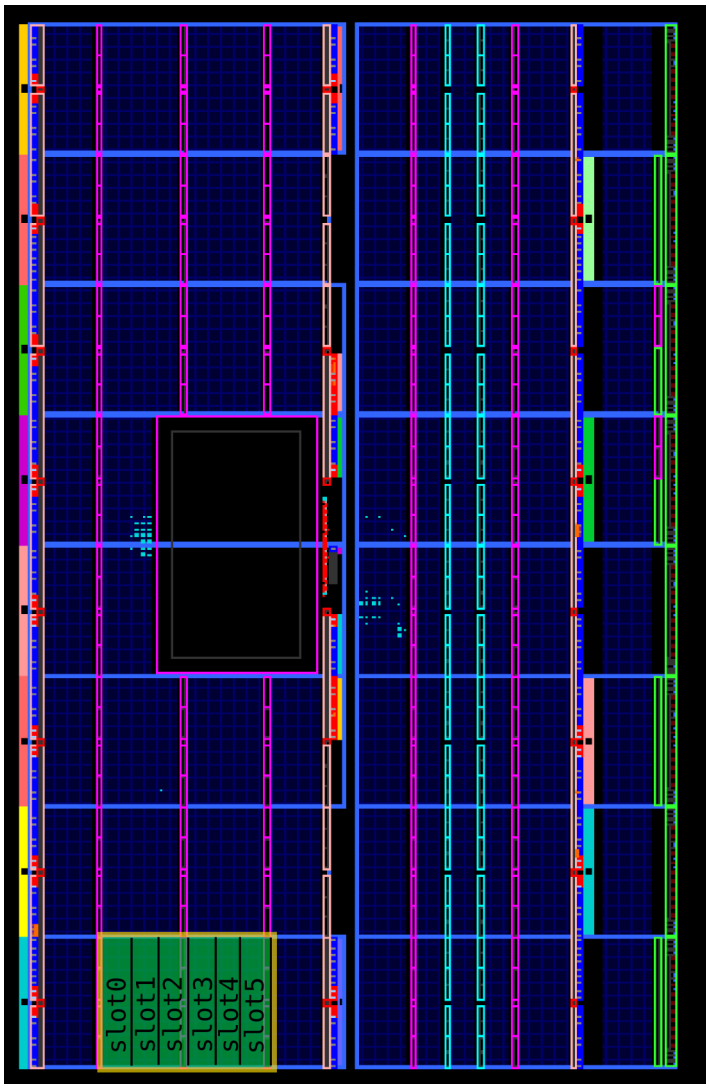

Ilustración 4-16: Localización del área reconfigurable en el dispositivo

La Ilustración 4-8 mostró la localización general de los pines de partición y las interfaces estásticas. Esta localización es importante. La interfaz de salida y la interfaz de entrada están ubicadas en un mismo extremo de la partición reconfigurable. De esta forma se evita la generación de rutas más largas y se facilita el enrutamiento. Para esta

implementación en particular, por la posición elegida para el área reconfigurable, no sería posible colocar interfaces estáticas en el otro extremo de los *slot*, pues se encuentran en el límite inferior del dispositivo. Aún si existiera espacio para colocar las interfaces en el extremo más alejado del procesador, esto no sería conveniente. La mayor parte del enrutamiento del bus plb se realiza en las cercanias del procesador; si alguna interfaz estuviera en el extremo de la interfaz mas lejano al procesador, PAR tendría que generar rutas que rodearan el área reconfigurable y podría necesitar mas recursos de enrutamiento.

La Ilustración 4-17 muestra la posición exacta de las interfaces y las terminales de partición para un *slot*. Los nombres que ostentan son los mismos que en el módulo que los implementa. Se muestra su posición por SLICE, no por CLB (recordemos que un CLB está formado por dos SLICE). Habrá 4 terminales de partición (en la Ilustración se muestra el intervalo dentro de la señal) por SLICE, es decir, se ocupan las 4 LUT de cada SLICE seleccionado .Se muestra también la posición de los bloques de interconexión (INT_TILE) con respecto a los SLICE. La interfaz de entrada se muestra al lado izquierdo de la Ilustración mientras la interfaz de salida se encuentra a la derecha.

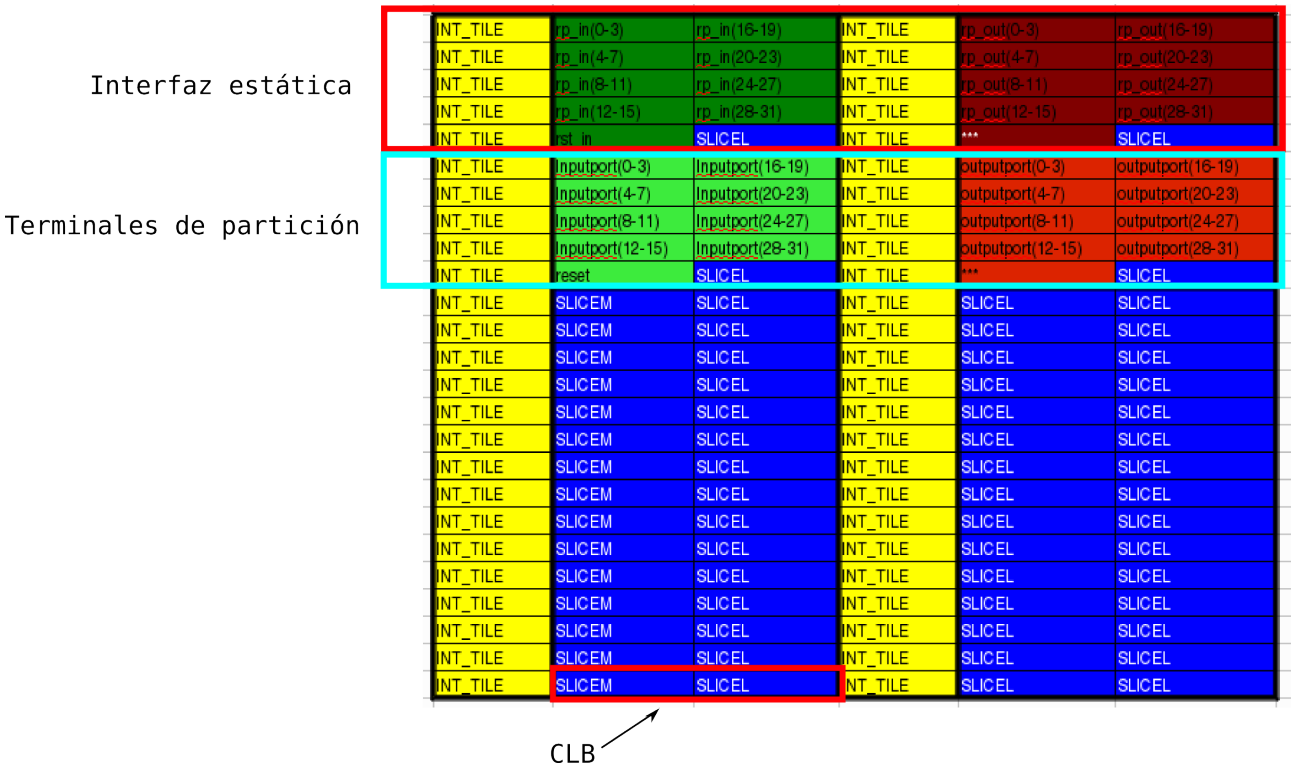

Ilustración 4-17: Posición de la interfaz estática y los pines de partición para un SLOT.

El Listado 4-3 muestra la definición de las restricciones AREA_GROUP, RANGE y de enrutamiento privado para un *slot* y las restricciones de colocación, bloqueo de pines para un pin de partición y su contraparte en la interfaz estática.

```
#---------------------------------------
# slot0_0
#---------------------------------------
INST "slot0_0/slot0_0/USER_LOGIC_I/rp_instance" AREA_GROUP =      \
       "pblock_slot0_0_USER_LOGIC_I_rp_instance";
AREA_GROUP "pblock_slot0_0_USER_LOGIC_I_rp_instance" RANGE=SLICE_X8Y0:SLICE_X11Y19;
AREA_GROUP "pblock_slot0_0_USER_LOGIC_I_rp_instance" PRIVATE=ROUTE;




PIN "slot0_0/slot0_0/USER_LOGIC_I/rp_instance.inputport<0>" LOC = SLICE_X8Y19;
PIN "slot0_0/slot0_0/USER_LOGIC_I/rp_instance.inputport<0>" BEL = A6LUT;
INST "slot0_0/slot0_0/USER_LOGIC_I/rp_instance/inputport<0>_PROXY" LOCK_PINS=ALL;

INST "slot0_0/slot0_0/USER_LOGIC_I/interfaz_in_instance/LUT6_inst0" LOC = SLICE_X8Y24;
INST "slot0_0/slot0_0/USER_LOGIC_I/interfaz_in_instance/LUT6_inst0" BEL = A6LUT;
INST "slot0_0/slot0_0/USER_LOGIC_I/interfaz_in_instance/LUT6_inst0" LOCK_PINS = ALL;
```

Listado 4-3: Definición de restricciones para un slot y sus interfaces

4.3.5. Generación de restricciones de enrutamiento directo: Diseño de las Herramientas SlotRouter y MimicRouter

Después de haber cubierto los pasos previamente enlistados y definir las variables el flujo de implementación como se detalla en [29], se ejecuta una primera fase del flujo de implementación para obtener un archivo .ncd de la primera configuración con la ubicación física de los elementos que conformarán el diseño completo. Como se ha descrito en la sección 4.2.3, las particiones no serán compatibles debido al enrutamiento. Es aquí donde este trabajo ofrece una solución con la estandarización del enrutamiento de las señales que comunican la interfaz estática con los pines de partición. Para tal fin se han desarrollado dos herramientas basadas en el API RapidSmith [25]. Estas herrramientas son un *router* con definición de recursos de enrutamiento permitidos por *net* (*slotRouter*) y un router de imitación (*mimicRouter*).

La herramienta *slotRouter* permite definir una lista de *nets* a enrutar y una región de

recursos de enrutamiento que la lista de *nets* tendrá permitido utilizar. Se utiliza para enrutar los *nets* de las señales entre interfaces para un *slot*. La definición de recursos permitidos evita que el enrutamiento generado invada las particiones reconfigurables del resto de los *slot* de un diseño.

La herramienta *mimicRouter* toma el enrutamiento generado por *slotRouter* y lo imita para las interfaces del resto de los *slots* presentes en el diseño. Es decir, utiliza recursos de enrutamiento que se encuentren en posiciones relativamente iguales para todos los *slots*. Al final de su ejecución entrega un enrutamiento compatible para todos los *slots* a partir del cual se obtendrán restricciones de enrutamiento directo que se añadirán al archivo de restricciones del flujo de diseño y que estandarizan el enrutamiento para todas las configuraciones que se implementen.

Ambas herramientas están basadas en la API de Java RapidSmith. Esta API permite manipular todos los elementos de un diseño en formato XDL como si fueran objetos. También permite manejar de igual forma los elementos de un dispositivo FPGA de Xilinx. Así existen objetos representativos de las instancias, de las *nets*, de las terminales, de los cables y de un concepto importante en la arquitectura de los dispositivos FPGA de Xilinx: el *tile*. Para los dispositivos Xilinx, todos los recursos agrupados en columnas tales como CLB, BRAM, IOB, DSP48 y las matrices de enrutamiento son un tipo de *tile* con coordenadas (x, y) que indican su posición en el dispositivo. Esta característica es de particular utilidad para el desarrollo de mimicRouter, como se verá más adelante.

La herramienta *slotRouter* basa su funcionamiento en un sencillo *router* implementado con un algoritmo de búsqueda por amplitud *(breadth first search,* BFS). BFS es un algoritmo de búsqueda en un grafo que inspecciona para cada nodo, partiendo de la raíz, a todos los vecinos que no hayan sido visitados. Añade a los vecinos a una cola de donde va tomando nodos por visitar. Sin entrar en detalles, BFS genera un árbol de búsqueda en el que se asegura que cada nodo encontrado y enlazado al árbol de búsqueda se encontrará a la distancia mínima (distancia medida en aristas) del nodo raíz.

Para nuestro caso particular, la estructura de enrutamiento del FPGA es un grafo, que como tal no existe en rapidSmith ni se genera en *slotRouter*, sino que se induce por la propia estructura de los dispositivos y de rapidSmith. Los nodos de este grafo son una

dupla (*tile*, cable). Cada *tile* tiene un conjunto de cables que la conectan con otros *tiles* o internamente con si misma. Estos cables pueden tener nombres diferentes en cada *tile* que conectan aún siendo físicamente el mismo cable. Cada cable puede conectarse con una cantidad variable de otros cables. Estas conexiones se realizan al activar un punto de interconexión programable (PIP) particular. Ilustración 4-18 muestra un ejemplo de esta relación de elementos.

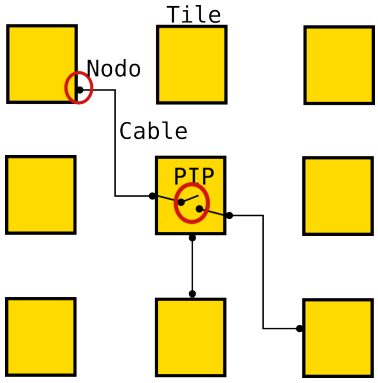

Ilustración 4-18: Tile, cable, nodo y PIP.

Es esta conjunción entre *tiles*, cables y PIP lo que induce la generación de un grafo de interconexión en el dispositivo. Es este grafo sobre el que BFS trabaja en *slotRouter*. Su función, para cada *net* que se quiera enrutar, será buscar un camino de conexión formado por una colección de nodos entre el nodo fuente de la red (la raiz para BFS) y el sumidero de la misma. Este tipo de *net* es un caso particular de enrutamiento, muchas *nets* para otros usos tendrán más de un sumidero y su resolución será más complicada. Para este caso, con BFS será suficiente, considerando además, que la información de latencias para los dispositivos de Xilinx no es información de acceso libre, por lo que no se puede esperar optimizar con las latencias como criterio. Más aún, ni siquiera parece ser necesario con *nets* tan cortas.

Se utiliza un archivo de configuración para *slotRouter.* Este archivo contiene las coordenadas que delimitan las regiones de entrada y salida del *slot* a enrutar. Los límites verticales de estas regiones se encuentran en la parte inferior de la región de pines de partición y la parte superior de la interfaz estática. Los límites horizontales están definidos por los límites de ambas regiones, que como se observa en la Ilustración 4-17, son los

mismos. Es decir, las regiones que se definen en el archivo de configuración de *slotRouter* son rectángulos que abarcan la unión de una interfaz estática y su par de pines de partición. Esta definición permite delimitar los recursos permitidos para el enrutamiento de los *nets* pertenecientes a este par interfaz/terminales de partición. Al enrutar una *net, slotRouter* utilizarán sólo los recursos delimitados por la región definida en su archivo de configuración, y en particular para esta implementación, aquellos que se encuentren entre las coordenadas verticales de la fuente y el sumidero de la *net*. La Ilustración 4-19 muestra esta idea.

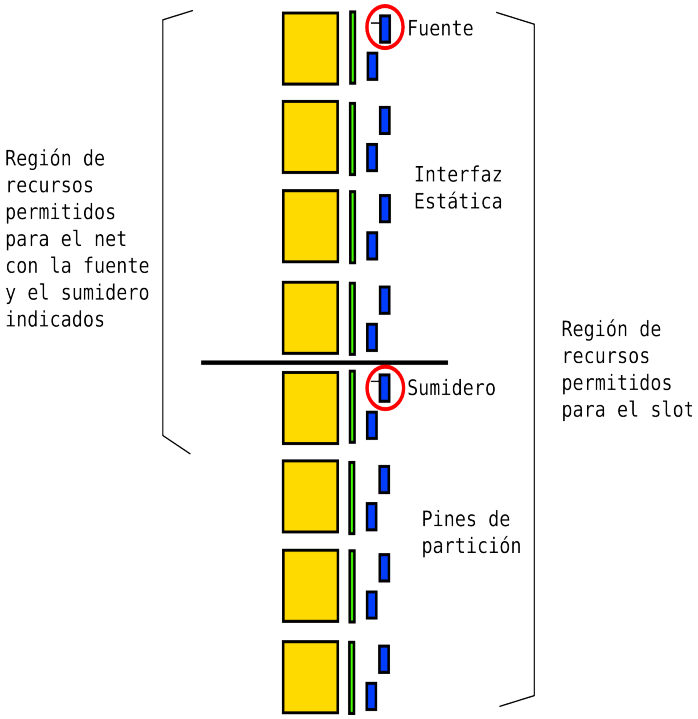

Ilustración 4-19: Región de enrutamiento para una net

El archivo de configuración define además de estas regiones de enrutamiento, la coordenada *x* del inicio de cada *slot*. Estas coordenadas permitiran, para un *net*, mediante el método *reserveCriticalNodes(Net net)* reservar de los recursos permitidos de enrutamiento para el *slot,* una serie de nodos que la *net* requiere forzosamente para resolver su enrutamiento.

SlotRouter itera sobre la colección de *nets* a enrutar, reserva sus nodos críticos, resuelve su enrutamiento mediante BFS, retira los nodos recientemente utilizados de la lista de recursos de enrutamiento disponibles y añade al *net* los PIP que deben ser activados para

generar el enrutamiento que se resolvió.

La herramienta *mimicRouter* toma el resultado de *slotRouter* y lo imita para el resto de los *slots* basándose en un archivo de configuración que le indica cuantos *slots* existen y en donde se ubica, y los nombres de las *nets* que debe imitar.

La idea es muy simple: *mimicRouter* tomará cada una de las *nets* definidas en su archivo de configuración y para cada una de ellas recorrerá su lista de PIP, obtendrá de cada PIP *pc* las coordenadas (*x, y)* de su *tile* y los identificadores de los cables que conecta el PIP. Calculará la distancia *diff* horizontal entre el *slot* enrutado por *slotRouter* y cada uno de los *slots* restantes y generará un PIP con los mismos cables obtenidos del PIP *pc* pero en un *tile* desplazado la distancia *diff* para cada *slot*. De esta forma asegurará el uso de recursos de enrutamiento que ocupen las mismas posiciones relativas a los *slots*. La Ilustración 4-20 muestra esta idea para un solo *net*.

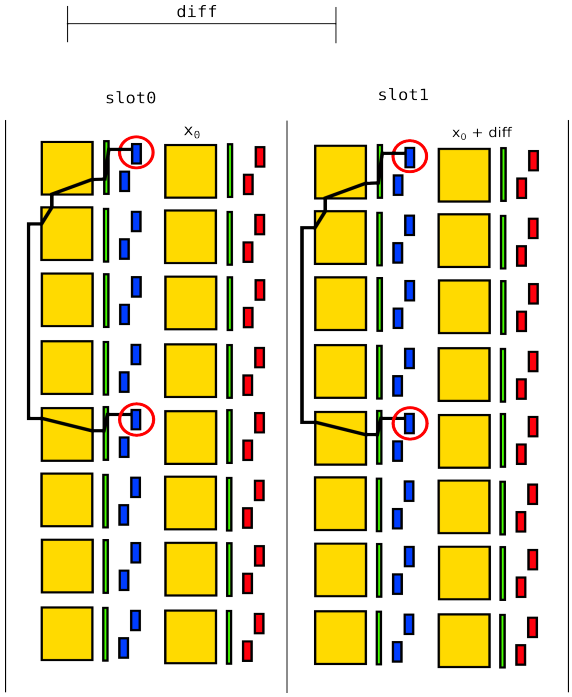

Ilustración 4-20: Funcionamiento básico de mimicRouter

Tras la ejecución de *mimicRouter* se obtienen las restricciones de enrutamiento directo para las *nets* de interfaz y se añaden al archivo de restricciones del diseño. Una vez que estas restricciones han sido añadidas, se ejecuta el flujo de implementación de forma normal. Al final se obtendrán los *bitstreams* completos y parciales para el SoPC.

Es importante tener en cuenta que esta implementación, tal como se ha descrito hasta este punto solo genera tareas de tamaño 1 *slot*, pero no está limitada al uso de tareas de ese tamaño. La generación de tareas de diferentes tamaños implica el armado de otros proyectos en EDK con las mismas características, salvo por las características de los *slots* y su cantidad, que tendrán que coincidir con las características de las tareas que se quieran implementar. El proceso es largo, no obstante, la mayor parte de los archivos utilizados en el flujo de implementación se pueden reutilizar sin grandes modificaciones. Un punto muy importante es que las restricciones de enrutamiento directo generadas por las herramientas aportadas por este trabajo se pueden re-utilizar si acaso con modificaciones a los nombres de los *nets* a las que estas restricciones pertenecen. Estas restricciones son completamente específicas del dispositivo y definen, en una sintaxis solo reconocible por las herramientas de Xilinx, los recursos específicos de enrutamiento que un *net* debe utilizar. Así, si en un nuevo flujo de implementación se desea generar una tarea que ocupe los dos primeros *slots* del área reconfigurable, las restricciones de enrutamiento directo generadas para esos dos *slots* en la primera implementación deberán usarse para conservar la compatibilidad de esta nueva tarea con el área reconfigurable definida en la primera implementación. Este proceso se repetirá tantas veces sea necesario de acuerdo a la cantidad y características del resto de tareas que se quiera implementar.

Con las consideraciones del párrafo anterior termina la descripción del proceso de desarrollo del SoPC capaz de soportar tareas relocalizables de tamaño y número de entradas variables. Como se ha hecho notar, es un proceso largo, pero no existen alternativas conocidas. El fabricante no soporta este tipo de implementaciones y aunque fuentes de Xilinx han declarado que se estudia la posibilidad de implementar el soporte para estas características, no parece estar cerca ese día, el flujo actual tiene limitaciones importantes y aún no hay estándares que las herramientas deban seguir. Es por ello que presentamos estas herramientas alternativas que aunque aún son perfectibles y sobre todo se pueden complementar con herramientas de automatización del flujo, abren caminos interesantes en este poco explorado terreno. Las posibilidades que éstas herramientas ofrecen a futuro se discutirán en el capítulo 6.

4.4. DESARROLLO DE SOFTWARE

La etapa de desarrollo de software comprende el diseño del administrador de recursos reconfigurables dentro de Xilkernel y de una aplicación multihilos en la que cada hilo solicita la programación en el dispositivo FPGA de una tarea de hardware y espera respuesta del administrador. Ante una respuesta positiva, el hilo solicitante toma posesión de la tarea de hardware y la utiliza. Una vez terminado su uso, solicita el borrado de la tarea de hardware al administrador. Ante una respuesta negativa, el hilo continúa su ejecución, y es su responsabilidad tomar una decisión pertinente ante la negativa.

Previo a esta etapa, se describe el desarrollo de un par de herramientas utilizadas durante las primeras fases de evolución de este trabajo orientadas a la interpretación de los archivos de configuración *bitstream*. Estas herramientas sirvieron como base para la implementación de la función de relocalización de *bitstreams* que permite cambiar las coordenadas de escritura de una tarea de hardware logrando con esto su relocalización.

4.4.1. Herramientas de interpretación y modificación de *bitstreams*

Tal como este capítulo inicia con un análisis de las posibilidades de la tecnología utilizada en este trabajo, el desarrollo del mismo inició con la creación de herramientas que nos permitieran explorar estas posibilidades de forma directa en los archivos de configuración del dispositivo. La primera herramienta desarrollada fue *readbitstream*, un programa que hace un volcado del contenido de un archivo *bitstream* traduciendo los comandos que aparecen en él y agrupando las palabras que forman los *frames* en una forma fácil de leer y en la medida de lo posible interpretar. Esta herramienta permitió confirmar la posición de la configuración de cada CLB en un *frame* y mediante la lectura de un *bitstream* de depuración, conocer la secuencia con la que el bitstream se escribe en memoria. El Listado 4-4 muestra un resumen del volcado de un bitstream como lo despliega *readbitstream*. Se puede ver la secuencia completa de configuración desde la palabra de sincronización, en forma de resumen la escritura de las 2393 palabras de configuración del dispositivo y en la parte final los comandos de inicialización del dispositivo.

```
Leyendo bitstream inv0.bit
SYNCWORD:      AA 99 55 66
20 00 00 00    Header type 1  NOP
30 00 80 01    Header type 1  WRITE   CMD     1
```

```
00 00 00 07
20 00 00 00    Header type 1  NOP
20 00 00 00    Header type 1  NOP
30 01 80 01    Header type 1  WRITE   IDCODE 1
03 2C 60 93
30 00 80 01    Header type 1  WRITE   CMD    1
00 00 00 01
20 00 00 00    Header type 1  NOP
30 00 20 01    Header type 1  WRITE   FAR    1
00 11 83 00
20 00 00 00    Header type 1  NOP
30 00 40 00    Header type 1  WRITE   FDRI   0 palabras
50 00 0B B1    HEADER TYPE 2  WRITE   2993 palabras
00 00 00 00    2993
00 00 00 00    2992
...
00 00 00 00    3
00 00 00 00    2
00 00 00 00    1
30 00 C0 01    Header type 1  WRITE   MASK   1
00 00 10 00
30 03 00 01    Header type 1  WRITE   CTL1   1
00 00 00 00
30 00 80 01    Header type 1  WRITE   CMD    1
00 00 00 03
20 00 00 00    Header type 1  NOP
20 00 00 00    Header type 1  NOP
...
30 00 20 01    Header type 1  WRITE   FAR    1
00 EF 80 00
30 00 00 01    Header type 1  WRITE   CRC    1
00 00 DE FC
30 00 80 01    Header type 1  WRITE   CMD    1
00 00 00 0D
20 00 00 00    Header type 1  NOP
```

Listado 4-4: Volcado de un bitstream como lo presenta readbitstream

Esta herramienta funcionó como base para el desarrollo de un visor de ocupación de recursos CLB y la primera función de relocalización de *bitstreams*. El visor analiza el contenido de los *frames* de configuración para CLB y marca a estos como libres u ocupados según la existencia de contenido de configuración en algún CLB. El desarrollo de esta herramienta permitió encontrar detalles sobre el formato *bitstream* que no están documentados en las guiás de usuario, como la existencia de un *frame dummy* al final de

cada fila de configuración. Este *frame dummy* se utiliza para perder tiempo mientras la lógica de configuración se ajusta para configurar la siguiente fila, este mismo comportamiento se verá en los *bitstream* parciales. La Ilustración 4-21 muestra al visor tras haber leido y analizado un bitstream. Los cuadros de color claro son CLB marcados como ocupados y el gran cuadrado oscuro en el centro representa al procesador PPC.

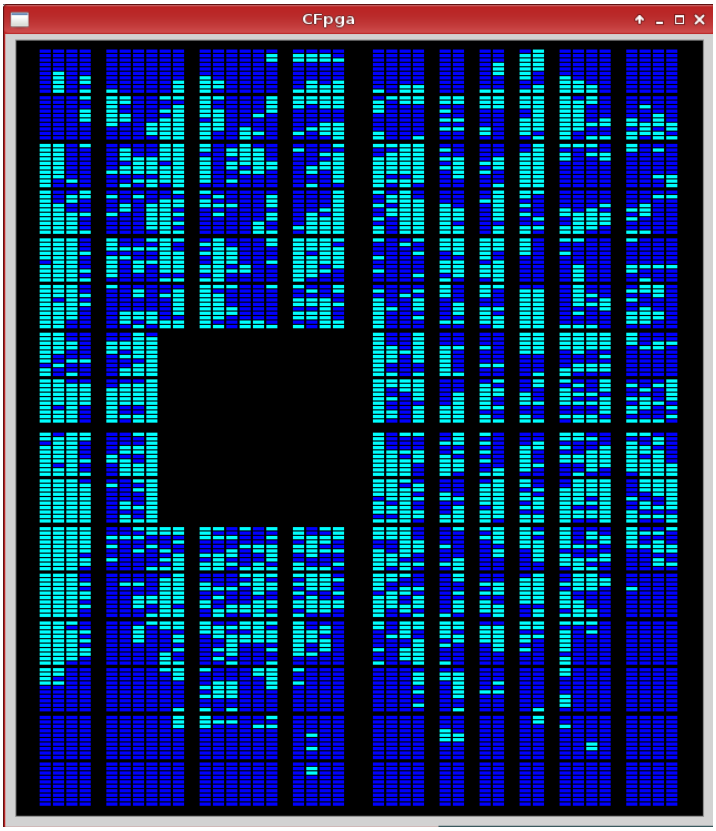

Ilustración 4-21: Visor de ocupación de CLB

La primera función de desplazamiento de *bitstreams* es una aplicación en modo línea de comandos para PC que toma un *bitstream* parcial y dos parámetros de desplazamiento: *desp_x* y *desp_y* y entrega el *bitstream* desplazado *desp_x* columnas en horizontal y *desp_y* filas en vertical. Se ofrece la posibilidad de desplazar el *bitstream* en sentido vertical aunque en nuestra implementación en hardware no se utiliza por el modelo de área utilizado.

La función de desplazamiento es realmente muy sencilla, utiliza la misma estructura que *readbitstream* para detectar la aparición del comando “escribe una palabra al *Frame Address Register* (FAR)”. Este registro, como se describió en el capítulo 3, indica a la

lógica de configuración la dirección del siguiente frame al que se le escribirá información de configuración. Una vez detectado este comando, la siguiente palabra que aparezca en el *bitstream* será la dirección a escribir en FAR. Esta palabra debe ser modificada de acuerdo a los desplazamientos indicados en los parámetros de entrada de acuerdo a la siguiente expresión:

$$FAR_{desplazado} = FAR + (despl\_x \; X \; 0x80) + (despl\_y \; X \; 0x8000)$$

Donde *despl_x* y y *despl_y* son respectivamente el desplazamiento horizontal en columnas y el desplazamiento vertical en filas que se desean aplicar al bitstream.

Una vez se ha obtenido el valor $FAR_{desplazado}$ este reemplaza el valor previo de FAR en el *bitstream.* Con esto se logra generar *bistreams* relocalizados que permiten realizar pruebas de la funcionalidad del modelo de área sin requerir el desarrollo de estas funciones en software embebido.

4.4.2. Administrador de recursos reconfigurables

La propuesta de diseño del administrador de recursos reconfigurables se ha mantenido casi inalterada desde el principio de este trabajo. Su estructura básica sigue muy fielmente el diagrama a bloques presentado en el capítulo 1. Mostramos de nuevo el diagrama de bloques del administrador de recursos en la Ilustración 4-22.

Una vez se han definido nuestros modelos de área y de tarea, la distribución de recursos del área reconfigurable y las características de los *slot*, el sistema operativo que ejecutará nuestro SoPC y el medio de almacenamiento de los *bitstream* parciales, podemos ahora aclarar los detalles de implementación que no se muestran en el diagrama de la Ilustración 4-22. Los puntos mas importantes a tener en cuenta son los siguientes:

- Descripción de las tareas de hardware
- Estructura y significado de las entradas de la lista de recursos libres
- Estructura de las entradas de la lista de tareas de hardware en ejecución
- Estructura de las entradas de la lista de tareas de hardware pendientes
- Comunicación y coordinación entre los hilos solicitantes de tareas y el administrador

- Determinación de disponibilidad de recursos y programación de la tarea de hardware
- Indicación de terminación de ejecución de una tarea de hardware y borrado de la misma

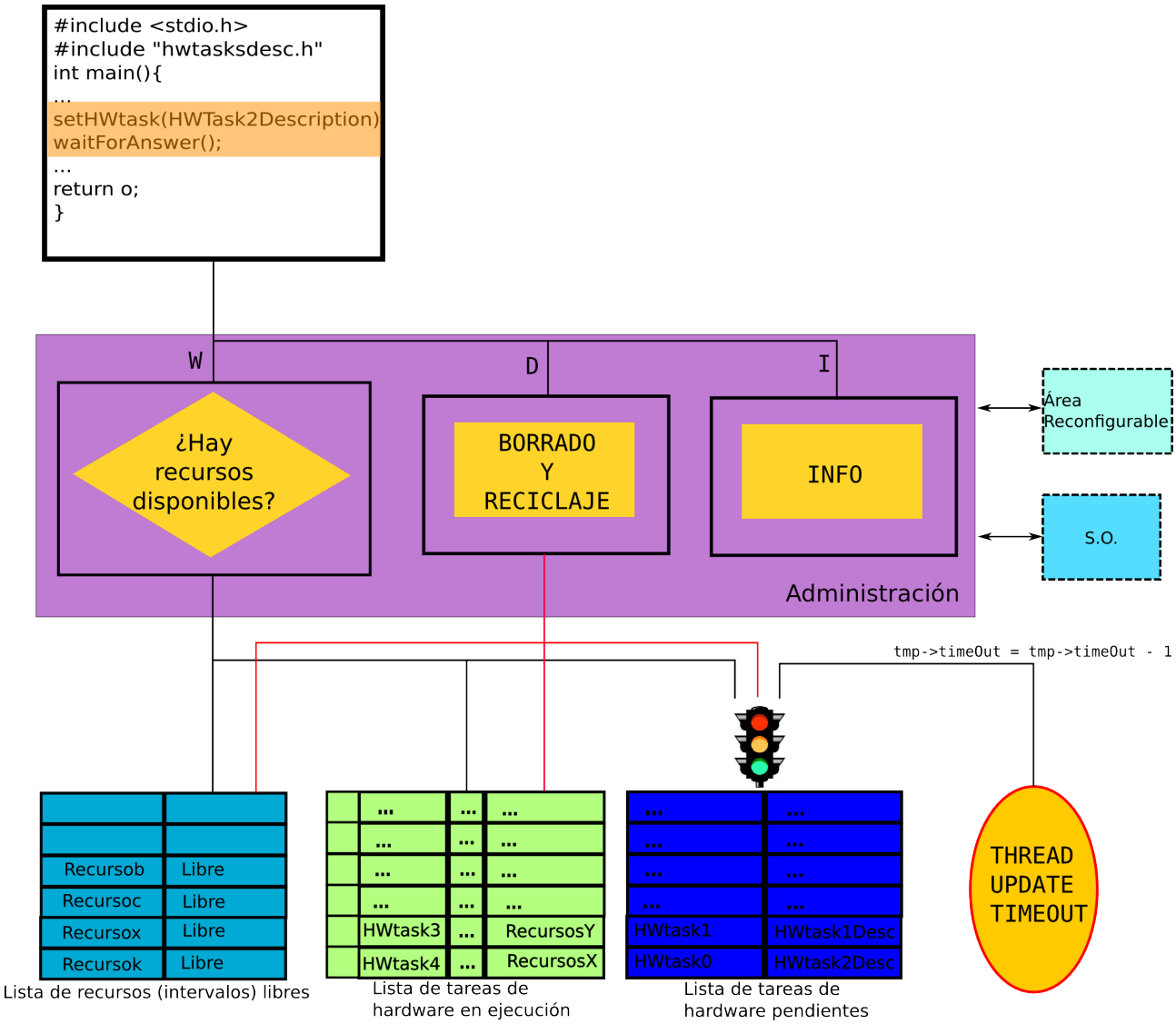

Ilustración 4-22: Diagrama de bloques del administrador de recursos

Descripción de las tareas de hardware

Cada tarea de hardware estará definida por cuatro parámetros:

- *Bitstream* parcial de la tarea de hardware – Una cadena de caracteres con el nombre del *bitstream* de la tarea de hardware.
- Mapa de recursos – Una cadena de caracteres donde cada caracter representa el

tipo de *slot* que la tarea de hardware utiliza en una posición específica. El mapa de recursos indica de izquierda a derecha los tipos de *slot* que forman la tarea igualmente de izquierda a derecha. La Ilustración 4-23 muestra un ejemplo de un mapa de recursos para una tarea de hardware.

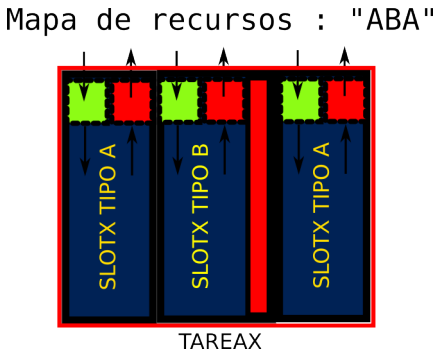

Ilustración 4-23: Ejemplo de mapa de recursos

- Tamaño en *slots* de la tarea de hardware
- *Slot* origen de la tarea – El primer *slot* a la izquierda de la tarea en la implementación inicial de esta.

En esta implementación la definición de todas la tareas hardware se concentra en el archivo de encabezado *hwtasksdesc.h.* El Listado 4-5 muestra estas definiciones para una tarea.

```
#define TASK_BUFFER_SIZE1_BITSTREAM  "buf0.bit" // Tarea buffer tamaño 1
#define TASK_BUFFER_SIZE1_MAP "A"
#define TASK_BUFFER_SIZE1_SIZE       1
#define TASK_BUFFER_SIZE1_ORG_SLOT   0
```

Listado 4-5: Definición de parámetros para una tarea

Estructura y significado de las entradas de la lista de recursos libres

La lista de intervalos libres es simplemente una lista enlazada ordenada. Cada entrada representa un intervalo de *slots* que actualmente se encuentran libres en el área reconfigurable. Es una lista ordenada por posición del *slot.* Este orden facilitará el re-armado de intervalos libres cuando el administrador libere recursos. El Listado 4-6 muestra la definición del nodo representativo del intervalo en la lista de intervalos libres.

```
// Nodo de la lista de intervalos libres
```

```
typedef struct fInterval{
	int size;		// Tamano en slots
	int startSlot;		// Slot inicial
	int endSlot;		// Slot final
	struct fInterval *next;
}freeInterval __attribute__ ((packed));
```

Listado 4-6: Definición del nodo de la lista de intervalos libres

Estructura de las entradas de la lista de tareas de hardware en ejecución

Cada nodo de la lista de tareas de hardware en ejecución identificará univocamente una tarea mediante un identificador y el Identificador de Proceso (PID) del thread al que pertenece. Indicará el nombre, posición y tamaño de la tarea. Esta lista estará ordenada por el identificador taskId. El Listado 4-7 muestra la definición del nodo de la lista de tareas en ejecución

```
// Nodo de la lista de tareas en ejecucion
typedef struct exTask{
	int taskId;		// Identificador de la tarea en ejecucion
	int ownerThreadPID;	// PID del thread dueño de esta tarea de hardware
	char *taskName;		// Nombre de la tarea de hardware
	int startSlot;		// Slot inicial del bitstream cargado
	int endSlot;		// Slot final del bitstream cargado
	int size;		// Tamaño en slots de la tarea hardware
	struct exTask *next;
} execTask __attribute__ ((packed));
```

Listado 4-7: Definición del nodo de la lista de tareas hardware en ejecución

Estructura de las entradas de la lista de tareas de hardware pendientes

Las entradas de la lista de tareas pendientes contienen toda la definición de una tarea cuya programación ha sido solicitada al administrador de recursos. Incluye además de los parámetros que definen a la tarea de hardware la información concerniente al thread dueño y las variables de intercomunicación con él. Además incluye un parámetro *timeout* que indica cuanto puede esperar esta tarea de hardware para ser configurada en el dispositivo antes de ser descartada. El Listado 4-8 muestra la definición del nodo de la lista de tareas pendientes.

```
// Nodo de la lista de tareas pendientes
typedef struct pTask
{
```

```
        key_t workerqkey;      // La llave para el message queue de admin -> worker
        sem_t* workersem;      // El semaforo de espera de respuesta del worker
        int ownerThreadPID;    // El PID del thread que solicita el uso de la tarea de hardware
        short int sizeInSlots;// El tamaño en slots de la tarea hardware
        short int origSlot;           // Slot origen del bitstream original
        char *bitfilename; // El nombre del bitstream para la tarea hardware
        char *resMap;          // El mapa de recursos de la tarea hardware representado en una cadena
        int timeout;           // Cuanto puede esperar la tarea para poder ser asignada
        struct pTask *next;
} pendTask;
```

Listado 4-8: Definición del nodo de la lista de tareas pendientes

Comunicación y coordinación entre los hilos solicitantes de tareas y el administrador

La Ilustración 4-24 muestra la arquitectura de comunicación que se ha propuesto para el administrador y los hilos solicitantes.

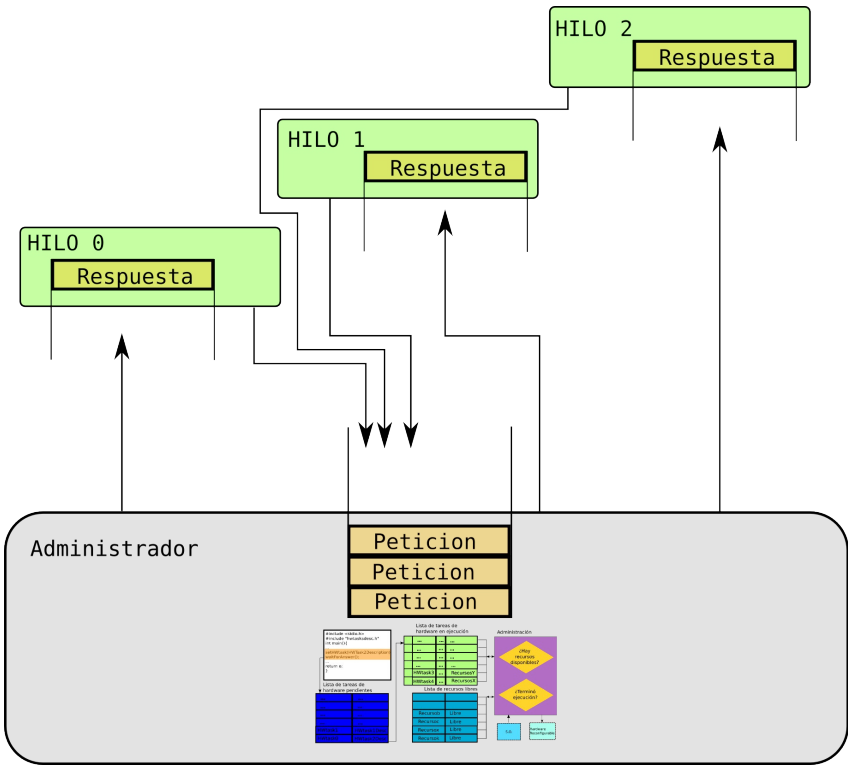

Ilustración 4-24: Arquitectura de comunicación entre el administrador y los hilos solicitantes

El administrador de recursos de hardware necesita algún mecanismo para recibir peticiones por parte de hilos que necesiten utilizar tareas de hardware. También necesita un mecanismo para contestar las peticiones de los hilos indicando si la petición pudo ser

cumplida o no. Los hilos deben tener una forma de esperar por estas respuestas por parte del administrador. Para solventar estas necesidades se utilizan las herramientas que Xilkernel facilita. Se ha optado por utilizar colas de mensajes como mecanismo de comunicación entre procesos y semáforos como mecanismo de sincronización.

El administrador cuenta con una cola de mensajes para recepción de peticiones a la cual todos los hilos solicitantes de tareas de hardware tienen acceso mediante la llave de la cola de mensajes que todos los hilos comparten. De forma complementaria, cada hilo solicitante tendrá una cola de mensajes donde recibirá las respuestas del administrador. Cuando el administrador tome una decisión sobre la tarea de hardware solicitada, contestará al hilo solicitante mediante su cola de mensajes y desbloqueará un semáforo en el que el hilo se bloquea esperando la respuesta. El administrador sabrá a que cola contestar y que semáforo desbloquear pues al recibir la petición por parte del hilo, el mensaje en el que llega tal petición contiene la llave de la cola de mensajes del hilo en cuestión y el semáforo que se deberá desbloquear. El Listado 4-9 muestra la definición de los mensajes que podrán enviarse a las colas de intercomunicación.

```
// Mensaje de thread trabajador a administrador
struct msgToAdmin
{
	key_t workerqkey;	// La llave para el message queue de admin -> worker
	sem_t* workersem;	// El semaforo de espera de respuesta del worker
	int ownerThreadPID;	// El PID del thread que solicita el uso de la tarea de hardware
	int taskId;			// TaskId de la tarea (para borrado)
	char command;		// El comando enviado al administrador (W - Write, D - Delete)
	short int sizeInSlots;// El tamaño en slots de la tarea hardware
	short int origSlot;		// Slot origen del bitstream original
	char *bitfilename; // El nombre del bitstream para la tarea hardware
	char *resMap;		// El mapa de recursos de la tarea hardware representado en una cadena
	int timeout;		// Cuanto puede esperar la tarea para poder ser asignada

};
// Mensaje de thread administrador a trabajador
typedef struct msgToWorker
{
	short int slot;		// El slot inicial al que la tarea ha sido asignada (-1 sino se pudo cumplir la peticion)
	short int hwTaskId;	// Identificador de la tarea hardware
} mesgToWorker;
```

Listado 4-9: Estructura de los mensajes para las colas de intercomuniación

Cuando el administrador no puede contestar inmediatamente, la información de la tarea solicitada se guarda en una lista de tareas pendientes. Un hilo ayudante del administrador actualiza los valores *timeout* de las tareas presentes en la lista de tareas pendientes. Cuando el valor *timeout* llega a cero, la tarea es borrada de la lista de tareas pendientes, y el hilo recibe el aviso de que la tarea fue rechazada.

Determinación de disponibilidad de recursos y programación de la tarea de hardware

El administrador incluirá un mapa de recursos especial para definir la estructura del área reconfigurable que administra. Por cada tarea solicitada al administrador, éste buscará un intervalo libre de tamaño suficiente para albegar la tarea solicitada. Si encuentra un intervalo adecuado, cotejará que los recursos presentes en el intervalo sean compatibles con los que la tarea necesita.

Si se cumplen las condiciones para que una porción de algún intervalo libre albergue la tarea solicitada, el administrador actualiza la lista de intervalos libres, retirando los recursos que ocupará la nueva tarea en ejecución. La tarea se añade a la lista de tareas en ejecución y se programa en el dispositivo FPGA mediante la función:

*int XHwIcap_CF2Icap(XHwIcap *hwicap, char* filename, int despx)*

El *bitstream* representativo de la tarea con nombre *filename* es leido de la tarjeta Compact Flash y es programado en la instancia del puerto *hwicap* que el administrador tenga inicializada. El parámetro *despx* indica cuantas columnas debe desplazarse el *bitstream* según el *slot* al que se haya asignado. Internamente, la función *XHwIcap_CF2Icap* contiene un código similar a la función *movebitstresam*, pero adaptado para solo permitir desplazamientos horizontales y funcionar en el procesador PowerPC.

Indicación de terminación de ejecución de una tarea de hardware y borrado de la misma

Dado que en esta implementación las tareas de hardware no tienen un método para avisar por si mismas que han terminado su ejecución, y que no son tareas de tiempo real, se considera que, como si se tratara de un archivo abierto, el hilo dueño de la tarea de hardware será el encargado de avisar cuando ha terminado de usar una tarea y solicitar al

administrador que sea borrada.

Un hilo solicita el borrado de una tarea enviando un mensaje a la cola de peticiones del administrador. Este mensaje contendrá el comando borrar, el identificador de la tarea y el PID del hilo solicitante. Si la tarea existe en la lista de tareas en ejecución y el hilo que solicita su borrado es su dueño, esta podrá ser borrada.

El borrado de una tarea implica la escritura de *bitstreams* de borrado en todos los *slots* que la tarea ocupaba. Una vez que la tarea ha sido borrada, el administrador recupera los *slots* que se han desocupado y actualiza la lista de intervalos libres. Si con los *slots* recuperados y los *slots* intervalos libres ya presentes en la lista de intervalos libres se pueden formar intervalos mas grandes, estos se generan, conservando siempre intervalos completos y evitando con esto una forma muy básica de fragmentación del área reconfigurable.

El borrado de una tarea provoca el intento por parte del administrador de recuperar tareas de la lista de tareas pendientes. Esta lista está ordenada por el parámetro *timeout,* por lo que el administrador siempre intenta recuperar la tarea a la que le queda menos tiempo para ser recuperada.

4.4.3. Aplicación de prueba

Se desarrolló una aplicación de prueba muy sencilla en la que dos hilos llamados *worker1* y *worker2* solicitan la programación de 5 tareas de hardware al administrador y después solicitan su borrado. Dadas las características de las tareas y el área reconfigurable, no todas las solicitudes son exitosas incluso siendo reservadas en la lista de taras pendientes.

La programación de esta y cualquier otra aplicación de prueba que se desee generar deberá considerar la inclusión de los archivos de encabezado *hwtasksdesc.h* cuya utilidad ya ha sido explicada, y del archivo *globals.h*. Este archivo contiene la definición de todas las variables globales que las aplicaciones necesitan para tener acceso al administrador de recursos reconfigurables.

La aplicación de prueba muestra sus resultados en la salida estándar mientras el administrador muestra por el mismo medio el estado de sus estructuras de datos y sus

diferentes operaciones a lo largo del tiempo. Esta aplicación ejemplifica de la forma mas simplificada posible la posibilidad de implementar trabajo colaborativo entre hilos que utilicen tareas de hardware. También es posible la colaboración entre este tipo de hilos e hilos que trabajen completamente con software.

Como un accesorio que permite la existencia de varias aplicaciones en el sistema, se ha añadido el shell básico para xilkernel que Xilinx provee. Este shell permite listar las diferentes tareas pre-cargadas en el sistema así como ejecutar cada una de ellas.

4.5 RESUMEN

Esta capítulo ha detallado el proceso de desarrollo seguido en este trabajo para la consecución de los objetivos planteados, que al final se resumen en la generación de un SoPC capaz de soportar tareas de hardware relocalizables y de tamaño y número de puertos variable y un método para administrar recursos reconfigurables que se ve reflejado en un administrador que se ejecuta como software embebido en el procesador. Se presentó todo el análisis necesario para establecer los alcances de la arquitectura del dispositivo. Este análisis guió el diseño de los modelos de área y tarea reconfigurables, la generación de un método de implementación específico para soportar estos modelos y la implementación del software administrador y una aplicación de prueba básica para el sistema. Todos estos, temas que se detallaron en este capítulo.

CAPÍTULO 5. PRUEBAS Y RESULTADOS

Este capítulo presenta las pruebas realizadas a la implementación propuesta y los resultados obtenidos de ellas. Las pruebas se dividen esencialmente en pruebas al método de implementación del SoPC y pruebas a la implementación del administrador de recursos reconfigurables. Se presentan pruebas de inspección mediante la herramienta FPGA_EDITOR, pruebas funcionales del sistema mediante la captura de la salida de consola del sistema y mediciones de tiempos mínimos de configuración, borrado y respuesta ante estos dos tipos de solicitud.

5.1. PRUEBAS A LA IMPLEMENTACIÓN DEL SoPC

Las pruebas realizadas a la implementación del SoPC buscan comprobar la compatibilidad de los *slot* reconfigurables generados por el nuevo flujo de diseño y la técnica de implementación propuesta. La Ilustración 5-1 muestra la distribución de los *slots* en el área reconfigurable definida para la implementación específica desarrollada como prueba de concepto. Esta implementación, como se mencionó en el capítulo 4, contiene 6 *slots* heterogeneos ubicados de forma continua dentro del área reconfigurable definida. El *slot* tipo A solo contiene recursos tipo CLB mientras el *slot* tipo B contiene la misma cantidad de recursos CLB que el *slot* tipo A más una columna de memoria BRAM.

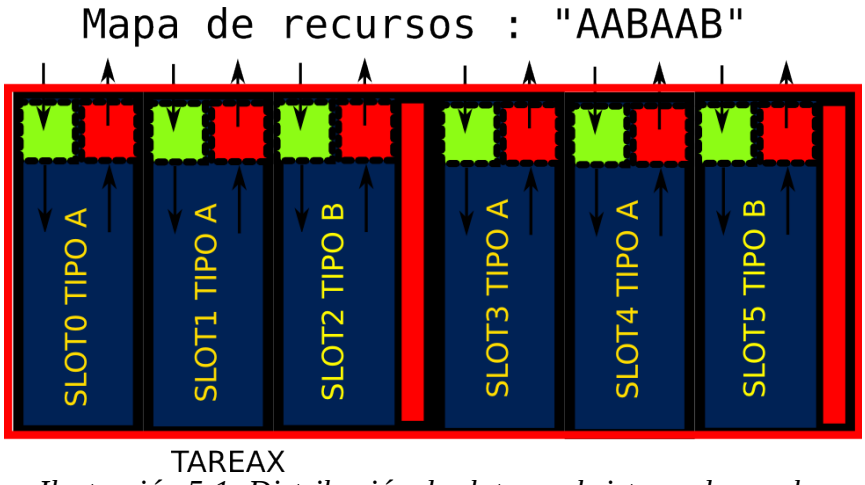

Ilustración 5-1: Distribución de slots en el sistema de prueba

5.1.1. Prueba de inspección en FPGA_EDITOR

La primera prueba realizada es la inspección de la implementación de las *nets* de interfaz con la herramienta FPGA_EDITOR para el SoPC desarrollado. FPGA_EDITOR muestra la forma que adoptan dentro del dispositivo las *nets* enrutadas mediante gráficas representativas de las matrices de enrutamiento del dispositivo y los cables que de ellas emanan. Los *nets* ocupan cables específicos de cada matriz de enrutamiento y todas estas matrices son iguales.

La Ilustración 5-2 muestra el enrutamiento de todas las *nets* de interfaz para los 6 *slots* implementados. Cada dos columnas de *nets* pertenecen a un *slot* específico ubicado en la misma linea vertical que las columnas, en esta Ilustración y las subsecuentes solo se muestran las interfaces, no así el resto del *slot*. Se puede observar, a pesar de la congestión de la figura, que los *nets* generan figuras iguales cada dos columnas. La igualdad de las figuras indica el uso de los mismos recursos relativos de enrutamiento. Es decir, para cada *net* con la misma función en cada *slot,* las matrices que ocupan posiciones iguales, relativas a su *slot* correspondiente, tendrán activados los mismos puntos de conexión programable y por tanto utilizaran los mismos cables.

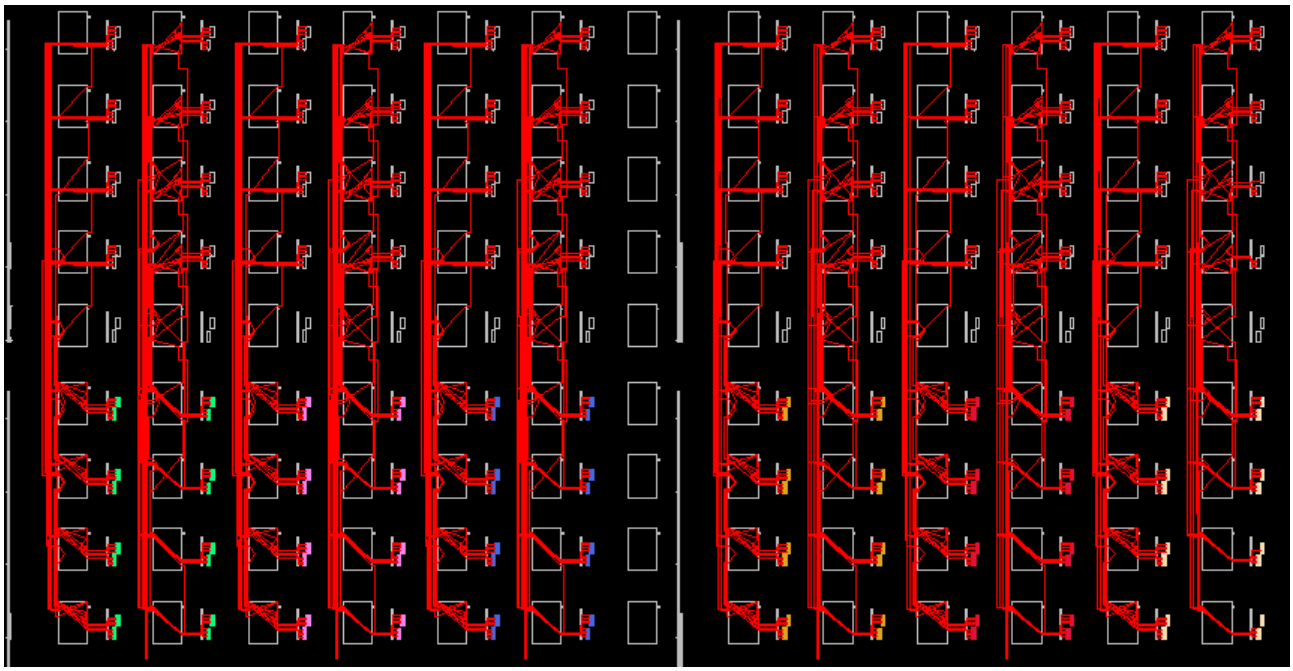

Ilustración 5-2: Enrutamiento de las nets de interfaz de todos los slots

Esta vista puede ser poco clara por la presencia de todos los *nets* de inferfaz enrutados. En las siguientes Ilustraciones se muestran los casos de tres *nets* compatibles. De nuevo obsérvese que la forma de todas las *nets* en cada Ilustración es la misma, lo que indica el uso de recursos de enrutamiento compatibles.

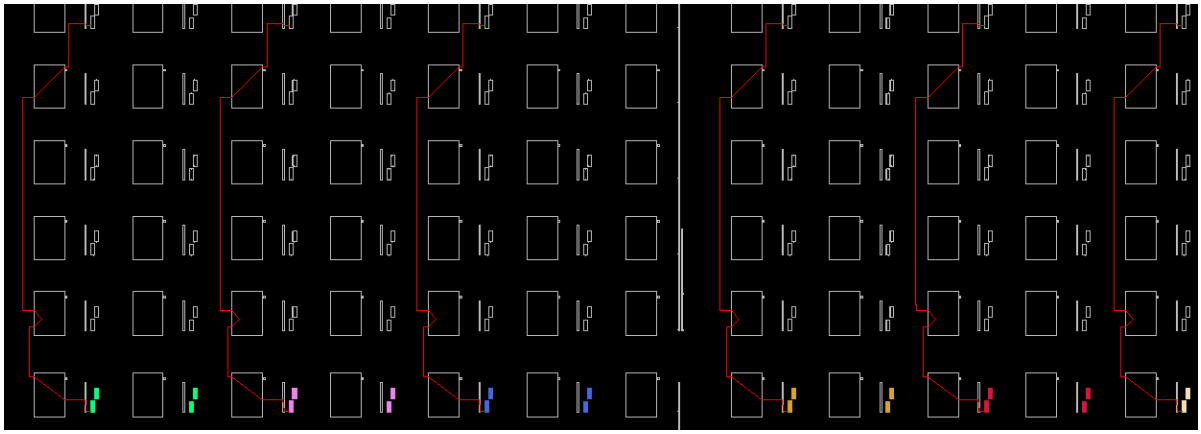

Ilustración 5-3: net compatible para todos los slots. Caso 1

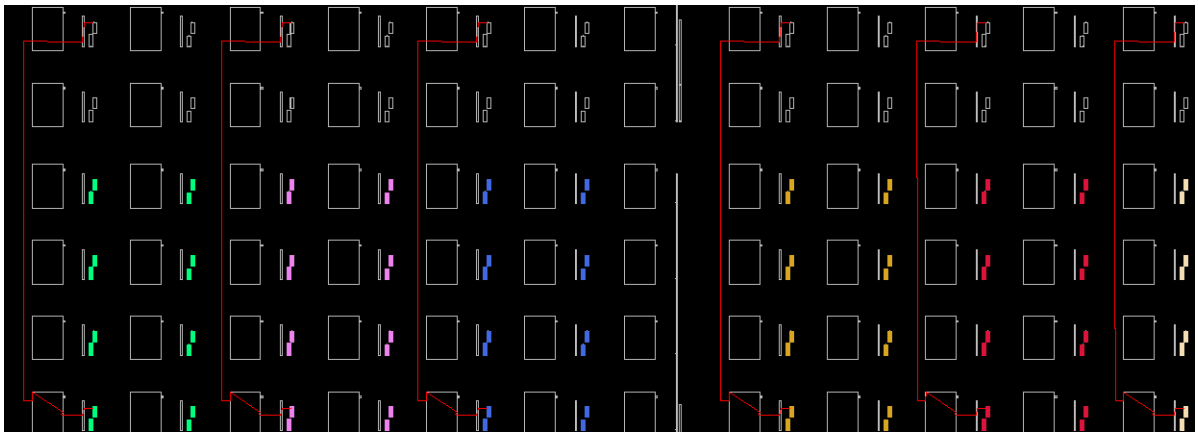

Ilustración 5-4: net compatible para todos los slots. Caso 2

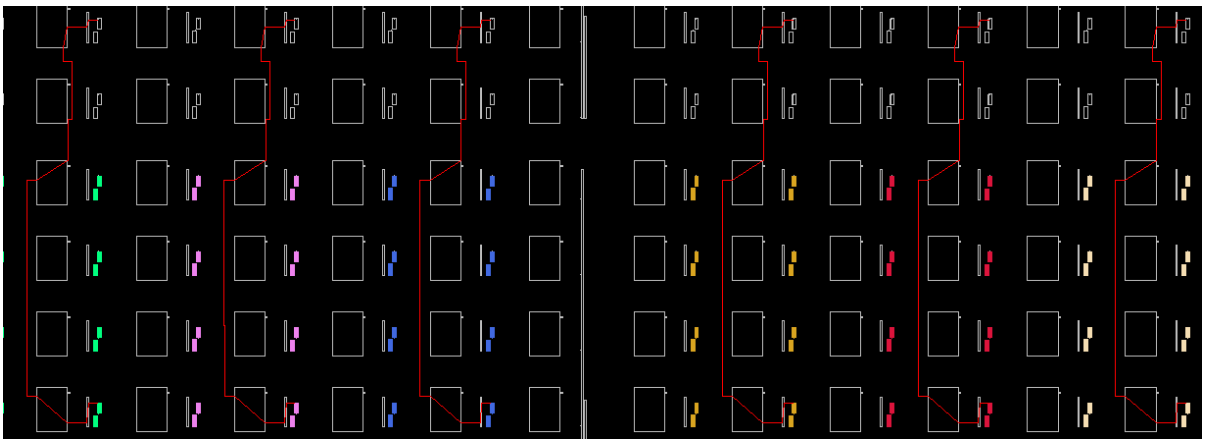

Ilustración 5-5: net compatible para todos los slots. Caso 3

Los tres casos anteriores son solo una muestra a detalle de tres *nets* generadas con el flujo y la técnica de implementación propuestos. El resto de las *nets* de interfaz con la misma función en sus respectivos *slots*, también muestran figuras iguales en FPGA_EDITOR y por tanto son compatibles en el mismo sentido que las *nets* mostradas previamente lo son, se omite mostrarlos por brevedad.

5.1.2. Prueba de funcionamiento *standalone*

La siguiente prueba consistió en el diseño de un pequeño programa *standalone* que probara el funcionamiento de tareas de hardware pre-relocalizadas del primer *slot* a algunos de los *slots* restantes presentes en el diseño. Es decir, a partir de un *bitstream* generado para el primer *slot,* se obtienen bitstreams con la misma configuración pero desplazados a las posiciones del resto de los *slots*. Esta pre-relocalización se realiza fuera del SoPC de manera previa en un entorno de escritorio mediante la herramienta move-*bitstream* y el programa de prueba solo se encarga de escribir los *bitstream* en el dispositivo y probar su funcionamiento con el envío de operandos y la lectura de resultados. El Listado 5-1 muestra la captura de consola para una ejecución de la aplicación *standalone*.

```
-------------------------------------------------------
      SLOT 0:
           - Pulse 1 para INVERSOR
           - Pulse 2 para BUFFER
           - Pulse 3 para BORRADO
           - Pulse 4 para cargar operando y mostrar resultados
      SLOT 1:
           - Pulse 5 para INVERSOR
           - Pulse 6 para BUFFER
           - Pulse 7 para BORRADO
           - Pulse 8 para cargar operando y mostrar resultados
      SLOT 2:
           - Pulse z para INVERSOR
           - Pulse x para BUFFER
           - Pulse c para BORRADO
           - Pulse v para cargar operando y mostrar resultados
      -----> PULSE Q PARA SALIR

      Cargando inversor en SLOT 0
Introduzca el operando: 45
Resultado: -46
```

```
      Cargando buffer en SLOT 0
Introduzca el operando: 45
Resultado: 45
      Cargando inversor en SLOT 1
Introduzca el operando: 89
Resultado: -90
      Cargando buffer en SLOT 1
Introduzca el operando: 98
Resultado: 98
      Cargando inversor en SLOT 2
Introduzca el operando: 65
Resultado: -66
      Cargando buffer en SLOT 2
Introduzca el operando: 36
Resultado: 36
```

Listado 5-1: Ejecución de la aplicación de prueba standalone

Al ejecutar este programa de prueba se corroboró la escritura de los bitstreams correspondientes y su correcto funcionamiento en todos los *slots* a donde fueron pre-relocalizados. Con esto se corroboró fehacientemente la validez del método de implementación al encontrar que los *slots* son en efecto compatibles fisicamente.

5.1.3. Evaluación de recursos para la técnica de implementación

La Tabla 5-1 muestra la evaluación de recursos extra utilizados por esta implementación como consecuencia de la técnica propuesta para lograr compatibilidad de *slots*.

Recurso	***Cantidad***
CLB por slot	40
LUT extra por *slot*	66
SLICE extra por *slot*	16 + 1 LUT
CLB extra por slot	8 + 1 LUT
LUT de pin de partición por *slot*	66
SLICE de pin de partición por *slot*	16 + 1 LUT
CLB de pin de partición por slot	8 + 1 LUT
CLB disponibles para lógica reconfigurable por *slot*	32

Tabla 5-1: Recursos utilizados por slot para esta implementación

Es evidente que la técnica propuesta impone gastos extra substanciales en una implementación de este tipo, no obstante hay que tener en cuenta que no existen muchas opciones prácticas para desarrollar sistemas con las características requeridas. En ese sentido, esta técnica consigue el objetivo con un costo que en futuros desarrollos puede,

sin lugar a dudas ser eliminado completamente. Mas aún, esta técnica puede ser expandida al punto de desarrollar un *router* para reconfiguración parcial basada en *slots,* que enrute el dispositivo completo, herramienta de la que no se conocen precedentes.

5.2 PRUEBAS DE LA IMPLEMENTACIÓN DEL ADMINISTRADOR

5.2.1. Prueba funcional en modo depuración

El Listado 5-2 muestra un historial de ejecución de la aplicación de prueba del sistema en modo depuración, es decir con todos los mensajes posibles para seguir la ejecución del administrador Se muestran listas de intervalos libres, tareas en ejecución, y tareas pendientes. Se observa la interacción entre el administrador y siete hilos trabajadores idénticos. Estos hilos trabajadores solicitan la configuración de una tarea de hardware cuya única función es regresar el mismo valor que se le envía es decir, es un buffer. Esta tarea está implementada en un *slot* tipo A. La tarea es utilizada por el thread trabajador y posteriormente borrada, tras estas dos operaciones, el thread trabajador termina su ejecución . El administrador despliega sus estructuras de datos para mostrar el estado en el que se encuentra tras determinadas operaciones. El sistema inicia con un intervalo libre desde el slot 0 hasta el slot 5. Conforme se le solicitan tareas, este intervalo se va separando en intervalos mas pequeños. Conforme se solicita el borrado de tareas, los recursos se recuperan y reagrupan hasta recuperar todos los recursos al final de la ejecución de la aplicación de prueba. En negritas y con letra mas grande se marcan puntos importantes en el desarrollo de la prueba y que se comentan a continuación.

1. El administrador termina su proceso de inicialización y muestra su lista de intervalos libres.
2. El primer thread solicita la primera tarea de hardware.
3. El primer thread recibe respuesta del administrador que ha podido configurar la tarea y ha entregado la posición donde dicha tarea fue colocada. Antes de enviar respuesta, el administrador muestra su lista de tareas en ejecución y la actualización de su lista de intervalos libres reflejando la carga de la tarea en cuestión.

4. El primer thread utiliza la tarea solicitada.
5. La primera tarea que no pudo ser ubicada por falta de recursos se coloca en la lista de tareas pendientes y el thread timeout entra en acción.
6. La primera tarea es borrada por el administrador a solicitud de su thread dueño y los cambios se reflejan en las listas del administrador.
7. Ante el borrado de una tarea, el administrador logra rescatar una tarea de la lista de tareas pendientes, tal tarea no había cumplido con su cuenta de timeout y por tanto aún está disponible para ser reciclada.
8. Termina el primer thread trabajador.
9. Terminan todos los threads trabajadores y el administrador muestra que todo el espacio reconfigurable ha sido recuperado y que no hay mas tareas pendientes de ejecución.

```
XMK: Starting kernel.
XMK: Initializing Hardware.
XMK: System initialization.
XMK: Process scheduling starts.
SHELL: Iniciando el reloj...
CLOCK: Successfully registered a handler for extra timer interrupts.
CLOCK: Configuring extra timer to generate one interrupt per second..
CLOCK: Enabling the interval timer interrupt...
SHELL: Iniciando el administrador de area reconfigurable...

SHELL:  Lanzando el thread administrador. --
ADMINTH: Iniciando configuracion
ADMINTH: Delay de lectura de ciclos: 137
ADMINTH: Controlador System ACE inicializado
ADMINTH: HWICAP Inicializado
ADMINTH: Message queue para recepcion de peticiones habilitada
ADMINTH: Direcciones de slots inicializadas...
        Slot 0 Address: C66A0000
        Slot 1 Address: C6680000
        Slot 2 Address: C6660000
        Slot 3 Address: C6640000
        Slot 4 Address: C6620000
        Slot 5 Address: C6600000
ADMINTH: Offsets de slots inicializados...
        Slot 0 Offset: 0
        Slot 1 Offset: 2
```

```
        Slot 2 Offset: 4
        Slot 3 Offset: 7
        Slot 4 Offset: 9
        Slot 5 Offset: 11
ADMINTH: Creacion exitosa de memory pool para INTERVALOS LIBRES
ADMINTH: Creacion exitosa de memory pool para TAREAS EN EJECUCION
ADMINTH: Creacion exitosa de memory pool para TAREAS PENDIENTES
ADMINTH: Lista de intervalos libres inicializada...
```

ADMINTH: Lista de intervalos libres... (1)

```
Size     Interval
-----------------------------------------------------
6        [S0 - S5]
-----------------------------------------------------

shell>list

Lista de programas cargados en este sistema
0: TestApp1 : Ejemplo 1 de uso del administrador de reconfiguracion
1: TestApp2 : Ejemplo 2 de uso del administrador de reconfiguracion
2: TestApp3 : Ejemplo 3 de uso del administrador de reconfiguracion
3: recoTime : Tiempos de respuesta del administrador de reconfiguracion
4: mthreadt : Tiempos de respuesta del administrador con varios threads

shell>run 1
-- shell yendo a modo de espera para hacer join con el thread lanzado.

TESTN2: Entrando a testapp1_main..
TESTN2: Inicializando semaforos...
TESTN1: Iniciando los threads!
TESTN1: Lanzando el thread trabajador 0. --
                                      WORKER1: Iniciando thread trabajador con PID: 5 workerKey: 1
TESTN1: Lanzando el thread trabajador 1. --
                                       WORKER1: Iniciando thread trabajador con PID: 6 workerKey: 2
TESTN1: Lanzando el thread trabajador 2. --
                                      WORKER1: Iniciando thread trabajador con PID: 7 workerKey: 3
TESTN1: Lanzando el thread trabajador 3. --
                                      WORKER1: Iniciando thread trabajador con PID: 8 workerKey: 4
TESTN1: Lanzando el thread trabajador 4. --
                                     WORKER1: Iniciando thread trabajador con PID: 9 workerKey: 5
TESTN1: Lanzando el thread trabajador 5. --
                                    WORKER1: Iniciando thread trabajador con PID: 10 workerKey: 6
TESTN1: Lanzando el thread trabajador 6. --
                                   WORKER1: Iniciando thread trabajador con PID: 11 workerKey: 7
TESTN1: Pulse una tecla para iniciar la aplicacion
```

WORKER1: Solicitando hardware task buf0.bit ... (2)

```
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
5        [S1 - S5]
-------------------------------------------------
ADMINTH: Escribiendo bitstream buf0.bit al puerto ICAP
                                                  WORKER1: Solicitando hardware task buf0.bit
ADMINTH: Bitstream buf0.bit listo en 0
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
-------------------------------------------------
1        buf0.bit        5       1       [S0 - S0]
-------------------------------------------------
                                                 WORKER1: Solicitando hardware task buf0.bit
```

WORKER1: buf0.bit Asignado en slot 0 con taskId: 1 ... (3)

```
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
4        [S2 - S5]
-------------------------------------------------
```

WORKER1: Calculo buf0.bit(45) = 45 ... (4)

```
                                                   WORKER1: Solicitando hardware task buf0.bit
ADMINTH: Escribiendo bitstream buf0.bit al puerto ICAP
ADMINTH: Bitstream buf0.bit listo en 1
                                                  WORKER1: Solicitando hardware task buf0.bit
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
-------------------------------------------------
2        buf0.bit        6       1       [S1 - S1]
1        buf0.bit        5       1       [S0 - S0]
-------------------------------------------------
                                                 WORKER1: Solicitando hardware task buf0.bit
                                                 WORKER1: buf0.bit Asignado en slot 1 con taskId: 2
                                                 WORKER1: Calculo buf0.bit(45) = 45
 ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
1        [S2 - S2]
2        [S4 - S5]
-------------------------------------------------
                                                   WORKER1: Solicitando hardware task buf0.bit
ADMINTH: Escribiendo bitstream buf0.bit al puerto ICAP
ADMINTH: Bitstream buf0.bit listo en 3
```

```
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
--------------------------------------------------
3        buf0.bit        8       1       [S3 - S3]
2        buf0.bit        6       1       [S1 - S1]
1        buf0.bit        5       1       [S0 - S0]
--------------------------------------------------
                                          WORKER1: buf0.bit Asignado en slot 3 con taskId: 3
                                          WORKER1: Calculo buf0.bit(45) = 45


ADMINTH: Lista de intervalos libres...
Size     Interval
--------------------------------------------------
1        [S2 - S2]
1        [S5 - S5]
--------------------------------------------------
ADMINTH: Escribiendo bitstream buf0.bit al puerto ICAP
ADMINTH: Bitstream buf0.bit listo en 4
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
--------------------------------------------------
4        buf0.bit        11      1       [S4 - S4]
3        buf0.bit        8       1       [S3 - S3]
2        buf0.bit        6       1       [S1 - S1]
1        buf0.bit        5       1       [S0 - S0]
--------------------------------------------------
                                          WORKER1: buf0.bit Asignado en slot 4 con taskId: 4
                                          WORKER1: Calculo buf0.bit(45) = 45
ADMINTH: timeoutrunning: 0
```

TESTN1: Lanzando el thread timeOut 1. -- ... (5)

```
TIMEOUT: timeout = 29
TIMEOUT: timeout = 28
TIMEOUT: timeout = 27
TIMEOUT: timeout = 26
TIMEOUT: timeout = 25
ADMINTH: Tarea buf0.bit para PID 10 no asignada, espacio insuficiente, anadida a la LISTA DE TAREAS
PENDIENTES
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
--------------------------------------------------
24       buf0.bit        10      1       0       A
--------------------------------------------------
TIMEOUT: timeout = 24
TIMEOUT: timeout = 23
TIMEOUT: timeout = 22
TIMEOUT: timeout = 21
```

```
TIMEOUT: timeout = 20
ADMINTH: timeoutrunning: 1
ADMINTH: Tarea buf0.bit para PID 9 no asignada, espacio insuficiente, anadida a la LISTA DE TAREAS PENDIENTES
TIMEOUT: timeout = 19
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
--------------------------------------------------
19       buf0.bit        10      1       0       A
29       buf0.bit        9       1       0       A
--------------------------------------------------
TIMEOUT: timeout = 29
TIMEOUT: timeout = 18
TIMEOUT: timeout = 28
TIMEOUT: timeout = 17
TIMEOUT: timeout = 27
TIMEOUT: timeout = 16
TIMEOUT: timeout = 26
TIMEOUT: timeout = 15
TIMEOUT: timeout = 25
TIMEOUT: timeout = 14
TIMEOUT: timeout = 24
```

ADMINTH: Tarea taskId 1, buf0.bit, ownerPID 5 Borrada ... (6)

```
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
--------------------------------------------------
4        buf0.bit        11      1       [S4 - S4]
3        buf0.bit        8       1       [S3 - S3]
2        buf0.bit        6       1       [S1 - S1]
--------------------------------------------------
TIMEOUT: timeout = 13
TIMEOUT: timeout = 23
TIMEOUT: timeout = 12
TIMEOUT: timeout = 22
TIMEOUT: timeout = 11
ADMINTH: Recuperando espacio reconfigurable...
ADMINTH: Lista de intervalos libres...
Size     Interval
--------------------------------------------------
1        [S0 - S0]
1        [S2 - S2]
1        [S5 - S5]
--------------------------------------------------
TIMEOUT: timeout = 21
TIMEOUT: timeout = 10
TIMEOUT: timeout = 20
```

```
TIMEOUT: timeout = 9
TIMEOUT: timeout = 19
TIMEOUT: timeout = 8
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
1        [S0 - S0]
1        [S2 - S2]
1        [S5 - S5]
-------------------------------------------------
TIMEOUT: timeout = 18
TIMEOUT: timeout = 7
TIMEOUT: timeout = 17
TIMEOUT: timeout = 6
TIMEOUT: timeout = 16
TIMEOUT: timeout = 5
TIMEOUT: timeout = 15
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
1        [S2 - S2]
1        [S5 - S5]
-------------------------------------------------
```

ADMINTH: RECICLE Escribiendo bitstream buf0.bit al puerto ICAP

ADMINTH: RECICLE Bitstream buf0.bit listo en 0 ... (7)

```
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
-------------------------------------------------
5        buf0.bit        10      1       [S0 - S0]
4        buf0.bit        11      1       [S4 - S4]
3        buf0.bit        8       1       [S3 - S3]
2        buf0.bit        6       1       [S1 - S1]
-------------------------------------------------
                                          WORKER1: buf0.bit Asignado en slot 0 con taskId: 5
                                          WORKER1: Calculo buf0.bit(45) = 45
ADMINTH: pendTaskQyt = 1
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
-------------------------------------------------
5        buf0.bit        10      1       0       A
15       buf0.bit        9       1       0       A
-------------------------------------------------
TIMEOUT: timeout = 14
TIMEOUT: timeout = 13
TIMEOUT: timeout = 12
```

```
TIMEOUT: timeout = 11
TIMEOUT: timeout = 10
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
-------------------------------------------------
9        buf0.bit        9       1       0       A
-------------------------------------------------
TIMEOUT: timeout = 9
TIMEOUT: timeout = 8
TIMEOUT: timeout = 7
TIMEOUT: timeout = 6
TIMEOUT: timeout = 5
TIMEOUT: timeout = 4
TIMEOUT: timeout = 3
TIMEOUT: timeout = 2
                                                        WORKER1: Borrando tarea buf0.bit taskId 1
                                                        WORKER1: Tarea buf0.bit taskId 1 Borrada
                                          WORKER1: ----------> Terminado ... (8)
ADMINTH: timeoutrunning: 1
ADMINTH: Tarea buf0.bit para PID 7 no asignada, espacio insuficiente, anadida a la LISTA DE TAREAS
PENDIENTES
TIMEOUT: timeout = 1
TIMEOUT: timeout = 29
TIMEOUT: timeout = 0
TIMEOUT: BORRANDO ownerId: 9
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
-------------------------------------------------
0        buf0.bit        9       1       0       A
29       buf0.bit        7       1       0       A
-------------------------------------------------
                                    WORKER1: La tarea buf0.bit no pudo ser asignada en hardware
                                    WORKER1: ----------> Terminado
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
-------------------------------------------------
29       buf0.bit        7       1       0       A
-------------------------------------------------
ADMINTH: Tarea taskId 2, buf0.bit, ownerPID 6 Borrada
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
-------------------------------------------------
5        buf0.bit        10      1       [S0 - S0]
4        buf0.bit        11      1       [S4 - S4]
3        buf0.bit        8       1       [S3 - S3]
-------------------------------------------------
```

```
TIMEOUT: timeout = 28
TIMEOUT: timeout = 27
TIMEOUT: timeout = 26
TIMEOUT: timeout = 25
TIMEOUT: timeout = 24
TIMEOUT: timeout = 23
ADMINTH: Recuperando espacio reconfigurable...
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
1        [S1 - S1]
1        [S2 - S2]
1        [S5 - S5]
-------------------------------------------------
TIMEOUT: timeout = 22
TIMEOUT: timeout = 21
TIMEOUT: timeout = 20
TIMEOUT: timeout = 19
TIMEOUT: timeout = 18
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
2        [S1 - S2]
1        [S5 - S5]
-------------------------------------------------
TIMEOUT: timeout = 17
TIMEOUT: timeout = 16
TIMEOUT: timeout = 15
TIMEOUT: timeout = 14
TIMEOUT: timeout = 13
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
1        [S2 - S2]
1        [S5 - S5]
-------------------------------------------------
ADMINTH: RECICLE Escribiendo bitstream buf0.bit al puerto ICAP
ADMINTH: RECICLE Bitstream buf0.bit listo en 1
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
-------------------------------------------------
6        buf0.bit        7       1       [S1 - S1]
5        buf0.bit        10      1       [S0 - S0]
4        buf0.bit        11      1       [S4 - S4]
3        buf0.bit        8       1       [S3 - S3]
-------------------------------------------------
```

```
                                        WORKER1: buf0.bit Asignado en slot 1 con taskId: 6
                                        WORKER1: Calculo buf0.bit(45) = 45

ADMINTH: pendTaskQyt = 0
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
--------------------------------------------------
--------------------------------------------------
                                                          WORKER1: Borrando tarea buf0.bit taskId 2
                                                          WORKER1: Tarea buf0.bit taskId 2 Borrada
                                                          WORKER1: ----------> Terminado
ADMINTH: Tarea taskId 3, buf0.bit, ownerPID 8 Borrada
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
--------------------------------------------------
6        buf0.bit        7       1       [S1 - S1]
5        buf0.bit        10      1       [S0 - S0]
4        buf0.bit        11      1       [S4 - S4]
--------------------------------------------------
ADMINTH: Recuperando espacio reconfigurable...
ADMINTH: Lista de intervalos libres...
Size     Interval
--------------------------------------------------
1        [S2 - S2]
1        [S3 - S3]
1        [S5 - S5]
--------------------------------------------------
ADMINTH: Lista de intervalos libres...
Size     Interval
--------------------------------------------------
2        [S2 - S3]
1        [S5 - S5]
--------------------------------------------------
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
--------------------------------------------------
--------------------------------------------------
                                                          WORKER1: Borrando tarea buf0.bit taskId 3
                                                          WORKER1: Tarea buf0.bit taskId 3 Borrada
                                                           WORKER1: ----------> Terminado
ADMINTH: Tarea taskId 4, buf0.bit, ownerPID 11 Borrada
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
--------------------------------------------------
6        buf0.bit        7       1       [S1 - S1]
5        buf0.bit        10      1       [S0 - S0]
```

```
-------------------------------------------------
ADMINTH: Recuperando espacio reconfigurable...
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
2        [S2 - S3]
1        [S4 - S4]
1        [S5 - S5]
-------------------------------------------------
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
4        [S2 - S5]
-------------------------------------------------
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
-------------------------------------------------
-------------------------------------------------
                                                          WORKER1: Borrando tarea buf0.bit taskId 4
                                                          WORKER1: Tarea buf0.bit taskId 4 Borrada
ADMINTH: Tarea taskId 5, buf0.bit, ownerPID 10 Borrada
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
-------------------------------------------------
6        buf0.bit        7       1       [S1 - S1]
-------------------------------------------------
                                                          WORKER1: ----------> Terminado
ADMINTH: Recuperando espacio reconfigurable...
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
1        [S0 - S0]
4        [S2 - S5]
-------------------------------------------------
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
1        [S0 - S0]
4        [S2 - S5]
-------------------------------------------------
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
-------------------------------------------------
-------------------------------------------------
                                                          WORKER1: Borrando tarea buf0.bit taskId 5
                                                          WORKER1: Tarea buf0.bit taskId 5 Borrada
```

```
ADMINTH: Tarea taskId 6, buf0.bit, ownerPID 7 Borrada
ADMINTH: Lista de tareas en ejecucion...
TaskId   Name    OwnerPID        Size    Interval
-------------------------------------------------
-------------------------------------------------
                                                        WORKER1: ----------> Terminado
ADMINTH: Recuperando espacio reconfigurable...
ADMINTH: Lista de intervalos libres...
Size     Interval
-------------------------------------------------
1        [S0 - S0]
1        [S1 - S1]
4        [S2 - S5]
-------------------------------------------------
ADMINTH: Lista de intervalos libres... (9)
Size     Interval
-------------------------------------------------
6        [S0 - S5]
-------------------------------------------------
ADMINTH: Lista de tareas pendientes...
Tout     Name    OwnPID          Size    OrigSlot        ResMap
-------------------------------------------------
-------------------------------------------------
                                                        WORKER1: Borrando tarea buf0.bit taskId 6
                                                        WORKER1: Tarea buf0.bit taskId 6 Borrada
                                                        WORKER1: ----------> Terminado
```

Listado 5-2: Historial de ejecución de la aplicación de prueba.

Esta prueba demuestra el funcionamiento del administrador y su interacción con hilos de trabajo. Ratifica además las pruebas de implementación del SoPC al demostrar que todos los *slots* son compatibles en cuanto al enrutamiento. Esto se comprueba al observar que todos los cálculos solicitados por los hilos trabajadores regresan un resultado correcto independientemente de la posición que la tarea de hardware correspondiente ocupe en el área reconfigurable.

5.2.2. Tiempos de configuración, borrado y respuesta del sistema

Además de la prueba funcional del sistema con la que se demuestra como maneja adecuadamente el espacio reconfigurable, se presentan mediciones del tiempo de configuración y borrado de tareas de todos los tamaños posibles,es decir 1 a 6 *slots*. Así también se presentan mediciones del tiempo de respuesta del sistema a una solicitud bajo

condiciones ideales. Es decir, la solicitud de una sola tarea y la respuesta exitosa del administrador que programa la tarea solicitada en el dispositivo FPGA y entrega al hilo solicitante la posición donde se encuentra la tarea. Esta medición se realiza igualmente para tareas de tamaño 1 a 6 *slots*. Todas las mediciones fueron realizadas en el sistema implementado con las siguientes características particulares a tener en cuenta: el puerto de reconfiguración ICAP trabaja con un reloj de 62.5MHz mientras el resto del sistema utiliza un reloj de 125MHz y la distribución de *slots* es la mostrada en la Ilustración Ilustración 5-1.

La Ilustración 5-6 muestra los tiempos de configuración y borrado para tareas de tamaño 1 a 6 *slots* cuyo primer *slot* es para todos los casos salvo los marcados T1B el *slot* 0. Estos tiempos de configuración o borrado solo implican la escritura del *bitstream* de la tarea correspondiente en el dispositivo o la escritura de los *bitstream* de borrado necesarios en su caso. En una operación de borrado no se escribe un solo *bitstream* para el borrado de toda la tarea, sino un *bitstream* de borrado particular para cada *slot* que ocupe la tarea a borrar.

Los primeros dos pares de datos para escritura y borrado corresponden a *slots* de tamaño 1. T1A es un *slot* tipo A, mientras que T1B es un *slot* tipo B. La diferencia de tiempo de configuración y borrado entre estos dos tipos de *slot* es sustancial, incluso como puede observarse, toma mas tiempo configurar o borrar un *slot* tipo B que dos *slots* tipo A, como muestran los datos marcados T2 correspondientes a una tarea de tamaño dos *slots* tipo A. De T3 a T5 se puede observar un crecimiento lineal en los tiempos de configuración. Esto es de esperarse considerando que cada nuevo slot añadido en este intervalo es un *slot* tipo A. La diferencia es nuevamente sustancial con la tarea T6 (tamaño 6) al añadir un *slot* tipo B. Para la tarea mas grande el tiempo de configuración es aproximadamente 1 segundo. Este tiempo será relevante en tanto las tareas sean usadas por tiempos dentro del mismo orden de magnitud que los tiempos de configuración y borrado. Así pues, para que tenga sentido utilizar reconfiguración parcial, el tiempo de uso de las tareas debería ser sustancialmente mayor a los tiempos de configuración y borrado.

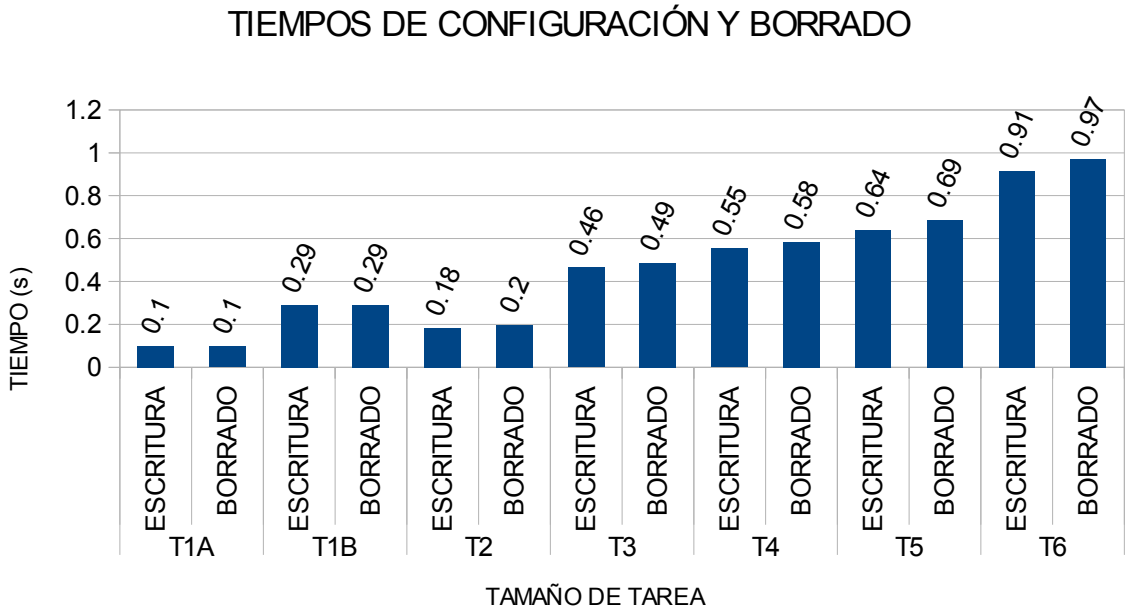

Ilustración 5-6: Tiempos de configuración y borrado

La Ilustración 5-7 muestra los tiempos de respuesta para reconfiguración y borrado. El tiempo de respuesta se considera el transcurrido desde que un hilo hace una solicitud de configuración o borrado hasta que el administrador responde tras haber realizado la configuración o borrado de la tarea. Es decir implica la escritura o borrado en el dispositivo y el tiempo que el administrador tarda en procesar la solicitud.

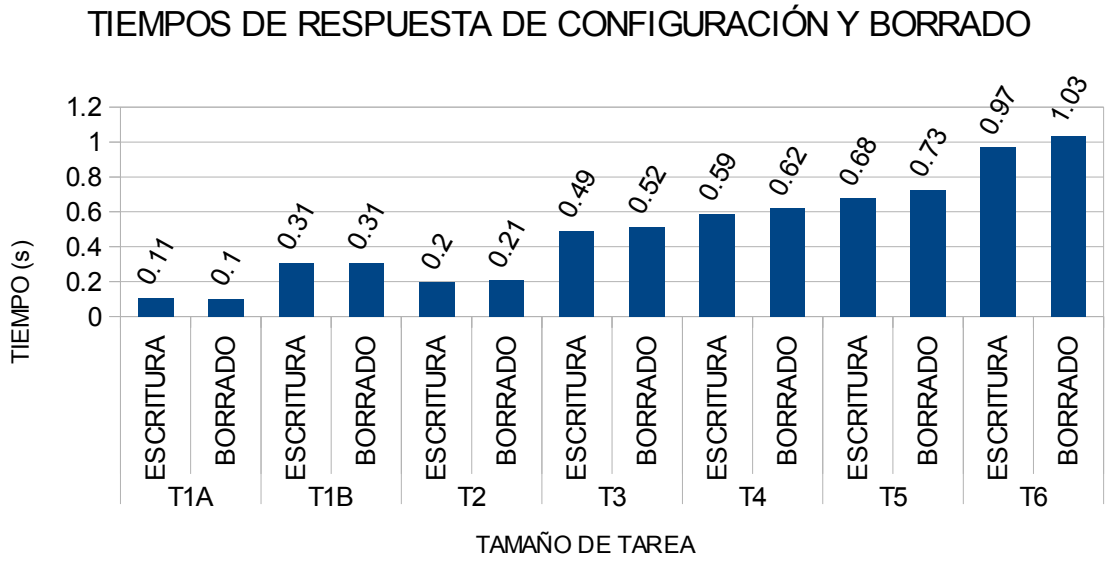

Ilustración 5-7: Tiempos de respuesta de configuración y borrado

Se puede observar un comportamiento muy similar al de la Ilustración 5-6. Los tiempos de respuesta son bastante cercanos a los tiempos de configuración. Esto indica que el tiempo de respuesta del sistema antes solicitudes de configuración y borrado, en tanto estas puedan ser satisfechas inmediatamente, está dominado en su mayor parte por el tiempo de configuración o borrado de las tareas para cada solicitud que se realice y el tiempo que tarda la administración como tal es comparativamente menor. Esto se observa en la Ilustración 5-8, la cual muestra la diferencia entre tiempo de respuesta y de configuración o borrado. Como puede observarse, esta diferencia está en el orden de centésimas de segundo mientras los tiempos de respuesta y configuración o borrado se encuentran en el orden de décimas de segundo. Es decir, el tiempo de respuesta del sistema, como se mencionó, está dominado por los tiempos de configuración y borrado.

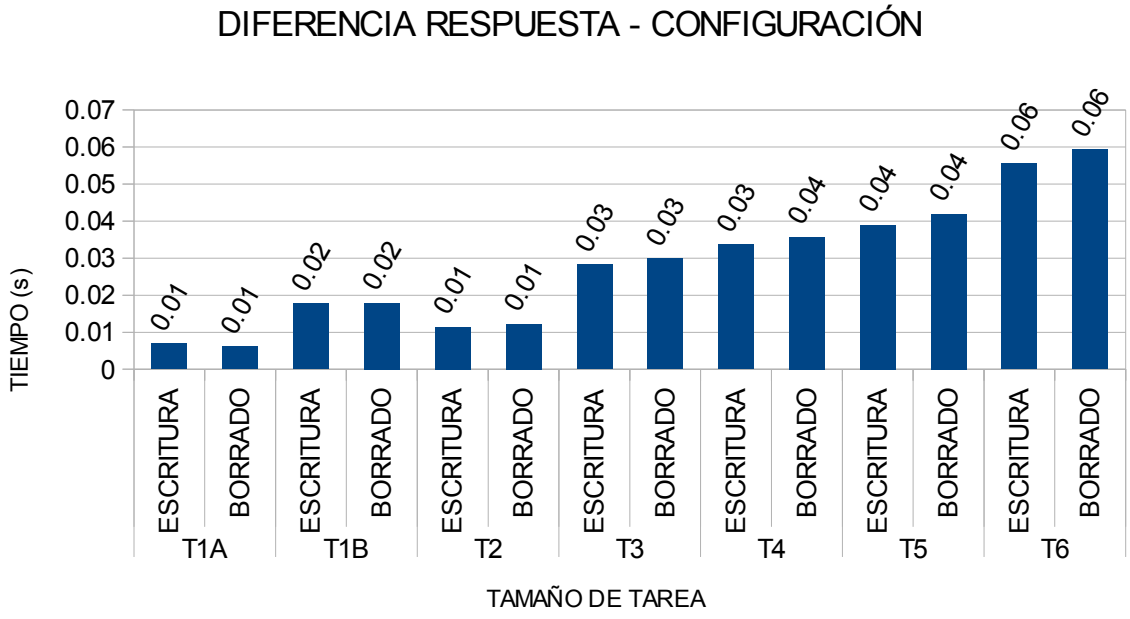

Ilustración 5-8: Diferencia entre Respuesta y configuración o borrado

5.3. RESUMEN

En este capítulo se ha demostrado mediante las pruebas funcionales realizadas al sistema desarrollado, tanto a su parte puramente hardware como a su parte software, la asignación de tareas de hardware desde un hilo de ejecución, la co-ejecución software-hardware y la compatibilidad de los slots. Las pruebas resultaron satisfactorias al demostrar la validez de la técnica de implementación desarrollada y la funcionalidad del método de administración de recursos. También se presentó un estimado de los recursos extra requeridos por la técnica de implementación desarrollada. Si bien lo ideal sería que la técnica propuesta no gastara recursos extra, esta ha sido sólo una primera aproximación que podrá ser depurada en trabajos futuros. Finalmente se presentaron mediciones de los tiempos de configuración y borrado y respuesta del sistema bajo condiciones controladas para obtener valores mínimos esperados para tareas de tamaño variable. Se observó que el tiempo de respuesta está dominado por los tiempos de configuración o borrado.

CAPÍTULO 6. CONCLUSIONES Y TRABAJO FUTURO

6.1. CONCLUSIONES

La primera fase de desarrollo de esta tesis consistió en el estudio de la estructura de los archivos de configuración *bitstream* para dispositivo FPGA Virtex 5. Como resultado de este estudio se ha generado un conjunto de herramientas para la interpretación de estos archivos que permiten interpretar al contenido de la información de configuración del dispositivo al nivel de comandos, *frames* y la ocupación de los recursos básicos. Se obtuvo también una función de relocalización de *bitstreams* capaz de reubicar la información de configuración del diseño mediante desplazamientos en dos dimensiones. Todas estas herramientas, listadas en la Tabla 6-1, resultaron básicas en el desarrollo de este trabajo y podrán servir de base para investigaciones posteriores y están disponibles para su descarga en **http://www.microse.cic.ipn.mx**

Herramienta	*Función*
readbitstream	Volcado en modo texto del contenido de configuración de un bitstream.
movebitstream	Desplaza un bitstream una cantidad *x, y* de columnas y filas
CFpga	Visor gráfico de ocupación de CLB para el FPGA utilizado en este trabajo.

Tabla 6-1: Herramientas de análisis desarrolladas

El estudio de la estructura y posibilidades de los archivos de configuración permitió el diseño de un elemento lógico de reconfiguración al que se llamo SLOT y la delimitación de un modelo de área y de tarea de hardware relocalizable realistas, con lo cual se introdujo el concepto de S*lot Reconfigurable*, mismos que pudieron ser implementados.

Para lograr esta implementación se desarrolló un método consistente en dos partes: primero, la definición de una interfaz intermedia entre la lógica estática y los pines de partición. La localización en el dispositivo de esta interfaz es entonces conocida. Segundo, la generación de restricciones de enrutamiento directo para las rutas entre la interfaz intermedia y las terminales de partición, mediante el uso de herramientas de enrutamiento diseñadas a medida para este tipo de aplicaciones.

Este método introduce un gasto extra de recursos para implementar la interfaz intermedia.

El gasto extra consiste en una unidad LUT por cada señal de interfaz existente. Para el SoPC desarrollado, este gasto es de 32 LUT de interfaz de salida + 33 LUT de interfaz de entrada por cada *slot* reconfigurable. Es decir, para el sistema completo se introducen 390 LUT extra en el diseño. No obstante el gasto extra de recursos, este método es fácil de implementar y trabajos posteriores pueden tomarlo de punto de arranque y eliminar las LUT extra.

Tomando como base los modelos de área y tarea reconfigurable desarrollados e implementados en el SoPC desarrollado se creó un administrador de recursos reconfigurables sobre el sistema operativo Xilkernel. Este administrador puede recibir peticiones de tantos hilos se requiera y contestar a tales peticiones, reciclar tareas que en un principio no pudieron ser programadas y lidiar con la heterogeneidad del dispositivo. Ofrece un método para administrar recursos basado en intervalos de *slots* libres y compatibilidad de recursos y es a nuestro juicio una opción básica pero valedera para la administración de recursos en un sistema como el propuesto de propósito general en el que los hilos propietarios gobiernen el tiempo de vida de las tareas de hardware que les pertenecen.

La aplicación de prueba que se ha desarrollado es realmente muy sencilla pues se buscaba simplemente demostrar el funcionamiento de las ideas planteadas en este trabajo en un ambiente de propósito general. Consideramos que este enfoque de generalidad solo requerirá cambios cuando la aplicación que se desee resolver esté bien definida y el tiempo de vida de las tareas dependa de otros factores como el tiempo de ejecución propio de la tarea o la existencia de parámetros de planificación como un tiempo de inicio forzoso para las tareas como en un sistema de tiempo real.

La evaluación de los tiempos mínimos de reconfiguración y borrado y respuesta del sistema a peticiones en condiciones ideales mostró que el tiempo de respuesta está dominado por los tiempos de escritura de los *bitstreams* necesarios y que para esta implementación particular el tiempo escritura de *bitstreams* es del orden de décimas de segundo y el trabajo del administrador del orden de centésimas.

6.2. Trabajo futuro

Salvo desconocer algún proyecto en particular, el presente proyecto puede ser considerado pionero en los temas que trata al nivel de nuestro Centro de Investigación. Es por ello que la mayor parte de este desarrollo es perfectible y considerando la novedad del trabajo, se puede pensar en que se abren caminos de desarrollo de investigación en varios frentes.

Algunas propuestas para trabajos futuros son las siguientes:

- Desarrollo de una herramienta CAD de implementación especializada en el flujo de diseño de reconfiguración parcial. Específicamente el desarrollo de un router capaz de enrutar un diseño completo con particiones reconfigurables que puedan ser compatibles.
- Implementación de tareas de hardware interrumpibles y reiniciables. Esto generaría ya no solo tareas sino procesos tipo software, capaces de guardar y restaurar su contexto. En este caso, la información de estado de las tareas de hardware.
- Desarrollo de una biblioteca de tareas de hardware de aplicación específica, por ejemplo para procesamiento digital de señales, y utilización de la misma para desarrollar aplicaciones reales.
- Proponer el uso de esta tecnología para el diseño de sistemas tolerantes a fallos.
- Perfeccionar y añadir herramientas gráficas al nuevo flujo de implementación desarrollado en este trabajo.
- Migración del desarrollo a un sistema operativo con mayores prestaciones.

Estos y otros temas mas pueden ser desarrollados tomando como base el presente trabajo.

Referencias

[1] DeHon A. & Wawrzynek J. (1999) Reconfigurable Computing: What, Why, and Implications for Design Automation. Design Automation Conference, Proceedings. 36th.

[2] Compton K. & Hauck S. An Introduction to Reconfigurable Computing. Invited Paper, IEEE Computer

[3] Compton K. & Hauck S. Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, 34 (2). 40.

[4] Todman T.J., Constantinides G.A., Wilton S.J.E., Mencer O., Luk W. & Cheung P.Y.K. (2005), Reconfigurable Computing architectures and design methods. IEE Proceedings - Computers and Digital Techniques 193-207.

[5] Sabeghi M. & Bertels K. (2008) Current Trends in Resource Management of Reconfigurable Systems 19th. Annual Workshop on Circuits, Systems and Signal Processing (ProRISC2008).

[6] Tanenbaum A. Sistemas Operativos Modernos (3a Edición). México: PEARSON EDUCACIÓN.

[7] Brebner G. (1996). A Virtual Hardware Operating System for the Xilinx XC6200. 6Th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers, Springer-Verlag.

[8] Burns, J., Donlin, A., Hogg, J., Singh, S. & Wit, M.d. (1997). A Dynamic Reconfiguration Run-Time System IEEE Symposium on FPGAs for Custom Computing Machines.

[9] Diessel, O., ElGindy, H., Middendorf, M., Schmeck, H. & Schmidt, B. (2000) Dynamic scheduling of tasks on partially reconfigurable FPGAs. Computers and Digital Techniques, IEE Proceedings, 147 (3). 181-188.

[10] Compton, K., Li, Z., Cooley, J., Knol, S. & Hauck, S. (2000) Configuration relocation and defragmentation for run-time reconfigurable computing. Transactions on Very Large Scale Integration (VLSI) Systems, IEEE, 10 (3). 209-220.

[11] Gericota, M.G., Alves, G.R., Silva, M.L. & Ferreira, J.M. (2002) On-line

Defragmentation for Run-Time Partially Reconfigurable FPGAs. Proceedings of the 12th International Conference on FieldProgrammable Logic and Applications. 302-311.

[12] Walder, H. & Platzner, M. (2003) Online scheduling for block-partitioned reconfigurable devices. Design, Automation and Test in Europe Conference and Exhibition. 290-295.

[13] G. B. Wigley, D. A. Kearney, & D. Warren. (2002). Introducing ReConfigME: An operating system for reconfigurable computing. Proceedings of the 12th International Conference on Field Programmable Logic and Application (FPL'02). Springer.

[14] Kwok-Hay So H. (2007). BORPH: An Operating System for FPGA-Based Reconfigurable Computers, Theses submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, California, EUA.

[15] Walder, H. & Platzner, M. (2003). Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations. Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA). 284–287.

[16] J.S. Jean, K. Tomko, V. Yavagal, J. Shah, and R. Cook, (1999). Dynamic Reconfiguration to Support Concurrent Applications, IEEE Trans. Computers, vol. 48, no. 6, pp. 591-602.

[17] P. Merino, J.C. Lopez, and M. Jacome (1998). A Hardware Operating System for Dynamic Reconfiguration of FPGAs, Proc. International Workshop Field Programmable Logic and Applications (FPL), pp. 431-435.

[18] H. Simmler, L. Levinson, and R. Manner, (2000). Multitasking on FPGA Coprocessors, Proc. International Conference. Field Programmable Logic and Applications (FPL), pp. 121-130.

[19] N. Shirazi, W. Luk, and P. Cheung, (1998). Run-Time Management of Dynamically Reconfigurable Designs, Proc. International Workshop Field-Programmable Logic and Applications (FPL), pp. 59-68.

[20] Steiger, C.; Walder, H.; Platzner, M. ,(2004). Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks, Computers, IEEE Transactions, vol.53, no.11, pp. 1393- 1407.

[21] K. Bazargan, R. Kastner, and M. Sarrafzadeh, (2000). Fast Template Placement for Reconfigurable Computing Systems, IEEE Design and Test of Computers, vol. 17, no. 1, pp. 68-83.

[22] Chun-Hsian Huang; Kai-Jung Shih; Chao-Sheng Lin; Shih-Shiue Chang; Pao-Ann Hsiung, (2007). Dynamically Swappable Hardware Design in Partially Reconfigurable Systems, Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on , vol., no., pp. 2742-2745.

[23] Alonso Adrian. Alligator OS: An embedded Operating System. Tesis de Maestría. IPN-CIC, México, 2010.

[24] Xilinx Virtex 5 Configuration User Guide

[25] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs in Proceedings of the 21st International Workshop on Field-Programmable Logic and Applications, Sept. 5-7, 2011.

[26] J. Suris, C. Patterson and P. Athanas, An Efficient Runtime Router for Connecting Modules in FPGAS, Proc. of the International Conference on Field Programmable Logic and Applications, 125-130, 2008.

[27] A. Sohanghperwala, OpenPR: An Open-Source Partial Reconfiguration Tool-Kit for Xilinx FPGAs, Masters Thesis, Virginia Tech, Dept. of ECE, December 2010

[28] C. Schuck, M. Kuhnle, M. Hübner, and J. Becker, "A framework for dynamic 2D placement on FPGAs," in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS '08), pp. 1-7, Miami, Fla, USA, 2008.

[27] Xilinx UG643: OS and Libraries Document Collection.

[28] Xilinx UG757: Using EDK to Run Xilkernel on a PowerPC 440 Processor.

[29] Xilinx UG702: Partial Reconfiguration User Guide.

[30] Xilinx UG744: PlanAhead Software Tutorial. Partial Reconfiguration of Processor Peripheral.

[31] D. Koch, C. Haubelt and J. Teich. Efficient Reconfigurable On-Chip Buses for FPGAs. Proceedings 16th Annual IEEE Symposium on Field-Programmable Custom

Computing Machines (FCCM 2008), pp. 287-290, Palo Alto, California, April 14-15, 2008.