UNIVERSIDAD NACIONAL DE SAN LUIS
FACULTAD DE CIENCIAS FÍSICO-MATEMÁTICAS Y NATURALES

MAESTRÍA EN INGENIERÍA DE SOFTWARE

TESIS DE MAESTRIA

“Nueva Metodología De Diseño De Sistemas Reconfigurables Basada En Sistemas Orientados A Objetos.”

Autor: Julio Daniel Dondo Gazzano
Director: Daniel Riesco
Co-Director: Germán Montejano

Noviembre 2007

A todos y cada uno de
mis seres queridos.

Indice General

Capítulo 1: Introducción

Capitulo 2: Estado del arte

Capitulo 3: Objetivos del Trabajo de Investigación

capitulo 1

Introducción

Introducción:

A través de los años y a medida que el avance tecnológico en el diseño de sistemas integrados (esto es sistemas que poseen componentes hardware y software en un mismo chip, o sistemas embebidos) ha permitido disponer de una mayor cantidad de recursos hardware en un mismo chip o por unidad de área, la complejidad en el proceso de diseño de estos sistemas se ha visto incrementada considerablemente.

Para poder hacer frente a tal incremento de complejidad y poder hacer uso de los recursos tecnológicamente disponibles han entrado en juego nuevas metodologías de diseño que han hecho posible un mejor aprovechamiento de estos recursos y un incremento en la productividad de estos sistemas integrados.

Diseño orientado al reuso de componentes hardware y/o software, y el diseño basado en plataforma, en los cuales se conjuga la reconfiguración y reusabilidad son ejemplos de tales corrientes.

Con la incorporación de Dispositivos Lógicos Programables como las FPGA (Field Programmable Gate Array) esta característica de reconfigurabilidad se vio notablemente incrementada. La posibilidad de reconfiguración parcial que presentan algunos de estos dispositivos permite la modificación de partes del diseño en tiempo de ejecución, o sea mientras otra parte del mismo continua funcionando.

Unir estas dos ventajas, reusabilidad de componentes previamente desarrollados y la reconfigurabilidad parcial de diseño, significa un avance muy importante en el diseño de sistemas integrados.

Este trabajo plantea la posibilidad de unir estas dos características utilizando un sistema de comunicación para la integración de componentes previamente desarrollados, en sistemas que permiten la reconfiguración parcial en tiempo de ejecución. Para ello adaptará funcionalidades de un sistema de comunicación basado en el paradigma de objetos distribuidos para incluir la reconfigurabilidad

parcial como servicio del sistema.

1.1 Diseño basado en Plataforma

El diseño basado en plataforma consiste en mantener una plataforma común que pueda soportar numerosas aplicaciones sobre la cual se desarrollarán los sistemas, permitiendo una movilidad y un reuso del diseño de manera de poder abaratar los costos del mismo, y disminuir el tiempo de puesta en escena del producto.

El diseño basado en plataforma plantea dos ortogonalizaciones: una entre la función (que representa lo que el sistema va a hacer) y la arquitectura (como lo va a hacer), y la otra entre comunicación y computación.[1] El mapeo de una función en una arquitectura implica la implementación de la funcionalidad del sistema en un circuito integrado, donde los costos de diseño, de verificación y fabricación inciden directamente en la calidad y costo del producto final. Este costo es posible reducirlo si se desarrolla una arquitectura común para una serie de aplicaciones diferentes, pudiendo aplicar esta arquitectura en otros diseños.

Existen varias propuestas al respecto, algunas correspondientes a arquitecturas o familias de arquitecturas fijas, sobre las cuales se implementan aplicaciones de software que pueden ser reutilizados para aplicaciones futuras. Una familia de arquitecturas que permite el reuso sustancial de software es lo que se conoce como Plataforma de Hardware[1].

Otras propuestas donde conviven plataformas de hardware que poseen partes fijas con otras no fijas (reconfigurables), sobre las cuales se implementan distintas funcionalidades del sistema, han entrado en escena con la aparición de las FPGAs (Field Programmable Gate Array), los cuales han alcanzado un nivel de desarrollo tal que permite que circuitos complejos, como core de procesadores PowerPC[40] o ARM[41] sean implementados en ellos. Estos dispositivos se pueden reconfigurar total o parcialmente en milisegundos, lo que aumenta de manera notable la flexibilidad en el diseño de sistemas integrados. Este tipo de

plataformas es conocido como Plataforma de Hardware Reconfigurable.

1.2 Plataformas de Hardware Reconfigurables

La introducción de Hardware reconfigurable y mas específicamente, Hardware reconfigurable Dinamico nos presenta nuevos aspectos, especialmente en el flujo de diseño a nivel de sistemas, en donde, además del problema tradicional de gestión de particiones Hardware/Software y de migración de tareas, se agrega ahora la complejidad de la gestión de componentes hardware reconfigurables dinámicamente en el proceso de diseño.

En el diseño sobre Plataformas de Hardware Reconfigurables, la reconfiguración parcial permite una flexibilidad de diseño similar a la encontrada en los sistemas software donde es posible la construcción, utilización y posterior destrucción de objetos software en tiempo de ejecución, según la necesidad del diseño. Análogamente, la característica de reconfigurabilidad parcial que presentan algunas FPGA actualmente hace posible la creación, implementación y utilización de objetos hardware y su posterior destrucción, sin que el sistema original que los contiene se detenga.

Bajo el mismo modelo de objetos software, es posible el uso de estos objetos hardware en otros diseños. Este uso múltiple contribuye de manera significativa a la disminución del tiempo de diseño, y por ende del costo de sistemas on chip.

Para hacer posible la incorporación de componentes hardware desarrollados por terceros sobre un diseño o sistema base, muchas veces es necesario adaptar el componente, puesto que no ha sido creado para ese diseño específicamente sino que se lo reutiliza adaptándolo para el diseño en cuestión. Esta adaptación, generalmente realizada por el diseñador, no es tarea fácil puesto que a veces no es posible contar con el código fuente para su modificación. Es imperioso, por lo tanto, encontrar una manera de adaptar los componentes desarrollados por terceros a nuestro sistema sin tener que modificar su código

fuente para lograr su utilización.

1.3 Sistemas de Comunicación

Una manera de poder utilizar componentes desarrollados por terceros consiste en usar un modelo de comunicación entre componentes basado en el paradigma de sistemas de objetos distribuidos. El modelo que se utilizará en el presente trabajo está basado en el protocolo de sistema de comunicación para sistemas distribuidos ICE (Internet Communication Engine).

En este protocolo, el esquema de comunicación se divide en dos partes, clientes y servidores. Los clientes solicitan servicios que los servidores entregan, pero con la diferencia de que, en este modelo, el cliente no se comunica directamente con el servidor sino que lo hace mediante un *proxy*. Un *proxy* es un elemento que representa al servidor, con su misma fachada, de manera que lo que el cliente ve es el servidor en cuestión. A su vez el servidor ve por su lado una representación del cliente, llamada *esqueleto*. Por otra parte, como un servidor puede transformarse en cliente, dicho servidor/cliente deberá tener además un *proxy* del servidor al cual solicita servicios.

La forma de llevar a cabo efectivamente el servicio solicitado por el cliente se explica en el capítulo 3.

1.4 La Gestión de la Reconfiguración

Desde el punto de vista físico, la gestión de la reconfiguración se refiere a la manera en que la estructura interna de conexiones y de lógica de la FPGa es detenida y reprogramada. En las FPGAs Virtex esto se realiza normalmente modificando en forma parcial el bitstream de configuración que fuera cargado inicialmente para la configuración total de la misma. Esta modificación, que se obtiene cargando bitstreams parciales, puede ser realizada externamente a través de JTAG o de Select Map Ports o bien internamente a través de un componente especial diseñado para este fin denominado ICAP (Internal Configuration Access

Port). En este último caso el controlador interno para la reconfiguración puede ser tan simple como una FSM (Finite State Machine) o un procesador embebido.

El procedimiento descrito simplemente habilita el uso de tecnología reconfigurable. Sin embargo la gestión de la reconfiguración involucra otras tareas de mayor nivel que simplemente el manejo de bitstreams. Hay al menos tres aspectos que deben ser considerados cuando se diseñan aplicaciones reconfigurables:

* *Consistencia de estado y estructural para poder garantizar una correcta implementación del sistema:* La consistencia de estado significa elegir el lugar y el momento correcto para que la reconfiguración tenga lugar. La consistencia estructural se refiere a la preservación del resto del diseño durante la reconfiguración.

* *Persistencia del Estado:* Se requiere para mantener el estado de los objetos reconfigurables, para que puedan continuar con la ejecución de sus métodos una vez reconfigurados.

* *Interrupción y continuación de la ejecución controladas*: Necesaria para que la ejecución pueda ser detenida únicamente cuando la consistencia de estado pueda ser garantizada, y que además permita continuar con la ejecución o reiniciarse posteriormente.

Por lo tanto, es necesaria una nueva metodología de diseño que incluya lo expresado anteriormente, la cual facilitará las tareas de reconfiguración teniendo en cuenta nuevos aspectos que se consideran en el presente trabajo. Esta metodología aúna y comprende el modelo de Gestión de la Reconfigurabilidad Dinámica, el sistema de comunicaciones, las plataformas reconfigurables dinámicamente. El modelo de Gestión de la Reconfigurabilidad Dinámica que se propone junto a la nueva metodología está basado en el paradigma orientado a objetos, en lugar del modelo de tareas que ha sido normalmente usado como modelo de programación para aplicaciones reconfigurables [77], [78], [79], [81],

[82]. Como se describirá, la tecnología de objetos se adapta mejor a las tecnologías de reconfiguración disponibles actualmente puesto que conduce a una manera sistemática para el tratamiento del estado, ejecución de métodos y encapsulamiento.

La definición de una interfaz de objetos clara, incluye una detallada descripción de las operaciones y de los parámetros entregados por el objeto, como así también de los atributos que definen el estado. Esta interfaz y el uso del modelo de comunicación de objetos distribuidos detallado en el apartado anterior, extendido para soportar reconfiguración dinámica, proveerá la ortogonalización necesaria entre la funcionalidad del objeto y las tareas de reconfiguración y comunicación. El objetivo principal es hacer completamente transparente el servicio de reconfiguración dinámica, tanto en la gestión del proceso de reconfiguración, como en su uso desde el modelo de programación.

Capítulo 2

Estado del arte

2.1- Problemática en el diseño de Systems on Chip.

El diseño de Systems On Chip ha pasado por numerosas etapas desde sus comienzos donde, debido al desarrollo tecnológico constante en la fabricación de circuitos integrados, fue lográndose una mayor integración al haberse hecho posible la incorporación de nuevos circuitos de complejidad y prestaciones diversas, en un mismo substrato.

La cantidad de transistores integrados por unidad de área ha ido incrementándose de acuerdo a lo establecido por la ley de Moore[56], llegándose en estos momentos a integraciones del orden de los 10 millones de transistores por mm2 [57]. Esto hace posible ahora la fabricación de circuitos compuestos de diferentes cores o núcleos integrados en un mismo substrato. En la actualidad es posible poner varias unidades de diseño por unidad de área, lo que implica un potencial de desarrollo de productos muy atrayente para diversas industrias como la de telecomunicaciones, automotriz, defensa, etc.

Estos avances tecnológicos generalmente conllevan un incremento en la complejidad del diseño de sistemas en chip, complejidad que no es asistida por herramientas apropiadas. Esto provoca dificultades al momento del diseño, puesto que muchas veces estos avances no están soportados por las herramientas correspondientes. Además, al ser tecnológicamente posible la implementación de nuevos y más complejos diseños, con mayor cantidad de prestaciones, los diseñadores intentan aprovechar los recursos tecnológicos presentes, para sacar al mercado un producto con características cada vez más novedosas y en el menor tiempo posible.

Otro de los problemas mayores que existen en el diseño de sistemas on chip es el costo de fabricación, puesto que el costo de fabricación de las máscaras para el proceso fotolitográfico aumenta a medida que se avanza en tecnología. La necesidad imperiosa de reducir el costo de diseño y el tiempo de puesta al mercado, hizo que muchos investigadores, académicos e industriales hicieran

importantísimos aportes en el tema.

Para intentar subsanar algunas de estas dificultades han aparecido en escena nuevos recursos, metodologías y corrientes de diseño. Varias empresas de desarrollo de software se han abocado al diseño de herramientas EDA (Electronic Design Automation) para el diseño de sistemas on chip, que incluyen herramientas de diseño y de simulación, como por ejemplo Cadence Design Systems, CoWare Inc, Mentor Graphics, Synopsys, entre otros.

Diversas empresas que se dedican a la fabricación de dispositivos lógicos programables ofrecen a sus usuarios paltaformas de desarrollos de sistemas y herramientas de desarrollo y simulación para facilitar el uso y aprovechamiento de los mismos, tales como Xilinx, Altera, Actel, etc.

El diseño basado en plataforma hizo su aparición en escena como una metodología creada para hacer frente a estos problemas.

2.2- Diseño basado en plataforma.

En [2] se define a una plataforma como una abstracción de una familia de micro arquitecturas. Es una abstracción que cubre varios refinamientos a nivel más bajo. En este trabajo se plantea que, para software embebido, una plataforma es una micro arquitectura fija, pero lo suficientemente flexible, que permite implementar un conjunto de aplicaciones, logrando de esta manera amortizar el costo de desarrollo en varios productos.

Es decir, una plataforma se define como una familia específica de micro arquitecturas orientadas a una clase particular de problemas. Por ejemplo, un sistema embebido estará construido en un circuito integrado, el cual es una instancia de esa familia de micro arquitecturas que mejor se adapta a la aplicación a implementar.

En [44] se presenta otra definición de plataforma en donde se expresa que una plataforma de sistema consiste en módulos de arquitecturas parametrizables

para computación e interconexión. La plataforma del sistema es visible al software de aplicación permitiendo de esta manera que la programación de la aplicación sea hecha en software.

El diseño de sistemas en chip presenta una característica muy importante que es la separación de aspectos en el espacio de diseño. Esta separación implica una doble ortogonalización de conceptos: una entre la función (que representa lo que el sistema va a hacer) y la arquitectura (cómo lo va a hacer), y otra entre comunicación y computación[1].

El mapeo de una función en una arquitectura significa la implementación de la funcionalidad del sistema en un circuito integrado. En este proceso los costos de diseño, de verificación y fabricación inciden directamente en la calidad y costo del producto final. Reducir el costo de diseño es posible si se desarrolla, por un lado una arquitectura común que pueda soportar una variedad de aplicaciones diferentes, pudiendo reusarse esta arquitectura en otros diseños y por otro lado desarrollar una serie de componentes que pueden ser utilizados en más de una arquitectura.

Para ello se propone en [1] una metodología de diseño que permite el reuso del software por parte de los diseñadores de sistemas. Para hacer esto posible es necesario que exista una plataforma, una arquitectura básica de la implementación, formada por cores programables, subsistemas de entrada/salida, memorias y conexiones de red, llamada Plataforma de Hardware. Esta plataforma debe ser lo suficientemente flexible de manera tal que permita adaptarse a diferentes funcionalidades sin cambios significativos en su arquitectura.

Para definirla, [2] propone comenzar por una plataforma general, y obtener instancias de la misma eligiendo o configurando componentes a partir de una biblioteca, hasta obtener una arquitectura que cumpla las restricciones de diseño. Estas restricciones están dadas por dos vistas distintas: la de las aplicaciones a implementar y las restricciones desde el punto de vista del hardware, por ejemplo, performance, y costo en función de los recursos usados. La intersección de estos dos conjuntos de restricciones definen la arquitectura de la plataforma. Figura[1]

Luego de definida la arquitectura, el proceso involucra la exploración del espacio de diseño definido por las restricciones de la plataforma. Posteriormente

se mapean las funciones de las aplicaciones en la plataforma. Este mapeo involucra el particionado Hardware/software, indicando qué funcionalidad será llevada a cabo en Software y cuál en Hardware.

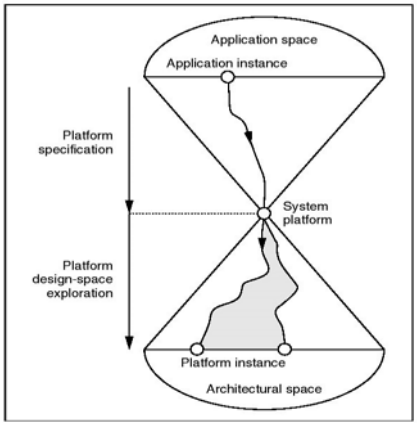

Figura 1

Para aumentar la reusabilidad del software los autores proponen abstraerse de esta plataforma y ocultar los detalles de la misma (arquitectura, implementación) a los desarrolladores, mediante una capa de software (wrapper) que envuelve los componentes principales de la capa de hardware. Esta capa de software (wrapper) contiene un sistema operativo en tiempo real (RTOS), Drivers para dispositivos I/O y un subsistema de comunicación en red. Esta capa de software es dependiente del hardware.

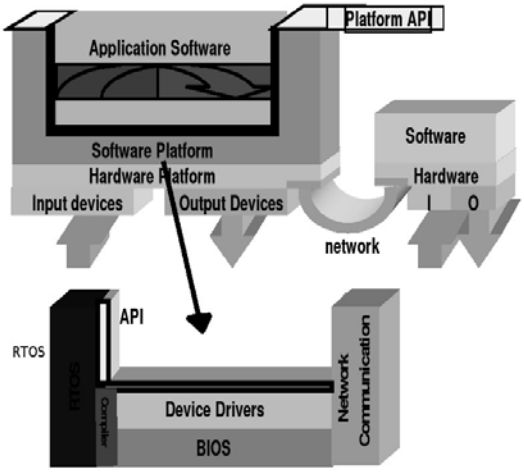

Figura 2

La interfaz entre esta capa de software dependiente del hardware y la capa

de aplicación se llama API (Application Program Interface). API entonces es considerada a su vez una plataforma que es una capa de abstracción que puede ser usada para diseño a mas alto nivel. Figura (2). Esta interfaz API es la única capa que “ve” el software de aplicación.

Además en este trabajo se plantea que es necesario considerar implementaciones intrínsecamente flexibles, que puedan cambiar rápidamente tendiendo hacia implementaciones basadas en software. Esto se logra gracias al incremento del poder computacional de los procesadores, correspondientemente con la disminución del costo de fabricación y del área que ocupan, lo que permite a los diseñadores llevar mayores funcionalidades al software.

Además, la elección de estas implementaciones flexibles, y la preferencia creciente de los diseñadores de circuitos integrados hacia chips que puedan aplicarse para varios diseños, es considerado como el nacimiento del diseño basado en plataforma en donde la reusabilidad y programabilidad son la clave [2].

Una extensión del concepto de diseño basado en plataformas aplicada para el diseño de sistemas distribuidos en red, es presentada en [14]. Aquí se parte de Network Platform, que es una librería de recursos, para formar una Instancia de Plataforma de Red. Lo mismo hace para plataformas Analógicas.

Este trabajo plantea que el valor del DBP (Diseño basado en plataformas) se multiplicaría con el uso de un set apropiado de herramientas y con un marco general de trabajo donde las plataformas puedan ser formalmente definidas en términos de rigurosa semántica, y manipuladas por herramientas de síntesis, optimización y verificación.

2.3- Clasificación de las plataformas de diseño.

Grant Martin y Frank Schirrmeister presentan en [5] una visión del diseño de sistemas embebidos, al 2002, donde exponen que el diseño basado en plataforma es la clave para mejorar costos y acelerar la cadena de diseño de sistemas embebidos. Los autores señalan 4 tipos de plataformas**: 1) full-Application platforms,** para desarrollo de productos derivativos. Incluyen una variada librería

de módulos de hardware con distintas configuraciones, mas una capa de software que separa la sección hardware del middleware y el software de la aplicación (API), de manera de probar diferentes optimizaciones de hardware sin tocar la aplicación. ej: Nexperia de Philips [59] para aplicaciones de telefonía móvil para Smartphones, OMAP de TI [60] para aplicaciones multimediales. Estas plataformas entregan un conjunto de bibliotecas hardware/software, presentan varios mapeos y ejemplos de aplicación y requieren los mayores esfuerzos de modelado [46]. **2) Processor-centric Platforms**, centradas en procesadores específicos, se enfocan en el acceso por software a un procesador, usándolo para modelar aplicaciones completas. Requieren elementos de hardware específicos. Tienen librerías de software para servicios de software claves. Ej: ARM Micropack, ST microelectronics. Estas plataformas poseen Cores de Procesadores y periféricos. Entregan drivers software y rutinas básicas de aplicación. El esfuerzo de modelado es menor que en (1) [46]. **3) Communication-centric platforms,** entrega una "fabrica" de comunicaciones optimizadas para el dominio de la aplicación. Pueden estar atadas a algún procesador pero son generalmente genéricas. El usuario debe agregar componentes para desarrollar una aplicación entera. Ej: Sonic uNetwork, ARM AMBA bus, IBM CoreConect. Estas plataformas poseen marcos de implementación para comunicación y periféricos. **4) Fully programmable platforms,** agregando lógica programable a la plataforma, van desde full-aplication platform con lógica reconfigurable embebida, a FPGAs con cores de procesadores. Ej: Xilinx Virtex II pro platform con powerPC[40], Altera Excalibur[41]. Incluyen Procesadores reconfigurables mas lógica programable.

Otra clasificación es presentada por Jauher Zaidi[66] quien establece dos categorías básicas: Plataformas genéricas y plataformas de aplicación específica. Plataforma Genérica es aquella que contiene un core CPU, controladores de memoria, Uart, timers, watchdog, Lcd controller, entradas/salidas de propósito general, con una arquitectura de bus para la interconexión de estos componentes. Las plataformas para aplicaciones específicas, además de los elementos anteriores, poseen bloques IP(Intellectual Properties) pre-integrados para una aplicación especifica, como por ejemplo la plataforma de IP Bluetooth de Parthus,

que contiene el controlador de banda base de Bluetooth [47].

2.4- Metodologías de diseño:

Para atacar el problema de la evolución constante de las especificaciones y la falta de herramientas de desarrollo de software embebido y de verificación del mismo, es necesario contar con una metodología rigurosa de diseño de software para sistemas embebidos y el diseño basado en plataformas. Es necesario una metodología de diseño de sistema y herramientas de soporte que permita resolver los problemas de integración de la cadena de fabricación, que considere métricas para el diseño de sistemas embebidos (costo, potencia disipada, peso, performance, etc), que trabaje a todos los niveles de abstracción, desde la concepción hasta la implementación, y favorezca el reuso de componentes.

Una propuesta de este tipo de metodología es presentada en [2] que, en resumen, consta de las siguientes etapas: a) Especificaciones, que deben incluir una descripción denotativa de la funcionalidad del sistema, el conjunto de restricciones de la implementación final y un conjunto de criterios de diseño; b) Análisis que permita la evaluación de los resultados intermedios con respecto a las restricciones de diseño, a través de métodos de análisis estadísticos y formales; c) Definición de la plataforma a elegir, mediante el uso de descripciones basadas en UML, que debe representar el rango total de servicios que la plataforma ofrece, y el uso de software de simulación como el SystemC 2.0 o 2.1 que permite una verificación más detallada de la plataforma a elegir, al insertar el diseño en el dominio de la plataforma hardware elegida; d) Mapeo, para ubicar las funcionalidades del diseño en los servicios que ofrece la plataforma, y e) Implementación, que incluya la posibilidad de obtener productos derivativos. Debe soportar síntesis de hardware, software e interfaces.

Una explicación detallada de la metodología presentada en el párrafo anterior puede ser encontrada con varios ejemplos de aplicación en [45].

Otro ejemplo de metodología de diseño es propuesta en [35], donde una **nueva ortogonalización en el proceso de diseño** es presentada. Li Li, et al, proponen un nuevo flujo de diseño basado en plataforma, en donde la idea de ortogonalización es explotada.

En su trabajo los autores plantean los siguientes problemas:

1) Plantean que las investigaciones y discusiones acerca del flujo de diseño basado en Plataformas es limitada e inadecuada.

2)Plantean una deficiencia en el flujo de diseño de SoC usando VCC (virtual components co-design) propuesto en el modelo de Princeton. En el modelo de Princeton la partición hardware/software está dividida en behavioural modeling, architectural modeling y mapping. Behavioural modeling detalla la funcionalidad, restricciones y especificaciones. El modelo de arquitectura es implementado eligiendo IP cores y módulos personalizados y conectándolos todos juntos. Mediante el mapeo del behavioral model en el modelo de arquitectura, VCC particiona las funciones especificadas y detalladas en el modelo behavioral. La deficiencia está planteada porque expresa que la ventaja y la idea esencial de la reusabilidad de los sistemas ya diseñados, y la extensión de los bloques IP basados en este sistema, no están representados en software y hardware.

Los autores dividen el diseño de sistema en tres partes: Diseño, Verificación y Soporte. Ver figura 3.

Para solucionar los problemas planteados los autores presentan una nueva ortogonalización entre el mapeo de plataformas y el mapeo de IP. La ortogonalización en la metodología de diseño es usada de la siguiente manera: La primera ortogonalización es entre diseño de sistema (función) y diseño de la arquitectura. La segunda ortogonalización es entre los dos niveles de mapeo de plataformas y mapeo de IPs en el diseño de la arquitectura. Ver Figura 3. La ortogonalización horizontal (entre computación y comunicación) no se modifica.

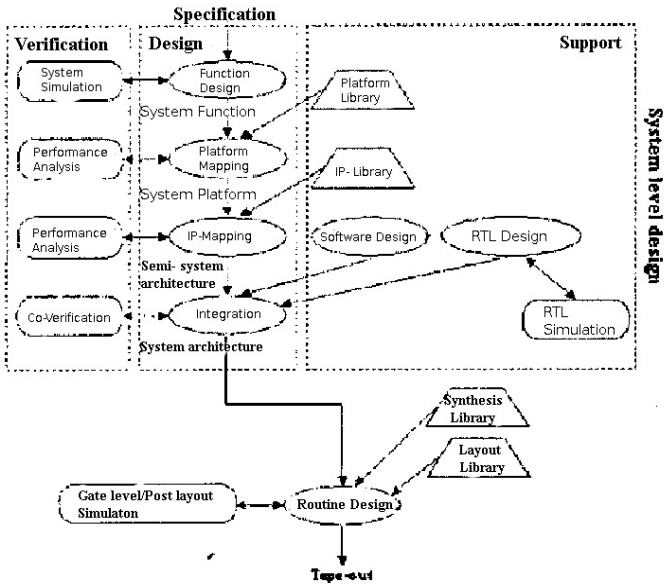

Figura 3

Este trabajo nombra los cuatro tipos de plataformas detalladas en [5], pero centra este planteo en plataformas Centradas en Procesador.

Una metodología iterativa de co-exploración a través de la ortogonalización procesador/comunicación es aportada por Wen Quan et. al, quienes presentan al codiseño hardware/software y al reuso del diseño a alto nivel como las claves para el diseño de SoC[16]. En este trabajo los autores muestran una metodología iterativa de co-exploración procesador/comunicación basada en ADL LISA (Architectural Description Language) y Modelado a Nivel de Transacción TLM (Transaction Level Modeling).

ADL LISA es usada para la descripción de la arquitectura del procesador de la plataforma, mientras que SystemC basado en TLM cycle accurate(CA), captura la arquitectura de la comunicación.

Describe el flujo de diseño basado en plataforma en 4 partes: Especificaciones del Sistema, Behavioral modeling, Architectural modeling, y refinamiento para satisfacer las necesidades de diseño. Para un análisis de performance inicial propone un Flujo de Diseño Revisado basado en plataforma,

donde, para dar solución al dilema de la primera generación de arquitectura, propone un mapeo inicial y se crea un modelo de arquitectura donde son insertados IPs disponibles y módulos propios. Para los IPs que no están disponibles se deriva de otros conocidos un modelo de performance que permita hacer una evaluación temprana de la performance total del sistema.

Presenta además la necesidad de configuración dinámica de SoC y la necesidad de verificación a nivel de sistema, proponiendo SystemC para esto. Presenta la co-síntesis hardware/software realizada en forma concurrente para alcanzar la mejor solución para las especificaciones y restricciones de diseño.

Zhihui Xiong et al, hacen su aporte a través de un método de diseño Jerárquico basado en plataforma para SoC [17]. Plantean una serie de problemas en el diseño basado en plataformas que son: a) Deficiencias en DBP para realizar síntesis directa desde el modelo del sistema al código de objeto SoC y descripción RTL, b) DBP no soporta revisiones de hardware dentro de la plataforma para la implementación de nuevas funcionalidades, solo revisión del software embebido. Falta de flexibilidad en el diseño. c) DBP no incorpora restricciones de performance dentro del proceso de diseño (first generation dilema). d) Modelado a nivel de sistema describe comportamiento y performance al nivel de algoritmo. Modelos tradicionales de sistema (redes de petri, modelo de evento discreto, modelo FSM, modelo Dataflow/control Flow) soportan la descripción behavioral pero no pueden ser usadas para describir restricciones de performance.

Para solucionar los problemas planteados presentan un método DBP Jerárquico que consta de 3 niveles de diseño y dos niveles de mapeo entre esos niveles de diseño:

1) Nivel de modelado de Sistema (behavior y performance a nivel de algoritmo) que usa CTG (Constrained Taskflow Graph) como modelo de SoC. Este modelo CTG es un tipo de modelo de descripción a nivel de algoritmo. Este modelo describe requerimientos de performance de tareas mediante la definición de atributos de restricciones (constraints attributes) y describe la funcionalidad de la tarea por mapeo algorítmico.

2) Nivel de componentes virtuales, que incluye componentes IP virtuales de

hardware, software y comunicación (VhwIP, VswIP y VcomIP).

3) Nivel de componentes reales que es el actual SoC hardware/software. Incluye la estructura y descripción de funciones de IP específicos, el set de instrucciones de procesadores embebidos, la red de interconexión, etc.

El mapeo a través de estos niveles es el siguiente: entre el nivel de modelado y el nivel de componentes virtuales se realiza a través de Design Planning, y el mapeo correspondiente entre el nivel de componentes virtuales y el de componentes reales es a través de virtual-real Synthesis. Los requerimientos de performance son tenidos en cuenta en los tres niveles de diseño y son transmitidos mediante los dos niveles de mapeo. Falta ver como construir teóricamente el first generation dilema.

Otro aporte es realizado por Qingxu Deng et al, en [22], donde presenta una metodología que provee una plataforma automáticamente, que incluye especificaciones de diseño a nivel de sistema, partición hardware/software , optimización hardware/software, para el diseño de System on Programable Chip (SoPC). Los autores proponen una metodología que ayuda al diseño de plataforma del sistema. Esta metodología incluye modelado del sistema, descripción formal y la provisión de plataforma del sistema. El modelo del sistema y la descripción formal la hacen usando FSM, EFSM, UML y se descompone el sistema en muchas tareas pequeñas. Posteriormente se particionan las tareas, se decide la construcción básica del hardware y se provee automáticamente una plataforma apropiada para hardware para los diseñadores. Como plataforma de software se diseña e implementa un RTOS con co-planificación hardware/software y soporte POSIX 1.1B. Para realizar esto usaron el kit de desarrollo de Xilinx Memec Virtex-4 LC

Gabriel Lipsa, et al [25], propone mecanismos de computación orgánica o autónoma para ser aplicados a métodos de diseño de hardware para SoC., para resolver la falta de metodologías y herramientas que incluyan en los procesos de diseño de SoC auto calibración, tolerancia a fallos y conceptos de auto curación.

Proponen un marco de trabajo y una metodología de diseño para SoC

autónomos donde la arquitectura está formada por dos capas lógicas: Una funcional, que contiene a los componentes IP, y la otra, la capa autónoma, que contiene los elementos autónomos que se comunican con cada IP y evalúan su estado (frecuencia, voltaje). Si uno falla, lo reemplaza otro que debe cumplir con las restricciones del sistema o lo reemplaza por un estado inactivo. Es reconfigurable.

2.5 Reusabilidad de componentes

La reusabilidad de componentes es uno de los elementos claves para reducir el costo de diseño, como se dijo. El desarrollo de plataformas con elementos reusables, y la búsqueda y elección de la arquitectura que mejor se adapte a un conjunto de aplicaciones, es motivo de varios trabajos de investigación. Uno de los aportes interesantes al respecto y que ha combinado esfuerzos entre diferentes grupos académicos es el presentado por Andrew Mihal.

Andrew Mihal et al. presentan en [4] el proyecto MESCAL (Modern Embedded Systems, Compilers, Architectures an Languages) que introduce una aproximación disciplinada para desarrollar plataformas reusables que pueden ser fácilmente programadas para cumplir con los requisitos de una variedad de aplicaciones. Se enfoca este trabajo particularmente en la exploración del espacio de diseño para aplicaciones de procesamiento en red, usando diseño de ASIP (Application-Specific Instruction Processor), cuya diversidad de arquitecturas posibles requiere una técnica que maneje la heterogeneidad de diseños posibles en ambos ejes, computacional y de comunicación.

Los autores hacen hincapié en una serie de problemas que ya han sido evidenciados en trabajos anteriormente presentados en esta revisión y que son: 1) Necesidad de reducción del time-to-market para un producto, 2) aumento de la complejidad de loscircuitos integrados, debido al aumento de cantidad de dispositivos que se integran, 3) aumento en los costos de fabricación para las nuevas tecnologías debido al aumento de costo de máscaras.

Una manera de amortizar los costos de diseño y de reducir el time-to-market

es hacer un solo chip que sirva para múltiples aplicaciones mediante reprogramación por el usuario. Este chip también puede ser parte de una plataforma reconfigurable que contenga recursos de hardware específicos. Estas plataformas existen en el mercado en algunos dominios de aplicación, pero el proceso de diseño en estas plataformas programables es en gran parte ad-hoc, son difíciles de programar y poco software de soporte está disponible.

Como una solución a estos puntos, los autores proponen el proyecto MESCAL que introduce una aproximación a la producción de plataformas arquitecturales reusables, fácilmente programables, que define y explora el espacio de diseño de una manera disciplinada a través de la combinación de modelos de Computación y de Arquitecturas con una metodología de mapeo disciplinado.

Los elementos claves de la metodología son: a) Desarrollo de benchmark representativos, que incluyan representatividad, especificaciones precisas y caracterización de la carga de trabajo, b) Definición del espacio de diseño teniendo en cuenta varios ejes, como grado de procesamiento paralelo, hardware de propósito especial, mecanismos de comunicación on-chip, arquitectura de memorias e integración de periféricos. c) Exploración del espacio de diseño (a través de una herramienta llamada TeePee) usando el modelo Y-chart que consiste en el mapeo de una aplicación en una arquitectura específica. Luego se analiza la performance y de acuerdo a los resultados se cambia o bien la estrategia de mapeo, o la arquitectura presentada o se retoca la aplicación. Esta exploración se realiza a través de una serie de vistas del modelo - micro arquitectura, operaciones, compilador, memoria, simulador, mapeo, aplicación, microprocesador. d) Verificación formal y consistencia entre las distintas facetas de la arquitectura propuesta, e) Mapeo de aplicaciones en la arquitectura definida.
Actualmente el proyecto MESCAL se realiza en colaboración entre miembros del UC Berkeley's CAD Group y el Princeton's Mescal Group. Más información acerca de este proyecto puede encontrarse en [58].

La selección e interconexión de componentes IP hardware-software es abordado por Cesario et al, en [3], quienes hacen un aporte en el diseño de SoC

presentando una **metodología y un ambiente para el diseño basado en componentes**. Este trabajo es una aproximación a la automatización del diseño basado en componentes para plataformas de MPSoC (Multiprocessors System on Chip).

Los autores observan los siguientes problemas en el diseño de SoC: 1) La complejidad de los sistemas de comunicación en plataformas MPSoC, debido a la heterogeneidad de los procesadores que pueden formar el sistema, diferentes protocolos y de diferente complejidad y topologías distintas. 2) La necesidad de mecanismos sofisticados de sincronización para el control de comunicaciones compartidas a través de varios masters no homogéneos. 3) El uso de diferentes sistemas de buses complejos o micro redes como interconexiones globales. 4) la falta de herramientas para el diseño de capas de abstracción hardware/software para la interconexión de IPs.

Los autores presentan un modelo conceptual de plataforma que está formado por cuatro tipos de componentes: Las aplicaciones en software, unidad núcleo (core) de microprocesador, cores de IP, y el componente de comunicación e interconexión dentro del chip. Para completar la plataforma los autores incluyen las capas de hardware y software que adaptan un componente con otro. Plantean el uso del mismo concepto de abstracción usado para el desarrollo de software de aplicación a través de la capa API, descritas en [1] y [2], para el desarrollo de software dedicado (firmware) que es el software que gobierna la plataforma.

Describen el proceso de diseño actual de SoC dividiéndolo en 5 etapas: 1) Especificación del sistema, 2) exploración de la arquitectura , donde a partir de un modelo ejecutable de las especificaciones e iterándolo hasta cumplir con las restricciones de performance, se realiza la partición hardware/software. Esta exploración se basa en una plataforma abstracta con modelos de componentes hardware/software. En esta etapa se determina el “Golden model architecture”, que es la arquitectura sobre la que se implementa el diseño. 3) Diseño de software, a través de la capa API, 4) Diseño del hardware, 5) Integración hardware/ software. El Golden model architecture especifica las restricciones de performance que debe cumplir la integración hardware/software. En esta etapa los diseñadores crean las interfaces para la interconexión de tales componentes.

La herramienta propuesta por los autores automatiza los pasos 3, 4 y 5 del proceso de diseño presentado. Comienza con una arquitectura virtual a partir del golden model descrito en el párrafo anterior, que es una netlist abstracta de componentes virtuales. Estos componentes usan wrappers para adaptar los accesos de los componentes internos (hardware o software) a los canales externos. Esta arquitectura virtual modela a los wrappers como un conjunto de puertos internos y externos, virtuales. que pueden diferir en el nivel de abstraccion, el protocolo de comunicacion o en el lenguaje de especificación.

Este flujo de diseño especifica la arquitectura virtual a partir de las librerías de SystemC C++ extendidas con las siguientes clases: Módulos virtuales (módulo y su wrapper), grupos de puertos virtuales, internos y externos, que tienen una relación de conversión, grupos de canales virtuales y parámetros para modificar para requisitos particulares de interfaces de hardware.

Todas las herramientas de diseño propuestas en este trabajo usan un modelo unificado de diseño que contiene una netlist abstracta hardware/software. La generación automática de wrappers hardware transforma el modelo de entrada en una arquitectura sintetizable. El generado de wrappers software produce un sistema operativo para cada procesador de la plataforma destino.

Este trabajo propone la generación de una arquitectura virtual y el uso de herramientas de generación automática de wrappers para las interfaces software/hardware.

Desde el punto de vista de los componentes creados por terceras compañías que pueden ser utilizados y reusados por otros desarrolladores en sus diseños basados en plataformas, la reutilización de estos IPs permite la reducción del tiempo de diseño para lanzar un producto al mercado en menor tiempo, logrando mejorar la problemática de Productividad lenta y corta vida útil de los SoC.

Pero esta ventaja se ve limitada por el problema de la falta de estándares en la creación de IPs, la dificultad encontrada en la búsqueda y calificación de los IPs para mejorar productividad y reusabilidad.

Hans-Jürgen Brand, et al,[21] presentan un método para facilitar la

búsqueda de IPs para el desarrollo de plataformas, en donde la aplicación de componentes reusables es considerada indispensable para la mejora de productividad en el diseño de sistemas. Para ello es necesario contar con un método de evaluación y selección de componentes que puedan ser añadidos al diseño, que tenga en cuenta una serie de parámetros y que permita encontrar el componente que mejor se adapte a la necesidad planteada.

Presenta el proyecto IPQ (Intellectual Property Qualifications). Método para buscar, seleccionar, evaluar, integrar y adaptar IP eficientemente. En este trabajo se presenta la creación del grupo IPQ que trabaja las siguientes áreas: 1) Estandarización de IPs y de especificaciones orientadas al re uso. 2) Selección y adquisición de IPs mediante técnicas provenientes del campo de “sistemas basados en conocimiento” o “inteligencia artificial”, en particular método de propagación de constraints y Case Base Reasoning. 3) Análisis y evaluación de los IPs seleccionados, teniendo en cuenta su compatibilidad con las condiciones tecnológicas y metodológicas de diseño. 4) Adaptación de IPs que no satisfacen completamente las características y restricciones de diseño.

Este proyecto surge de la cooperación entre empresas y universidades. Poseen varias herramientas EDA desarrolladas.

2.6 Plataformas reconfigurables

En plataformas reconfigurables existen varios aportes que se han realizado donde se presentan diversas metodologías de diseño teniendo en cuenta la posibilidad de reconfiguración que presentan los dispositivos lógicos programables. Una plataforma reconfigurable está compuesta por elementos de hardware que permiten una reconfiguración dinámica en tiempo de ejecución sin necesidad de detener la aplicación. Las plataformas fijas extienden su campo de aplicación al poseer procesadores donde correr el software de aplicación, son de uso general. La incorporación de elementos de hardware reconfigurable permite ampliar el campo de aplicación de las plataformas puesto que permite una reconfiguración de componentes en tiempo de ejecución, lo que aumenta de

manera considerable la versatilidad de la arquitectura y permite reducir aún mas el tiempo de diseño.

Esta reconfigurabilidad también es necesaria para la elección de la arquitectura más adecuada cuando es necesario cumplir con restricciones de diseño, como por ejemplo performance, consumo de potencia, etc. Krishna Sekar et al [8], hacen su aporte al diseño basado en plataformas introduciendo el concepto de Gerenciamiento Dinámico de Plataformas. Una metodología que permite modificar para requerimientos particulares, en tiempo de ejecución, una plataforma de propósito general configurable para mejorar la eficiencia en consumo y performance, en el diseño de sistemas embebidos.

Los autores presentan en este trabajo una metodología para gerenciamiento dinámico de una plataforma de uso general que permite configurar, en tiempo de ejecución, una arquitectura flexible donde los parámetros factibles de modificación son la memoria, la frecuencia de operación y la tensión de alimentación.

Problemas como limitaciones en costo de diseño, tamaño, consumo de potencia, necesidades de alta performance (sobre todo en wireless applications) se pueden solucionar con las plataformas configurables, pero tienen el inconveniente de que éstas se configuran estáticamente y posteriormente, si hicieran falta cambios u optimizaciones, estos se harán sobre la plataforma hardware subyacente.

Una forma de solucionar esto es mediante el Gerenciamiento de plataformas dinámicas, implementado a través de una capa de software que entiende y explota conocimiento de la aplicación y sus características, para gerenciar y configurar en forma óptima los recursos de la plataforma.

La arquitectura de la plataforma soporta partición de datos en forma dinámica entre la memoria rápida on-chip SRAM y la memoria externa mas lenta. Esto provee la facilidad de decidir en tiempo de ejecución cuál es el conjunto de datos más conveniente para ser almacenado en la memoria rápida on chip.

El espacio de configuración de la plataforma es bidimensional. Una dimensión está determinada por diferentes pares de valores frecuencia de reloj-tensión de alimentación asociadas. La segunda dimensión está dada por las diferentes opciones en que el conjunto de datos es particionado entre la memoria

on-chip y la externa.

Este gerenciamiento es únicamente referido a frecuencia, voltaje y datos en memoria para definir cuáles datos dejar en memoria interna y cuáles no, de acuerdo al porcentaje de uso de la CPU para optimizar el consumo. La prueba de esta metodología está aplicada en el diseño de un sistema de procesamiento de seguridad de acceso dual UMTS/WLAN asumiendo una plataforma subyacente compuesta de: Voltaje y frecuencia seteable en tiempo de ejecución, un procesador embebido STRONG ARM1, y arquitectura de memoria flexible conteniendo una rápida y pequeña memoria on chip.

Integrar la reusabilidad de componentes y la facilidad de reconfiguración dada por las plataformas con componentes de lógica reconfigurable, resolver los problemas de la comunicación entre los IPs que componen el diseño y el de la falta de estandarización de los mismos, plantea la necesidad de diseño y optimización de una arquitectura eficiente que permita Reconfigurabilidad y programabilidad. Esta arquitectura debe permitir aplicaciones de alto rendimiento y bajo consumo, eficientes en area, consumo de potencia y en uso de recursos, para poder mantener el numero requerido de capacidad de computo (1000 MIPS mínimo) integrando múltiples IP cores o unidades funcionales con velocidades de procesamientos de paquetes de 10Gbps.

Además, es necesario que esta arquitectura solucione los problemas que se presentan con el uso de buses compartidos debido a las latencias en la comunicación, la necesidad de buffers grandes para reducir las condiciones de overflow, los problemas físicos de diseño por problemas de limitación de tamaño de buses y las complicaciones de timing de los mismos para las comunicaciones entre las interfaces (IPs)

Este problema es abordado por Pascal Nsame, e Yvon Sacaria en [15]. Los autores presentan una plataforma reconfigurable para la interconexión de IPs que puede ser modificada para requisitos particulares, de propósito general, donde la estructura de comunicación está basada en Virtual Channels, usando colas (queues) y transacciones basadas en paquetes, para la comunicación entre unidades funcionales (IP Cores).

La plataforma propuesta es una Plataforma de conexión de IPs (multiprocessor, realtime application, etc) basada en VCCA (Virtual clear channel avoidance) que permite reusabilidad, integración de SoC, optimizaciones en performance y potencia. Combina interconexiones on-chip y funciones a nivel de chip. Consta de: a) Unidades de control y Datapath optimizados para 32, 64, y 128 bits formados por fifos/queues virtuales, DMA multichannel y motor configurable de links (CLE-Configurable link engine) para mensajes de I/O. b) Macro de interface de software y reloj con control de interrupciones, reloj de tiempo real, PLL.

Cada IP se conecta plug & play, a través de enlaces full-duplex mediante un PBI (Packet Based Interface) de 64 bits que funciona a 800 MHZ, a la plataforma VCA. Cada PBI se adapta al IP que atiende. Cada canal virtual tiene sus recursos de buffers de paquetes.

Aseguran que elimina la necesidad de wrappers en el bus y provee soporte de hardware para múltiples transacciones concurrentes en canales virtuales.

Otro aporte que integra la reconfigurabilidad dinámica de plataformas y la utilización de componentes pre diseñados es presentada por Markus Visarius et al, en [20] Propone una reconfiguración dinámica de sistemas basados en IP.

Presenta un flujo de diseño para automatizar el diseño de sistemas embebidos. Comienza con la especificación del sistema, búsqueda y selección de IP (para esto usa IPQFORMAT, que puede ser usado para intercambiar IPs entre proveedores y usuarios de manera estandarizada). Provee además diseños de IP y la posibilidad de editarlos a través de una GUI y permite el control de IP basado en servicios web.

Una vez encontrados los IPs posibles se evalúa la mejor composición mediante un algoritmo basado en CBR (Case Base Reasoning) de IPQ y mediante una función “costo” que incorpora criterios de decisión para minimizar los posibles conjuntos. Estos IPs se combinan y conectan para crear el sistema específico.

Para que sea un sistema reconfigurable, debe ser posible mapear IPs mientras otros IPs están activos. Este proceso se llama Partición de la Configuración. Luego se generan los canales de la comunicación, el controlador de reconfiguración y por ultimo la generación de código (hwMask o bitstream). Este

trabajo está orientado solo a IPs, no conjuga software embebido en la problemática.

Patrick Lysaght presenta un aporte en [6] donde describe diferentes aspectos a ser tenidos en cuenta en el diseño basado en plataformas. Dentro de los 4 tipos de plataformas señalados por Martin y Schirrmeister en [5], este trabajo está orientado al cuarto tipo Full Programmable Platform, al cual se refiere como Meta-Plataformas,

El autor plantea mejorar la metodología de diseño basado en plataforma para ASICs, presentando una nueva metodología de diseño a través del desarrollo de meta-plataformas. Meta-plataformas incrementa la creación de metodologías de diseño basado en plataformas permitiendo a los diseñadores crear su propias plataformas, a partir de la cual pueden instanciar sus propios diseños derivativos.

Dentro de los componentes reconfigurables existentes en las plataformas, se incluyen los dispositivos de lógica programable como las FPGA, en los cuales es posible implementar en forma estática y/o dinámica en sus circuitos lógica definida por el usuario, interfaces, núcleos de procesadores llamado procesadores embebidos sobre los que corre el software de aplicación del sistema, etc.

Estos componentes reconfigurables es posible tenerlos en el mismo chip, junto con cores hardware de procesadores, memoria, etc.

Este incremento en las capacidades de los circuitos integrados, dado por los avances tecnológicos, implica un aumento en la complejidad en los diseños de systems on chip. Este incremento de complejidad en el diseño hace necesaria la existencia de métodos de diseños consistentes y bien organizados y una mayor y efectiva verificación del mismo, para lanzar un producto al mercado lo antes posible.

Un aporte en este campo, donde se presenta una metodología de diseño de SoC basado en plataforma con FPGAs es realizado por Nobuyuki Ohba, et al en [37].

Propone una metodología de diseño para SoC que hace uso completo de las capacidades de las FPGAs. Los módulos de diseño en diferentes niveles de

abstracción son implementados en una FPGA, conectados y combinados unos con otros y ejecutados juntos en un sistema de prototipado con FPGA que emula el SoC final. El diseño es realizado en forma top-down:

1) Diseño al nivel de especificaciones: El SoC es descrito a nivel de especificaciones, luego creado y verificado usando una PC y transferido a un sistema de prototipado con FPGA, para que pueda correr en los cores de powerPC embebidos.

2) Diseño a nivel de transacción: Los módulos son descritos a nivel de transacción y luego simulados en los cores de hardware de PowerPC en la FPGA. El diseño a nivel de transacciones es simulado por dos de los cores de powerpc embebidos y se conectan al bus del sistema a través de wrappers hardware.

3) Diseño a nivel de Cycle Accurate: Para ejecutar módulos a nivel de Cycle Accurate en los core de hardware de powerpc, debemos asegurarnos que "mantengan el paso"con otros módulos escritos a diferentes niveles. Para hacer esto, cada powerpc manda una señal al marcador de paso del reloj cuando termina la ejecución de las tareas correspondientes a un ciclo de reloj. El marcador de paso (que es una compuerta AND que recibe las señales de Ciclo Completo proveniente de los módulos simulados) chequea si todos los trabajos a ejecutarse en ese ciclo están terminados y envía una señal a cada módulo para que ejecuten los trabajos correspondientes al ciclo siguiente.

El diseño es a nivel de RTL. Todos los módulos de diseño en RTL son sintetizados y mapeados en la FPGA. No hace análisis de consumo ni evalúa performance del SoC. Evalúa performance de la simulación y es una alternativa para mejorar los tiempos de diseño. Es para hacer prototipo y simular. Declara que deben incorporarse simulaciones intensivas de timing debido a las diferencias entre los tiempos de setup y holding entre las FPGAs y los ASICs, y a la diferencia de comportamientos entre uno y otro debido a la dependencia tecnológica.

Otro ejemplo referente a modelos de plataformas reconfigurables es presentada por Francesco Lertora et al, en [19], quien plantea que la complejidad alcanzada por las ultimas generaciones de aparatos reconfigurables no está balanceada con las herramientas y flujos necesarios para mapear aplicaciones en

esas arquitecturas. El trabajo para obtener el máximo potencial es difícil y se obtiene a través de soluciones de mapeo específicas para la aplicación. Existe además en el mercado la necesidad de ejecutar aplicaciones de software muy complejas con alta relación performance/consumo.

En este trabajo se presenta una plataforma para SoC integrado por un procesador, 3 FPGA y un NoC de 8 puertos que es una arquitectura basada en un ARM, que se conecta a una plataforma reconfigurable formada por 3 FPGA, a través de buses AMBA, interrupciones y canales de coprocessor. La infraestructura de comunicación está basada en una arquitectura de mensajes pasantes (message passing) implementada usando un NOC.

La plataforma del procesador está basada en un ARM926EJS, buses AHB AMBA para interfaces y APB bus para la interconexión de IP a través del puente AHB/APB. Posee IPs conectados que son GPIO, timers, reloj de tiempo real, uart, I2C master.

La Plataforma reconfigurable está formada por: 3 FPGA, SRAM, Register file, unidad VMI (Versatile message interface) que conecta los puertos de mensajes de la FPGA con el NoC, un Noc de Fat 3 de 3 niveles y 8 puertos.

La plataforma reconfigurable puede manejar a lo sumo 5 procesadores hardware autónomos en cada FPGA.

El VMI es la clave del diseño: Decodifica y direcciona los paquetes de NoC al puerto de mensajes correcto de la FPGA. Setea el canal de comunicación comenzando a partir del pedido hecho por el puerto de mensaje de la FPGA. Implementa un procedimiento automático para reconocer si un proceso hardware está conectado a una interface de mensaje de una FPGA dada.

Cada proceso de hardware dentro de la FPGA es manejado por una FCU (FPGA Control Unit) y está interfaceado con un puerto VMI.

Desarrolla una metodología (REAL Scope) Reconfigurable Algorithm usan Swappable Coarse-grain IPs, que analiza los Kernels de aplicaciones que necesitan ser reemplazados por hardware. Los IPs mapeados son manejados por un microcontrolador específico cuyo microcódigo está almacenado en las SRAM disponibles para cada FPGA.

Construido en tecnología CMOS de 90 nm.

Las restricciones de diseño cada vez más exigentes para los sistemas embebidos para comunicación wireless (mecanismos de acceso, tasas de error y de transmisión, ancho de banda, consumo de potencia, conexión con otros aparatos con diferente protocolo), y el problema de desarrollar ASICs para aplicaciones que sufren la constante evolución de los estándares y especificaciones de comunicación, han llevado a trasladar el desarrollo basado en plataformas para el dominio de aplicaciones orientadas a comunicación.

Al respecto Visvanathan Subramanian et al [9] hacen su aporte presentando una metodología para el desarrollo de una plataforma reconfigurable para el diseño de sistemas embebidos para comunicaciones.

Divide el diseño basado en plataformas en tres fases: Concepción, instanciación e Implementación de plataformas. La concepción es el proceso de desarrollar las plataformas de hardware y software partiendo de las especificaciones, para extraer las funcionalidades comunes de la aplicación y determinar la mejor arquitectura para esas funcionalidades de acuerdo a las restricciones del diseño. La Instanciación corresponde al proceso de mapeo que involucra el desarrollo de plataformas a partir de modelos de arquitecturas, en la plataforma del sistema y elegir la configuración óptima. La Implementación es básicamente la compilación de los programas de aplicación y la síntesis del hardware para crear la aplicación.

En este trabajo los autores proponen el uso de plataformas reconfigurables para el diseño de sistemas embebidos para comunicación.

Presenta un ejemplo para manejar y reconfigurar comunicaciones y conectividad en caso de emergencia y desastre, y estudia tres alternativas de espacio de diseño: a) Procesador simple, (CPU, memoria, periféricos y controlador de periféricos/dma), b) Procesador de propósito general más FPGA como coprocesador actuando como controlador DMA, c) Arquitectura centrada en memoria, combinada con una FPGA y un procesador especializado en comunicación. Esta última alternativa fue usada porque permite la implementación de aplicaciones parcialmente en software y parcialmente en hardware en la FPGA y se puede reconfigurar dinámicamente para actualizarse a la aplicación.

2.7- Herramientas de diseño, modelado y verificación:

La implementación efectiva de aplicaciones en arquitecturas de plataformas con FPGAs no se puede llevar a cabo sin un conjunto completo de herramientas que permita implementar la lógica teniendo en cuenta las ventajas y características de la FPGA disponible, de manera de aprovechar los recursos existentes de manera óptima, y su integración con los demás componentes del sistema.

La reconfigurabilidad de las plataformas debe venir acompañada de herramientas que permitan aprovechar al máximo sus capacidades de diseño. Además es necesaria una plataforma de simulación que provea la evaluación de la performance requerida para el desarrollo de SoC, y que permita la validación de la arquitectura en etapas tempranas de diseño.

Existen varias plataformas de simulación y prototipado, pero presentan algunos inconvenientes, como por ejemplo, una plataforma de simulación de software puro no puede proveer información acerca de la performance requerida. Varias plataformas de emulación de multiprocesadores como ARM integrator o Hunt Engineering's Heron no permiten la adición de IPs creados por terceros [23]. Soluciones existentes no pueden evaluar diferentes arquitecturas porque su topología de interconexión no puede ser modificada. Las plataformas de emulación existentes son lentas, caras y no proveen las herramientas requeridas por los desarrolladores de software.

En este campo de investigación, Mario Diaz Nava et al, [23] presentan una plataforma de emulación de MPSoC reconfigurable desarrollada por STMicroelectronics para diseño y verificación.

ARCHIFLEX SYSTEM PLATFORM, plataforma de emulación cuya arquitectura consiste en 3 componentes principales: 1) El nodo de computo, que incluye un procesador ST230 core (VLIW), diferentes niveles de cache, y puertos

I/O con DMA. Un STBus interconecta los componentes. 2) La interface de Red que conecta el nodo de computo a la red de comunicación o al sistema interconectado. Incluye una FPGA para el soporte de protocolos diferentes e interfaces de conexión. 3) Un sistema de interconexión reconfigurable que soporta topologías de distinta complejidad, que permite la evaluación de distintos esquemas de interconexión y topologías y elegir el que mejor se adapte al sistema evaluado,

Está formado por 2 placas (boards), la del nodo de computo con un SoC asociado (TC4SoC) y la placa de intercomunicación (Communication Network Board) con 2 versiones: una con 1 FPGA con más de 1000 I/O pines para mapear las interconexiones y la otra para diseños más complejos de MPSoC con un array de FPGAs para las interconexiones. Tiene facilidades para debugging hardware y software y viene con un set de herramientas de software para crear programas para el core ST 230

Otra visión desde el punto de vista del modelado es presentada por Annti Pelkonen et al en [28].

Este trabajo presenta una metodología de modelado a nivel de sistema y las herramientas asociadas capaces de realizar una rápida exploración del espacio de diseño, teniendo en cuenta las propiedades del hardware dinámicamente configurable. Está basada en SystemC.

Presenta una clasificación de las arquitecturas reconfigurables con respecto a los siguientes parámetros: a) Esquema de reconfiguración: plataformas estática o dinámicamente reconfigurables, b) Acoplamiento: reconfigurabilidad fuertemente acoplada con el procesador anfitrión (la unidad reconfigurable está en el datapath del micro), o reconfigurabilidad pobremente acoplada (la unidad reconfigurable actúa como un coprocesador); y c) granularidad de los elementos de procesamiento, referido a los niveles de manipulación de datos.

Presenta el flujo de diseño adaptado para soportar el desarrollo de hardware reconfigurable dinámicamente y la metodología de modelado a nivel de sistema basada en SystemC. Esta metodología toma un módulo de SystemC como entrada y transforma las instancias del módulo y los componentes jerárquicos que crearon dicho módulo para usar un componente llamado

Dynamically reconfigurable fabric (DRCF). Esta metodología y sus herramientas proveen una manera de ensayar los efectos de implementar algunos componentes en hardware dinámicamente reconfigurable. El DRCF reemplaza a los componentes elegidos, implementando todas las funcionalidades del componente candidato con parámetros que incluyen el modelado del tráfico del bus de memoria.
Limitaciones: La metodología presenta algunas limitaciones reconocidas por el autor específicas de SystemC.

Otro aporte en simulación y verificación de diseños SoC la realiza en [12] R.Henftling, et al, en cuyo trabajo presenta una nueva tecnología basada en plataformas que acelera la Verificación de Sistemas. Este trabajo apunta a solucionar una serie de problemas de verificación del diseño provocados por: a) la creciente complejidad en los testbenchs necesarios para una prueba completa del diseño, debido al aumento de prestaciones de los SoC, b) la enorme cantidad de tiempo que lleva la verificación del sistema completo, formada por la construcción del testbench, el tiempo de simulación puro y el tiempo de los ciclos de prueba y c) la falta de estándares para los tests de pruebas, generadores de protocolos o componentes virtuales, que se necesiten para verificación.

Presenta el uso de plataformas para la generación de un testbench configurable y sintetizable en una arquitectura, partiendo de un testbench behavioral, que puede ser mapeado en emuladores o en una FPGA.

Este método usa unidades de control programables, interfaces bien definidas, generadores de protocolos específicos y elementos de testbenchs sintetizables. Parte de un testbench behavioral, y lo sintetiza para conectarlo al diseño bajo test y probarlo.

La exploración del espacio de diseño para encontrar la configuración óptima en la mayoría de los casos no es posible realizarla teniendo en cuenta todas las posibles configuraciones de la arquitectura de una plataforma. Existe una falta de herramientas para la búsqueda de configuraciones óptimas. Esta configuración es un problema de optimización multi-objetiva, puesto que existen varios parámetros

de evaluación, tales como tiempo de ejecución, consumo de potencia, área utilizada, etc. Además existe una carencia en herramientas de exploración automática y búsqueda de soluciones en un espacio de soluciones multidimensional para la selección de las mejores configuraciones. [11], [27] y [29] realizan aportes en este problema

K. Ghali et al, [11] presentan un método para la evaluación de configuraciones de procesadores embebidos en plataformas FPGA.

Propone una forma de evaluación basada en ejecución directa de la mejor alternativa de procesador que se pretende embeber en una FPGA, teniendo en cuenta parámetros como área, consumo de potencia, cantidad de ciclos de ejecución necesarios para aplicaciones de prueba. Usa ISE 6.1 de Xilinx y una plataforma de diseño con Virtex II. Cuando se cumplen con las restricciones de diseño impuestas estos procesadores se implementan, se ejecutan programas de pruebas y se elige la mejor configuración de procesador embebido encontrada. Este trabajo está basado en el Procesador RISC embebido LEON-2, con 5 etapas de pipeline, compatible con sparc v8, diseñado para aplicaciones embebidas. Se intenta encontrar la mejor configuración de este procesador para ser embebida.

Claudio Talarico et al, presentan en [27] una nueva estrategia para la exploración del espacio de diseño (DSE) de plataformas SoC.

Introducen una aproximación para la exploración multi-objetiva, del espacio de configuración de un SoC parametrizado, (teniendo en cuenta el consumo de potencia y el tiempo de ejecución). Proponen usar la heurística **recocido simulado** para computar una aproximación precisa al conjunto de soluciones óptimas de Pareto. Se compara este método con el de búsqueda exhaustiva y se encuentra que, analizando un espacio de soluciones mucho menor, se obtienen configuraciones muy aproximadas.

Yang Qu et al, [29] presentan una continuación del trabajo [27]. Se basa en una aproximación de diseño a nivel de sistema reconfigurable basado en SystemC, que puede ser fácilmente embebida en un flujo de diseño SoC para permitir rápida exploración del espacio de diseño para diferentes alternativas de reconfiguración sin entrar en detalles de implementación.

Presentan una metodología de diseño a nivel de sistema para

Reconfigurable System on Chip (RsoC), cuyo objetivo es proveer un mecanismo que permita a los diseñadores evaluar los efectos de la implementación de algunos componentes en hardware reconfigurable. Soporta estimación y análisis para la exploración del espacio de diseño y particionado del sistema, provee herramientas para la generación de modelos de hardware reconfigurable.

Limitaciones: No tienen generación automática de códigos para diseño a bajo nivel. Hay que usar otra herramienta o hacerlo manualmente. (de systemC a C para software o VHDL para hardware)

En el espectro completo del diseño de SoC, arquitecturas heterogéneas se han vuelto cada vez más populares, por lo que el problema de la partición hardware/software es un problema que incluye asignación automática de aplicaciones en los diferentes recursos computacionales, optimización de la performance del sistema bajo restricciones, y el codiseño hardware/software [47]. Existen varios aportes al respecto [18] [47] [48].

G. Wang et al, proponen en [47] el desarrollo de un método para el problema de particionado de recursos a nivel de tarea. Esta aproximación, basada en el algoritmo de Colonias de Hormigas permite encontrar para una aplicación dada, la partición óptima de recursos hardware-software bajo ciertas restricciones de sistema: Abstracción a nivel de tareas, que permita el mapeo de tareas en procesadores de propósito general o en lógica configurable, y en donde exista un conocimiento previo de los recursos computacionales.

Una modificación a este trabajo es presentado en [18] donde Zhihui Xiong et al, presenta un método de particionado hardware/software para diseño basado en plataformas, basado en el algoritmo de Colonias de Hormigas con Feromona Inicial para PBD.

La falta de feromona inicial limita futuras mejoras de performance en la aplicación del algoritmo de colonias de hormigas usado por el método WANG[47] para la partición optima hardware/software en diseños SoC en procesadores de propósito general o en lógica reconfigurable.

Este método aplica el mismo método Wang, pero usando como feromona inicial, el resultado de una partición inicial. Para obtener esta partición

hardware/software inicial utiliza las partes del nuevo diseño que coinciden o son comunes a un diseño de referencia, a través de fórmulas, y las partes que no coinciden o no son comunes a través de otras fórmulas. A partir de acá se aplica el algoritmo que aseguran que mejora la eficiencia en un 40% con respecto al algoritmo sin feromona inicial. Utiliza un grafo de partición de tareas donde cada nodo representa una tarea con la cantidad de computo a realizar, teniendo como restricción el área a utilizar de la parte lógica reconfigurable. La partición que necesite menor tiempo de ejecución total y use menor área será la elegida.

En [33] se propone una nueva metodología para el co-diseño hardware-software para sistemas embebidos, usando lenguajes de programación de alto nivel. Se presenta una metodología de diseño basado en un grupo de herramientas (FLECOS) para co-diseño hardware-software, basada en una arquitectura reconfigurable que parte de un procesador estándar conectado a uno o más Datapath reconfigurables, con estructura de pipe, para trabajar como co-procesador. La herramienta selecciona, a partir del código en software únicamente, qué etapa de la funcionalidad es la que debe implementarse en hardware teniendo en cuenta diversos parámetros como performance u otro. Luego la herramienta genera la secuencia de configuración para cada etapa del pipeline del datapath.

Otro aporte orientado a herramientas CAD es presentado en [48]. En este trabajo se presenta un software de partición de hardware temporal, incluido en una metodología de diseño que usa las posibilidades de reconfiguración de las FPGAs para los diseños de sistemas SoC. Esta herramienta de partición llamada DAGARD minimiza el número de celdas necesarias para implementar una aplicación bajo restricciones de tiempo, teniendo en cuenta las necesidades de ancho de banda y tamaño de memoria. Esta aplicación evita el sobredimensionado de los recursos de implementación necesarios. Permite además una partición automática del Data Flow Graph completo.

Herramientas de análisis formal de diferentes propiedades de sistemas

heterogéneos basado en plataforma se presentan en [50]

En [49] se presenta una nueva aproximación para el gerenciamiento de la ubicación de funciones, soportando múltiples dispositivos reconfigurables. Es una aproximación rule-based que considera el uso de recursos y consumo de potencia. Posee una unidad de recuperación para el pedido de funciones que adopta metodologías del dominio del sistema basado en conocimiento, teniendo en cuenta los parámetros relacionados con la calidad de servicio, a partir del pedido de función de la aplicación

2.8 Reconfigurabilidad Parcial Dinámica

Se encuentran en la literatura diversos trabajos que tratan el tema de la reconfiguración parcial dinámica, donde se ofrecen técnicas diversas de reconfiguración y metodologías de diseño, pero altamente dependientes de la tecnología a usar, como en [10], [75] y [76].

Dentro de este campo de investigación se encuentran diversos trabajos que se pueden clasificar en las siguientes áreas: 1) Gestión de la reconfiguración, 2) ubicación (placement) de los módulos parcialmente reconfigurables, 3) scheduling de tareas y 4) partición hardware/software.

Con respecto a la Gestión de la Reconfiguración existen varias aproximaciones, como la de J.Resano et. al. [67], en donde se propone un modelo de Gestor de reconfiguración diseñado para reducir la latencia de reconfiguración usando dos técnicas diferentes en tiempo de ejecución para esto: prefetch scheduling y reemplazo de tareas. En la técnica de prefetch_scheduling el gestor recibe una secuencia programada de tareas y un conjunto de pedidos de reconfiguración y con esos datos genera una nueva programación de tareas. Una vez generado esto el gestor de reconfiguración aplica una técnica de optimización entre tareas. Junto con esto el gestor de reconfiguración tiene un modulo de reemplazo de tareas que determina cuáles son las que se van a reemplazar de

acuerdo a su criticidad. La implementación la realiza sobre una FPGA de Xilinx donde divide la FPGA en un arreglo de celdas idénticas que sirven como unidades de procesamiento de reconfiguración. La comunicación entre estas celdas se hace a través de mensajes sobre la red de interconexión como una NoC [70]. Cada una de estas unidades de procesamiento de reconfiguración posee una interface para conectarse al sistema de comunicación. Con este esquema de celdas uniformemente distribuidas se evita la sobrecarga de place and route que se necesitaría para la interconexión de tareas. No permite la incorporación de módulos diseñados por terceros y posee la restricción de que no se pueden agrupar dos o más celdas para alojar tareas más grandes.

En[68] se presenta una técnica para manejar reconfiguración dinámica del hardware basada en un scheduler en tiempo de ejecución. Este scheduler implementa un algoritmo llamado Multiplexado Dinámico de Hardware, algoritmo que permite un scheduling adaptativo espacial de tareas en una arquitectura reconfigurable. La configuración del sistema se realiza en tiempo de diseño, pero en tiempo de ejecución este scheduler tiene en cuenta la ubicación y modifica la configuración inicial de manera de adaptar los recursos computacionales para manejar varias aplicaciones o implementar diferentes calidades de servicio, abstrayendo de este modo las restricciones de diseño a tiempo de ejecución. Este algoritmo está implementado en software.

En [69] se presenta un modelo de Controlador de configuración para sistemas reconfigurables, desarrollado en hardware en lugar de las implementaciones realizadas en software presentadas en la mayoría de las aproximaciones al momento de la publicación. En [77] se propone un algoritmo de ubicación y un esquema de reubicación para tareas reconfigurables, donde ubicación y problemas de migración de tareas se discuten, basado en modelo de tareas.

Todas estas aproximaciones presentan el problema de la adaptación de los módulos reconfigurables al resto del sistema estático, ya que debe mantenerse la integridad y consistencia de las señales que entran y salen de los módulos

dinámicamente reconfigurables, para garantizar la comunicación entre estos módulos y la parte estática del sistema. Por lo tanto todos los módulos que se implementen en una zona reconfigurable deben presentar la misma interfaz.

Shibamura et. al [10] realiza su aporte en plataformas dinámicamente reconfigurables presentando un modelo de plataforma llamada Express-1 para el desarrollo de sistemas embebidos, usando FPGA y procesadores embebidos en la misma, con capacidad de reconfigurabilidad dinámica.

La plataforma propuesta en este trabajo está formada por una FPGA de Altera, usando un ARM-Excalibur embebido en ella [41]. La FPGA junto con la lógica de usuario y los IPs forman la plataforma hardware del sistema, y el RTOSy aplicaciones de usuario actúan como plataforma Software.

Las plataformas reconfigurables existentes que adoptan o utilizan FPGAs convencionales totalmente reconfigurables (esto es que no permiten reconfiguración parcial) tienen tiempo de reconfiguración lento que pueden provocar suspender el sistema en la reconfiguración. Estas FPGAs no están disponibles durante la reconfiguración, por lo que operaciones del sistema que dependen de la misma son a menudo suspendidas.

Para evitar esto se propone un sistema de gerenciamiento de reconfiguración. Este sistema provee funciones primitivas para reconfiguración dinámica y un Sistema de Gerenciamiento de Archivos, que maneja los archivos de configuración en la memoria externa anexada a la placa. Lo hace de la siguiente manera: El ARM prepara la parte de lógica configurable a través de registros de configuración. Luego un archivo de configuración (con extensión *sbi*) es leído de memoria y escrito en la lógica reconfigurable al final del archivo. Luego se deshabilita el modo de configuración.

Para reconfigurar el sistema sin detenerlo, introduce un mecanismo de Reconfiguración On the Fly, que oculta las tareas de configuración, detrás de otras tareas para mantener funcionando el sistema. Lo que hace es hacer las dos cosas alternadamente: reconfigurar - ejecutar, conmutando las tareas para no detener el procesador.

Además propone Reconfiguration On Demand (ROD) y mecanismos de

Ejecución Transparente para obtener las funciones necesarias para una aplicación, en hardware o software, de cualquier lado y proceder con la configuración. Para ello, si la aplicación necesita una función, el micro consulta una tabla de identificadores de funciones en memoria. Esta tabla consulta a las librerías de funciones implementadas en hardware y software. Si la función existe en software, la coloca en memoria y hace el link dinámicamente al programa principal. Por el contrario, si la función está en hardware ya implementada, la ejecuta y si no, busca en la librerías de hardware y la implementa.

Este trabajo analiza performance usando reconfiguración “on the fly” y reconfiguración bajo demanda (ROD) pero no analiza consumo de potencia ni área involucrada necesaria.

El problema de la asignación de recursos (place and route) y la programación de tareas en sistemas reconfigurables es tratado en diversos trabajos comoen [78], [79], [80] y [81].

La evaluación de disponibilidad de área significa detectar, dentro de los espacios de reconfigurabilidad parcial posibles, cuál es el que mejor se adapta a la tarea o función a implementar. Existen varios trabajos al respecto, en donde el área reservada para reconfiguración puede ser tratada como un espacio dinámico 2D, que puede ser compartido por todos los componentes y tareas dinámicamente reconfigurables como un mismo recurso. Esto permite gran flexibilidad de ubicación de componentes pero incrementa notablemente la complejidad de implementación en tecnologías actuales, como se detalla en [79]. Allí el autor propone un modelo de sistema operativo reconfigurable para manejar los problemas de ubicación en forma dinámica.

En [80] se propone un modelo de reconfigurable computing system usando el mismo modelo 2D, y utiliza un motor de ubicación (placement engine) que se invoca de dos maneras, para la inserción o para el borrado del módulo. Se asume que no existe comunicación entre los módulos y que los datos, a ser procesados por tales módulos, son previamente cargados en la unidad funcional reconfigurable (RFU) antes de que el módulo comience su ejecución. En nuestro trabajo

utilizamos el modelo 2D previamente particionado en áreas reconfigurables, ya propuesto por [4] de manera de simplificar la ubicación de los objetos dinámicamente reconfigurables

Dentro de los modelos de comunicación entre componentes o entre áreas dinámicamente reconfigurables existen varias alternativas. En [70] se adopta un modelo de red de interconexión que soporta ubicación de tareas, comunicación inter-tareas (incluyendo comunicación hardware/software) y soporte para sistema operativo. Este trabajo está íntimamente ligado con el [67] y presenta los mismos inconvenientes que enumeramos para aquél.

La reconfiguración dinámica utilizada en sistemas software para modificar y extender funcionalidades de un sistema en tiempo de ejecución es ahora posible trasladarla al hardware. Modelos como los de recuperación de fallas en sistemas basados en componentes distribuidos o sistemas expertos [71] para redundancia en sistemas críticos son ahora posibles en hardware para sistemas en chip.

Capitulo 3

Objetivos y Contextualización

3.1 Objetivo del trabajo de tesis:

Desarrollar una metodología de diseño de sistemas integrados tomando como base sistemas orientados a objetos, que consta de un servicio de Gestión de Reconfiguración Parcial Dinámica de Hardware para el diseño de Sistemas integrados, que pueda ser utilizado de forma independiente de la tecnología, y soportado por una arquitectura unificada de comunicaciones inspirada en el paradigma de los sistemas de objetos distribuidos. Esta arquitectura proporciona, desde el punto de vista de la comunicación, una abstracción homogénea de los diferentes elementos que forman parte del sistema, independientemente de que sean implementados en hardware o software. Además, esta arquitectura ofrece total inter-operabilidad con otros sistemas de objetos distribuidos, permitiendo por lo tanto la interacción con objetos externos al sistema integrado.

3.2 Objetos en lugar de tareas.

Desde el punto de vista lógico, podemos definir objeto como el "encapsulamiento de un conjunto de operaciones (métodos) que pueden ser invocados externamente, y de un estado que recuerda el efecto de los servicios". **[Piattini et al., 1996]**.

Un método se define como una operación que se aplica a un objeto, que lleva una serie de parámetros más un valor de retorno. Tanto los parámetros como los valores de retorno son opcionales. Estos son los servicios invocados por clientes de este objeto (otros objetos) o por otros métodos dentro del mismo objeto. Todo objeto presenta una interfaz definida por sus métodos para permitir la comunicación entre objetos. Esta comunicación se realiza mediante la invocación del método correspondiente.

El estado de un objeto está definido por sus atributos. Estos atributos definen el estado en que se encuentra un objeto en un determinado momento.

A los componentes hardware es posible modelarlos como objetos partiendo de las siguientes consideraciones:

La interfaz está definida por la entidad del componente y sus métodos y argumentos son traducidos a señales que se conectan en la entidad. Los parámetros se envían junto con las señales y pueden tener o no valores de retorno como los objetos.

Si los objetos están conectados entre si estas invocaciones son señales entre los mismos. Si están conectados mediante algún canal de comunicación estas invocaciones son realizadas remotamente mediante adaptaciones de los protocolos de comunicación correspondientes.

En vhdl se modela un componente hardware mediante una entidad y una arquitectura. La entidad es la interfaz del componente, representada por las señales que llegan y salen del mismo. Por estas señales llegan las invocaciones correspondientes. La arquitectura por su parte define el comportamiento del componente, similar al comportamiento del objeto, que se encuentra encapsulado en la definición del mismo.

Como se observa, la correspondencia entre objetos y componentes hardware es directa, y además, esta granularidad en la definición de componentes permite el reuso de los mismo

3.2.1Tareas:

Actualmente, la mayor parte de aplicaciones dinámicamente reconfigurables utilizan el modelo de programación basado en tareas. Desde el punto de vista de la reconfiguración, este modelo tiene algunos inconvenientes serios:

- Cada tarea tiene una interfaz propia, que raras veces es común al de cualquier otra tarea. Por otro lado, la tecnología de reconfiguración normalmente obliga a definir una interfaz invariable, puesto que ese es el

único modo de garantizar que el área dinámicamente reconfigurable está convenientemente aislada del resto del circuito durante el proceso de reconfiguración, y que, por lo tanto, este proceso no afectará a la integridad del sistema. Esto hace necesario la definición de interfaces compatibles entre las diferentes tareas que pueden ser asignadas al mismo área reconfigurable. Esta definición no es trivial, e incluso puede implicar una planificación muy rígida de las tareas mutuamente excluyentes y su ubicación dentro del dispositivo reconfigurable. Tampoco resulta fácil modificar tareas existentes o incluir nuevas tareas, una vez que el sistema ha sido implementado.

- Debido a la granularidad de las tareas, puede ser necesario desalojar la tarea antes de que su ejecución haya finalizado. En tal caso, la interrupción de la ejecución debe realizarse de manera que pueda reanudarse posteriormente en el mismo punto en el que fue detenida. La solución habitual consiste en definir explícitamente en el diseño los puntos de interrupción (checkpoints) de la tarea, en los que es seguro suspenderla para su reconfiguración. Por lo tanto, una petición de reconfiguración no será atendida hasta que la tarea alcance uno de los checkpoints definidos. Si la tarea es recargada posteriormente, la ejecución continuará a partir del checkpoint en el que fue detenida. La definición de los checkpoints es, por lo tanto, completamente dependiente de la implementación, y en consecuencia difícilmente automatizable.
- Una vez que la tarea ha sido detenida, un problema adicional es el de definir la cantidad de información que define el estado, y que debe almacenarse si se quiere garantizar la persistencia del objeto. El modelo de tareas no permite definir qué parte de la información almacenada en registros es realmente el estado, y qué parte es simplemente almacenamiento temporal utilizado para almacenar algún cálculo intermedio. Es, de nuevo, el diseñador el responsable de gestionar explícitamente la transferencia de los datos que considere relevante.

El modelo de tareas, por lo tanto, no proporciona un desacoplamiento suficiente

entre la tecnología de reconfiguración y el modelo de aplicación, por lo que diseñar componentes dinámicamente reconfigurables no es una tarea inmediata. Alternativamente se propone el uso del paradigma de objetos distribuidos.

3.2.2 Objetos:

La aplicación del modelo de objetos a los sistemas dinámicamente reconfigurables proporciona algunas soluciones a los problemas descritos anteriormente, ya que:

- La interfaz del los objetos está perfectamente definida desde las etapas iniciales del desarrollo de la aplicación. Además, la definición de un objeto incluye la lista de atributos que se consideran su estado. Se establece, por lo tanto, una clara separación entre el estado del objeto y la información temporal almacenada durante la ejecución de las operaciones.
- a diferencia del modelo de tarea, en el que las operaciones no se especifican explícitamente, los objetos proporcionan funcionalidad a través de un conjunto de métodos, también declarados en la interfaz del objeto. Los métodos son invocados en un instante determinado, realizan una serie de operaciones y finalizan. Los objetos, por lo tanto, no tienen un hilo de ejecución propio, sino que utilizan el del objeto que realiza la invocación. Gestionar la parada del objeto para garantizar su coherencia en el proceso de reconfiguración es mucho más simple. No es necesario que el diseñador especifique el momento y lugar en el que el objeto puede ser reconfigurado. Sólo se requiere esperar hasta que no haya ningún método en ejecución. Además este mecanismo es suficientemente general como para que pueda ser generado automáticamente para cualquier objeto dinámicamente reconfigurable.

La coherencia de estado está garantizada simplemente por el hecho de que si no hay métodos en ejecución los atributos no pueden ser modificados. Como además el estado es parte de la definición del objeto, la extracción y reinyección

es susceptible de ser completamente automatizada. Todo ello, de nuevo, de forma totalmente transparente al diseñador.

3.3 Arquitectura unificada de comunicaciones

Las aplicaciones se pueden considerar como una secuencia de tareas y estas son llevadas a cabo por objetos, La tarea la realiza el objeto activo, que se implementa en tiempo de ejecución. Este objeto realiza la tarea que sus clientes le invocan mediante metodos.

Los objetos se comunican con otros objetos invocando el correspondiente método. Para objetos softwares ejecutados en una misma CPU esta invocación se traduce en una llamada a procedimiento.

Pero este mismo modelo puede ser extendido a objetos que no estén cercanos físicamente, mientras exista algún canal de comunicación entre ellos. Este es el caso de los sistemas de objetos distribuidos, donde el mecanismo de comunicación es una invocación al método remoto (RMI). En estos sistemas, la ortogonalidad entre la funcionalidad del objeto y las comunicaciones entre los mismos esta garantizada por el uso de un middleware de comunicación. Por lo tanto este middleware es el responsable de que la comunicación entre objetos sea totalmente transparente, de manera que para los objetos sea imposible determinar si la invocación ha sido remota o local.

El modelo se representa en la figura 4. Allí se observan los actores que intervienen en la comunicación remota (usando la aproximación cliente-servidor) y el entorno al cual pertenecen (aplicación o middleware). Cualquier invocación al método debe tener lugar entre un *proxy* y un *esqueleto.* Desde el punto de vista del Cliente, un *proxy* es el propio objeto al cual se está invocando, puesto que provee exactamente la misma interfaz física. Desde el punto de vista del objeto servidor, éste no necesita preocuparse por la ubicación del objeto cliente que requiere sus servicios, puesto que se comunica con una interfaz del objeto cliente llamada *esqueleto.*

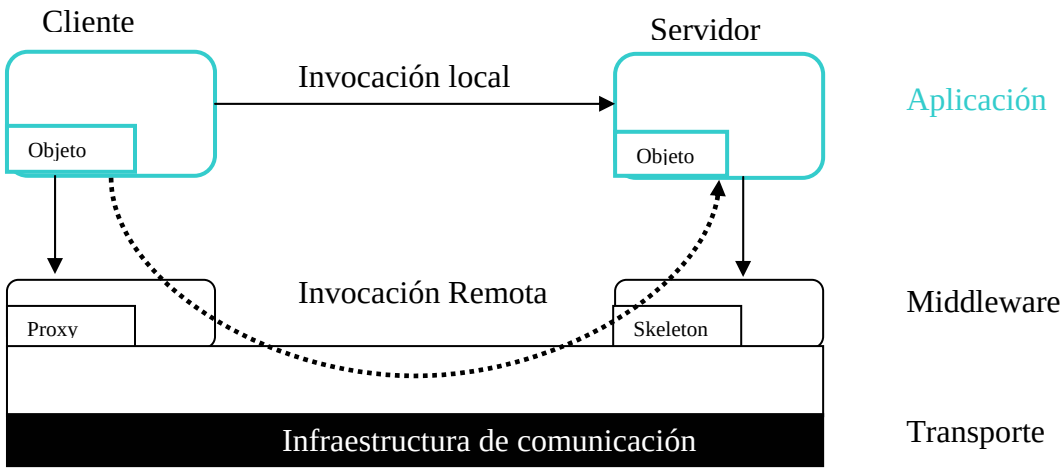

Figura 4

El *proxy* se comunica entonces por un lado con el cliente como si fuera el sirviente al cual le están requiriendo una operación o servicio y por otro lado con el *esqueleto* que representa al cliente, para trasladar las peticiones de éste al servidor como si fuera una petición local. El *esqueleto* por su parte se comunica por un lado con el servidor como si fuera el cliente que está realizando la petición y por el otro con el proxy para trasladar la respuesta del servidor al cliente.

La comunicación entre clientes y sirvientes se desarrolla de la siguiente manera:

1) Un cliente requiere una operación o tarea al *proxy* local del objeto

2) El *proxy* traduce el requerimiento en un conjunto de asignaciones de señales que dependerá de la arquitectura del canal de comunicación entre cliente y servidor.

3) El *esqueleto* recibe el pedido del *proxy* y las traduce de nuevo como pedido de operación al sirviente.

4) El sirviente completa con la ejecución de la operación.

Son funciones del *proxy* entonces:

1) codificar invocación y argumentos en un mensaje y

2) esperar respuesta y decodificar información de retorno,

mientras que son funciones del esqueleto:

1) Esperar invocación y decodificar método y argumentos

2)Invocar el método real

3) Codificar el valor de retorno en mensaje de respuesta.

Una de las principales ventajas del modelo descrito es que la mayor parte de la infraestructura necesaria para la comunicación puede ser generada automáticamente a partir de descripciones sencillas de la interfaz de los objetos, combinando de esta manera la flexibilidad de esta aproximación con una implementación eficiente en tiempo de ejecución.

Este mismo paradigma puede ser aplicado a sistemas heterogéneos Hardware/Software, como se describe en [72][73][74], puesto que cualquier componente hardware puede ser considerado un objeto, en donde la entidad del componente podría ser equivalente a la interfaz del objeto y donde las señales de interconexión implementan operaciones y parámetros de entrada/salida. El estado del objeto podría estar entonces almacenado en registros o memoria interna, encapsulado y hacerlo solo visible o disponible a través de señales en la entidad.

Continuando con nuestra aproximación, un *esqueleto* hardware podría consistir en la lógica que adapta la interfaz del objeto hardware (la entidad) al medio de comunicación, por ejemplo a la tecnología del bus donde se conectará. Esta adaptación consiste en traducir las transacciones de lectura y escritura en el bus en operaciones de activación de objetos. Estas traducciones significan adaptar el ancho de palabra del objeto al ancho del bus y convertir los datos serializados del bus en parámetros del objeto, más cualquier otra tarea que ayude a abstraer el flujo de comunicación de la funcionalidad del objeto.

Simétricamente un *proxy* deberá traducir operaciones de invocación a una secuencia de transacciones de lectura y escritura que deberán ser entregadas al *esqueleto* del servidor al cual están dirigidas. Esto implica que el proxy deberá incluir información para localizar el servidor, por ejemplo, la dirección en el bus.

Trasladando este modelo de objetos distribuidos a sistemas heterogéneos Hardware/Software se ofrece, desde el punto de vista del diseñador de aplicaciones una vista homogénea del sistema como una colección de objetos que

se comunican entre sí. Desde el punto de vista de la implementación, ofrece transparencia en la comunicación para cualquier clase de objeto, independientemente de cómo sea su implementación: en hardware o en software.

3.3.1 Objetos Dinámicamente Reconfigurables

La arquitectura de comunicación descrita en el apartado anterior también es aplicable a sistemas dinámicamente reconfigurables, donde cualquier objeto hardware estático puede ser convertido en un Objeto Dinámicamente Reconfigurable.

Esto implica dos modificaciones principales. Primero es necesario extender la funcionalidad de los *esqueletos*, de manera de que sean capaces de manejar la parada de la ejecución y la transferencia del estado del objeto a lugares de almacenamiento permanente, en el caso de la descarga de objetos, y recuperación de estado y reinicio de la ejecución cuando el objeto es reconfigurado. Segundo, debe ser definido un servicio de gestión de reconfiguración cuyas responsabilidades serán notificar las solicitudes de carga y descarga de objetos, conocer el estado actual de cada objeto y proveer almacenaje asignado para aquellos objetos que necesiten guardar su estado.

3.4 El proceso de reconfiguración dinámica

3.3.1 Reconfiguración parcial dinámica

Algunas FPGAs presentan actualmente la posibilidad de reconfiguración de solo una parte de su estructura interna. Esta reconfiguración parcial es posible realizarla de forma estática de la misma manera que se realiza la reconfiguración total, o sea deteniendo la FPGA, o en forma dinámica mientras el dispositivo está funcionando. Esta última característica de reconfiguración de sólo una parte del circuito mientras otras partes continúan funcionando, permite al diseñador considerar nuevas perspectivas para el diseño de Sistemas Integrados en las que

se puede tomar ventaja del uso de estas zonas de la FPGA que pueden ser remodeladas o reconfiguradas en tiempo de ejecución.

3.3.2 Problemática actual del diseño de sistemas dinámicamente reconfigurables

La capacidad de reconfiguración dinámica que ofrecen las FPGAs hoy en día, abre novedosas posibilidades de diseño tales como la factibilidad de quitar o insertar componentes u objetos hardware en nuestro diseño, mientras parte del mismo sigue funcionando. Esto significa que es posible modificar y adaptar nuestro diseño a nuevas aplicaciones insertando nuevos objetos hardware mientras el sistema está en ejecución, sin necesidad de detener los objetos que se estén ejecutando excepto aquellos que vayan a ser modificados o reemplazados.

Esto permite una flexibilidad de diseño sin precedentes, en donde se conjugan varios factores: ahorro de área, de potencia consumida, reducción del coste y tiempo de diseño, posibilidad de adaptación de diseño a futuras versiones sin necesidad de rediseño total del nuevo sistema, incorporación de nuevas funcionalidades en el mismo espacio en tiempo de ejecución de manera de cumplir con los requerimientos variables de las nuevas aplicaciones, etc.

Un sistema hardware reconfigurable dinámicamente, en general está formado por dos partes, una estática y otra dinámica. Las dos son creadas en tiempo de diseño, pero solo la parte dinámica se puede modificar en tiempo de ejecución sin necesidad de detener el sistema completo. La parte estática está compuesta de objetos hardware que serán indispensables para el funcionamiento del sistema, y por aquellos componentes que son requeridos de manera casi continua por el mismo. La parte dinámica en cambio, estará formada por aquellos objetos hardware que debido a su poco uso no justifican estar incluidos en la parte estática del sistema ocupando recursos de manera poco eficiente, sino que compartirán área con otros objetos dinámicamente reconfigurables.

Actualmente la reconfiguración dinámica se lleva a cabo de manera casi artesanal, altamente dependiente de la tecnología a utilizar [83]. Esta capacidad no

se presenta en todas las FPGA existentes en el mercado actual, pero no demorará en llegar tal tecnología a aplicarse de forma generalizada.

En términos de hardware reconfigurable específicamente, la reconfiguración puede ser realizada off-line, con la detención del sistema completo (reconfiguración estática) o en tiempo de ejecución (reconfiguración dinámica).

Consideraremos dos escenarios para la reconfiguración dinámica: el primero realizado en tiempo de compilación, donde los módulos a reconfigurar son definidos y diseñados junto con el sistema completo. Para el ahorro de área, estos módulos compartirán un espacio dentro de la FPGA y serán agregados o removidos en el diseño de acuerdo a las necesidades de las aplicaciones a ejecutar. La ubicación del area reconfigurable dentro del diseño y de los objetos reconfigurables dentro del area reconfigurable es realizada en tiempo de compilación usando herramientas EDA provistas por los fabricantes de los dispositivos programables. Estas herramientas permiten la definición del área dinámicamente reconfigurable en tiempo de diseño teniendo en cuenta los módulos a ser instanciados dentro[84], y teniendo en cuenta además las restricciones del sistema. Cada módulo a reconfigurar es diseñado, sintetizado y luego verificado en el diseño total, por separado, para asegurar su funcionamiento y el cumplimiento de las restricciones del sistema. [84]. El bitstream de reconfiguración correspondiente a cada módulo es almacenado en memoria y de allí es cargado en la FPGA cuando se realiza la reconfiguración.

El segundo escenario es la reconfiguración de nuevos objetos hardware que no fueron previstos en tiempo de compilación, sino que corresponden a adaptaciones del diseño original a nuevas funcionalidades del sistema. Esto es, objetos que se agregan al diseño original para aumentar sus prestaciones y logrando una notable disminución del costo de diseño de un sistema, por ejemplo en sistemas móviles o multimediales, con aplicaciones de video digital que presentan una carga de trabajo muy variada, entre otras.

En este segundo escenario, se pueden considerar dos situaciones: reemplazo de objetos por versiones mas actualizadas, o, por otra parte agregado de nuevos objetos con nuevas funcionalidades. En cualquier caso es necesario soporte on-line para la ubicación de los objetos y soporte de scheduling. La

ubicación de los objetos nuevos y su enrutamiento dentro del diseño de manera on-line es objeto de estudio en muchos trabajos de investigación[80]. El agregado de nuevos objetos con nuevas funcionalidades implica un ordenamiento en la reconfiguración de dichos objetos ya que existe un mayor número de objetos pendientes de reconfiguración con distintas características de parada o de activación. Al respecto existen diferentes aportes referidos al scheduling de tareas en hardware reconfigurable dinámicamente.[70],[79].

En ambos escenarios descritos más arriba es necesario que los objetos hardware Dinámicamente Reconfigurables cumplan con las restricciones impuestas por el diseño al momento de ser implementados (timing, área, etc) y que no violen ningún principio de integridad del mismo al momento de la detención del objeto o al momento de la reconfiguración.

Para ello es necesario garantizar la integridad estructural del diseño y la integridad del objeto reconfigurable durante la reconfiguración.

Esta garantía se obtiene logrando una separación entre las zonas donde serán implementados los objetos dinámicamente reconfigurables y con la zona del diseño estático. Este aislamiento es necesario para evitar que existan interrupciones o variaciones en las señales que corresponden a la parte estática, lo que produciría fallos en el circuito implementado en el momento del desacople de los módulos, al retirar o al acoplar módulos nuevos. Para ello se recurre a macros desarrollados en hardware que permiten acoplar los módulos dinámicamente reconfigurables al resto del circuito.

En las FPGA de Xilinx, estos macros se denominan BUSMACROS y son Macros pre-ubicados y pre-ruteados que fijan las conexiones entre las regiones parcialmente reconfigurables y el diseño base, haciendo que estas regiones sean pin compatibles. Por estos busmacros deben pasar todas las señales excepto las globales como el clock y reset. Figura 5

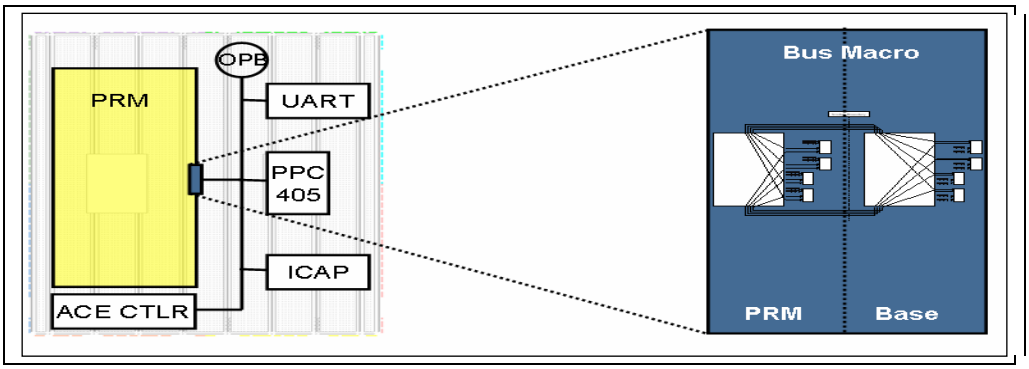

Fig 5

En esta figura se observa los elementos que componen el diseño base, y su conexión con el Módulo Parcialmente Reconfigurable (PRM) mediante los Busmacros descritos.

Esta compatibilidad necesaria a nivel de pines exige que los módulos a reconfigurar posean la misma interfaz, y que esta interfaz además va a depender del resto del sistema al cual está conectado, por lo que los módulos a ser reconfigurados en una misma región deben todos poseer los mismos puertos de conexión. Esto limita mucho el espacio de diseño del sistema completo.

Además, es necesario que los objetos hardware dinámicamente reconfigurables puedan ser detenidos en algún lugar de su proceso de ejecución sin que pierdan consistencia y que, además, su estado sea transferido a memoria para su posterior utilización en una nueva reconfiguración, mientras el resto del sistema sigue funcionando sin alteraciones.

Capítulo 4

Desarrollo

4.1 Las bases de la Metodología

Para poder resolver los problemas que han sido detallados en el capítulo anterior al hacer uso de la capacidad de reconfiguración parcial que poseen las FPGA, este trabajo propone una nueva metodología de diseño de sistemas dinámicamente reconfigurables

La nueva metodología propuesta permitirá la generación de objetos, su integración al sistema y su implementación de una manera que sea completamente transparente para el usuario, independientemente de si los objetos son objetos hardware estáticos, reconfigurables u objetos software.

Esta metodología permitirá la integración de nuevos componentes diseñados por terceros en nuestro sistema reconfigurable lo que aumentará considerablemente la capacidad de desarrollo de sistemas y la disminución del tiempo de diseño.

Esta metodología esta basada en sistemas orientados a objetos, que se apoya en una arquitectura de reconfigurabilidad dinámica llamada Reconfiguration Manager (RM) que puede ser descrita mediante un modelo de estructura de capas lógicas, la cuál será responsable de todas las tareas necesarias para la Reconfiguración Dinámica del Sistema. (Figura 6)

Esta arquitectura utiliza un sistema de comunicación para la integración de componentes previamente desarrollados, en sistemas que permiten la reconfiguración parcial en tiempo de ejecución, adaptando para ello funcionalidades del sistema de comunicación basado en el paradigma de objetos distribuidos para incluir la reconfigurabilidad parcial como servicio del sistema.

4.1.1 Reconfiguration Manager

Desde el punto de vista del sistema el RM deberá ser capaz de:

- Determinar la disponibilidad de recursos para la reconfiguración dinámica(por

ejemplo: área necesaria, memoria, ubicación de las diferentes zonas parcialmente reconfigurables, etc)

- Determinar la ubicación óptima dentro de la FPGA del Objeto hardware DR para cumplir con las restricciones de timing o performance requeridas por el diseño.
- Determinar el orden de los Objetos hardware DR a ser implementados, y planificar la cola de implementación según las aplicaciones a ejecutar.
- Determinar si el objeto a ser activado necesita otras funcionalidades del sistema.
- Determinar cuándo cambiar un objeto cargado si hay fallos en el mismo por otro Objeto hardware DR u otro objeto Software.
- Determinar si un objeto ha podido ser cargado sin falla, registro de objetos defectuosos.
- Determinar la ubicación o localización del objeto a ser cargado (Memoria Flash, DDR, remoto, etc)
- Verificar si es necesario redundancia de objetos PR (duplicación de objetos, tolerancia a fallos)

Desde el punto de vista del objeto:

- Detener un objeto reconfigurable que este ejecutándose.
- Guardar el estado del mismo
- Guardar los valores intermedios del objeto
- Cargar un nuevo objeto hardware DR
- Recuperar el estado del objeto hardware DR, para continuar la ejecución de un objeto que haya sido detenido.
- Recuperar los valores intermedios del objeto reimplantado
- Proteger datos de objetos compartidos.

Desde el punto de vista de la Aplicación:

- Dar información acerca de la ubicación objeto hardware DR o de su versión en software.

- Responder ante un pedido de activación ya sea explícito o implícito.
- Evaluar si el tiempo de reconfiguración del Ob. hardware DR mas el correspondiente al de ejecución es mayor que el tiempo de ejecución del mismo pero en su versión software.
- Dar aviso de duplicación de objetos, si hubiera, para casos de tolerancia a fallos.
- Dar aviso de la disponibilidad del Ob. implementado.

4.1.2 Objetivos del Servicio de Reconfiguración:

El Servicio de Reconfiguración deberá cumplir con los siguientes objetivos

- Gestión transparente de la reconfiguración dinámica
- Gestión de la persistencia
- Ofrecer servicios de mayor nivel a las aplicaciones/sistema operativo, de manera que éstos puedan realizar una planificación/gestión de alto nivel.

4.2 El soporte de la Metodología

Para llevar a cabo los objetivos descritos e implementar la metodología propuesta se propone una arquitectura basada en 4 capas lógicas. Cada capa brinda servicios a la capa superior, basándose en los servicios de la capa inferior (figura 6). Así, en el nivel inferior se encuentran los objetos dinámicamente reconfigurables, que además de la funcionalidad hardware propiamente dicha, disponen de una interfaz muy simple para su parada y activación, su reconfiguración, o la extracción e inserción del estado. Sobre esta capa se encuentra la capa de activación, responsable de gestionar correctamente la secuencia de reconfiguración de un objeto, así como de garantizar su persistencia, en caso de que sea necesario. En un tercer nivel se encuentra la capa de planificación de la reconfigurabilidad, que proporciona los servicios de alto nivel, tales como la gestión de los recursos dinámicamente reconfigurables, la

localización de los objetos dinámicamente reconfigurables o la planificación de la reconfiguración. Finalmente, la última capa corresponde al nivel de aplicación, o de sistema operativo si lo hubiera, que puede utilizar los servicios de reconfiguración de forma implícita o explícita.

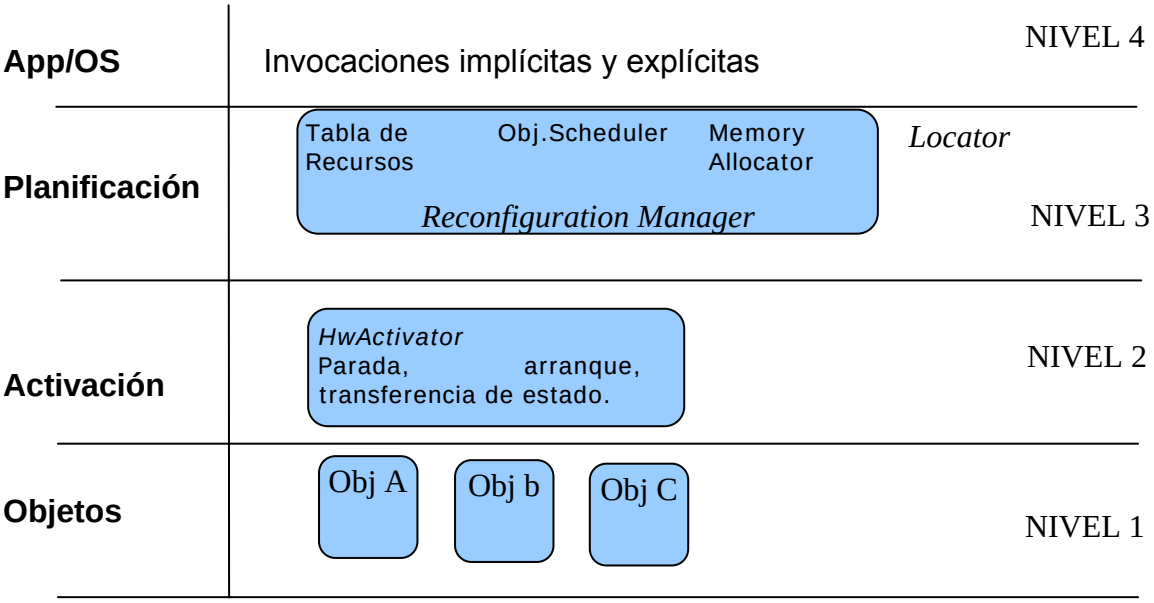

FIG (6)

En los siguientes apartados se describe con mayor nivel de detalle la funcionalidad de cada una de estas capas y las interfaces que proporcionan a los siguientes niveles.

Para una visión de la arquitectura propuesta, la siguiente figura muestra un diagrama de dominio en UML, del modelo propuesto, con las relaciones entre los dominios y las clases existentes dentro de los dominios.

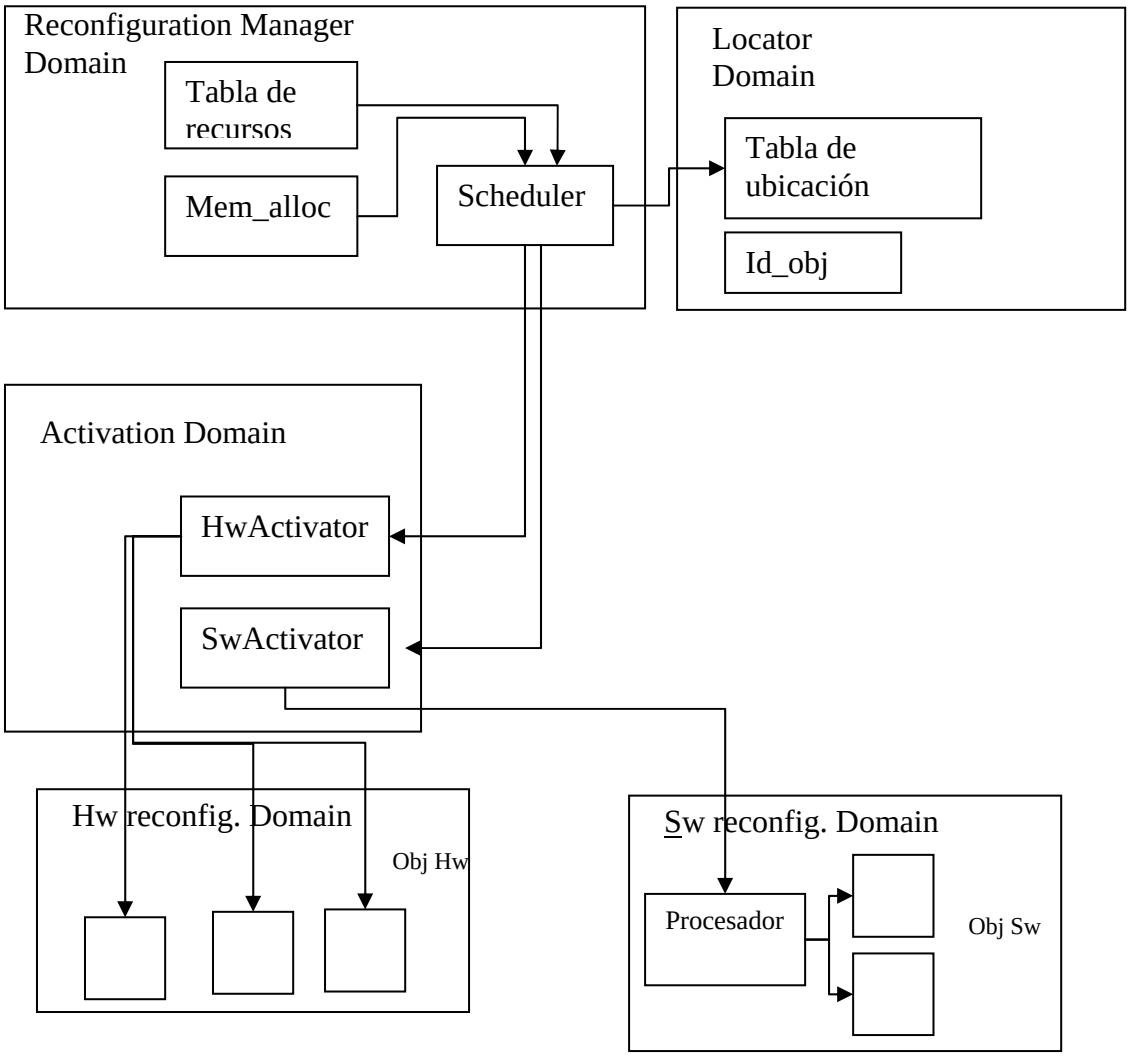

Diagrama de Dominio del Servicio de Reconfiguración

4.2.1 Nivel 1: Objetos dinámicamente reconfigurables

Desde el punto de vista de la funcionalidad que proporciona, un objeto dinámicamente reconfigurable no se diferencia en nada de otro estático. Es decir, ambos ofrecen exactamente la misma interfaz funcional al usuario, que de hecho puede ignorar el tipo de objeto con el que está tratando. La principal diferencia radica en que éstos objetos disponen de la capacidad de ser creados y destruidos

en tiempo de ejecución, de forma similar a cómo sucede con los objetos software.

La reconfiguración es un proceso complejo, que involucra no solo al objeto que está siendo reconfigurado, sino que requiere algún tipo de gestión de los tiempos y los recursos involucrados. Una forma de acotar esta complejidad consiste en establecer claramente las responsabilidades de cada elemento que interviene. En el caso de los objetos podríamos indicar las dos siguientes:

1. garantizar la integridad estructural del sistema y la coherencia del estado del objeto durante la creación y la destrucción del objeto, y atender a las peticiones que desencadenarán esos procesos.
2. gestionar el almacenamiento y la recuperación del estado, como soporte básico de los mecanismos de persistencia.

Uno de los principales problemas del proceso de reconfiguración es que éste afecta a la estructura del diseño, que se modifica dinámicamente. Para que la reconfiguración no afecte a los elementos del sistema que no están siendo modificados, y por lo tanto se mantenga la integridad estructural de éste, es necesario garantizar que el área reconfigurable está convenientemente aislada del resto de componentes. Tal y como se describe en apartados anteriores, esto tiene repercusiones en el modelo de programación utilizado, puesto que la manera habitual de garantizar el aislamiento es el de mantener una interfaz constante. Tanto en el caso de las tareas como en el de los objetos, sus interfaces varían de una a otra entidad. Difícilmente se encuentran en el diseño dos tareas o dos objetos con exactamente la misma interfaz. Esto significa que a la hora de ser convertidos en elementos dinámicos, su implementación requerirá adaptar la interfaz lógica del elemento a la interfaz física (invariable).

La ventaja del modelo de objetos distribuidos es que estos adaptadores ya existen (proxies y esqueletos), son sistemáticos y son generados automáticamente durante el proceso de síntesis de la infraestructura de comunicaciones. En realidad es que la entidad reconfigurable no es solamente el objeto propiamente dicho, sino que incluye sus adaptadores, que sin el objeto no tienen sentido de existir. Estos adaptadores, a diferencia del objeto, cuya interfaz varía de un tipo a otro, siempre tienen una misma interfaz, la del bus a la que se conectan. Tan sólo es necesario,

por lo tanto, fijar como interfaz del área dinámicamente reconfigurable la del protocolo de transporte utilizado. Así, cualquier core (entendido como el objeto más sus adaptadores), incluso los generados posteriormente al despliegue del sistema, puede ser reconfigurado en cualquier área reconfigurable, con la única limitación de que no exceda el tamaño del área.(Figura 7)

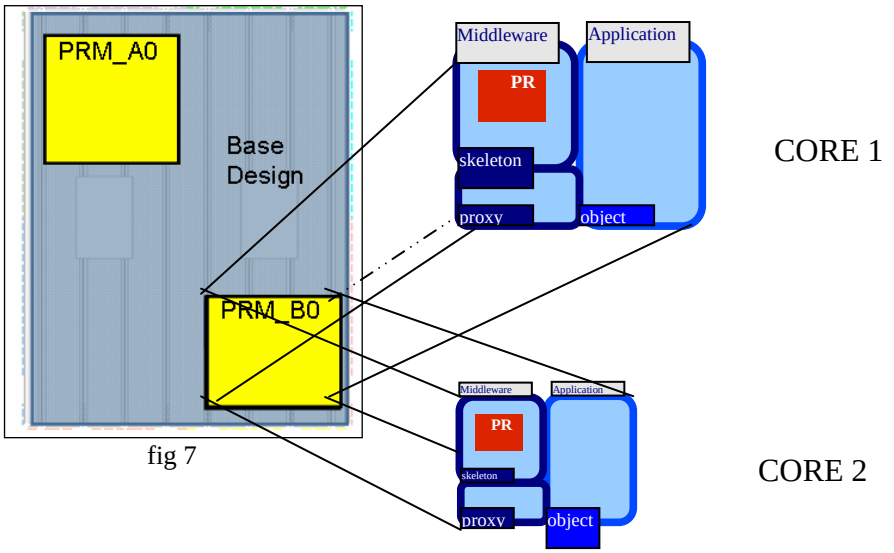

fig 7

En el caso de las tareas, la solución también pasa por adaptar las interfaces, pero puesto que no existe una semántica sobre cómo debe hacerse esta adaptación, este proceso depende totalmente del diseñador, que además debe prever de antemano qué tareas deben compartir una misma área reconfigurable.

4,2,1,1 Persistencia del Objeto

La persistencia de un objeto es un sinónimo del almacenamiento y recuperación del estado de un objeto. Como el estado es conocido, es decir se especifica explícitamente desde las primeras etapas del proceso de diseño, este proceso puede ser completamente automatizado. Tan sólo es necesario disponer de acceso desde el objeto a algún tipo de memoria externa, y definir un modo de transferir en serie el valor de los atributos hacia o desde ella. Todo ello sin la intervención del diseñador, que sólo debe definir el número y el tipo de atributos del objeto.

Por otro lado, el uso del modelo de objetos implica una separación entre el estado del objeto y las operaciones. Esto facilita enormemente gestión de la reconfiguración, puesto que puede garantizarse la coherencia del estado simplemente verificando que no existe ninguna operación en ejecución en un determinado momento.

4,2,1,2 Esqueletos Dinámicos

Las tareas descritas anteriormente no son estrictamente responsabilidad del objeto dinámicamente reconfigurable. El objeto propiamente dicho no debería diferenciarse en nada de su misma versión estática. En realidad son los proxies y esqueletos los que extienden su funcionalidad para asumir la gestión de la reconfiguración del objeto. Esto tiene dos consecuencias: 1) al no modificarse el objeto, tanto el proceso de diseño de éste, como el uso de sus operaciones desde otros clientes no se ve afectado (es transparente). 2) Ambos elementos se generan de manera automática, por lo que la gestión de la reconfiguración es también transparente.

Los esqueletos de objetos dinámicamente reconfigurables incluyen una máquina de estados adicional, encargada de diferenciar entre las situaciones en las que se puede encontrar un objeto reconfigurable.(Figura 8)

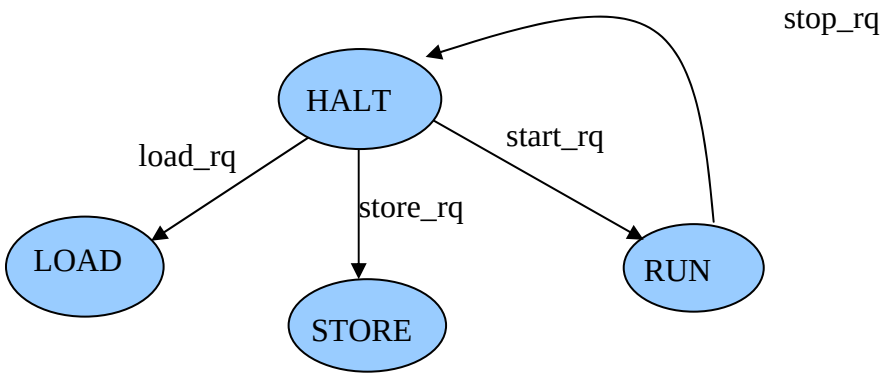

Fig. 8

- HALT: se da cuando el objeto dinámicamente reconfigurable no está activo, es decir, no atiende invocaciones, pero se encuentra cargado en el sistema y listo para comenzar la ejecución. Éste es el estado inicial cuándo se produce la carga del objeto.
- LOAD: la transición a este estado se produce cuando se le solicita al objeto la recarga de su estado desde cierta posición de memoria.
- STORE: similar al anterior, pero en este caso se transfiere el estado del objeto hacia la memoria.
- RUN: cuando el objeto se encuentra activo, y atiende cualquier tipo de invocación.

Por otro lado, estos esqueletos incluyen una interfaz adicional (además de la del bus del sistema mediante la cual atienden las invocaciones), a través de la cual se ofrecen los servicios de reconfiguración de la capa de nivel 1 hacia la de nivel 2. En concreto se implementan siguientes operaciones (figura 9) que son solicitudes entregadas por el HwActivator:

- load_rq(*state): petición de carga del estado del objeto desde la posición de memoria indicada por *state.
- store_rq(*state): petición de almacenamiento del estado del objeto en la posición de memoria indicada por *state.
- start_rq(): activación del objeto.
- stop_rq(): parada del objeto.

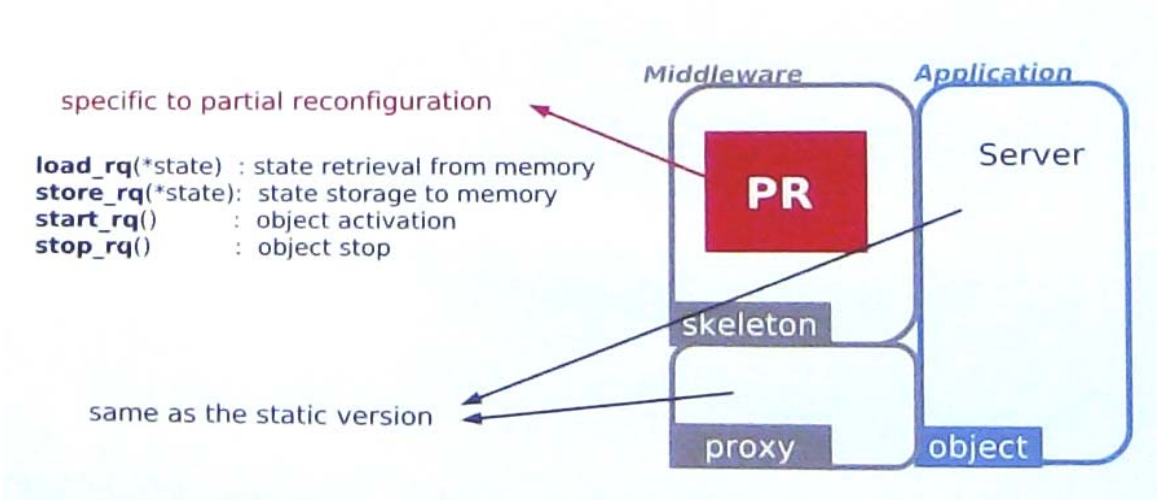

Fig. 9

Cada una de estas operaciones desencadena algún cambio de estados en la máquina descrita en la figura 8. Las operaciones de load y store sólo son atendidas cuando el objeto se encuentra parado (HALT), para garantizar la coherencia del estado. De igual forma, start_rq requiere que el objeto esté parado para activar la recepción de invocaciones. A diferencia de las anteriores, la operación de parada no tiene efecto inmediato sobre la máquina. Solamente es atendida cuando el objeto se encuentra en ejecución (RUN), pero antes de que tenga lugar la transición al estado HALT debe esperarse a la finalización de todos los métodos que se encuentren en ese momento en ejecución. Una vez finalizados, el estado ya es coherente, y por tanto el objeto puede ser detenido.

4.2.2 Nivel 2: Capa de activación

Esta capa se encarga del proceso de reconfiguración dinámica parcial del sistema y de solicitar la modificación del estado de un Objeto Dinámicamente Reconfigurable que permita su parada, activación o su reconfiguración, de acuerdo a las peticiones recibidas de la capa superior inmediata.

Esta capa de Activación actúa por pedido de la capa 3 o capa de Planificación, la cual envía una solicitud de reconfiguración, parada o creación de Objeto Dinámicamente Reconfigurable. Junto con esta solicitud se entrega la identificación del objeto a reconfigurar, su ubicación final y el lugar de memoria

donde guardar o desde donde cargar su estado.

A partir de estas solicitudes de servicio, la capa de activación solicita los servicios de parada(**stop_rq**), almacenamiento de estado (**store_rq**), carga de estado (**load_rq**) o activación (**start_rq**) del objeto según sea el caso, a la capa de Objetos.

Para ello se utiliza un componente, implementado en hardware llamado **HwActivator,** que entrega las solicitudes necesarias a la capa inferior mediante señales conectadas a los objetos, a través de sus esqueletos (Figura 9).

Por otra parte, un objeto software en ejecución también es factible de ser reemplazado por un objeto hardware y viceversa. Para poder realizar tales servicios es necesario otro objeto en software que realice las mismas tareas que lleva a cabo el HwActivator con los objetos DR, de la capa 1. Este objeto se llama SwActivator.

El HwActivator consta de una máquina de estados que se encarga del proceso de activación, parada, o del manejo de la persistencia de los objetos según sea la situación en que se encuentren los mismos, y aparte del bus del sistema de donde el activador recibe las solicitudes, está conectado a través de un bus dedicado a cada una de las áreas parcialmente reconfigurables disponibles en el dispositivo (figura 10). Con esto es posible liberar el bus del sistema mientras un pedido de reconfiguración esta siendo procesado por el activador.

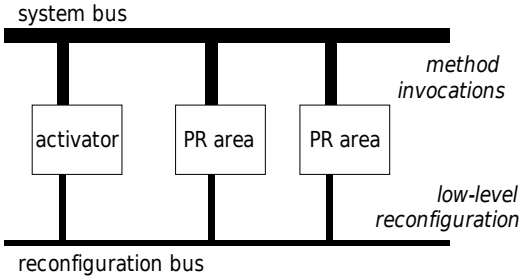

fig 10

Para obtener los servicios de la capa de Activación, la capa de Planificación envía 3 tipos de solicitudes:

Freeze_rq(Object_id, Store[], stop-run),

reconfig_rq(Object_id, Area reconf, Store[], bitstream_location),

create_rq(Object_id, Area_reconf, bitstream_location)

La capa de Activación entrega los siguientes servicios a la capa de Planificación dependiendo del tipo de solicitud entregada por ésta.

a) *Servicio de reconfiguración:* Si un objeto necesita ser removido, se solicita su remoción con *freeze_rq*. El HwActivator solicita entonces a la capa de objetos la parada del objeto correspondiente y el almacenamiento de su estado en el rango dado por Store[].

Luego de removido el objeto, la solicitud *reconfig_rq* envía al HwActivator la dirección del área reconfigurable en donde se reconfigurará el nuevo objeto identificado por Object_id, y la dirección de ubicación del bitstream correspondiente al nuevo objeto.

El HwActivator copia entonces el bitstream en el bloque RAM dado por el parámetro Area_reconf. Una vez copiado carga el estado almacenado en la dirección dada por Store[].

b) Servicio de migración software/hardware o hardware/software: Si un objeto software está ejecutándose y se requiere su reemplazo por un objeto hardware que realice lo mismo que el anterior, la capa de Planificación envía una solicitud de *Freeze_rq* al SwActivator, el cual detiene las invocaciones al objeto a reemplazar, y envía a su vez un pedido de *reconfig_rq* o *create_rq* al HwActivator para reconfigurar o crear un nuevo objeto, según sea el caso que reemplace al objeto Software inicial.

c) Servicio de creación de objeto: Si se desea la implementación por primera vez de un objeto reconfigurable, la capa de Planificación, a través Scheduler, envía la solicitud *create_rq* con los parámetros del lugar de memoria donde se encuentra el bitstream correspondiente y la ubicación del área donde colocar el ODR. Luego de realizar cualquier servicio, la capa envía una señal indicando la finalización del mismo con éxito, o un código de error al Scheduler de la capa de Planificación.

Para poder llevar a cabo los servicios requeridos por la capa 3, el HwActivator realiza una serie de solicitudes a la capa inferior de objetos dinámicos.

Estas solicitudes varían de acuerdo a la situación en que se encuentre el objeto, como por ejemplo: para solicitar el almacenamiento del estado del objeto es necesario que el mismo se encuentre en el estado Halt, para lo cual el HwActivator enviará la solicitud de parada primero y luego la solicitud de store_rq(*state) con la dirección de memoria correspondiente para el almacenamiento del estado. Con estas solicitudes, la capa de nivel 1 implementará los servicios correspondientes. (Figura 11)

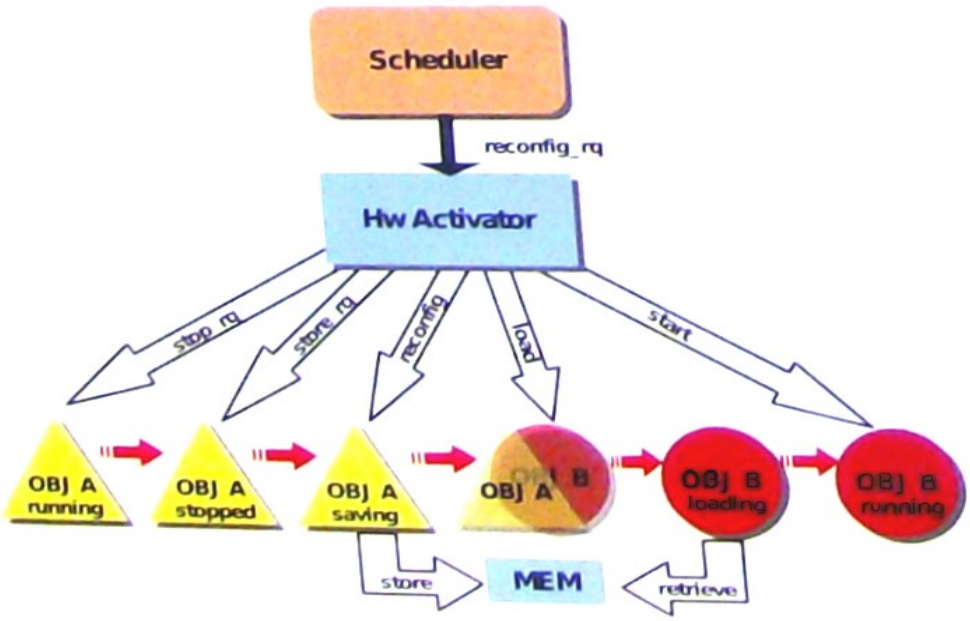

Fig. 11

La contraparte software del HwActivator llamada SwActivator posee funciones que son equivalentes a las del HwActivator. Este componente realiza las mismas tareas con los objetos software que el HwActivator con lo objetos hardware.

4.2.3 Nivel 3: Capa de planificación- Gerenciamiento de Objetos Dinámicos

Esta capa está formada por dos componentes principales: El Reconfiguration Manager y el Locator. El primero utiliza los servicios provistos por el Hwactivator y por el Swactivator y toma en cuenta el estado actual del sistema y

las características de los objetos estáticos para proveer scheduling a nivel de sistema de los objetos dinámicos. Este scheduling involucra la creación de nuevos objetos hardware o software, reconfiguración de objetos hardware con recuperación de estado o la migración de objetos hardware-software o viceversa. El Locator conoce la ubicación y el estado de todos los objetos dinámicamente reconfigurables o migrables del sistema.

El Reconfiguration Manager esta compuesto por tres elementos: La Tabla de Recursos, el Allocator de memoria y el Scheduler. La tabla de recursos captura las características de cada tipo de objeto dinámico (tabla 1). Cuando un nuevo tipo de objeto es definido, debe insertarse una nueva entrada en la tabla de recursos.

obj_type	: object type identifier
obj_size	: size (area) and shape of the object
state_size	: size in words of the object state
bitstream_loc	: size in KB of the partial bitstream

Tabla 1. Resource table fields

El Allocator provee bloques contiguos de memoria del tamaño requerido, de la memoria disponible. Estos bloques son usados para el almacenamiento de estado o de los bitstream de reconfiguración.

Por último, el Scheduler es el encargado de tomar las decisiones concernientes a la ubicación de nuevos objetos, la migración entre implementaciones software y hardware, o la reconfiguración de objetos existentes. Esta planificación que realiza el scheduler está basada en tres fuentes de información: 1) Las características estáticas de los objetos en la tabla de recursos, 2) El estado actual de los objetos dinámicos (almacenados en el locator) y 3) un modelo del comportamiento, en terminos de área y tiempos del servicio de reconfiguración implementado en una cierta tecnología.

En el presente trabajo de maestría no seguimos ninguna política concreta de scheduling puesto que está fuera del alcance de nuestro trabajo. En este trabajo se trata de proveer los elementos para poder usar cualquiera de las políticas que

se proponen en la literatura. Sin embargo, para nuestro trabajo hemos restringido nuestra implementación a un modelo 2D pre-particionado donde las áreas PR ya están definidas [81]. Esto simplifica no solamente la ubicación de los objetos reconfigurables sino tambien la caracterización del servicio (tercera fuente de información). No obstante, esto puede significar un desperdicio de área.

El servicio de reconfiguración puede ser aplicado también a otras clases de modelo de partición, tales como los descritos en [78][79] donde el área reconfigurable no se encuentra predefinida y la ubicación del objeto dinámico se realiza en tiempo de ejecución. Como consecuencia, el modelo de servicio podría ser actualizado para reflejar la arquitectura física subyacente.

El componente Locator posee la información de la situación actual de todos los objetos dinámicos del sistema. Esta información es almacenada en una tabla de ubicación (tabla 2) y es usada por el Scheduler como se mencionó anteriormente. Además de esto, el Locator posee otra responsabilidad que es la de proveer la ubicación real de los objetos dinámicos a cualquier otro objeto que necesite invocarlo. De esta manera los objetos clientes pueden acceder de manera transparente a objetos dinámicos. Ilustraremos lo expresado con el siguiente ejemplo. Supongamos que un objeto estático en un sistema necesita invocar un método de un objeto dinámico que nunca ha sido utilizado. Aquí se presentan dos posibilidades en la creación del objeto, hacerlo de manera explícita o implícita. En el primer caso, la aplicación debería requerir la creación explicita del objeto al Reconfiguration Manager. El Reconfiguration Manager debería entonces chequear la tabla de recursos, consultar al Allocator por el espacio de memoria del estado, y decidir qué ubicación, dependiendo de la disponibilidad de área reconfigurable y una política concreta de scheduling. Luego el Locator debería ser notificado de manera de que el nuevo estado del objeto se actualizado convenientemente. Ahora el objeto estático puede enviar la invocación pero no directamente ya que el cliente no conoce el lugar donde fue ubicado el objeto invocado. Para solucionar esto se recurre a una invocación indirecta, donde el locator es interrogado primero acerca de la dirección real del objeto y en una segunda etapa el objeto es contactado directamente.

Los *proxies* para los objetos dinámicos deberían ser siempre proxies indirectos. Todos ellos saben como contactar al locator, pero ellos también pueden almacenar en buffers la respuesta, de tal manera que la indirección realmente suceda cuando el objeto haya sido movido.

La única diferencia en una invocación implícita es que los objetos estáticos realiza la invocación sin previo conocimiento del estado de un objeto. El proxy dinámico en el objeto estático se pone en contacto con el Locator, quien detecta que el objeto no ha sido desplegado, entonces un pedido de creación es enviado al Reconfiguration Manager. Una vez creado el objeto el Locator devuelve la ubicación y la invocación, y sigue con los pasos ya descritos.

obj_id	: object identifier
location	: object placement (PR area/Sw processor)
status	: running, stopped or frozen
implementation	: hardware or software
storage	: memory space assigned for persistent storage

Tabla 2. Location table fields

Teniendo en cuenta la implementación, el Locator podría ser un objeto hardware. Por su parte, el Reconfiguration Manager podría depender de la política de scheduling elegida y de la inteligencia del Memory Allocator.

4.2.4 Nivel 4: Capa de aplicación

Esta capa ofrece a las distintas aplicaciones o al sistema operativo el acceso a los servicios de reconfigurabilidad dinámica dado por nuestro sistema, mediante invocaciones implícitas o explícitas, pero no será objeto de nuestro estudio.

4.3 Metodología de Diseño

Una vez desarrollados los elementos descritos, aplicamos la metodología de diseño objeto de esta tesis, propuesta para mejorar de manera sustancial el proceso de diseño y reducir el tempo del mismo, de sistemas reconfigurables, Esta metodología nos permite además incorporar componentes u objetos desarrollados por terceros, con un proceso de adaptación al canal de comunicación que se realiza en forma totalmente automatica.

Al proceso de diseño lo dividimos en tres etapas principales : En la primera etapa se definen las especificaciones. Aca se incluyen los elementos necesarios para el despliegue de la arquitectura (architectural deployment) que son: los tipos de objetos, sus relaciones, recursos y criterios de mapeo de objetos. La segunda etapa de diseño comprende la generación del middleware para vincular estos objetos con el canal de comunicación y por ultimo la tercera etapa que comprende implementación del sistema. Ver figura 12

4.3.1 Especificaciones

Si nos referimos a la metodología clásica de diseño de aplicaciones orientadas a objetos, los primeros pasos del proceso de diseño no difieren radicalmente de la misma. Así pues, las primeras especificaciones comprenden definir las clases (tipos de objetos que forman parte de la aplicación), así como las relaciones entre ellos. Estas dos descripciones, clases y relaciones, no incluyen ninguna información referente a la arquitectura sobre la que debe ser desplegada la aplicación, sino que son puramente funcionales.

Estos dos elementos, sumados a las especificaciones de recursos disponibles y al criterio de mapeo del objeto, nos permite tener un despliegue de arquitectura inicial que es incremental y que es independiente de la funcionalidad del objeto y de sus relaciones.

En las especificaciones de recursos se describen los recursos de la arquitectura, en terminos de: a) recursos ya implementados de fabrica (por

ejemplo Core de PowerPC), b) de áreas estáticas y reconfigurables, y c) cuál es el criterio para ubicar cada objeto en alguno de esos recursos, (criterio de mapeo).

El criterio de mapeo especifica que tipo de implementación llevan los mismos: si son Objetos Software, Objetos Hardware u Objetos Dinámicamente Reconfigurables. Por lo tanto, dependiendo de sus caracteristicas, pueden mapearse sobre los procesadores software utilizados como base de la arquitectura, o pueden convertirse en cores hardware, e incluso pueden ser implementados como cores dinámicamente reconfigurables.

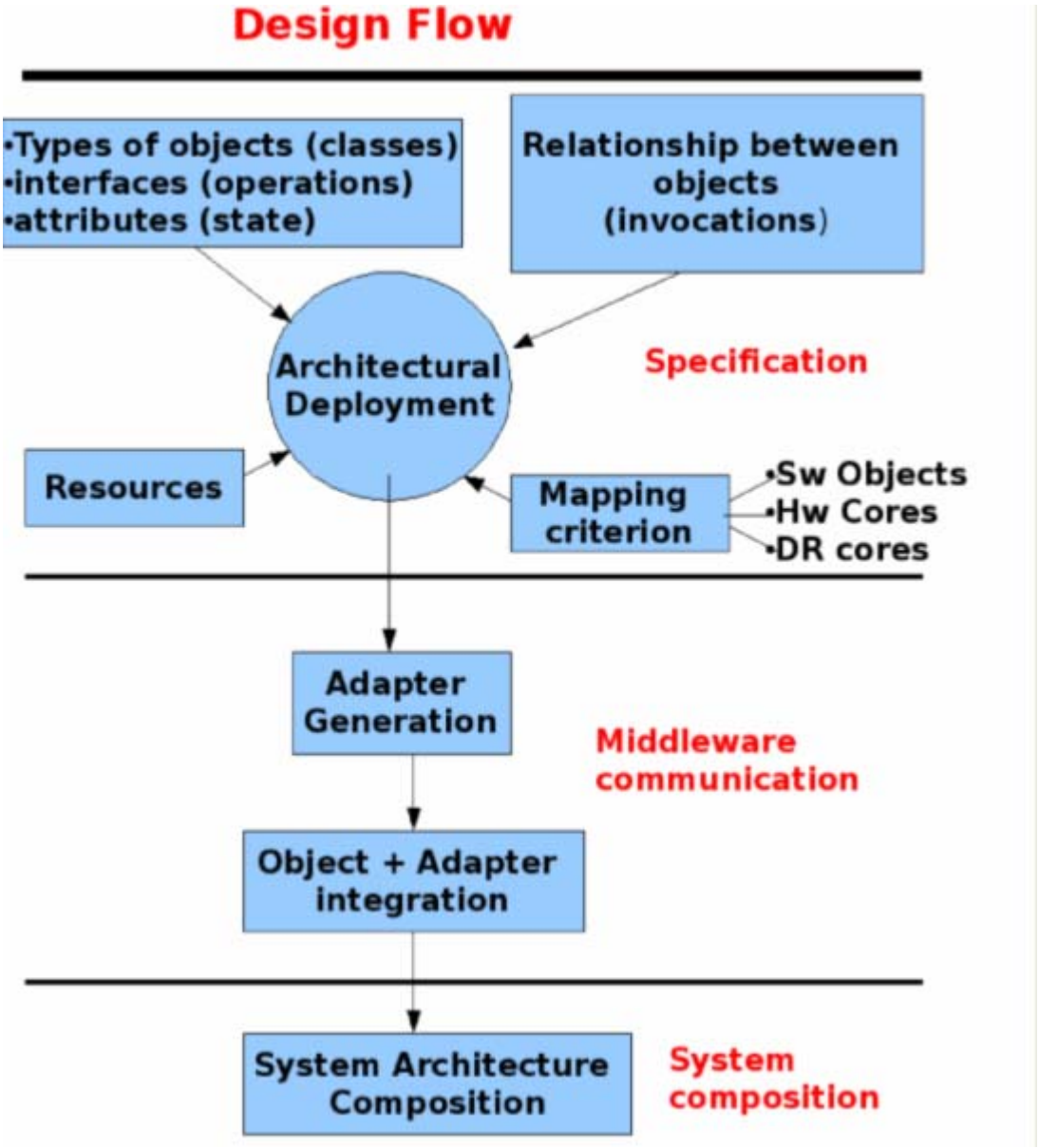

Fig. 12. Flujo de Diseño

4.3.2 Generación de Middleware

A partir de las especificaciones descritas, en una segunda etapa del proceso se genera de forma automática el middleware de comunicaciones. La generación tiene en realidad dos partes:

1) por un lado se analizan los recursos existentes, y se define la arquitectura de comunicaciones de bajo nivel, es decir, aquella que corresponde a los buses a los que deben conectarse los componentes, los bridges que permiten el paso de uno a otro, y demás elementos de interconexión física. Esta arquitectura se genera a la medida, en función de las relaciones entre los objetos, de manera que si, por ejemplo, un objeto que se ejecuta sobre un PowerPC debe comunicarse con un core en un bus OPB, automáticamente se definirá el bus PLB del PowerPC, el bus OPB del core y el bridge que los interconecta.

2) por otro lado se genera el middleware propiamente dicho. Esta generación depende, de nuevo, de las características concretas del diseño. En el caso de los adaptadores (proxies y esqueletos) de los objetos, si un objeto es, por ejemplo, invocado desde un objeto software y otro objeto hardware, se generará tanto un proxy software como otro hardware. En el caso de los objetos dinámicos, los esqueletos generados no solamente incluyen la lógica de gestión de las invocaciones, sino que se añade la correspondiente a la gestión del proceso de reconfiguración. Durante esta fase se incluyen también el resto de servicios básicos del middleware, tales como el HwActivator, los adaptadores de objetos remotos, etc.

Luego, una vez analizados los recursos y haberse generado el middleware correspondiente se procede a la integración del objeto y su adaptador. Este proceso se realiza de manera automática y totalmente transparente al usuario.

4.3.3 Composición del Sistema

Finalmente, tanto la arquitectura del sistema como el middleware de comunicaciones se sintetizan para obtener la plataforma hardware final, en el caso de objetos hardware, y objetos Dinámicamente reconfigurables. Para el caso de los Objetos software, estos se compilan para obtener el código que debe ejecutarse sobre la plataforma.

Una de las ventajas de la metodología propuesta es que permite desacoplar el proceso de diseño y prototipado de la funcionalidad de la aplicación de una implementación arquitectural completa. Es perfectamente posible abordar, en primer lugar, el diseño desde el punto de vista puramente software y no distribuido, de manera que todos los objetos pueden ejecutarse en un sistema que sólo disponga de un procesador y la memoria necesaria.

En estas primeras etapas, en las que se trataría de depurar la funcionalidad y las relaciones entre los objetos, la generación del prototipo es casi inmediata, ya que simplemente requiere la compilación del código fuente. Es posible, además, depurar este código en un procesador y un entorno de desarrollo completamente distinto al que finalmente será utilizado en la implementación definitiva.

Una vez verificada la funcionalidad del diseño, puede pasarse al despliegue sobre una arquitectura concreta. Para ello simplemente se deberá modificar la especificación de despliegue, y seguir el flujo descrito en los párrafos anteriores. El despliegue, por lo tanto, puede realizarse de manera totalmente incremental, sin que ello tenga consecuencias sobre el diseño de los objetos (en realidad sobre su interfaz) ni las relaciones entre ellos.

Una de las principales ventajas de la metodología es que permite el uso transparente de objetos dinámicamente reconfigurables. Estos objetos pueden haber sido desarrollados en tiempo de compilación, o pueden añadirse posteriormente al diseño, una vez que la plataforma se encuentra en ejecución. Desde el punto de vista de su uso, el interfaz del objeto invocado es siempre el mismo, ya se trate de un objeto software, hardware, o dinámicamente reconfigurable, por lo que en ninguno de los dos casos se ve afectado. En cuanto

a la interfaz física, tal y como se ha descrito en el apartado anterior, ésta es en realidad la del bus del sistema, y por lo tanto invariable, ya que la unidad reconfigurable no es solamente el objeto, sino que incluye sus adaptadores (ya que no tiene sentido mantenerlos si el objeto es desalojado).

Para añadir un nuevo objeto una vez que el sistema está en marcha el proceso sería el siguiente:

1) definir la interfaz y modelar la funcionalidad del objeto propiamente dicho
2) generar los adaptadores correspondientes
3) componer el core con el objeto, el esqueleto dinámicamente reconfigurable y tantos proxies como tipos de objetos puedan ser invocados desde el nuevo objeto
4) generar el bitstream parcial para su ubicación en un área reconfigurable, y almacenar en un lugar accesible a la aplicación
5) por último será la aplicación la que ponga en marcha el proceso de reconfiguración, ubicando el nuevo core en el área reconfigurable que considere oportuna

Capítulo 5

Prototipado y Validación

5.1 Prototipado y Validación

Para aplicar la metodología se diseñaron los componentes vinculados. El diseño del Reconfiguration Manager se ha implementado en una plataforma de desarrollo Xilinx University Program, VirtexII Pro Development System. Las características de esta plataforma de desarrollo se describen en [85]. Estas placas poseen una FPAG VirtexII Pro, que permite reconfiguración parcial dinámica.
Para realizar estas reconfiguraciones es necesario definir previamente las zonas o áreas dentro de la FPGA donde será implantada la lógica reconfigurable (PR Modules). Nos referiremos a estas zonas como Regiones Parcialmente Reconfigurables (PRR) según la nomenclatura usada por Xilinx.

El principal problema con las PRRs es que durante la reconfiguración, estas regiones están deshabilitadas, no pudiendo responder a ninguna interacción con el resto del circuito. Para poder garantizar la integridad estructural del diseño, las PRR deben estar aisladas del resto para que su reconfiguración no interfiera en el funcionamiento normal del resto del circuito.

En el caso de las FPGA Virtex, buffers especiales llamados Busmacros son insertados entre la parte dinámica y la estática del circuito (figura 12) para todas las señales excepto las señales globales. De esa manera se convierten en una verdadera interfaz entre ambas partes. Estos Busmacros cortan el acceso a las PRR durante el proceso de reconfiguración de manera segura. Para hacer que la parte reconfigurable sea pin compatible con el resto del circuito, estos macros son pre-ruteados y pre-ubicados en tiempo de diseño, de manera tal que la cantidad de busmacros y su ubicación deben permanecer invariables durante el ciclo de vida de la aplicación.

La característica de compatibilidad de pines entre las PRR y el resto del circuito obliga a que los módulos a ser instanciados dentro de las PRR posean la misma interfaz, o sea la misma entidad (figura 13) lo que condiciona mucho la elección y el diseño de los objetos dinámicamente reconfigurables.

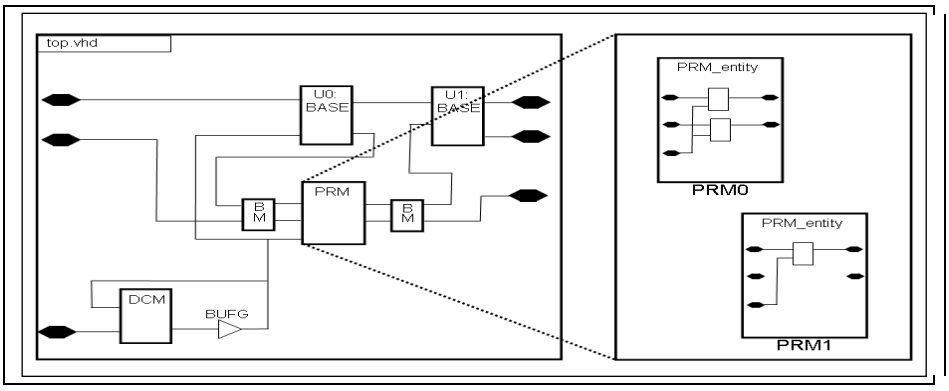

Fig 13

5.2 Flujo de diseño de Virtex PR

La figura 13 muestra el flujo de diseño para aplicaciones parcialmente reconfigurables. Una descripción detallada puede encontrarse en [83]. Brevemente, consiste en dividir el sistema en una parte estática y una parte dinámica e implementarlas como diseño no reconfigurable parcialmente.

La implementación inicial servirá como el diseño base donde la ubicación de la parte estática y dinámica ya está definida. Posteriormente se hara una implementación de cada módulo por separado teniendo en cuenta que la ubicación debe ser hecha en las PRR predefinidas.

Finalmente cada modulo PR será mezclado (merge) con el diseño base para generar los bitstreams parciales. Todo este proceso requiere de la definición de un conjunto de directorios y archivos muy rígido, donde el nombre de los componentes ya sintetizados al aplicar la herramienta de implementación, debe ser el mismo, por lo que la identificación de a qué modulo corresponde dependerá de su ubicación dentro del sistema de archivos.

Requiere además que el número de puertos de los módulos PR debe ser el mismo para todos los módulos que comparten la mismas PRR.(Figura 13)

Todo este proceso es manejado a través de modificación manual de los archivos de diseño (para la colocación de los macros, por ejemplo) y de la ejecución de un conjunto de scripts que no están integrados a ninguna de las herramientas regulares.

Este mismo flujo puede ser también aplicado al diseño de sistemas

embebidos usando las herramientas del Embedded Development Kit (EDK) de Xilinx. En este caso cada módulo PR se convierte en un periférico reconfigurable que puede conectar a cualquiera de los buses disponibles del sistema OPB o PLB. Una plantilla del periférico puede ser generado automáticamente mediante un programa de ayuda (wizard). Sin embargo debe ser modificado manualmente para incluir a los busmacros. Entonces debe hacerse una copia del periférico para cada instanciación diferente y debe realizarse un proceso de síntesis por separado, como así también varias otras modificaciones en el módulo Top, en los arcivos de restricciones del sistema, etc. Este proceso no es trivial y requiere demasiada manipulación por parte del diseñador.

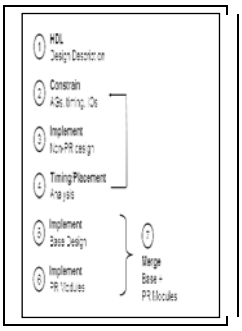

fig 14

Un flujo alternativo consiste en considerar el diseño EDK como un módulo simple el cual es exportado como componente para ser incluido en un sistema mayor. Este componente es incluido entonces en un proyecto ISE como parte de estática del diseño top, el cual incluye componentes reconfigurables y los buffers necesarios de bajo nivel (I/O o Busmacros)

Para poder tener acceso al bus del sistema (el bus OPB en este caso) y de este modo poder conectar los objetos reconfigurables se define un objeto para conexiones generales llamado PRP (Partially-Reconfigurable Peripheral) que simplemente lo que hace es rutear las señales del bus a la interfaz del componente EDK, de manera que el circuito final pueda ser ensamblado usando ISE. La ventaja de esta aproximación es que podemos conectar cualquier objeto al sistema base sin realizar ninguna modificación. Durante el proceso de verificación podemos conectar la versión estática del core bajo desarrollo. Una vez verificado podemos reemplazar la versión estática por la dinámica y completar el flujo de

diseño hasta la generación del bitstream inicial y los bitstreams parciales para cada core reconfigurable. (Figura 15)

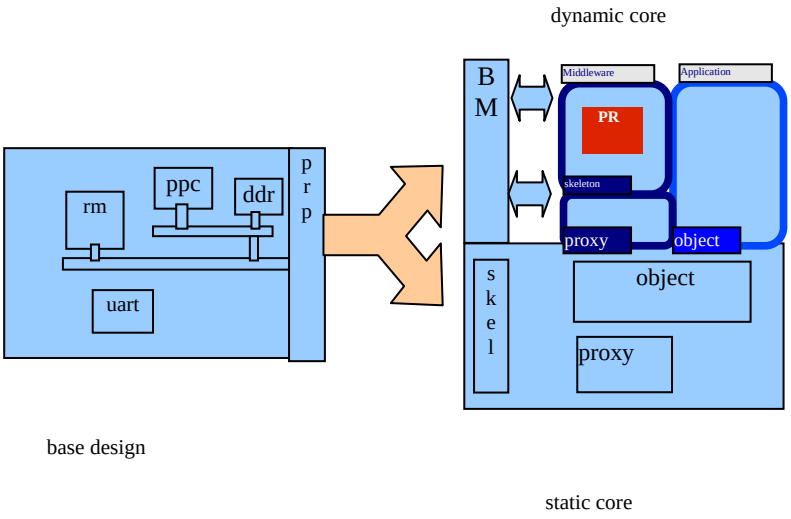

Fig 15

La cantidad de lógica que se agrega al circuito base al conectarle un PRO es mínima. La tabla 3 muestra la lógica utilizada por el sistema base con y sin PRP.

	Design with PRP	Design without PRP
Logic Utilization: Number of Slice Flip Flops:	2,543 out of 27,392 9%	2,460 out of 27,392 8%
Number of 4 input LUTs:	2,468 out of 27,392 9%	2,332 out of 27,392 8%
Logic Distribution: Number of occupied Slices:	2,585 out of 13,696 18%	2,475 out of 13,696 18%
Number of Slices containing only related logic:	2,585 out of 2,585 100%	2,475 out of 2,475 100%
Total Number 4 input LUTs:	3,219 out of 27,392 11%	3,067 out of 27,392 11%
Number used as logic:	2,468	2,332
Number used as a route-thru:	89	87
Number used for Dual Port RAMs:	472	462
Number used as Shift registers:	190	186
LUTs used per Dual Port RAM	2	

Tabla 3

5.3 Proceso de reconfiguración Xilinx

La gestión de la reconfiguración en la fpga que utilizamos se realiza por software utilizando primitivas que han sido desarrollada para tal fin.

Las fpga Virtex2pro poseen un componente denominado HWICAP que posee la capacidad de cargar los bitstream generados por las herramientas ISE. Los bitstreams son transferidos de la memoria local al ICAP (Internal Configuration Access Ports). Mediante este componente, los bitstream parciales son almacenados en la memoria de reconfiguración.

En este proceso los bitstream son leídos de memoria y se graban en la memoria de reconfiguración usando una técnica de leer-modificar-escribir. Para modificar el circuito de una FPGA, el HWICAP determina los frames de configuración que deben ser modificados, y entonces lee cada frame uno por vez en un BLOCK RAM. El contenido de cada frame es modificado antes de ser escritos de nuevo en el ICAP. Este ciclo leer-modificar-escribir se realiza un frame por vez. El HWICAP también permite la lectura de los estados de los recursos de configuración,. En este caso los frames son leídos en los BLOCKS RAM y los bits correspondientes son extraídos de allí.

Un frame es la cantidad menor de información de configuración que puede ser leída o escrita en un FPGA. Un frame de configuración es una colección de bits que llena completamente una columna de la FPGA y que posee un ancho de un bit. Los frames de configuración en el espacio de las CLB también contiene datos de configuración de los bloques de entrada salida (IOB) en la parte superior y en la inferior, los cuales configuran los IOBs superiores e inferiores de la FPGA. Una simple columna de CLB contiene múltiples frames de configuración.

Este mecanismo requiere que una columna completa sea leída en un BLOCK RAM. Esto implica que otra lógica de la misma columna puede ser modificada pero en la mayoría de los casos este efecto puede ser ignorado. Cuando el frame es reescrito en la memoria de configuración, las secciones de la columna que no han sido modificadas son reescritas con la misma información

Los bitstreams generados con las herramientas ISE poseen una parte de información de identificación, formada por un encabezado en donde se encuentra una serie de datos concernientes al bitstream como la longitud del nombre de diseño, la longitud de los datos, a que dispositivo corresponde, fecha, hora, la longitud del encabezado y la longitud del bitstream. Esta información es separada

al ser almacenado en memoria, quedando únicamente los datos efectivos de reconfiguración en memoria.

El proceso de reconfiguración parcial lo inicia el procesador cuando recibe el requerimiento por parte del usuario o por parte de la aplicación. El procesador inicia una serie de primitivas que manejan el HWICAP, el cual busca de memoria los bitstream para la reconfiguración parcial de la fpga.

Los tiempos de reconfiguración son del orden de los 15 ms si el bitstream esta alojado en memoria DDR hasta valores del orden de 160ms si están alojados en la compact flash

5.4 Ensayos en Laboratorio

Para poder realizar reconfiguración parcial dinámica utilizando la aproximación descrita en el capítulo 4 sobre la FPGA VirtexIIPRO se procede con un ejemplo de reconfiguración parcial formado por cuatro componentes que comparten un área de reconfiguración. Estos componentes son un contador ascendente, uno descendente, un registro de desplazamiento a la izquierda, y un registro de desplazamiento a la derecha, todos de cuatro bits. Estos módulos son reconfigurados en una misma región reconfigurable separada por 4 busmacros del resto del circuito. El tipo de busmacro utilizado se muestra en la figura. 16

Para realizar la reconfiguración se utilizan dos aproximaciones: una siguiendo el flujo de diseño propuesto por Xilinx y realizando la reconfiguración parcial por software. La segunda aproximación toma a estos mismos objetos pero se los convierte en objetos dinámicamente reconfigurables utilizando el modelo de comunicación indicado en el capítulo 3 y utiliza el dispositivo HWActivator para gestionar el proceso de reconfiguración.

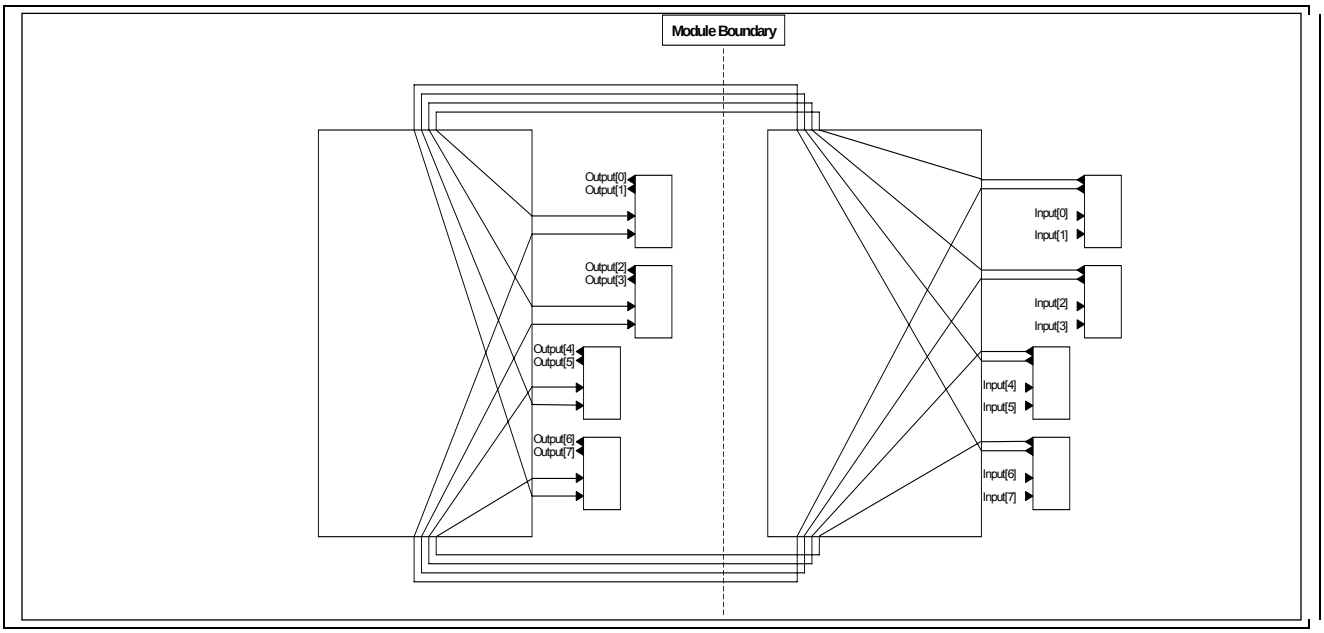

Fig 16

5.4.1 Reconfiguración Parcial usando metodología de Xilinx

Para la primera aproximación, se crea en EDK un diseño base formado por un procesador powerpc, una memoria DDR de 256MB, una compact flash, una uart para la comunicación serie, memoria interna conectada al bus PLB, el componente HWICAP conectado al bus OPB, que controla los puertos de acceso de configuración interna (ICAP) y un timer para controlar el tiempo de reconfiguración. Este sistema es luego sintetizado y exportado al entorno ISE donde se lo acopla a un diseño TOP que posee además lógica que compone la parte estática del diseño, los busmacros y los módulos reconfigurables.

El proceso de diseño se realiza ejecutando un script que reúne todas las operaciones de síntesis, implementación y Merge de cada uno de los módulos reconfigurables con el diseño estático, para generar el bitstream completo de configuración del la FPGA, mas los bitstreams parciales correspondientes a cada módulo a reconfigurar. El diseño estático mencionado es el formado ahora por el diseño base exportado del EDK mas la lógica que compone la parte estática.

Luego de generados los bitstreams, se configura la FPGA y se procede a realizar las diferentes reconfiguraciones parciales. Los tiempos obtenidos de

reconfiguración rondan los 17ms para los bitstream alojados en memoria ddr y alrededor de 160 ms para los bitstreams alojados en la compact flash.

La lógica utilizada por el sistema base se muestra en la tabla 4

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	2856	13696	20%
Number of Slice Flip Flops	2861	27392	10%
Number of 4 input LUTs	3552	27392	12%
Number of bonded IOBs	140	556	25%
Number of BRAMs	33	136	24%
Number of GCLKs	5	16	31%
Number of PPC405s	2	2	100%
Number of DCM_ADVs	2	8	25%

tabla 4- Reporte de recursos usados en el Sistema base

La reconfiguración se realiza completamente por software. Durante este proceso es el procesador el que invoca la carga de los bitstreams parciales al modulo HWICAP siguiendo el proceso descrito en el apartado 5.3

5.4.2 Reconfiguración siguiendo modelo propuesto

Para poder realizar un prototipo de nuestra propuesta de gestión de la reconfiguración dinámica usando nuestra aproximación basada en el sistema de comunicación detallado en el capítulo 3, el trabajo de laboratorio se dividió en varias partes.

Primero el refinamiento del modelo básico de objeto (Interfaz y estado) para permitir los mecanismos de activación implícita y Persistencia, segundo la generación automática del wrapper que adapte a cualquier objeto a nuestro sistema de comunicación, tercero la gestión de la detención y del estado en el proceso de reconfiguración parcial y cuarto la realización del proceso de reconfiguración por hardware usando el Hwactivator en lugar de por software como el primer ejemplo.

5.4.2.1 Refinamiento del modelo de objeto: Manejo de estado

La redefinición de los componentes que habían sido usados para reconfigurabilidad parcial en el primer ejemplo para convertirlos en objetos reconfigurables se realizó agregándoles a la funcionalidad inicial de los mismos dos métodos para la persistencia del estado, ***Set_state*** y ***get_state***.

El método ***Set_state*** tiene como parámetro a ***set_state_data,*** y un valor de retorno dado por ***set_state_done,*** mientras que el método ***get_state*** no tiene parámetros pero tiene los valores de retorno ***get_state_done*** y ***get_state_data.***

El método ***set_state*** le dice al objeto que el estado inicial esta dado por la entrada ***set_state_data***. Cuando el objeto leyó la entrada y actualizó su estado envía la respuesta ***set_state_done***.

Por su parte, ***get_state*** es un método para la obtención del estado que será entregado por la salida ***get_state_data***. La salida ***get_state_done*** se activará una vez que el método *get_state* haya sido ejecutado.

Con estos métodos ya es posible gestionar el estado del objeto. En estos objetos se definió el estado como el estado de los bits de salida de los contadores o de los registros de desplazamiento al momento de la detención de los objetos. Este estado es el que se guarda en memoria y es que luego se carga cuando se reactiva el objeto en una nueva reconfiguración.

5.4.2.2 Adaptación al sistema de comunicación

Una vez resuelta la gestión del estado falta la adaptación de los objetos al sistema de comunicaciones de nuestro modelo. Para ello se genera un esqueleto utilizando una herramienta desarrollada en Python. Esta herramienta denominada skeleton.py genera el esqueleto del objeto para ser conectado al sistema de comunicaciones.

Genera un core que esta formado por el objeto a adaptar, un proxy de la memoria donde se va a almacenar el estado y un esqueleto que adapta el objeto (sus métodos) al bus OPB en nuestro caso (nuestro sistema de comunicación) y al

gestor de reconfiguración, el HwActivator. Esta doble adaptación es la funcionalidad extra del esqueleto con respecto a un esqueleto de un objeto estático.

Por lo tanto esta herramienta adapta el objeto con sus métodos a las solicitudes del HwActivator y al bus de conexión.

5.4.2.3 Pruebas

Se realizaron pruebas para almacenar y recuperar el estado utilizando dos objetos de los cuatro anteriores, el contador ascendente y el registro de desplazamiento a la izquierda. En ambos casos se guardó el estado en una direccion de memoria que se entrega junto con la petición *store_rq* y luego se cargó al objeto con otro estado almacenado en otra dirección de memoria, indicada ahora con la solicitud *load_rq*.

Para llevar a cabo estas pruebas se generó un diseño en EDK creando un sistema formado por un procesador PowerPC, memoria DDR conectada al bus PLB, un puente OPB2PLB para poder acceder a la memoria a través del bus OPB, que es el bus donde se conectan los módulos reconfigurables, el HwActivator, el PRP, y el Core formado por el objeto mas el proxy y el esqueleto. El estado del objeto se conecto a los leds de la placa para su apreciación. Además se capturó las formas de onda de las señales del bus usando la herramienta Chipscope de Xilinx.

La activación del HwActivator se realiza por dirección de memoria. Las solicitudes de servicio que le envía la capa 3 de nuestro modelo de capas al HwActivator se realizan poniendo, en el bus de direcciones del bus OPB, la dirección del HwActivator más un offset que corresponde a cada una de las peticiones que puede recibir. En el caso de la petición Reconfig_rq la dirección es la dirección base (offset=0). Con esta dirección en el bus de direcciones del OPB el HwActivator se activa y el dato que esta en el bus de datos en ese momento es la dirección de donde hay que cargar el estado (dirección fuente), por lo que es necesario enviar la dirección de destino (donde guardar) el estado, antes de la

activación, a una dirección fija dentro del rango de direcciones correspondiente al HwActivator.

En ambos casos la cantidad de ciclos necesaria para la detención del objeto, el almacenamiento del estado, la recarga del mismo desde otra dirección y la señal de activación del objeto es de 30 ciclos cuyo detalle se añade a continuación:

Ciclo 0- activación del Hwactivator mas dirección fuente.

Ciclo 16. Reconfig_rq='1' recibida por el Hw activator solicitada por el procesador.
Ciclo 18: Store_rq='1' pedido de almacenamiento del estado. Objeto en Halt
Ciclo 19: Objeto en estado Store
Ciclo 22: Estado en bus
Ciclo 23: Estado almacenado (get_state_done='1')
Ciclo 24: load_rq='1', pedido de carga de estado. Objeto en estado Halt
Ciclo 25: Objeto en estado Load
Ciclo 28: Dirección fuente en reconfiguration_bus (ver Figura 10)
Ciclo 30: Estado en Objeto (set_state_done ='1')

La cantidad de lógica que agrega el esqueleto al objeto para su uso como objeto reconfigurable esta dada por la tabla 5

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	54	13696	0%
Number of Slice Flip Flops	58	27392	0%
Number of 4 input LUTs	96	27392	0%
Number of bonded IOBs	158	556	28%
Number of GCLKs	2	16	12%

Tabla 5: Sobrecarga de los objetos

Como se observa, esta sobre carga de lógica que se agrega a los componentes para transformarlos en objetos reconfigurables es prácticamente despreciable.

5.4.2.4 Gestión de la reconfiguración dinámica usando HwActivator

La Gestión de la reconfiguración dinámica se realiza reemplazando las actividades de reconfiguración que realiza el procesador por una arquitectura dedicada, formada por el HwActivator, un DMA, y el HWICAP.

Se parte realizando en EDK el mismo diseño base que la primera aproximación, pero ahora se le agregan unos componentes necesarios para la gestión de carga de los bitstream en la memoria de reconfiguración de la FPGA. formado por un powerPC, memoria DDR conectada el procesador a través del bus PLB. Estos componentes son el HwActivator, y un controlador DMA para el bus OPB para la transferencia de información.

El HwActivator, desarrollado en VHDL, posee una máquina de estado que controla tanto el proceso de carga y descarga de estado como así tambien el proceso de reconfiguración. Este componente diseñado para conectarse al bus OPB, posee como entradas las señales que transportan los pedidos de reconfiguración enviados por el Recofiguration Manager de la Capa 3 de nuestro modelo, según lo explicado en el apartado 4.2.3, y las señales de "*ack*" y "*ready*" enviadas por los objetos dinámicamente reconfigurables. Como salida entrega las señales *Stop_rq*, *store_rq*, *load_rq* y *start_rq*.

La Tabla 6 nos muestra la sobrecarga al diseño original provocada por el HwActivator. Como se aprecia esta sobrecarga es minima.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	175	13696	1%
Number of Slice Flip Flops	214	27392	0%
Number of 4 input LUTs	285	27392	1%
Number of bonded IOBs	216	556	38%
Number of GCLKs	2	16	12%

Tabla 6: Sobrecarga producida por el HwActivator

El proceso de reconfiguración lo realiza de la siguiente manera: primero se entrega la señal *stop_rq* al objeto que se va a reemplazar. Este objeto responde con un **ack** que indica petición recibida y continua con su ejecución hasta que no quede ningún método sin ejecutar. Una vez hecho esto se coloca en el estado

HALT y coloca la señal *ready*='1'.

Esto indica que el objeto ya está detenido. A partir de este momento el HwActivator le envía el pedido de guardar el estado del objeto con *store_rq* junto con la dirección donde almacenar el estado como se describió en el apartado 5.4.2.3. utilizando un bus dedicado para la transferencia del estado.

Luego comienza el proceso de carga de los bitstreams. Para ello el HwActivator coloca en el bus de direcciones OPB la dirección correspondiente al controlador de DMA incorporado al diseño para solicitar su servicio de transferencia de datos.

Este controlador DMA denominado OPB Central DMA Controller provee servicios de acceso directo a memoria para periféricos y memora conectados al bus OPB.

Este componente está formado por registros programables en donde uno coloca las direcciones fuente y destino, la longitud de datos a transferir, y el tamaño de la palabra de datos. Posee un buffer interno que soporta transferencias en ráfaga.

Estos registros están direccionados a partir de la dirección base del DMA mas un offset para cada uno de ellos. El DMA se activa al colocarle la cantidad de datos a transmitir en el registro de longitud de datos, por lo que este registro es el último en llenar.

Para poder comunicarse con la memoria DDR que está conectada al bus PLB, es necesario otro componente extra que es unpuende de conexión entre el bus OPB y el bus PLB, llamado OPB2PLB bridge, puesto que el DMA está conectado al bus OPB.

5.4.2.5 Pruebas realizadas en hardware.

Se realizaron varias pruebas utilizando los objetos desarrollados según se explicó en el apartado 5.3. El sistema base, formado ahora por el sistema original, mas los componentes mencionados en el punto anterior.

La cantidad de lógica total ahora del sistema base se indica en la tabla 7

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	3907	13696	28%
Number of Slice Flip Flops	4151	27392	15%
Number of 4 input LUTs	4796	27392	17%
Number of bonded IOBs	144	556	25%
Number of BRAMs	33	136	24%
Number of GCLKs	5	16	31%
Number of PPC405s	2	2	100%
Number of DCM_ADVs	2	8	25%

Tabla 7. Recursos utilizados por el sistema base con HwActivator, DMA

La sobrecarga con respecto al sistema inicial es la siguiente:

en Slices 3907/ 2856 = 1,36

En Slices Flip Flops = 4151/2861= 1,45

En LUT de 4 entradas: 4796/3552= 1.35

El incremento de lógica en el sistema base si bien es relativamente grande comparado con el sistema original (un 45% en algunos de los casos), es pequeño teniendo el cuenta la disponibilidad de recursos de la FPGA .

Mediante este componente el procesador es liberado de la carga de la reconfiguración parcial que ahora se realiza por Hardware.

Capítulo 6
Conclusiones y Trabajos futuros.

6.1 Conclusiones

La metodología presentada en la presente tesis presenta una serie de ventajas y soluciones a la problemática que existe en torno a la reconfiguración parcial dinámica del hardware que compone un sistema integrado.

Esta problemática vinculada con la falta de herramientas de desarrollo y con la dependencia tecnológica, hace que la reconfiguración parcial dinámica de las FPGAs no sea una tarea trivial. Presenta restricciones muy rígidas para el diseño de los componentes que van a compartir una zona parcialmente reconfigurable, y no ofrece alternativas para la adaptación de componentes realizados por terceros.

Las ventajas que otorga esta propuesta son las siguientes.

1) Provee una metodología de desarrollo simple, que contempla la gestion de la reconfigurabilidad parcial dinámica de los elementos hardware de un sistema de manera independiente de la tecnología.
2) Este gestor permite además el remplazo de cualquier objeto software que sea invocado por el procesador, por su contraparte en hardware y viceversa, debido al sistema de comunicación sobre el cual está montado este servicio.
3) Permite la incorporación de casi cualquier componente a nuestro sistema, ya que posee herramientas para la generación automática de su adaptador a nuestro sistema.
4) Esta adaptación no sobrecarga al componente.
5) Permite con esta adaptación su uso como componente reconfigurable con la única restricción de que su tamaño físico no supere el de la zona parcialmente reconfigurable de la FPGA.

Esta propuesta no está terminada, por el contrario, es solo la base para futuros trabajos de investigación y como en todo proyecto de consideración, lo realizado hasta acá permite acometer nuevos trabajos y además plantea nuevos interrogantes.

6.2 Trabajos Futuros

Existen varios aspectos a desarrollar y profundizar todavía dentro de esta propuesta. La aplicabilidad de esta metodología a sistemas integrados de redes en chip, y a los procesos de reconfiguración de Grids de FPGAs para computación de alta performance son algunos ejemplos de ello. El desarrollo de herramientas que permitan automatizar el proceso de diseño, el desarrollo de distintas técnicas de scheduling para la programación de las reconfiguraciones de acuerdo a su criticidad, el desarrollo de técnicas y algoritmos de ubicación (placement) en tiempo de ejecución de los objetos reconfigurables y no solamente en tiempo de compilación para darle mayor libertad al diseñador, son muestras de algunos de los aspectos que necesitan investigación y desarrollo.

Otro punto interesante a considerar, es la extensión de esta metodología de diseño que permita la gestión de la reconfiguración a sistemas formados por arreglos (grids) de FPGAs, en donde la gestión de la reconfigurabilidad y la ubicación de los objetos dinámicamente reconfigurables dejaría de ser local y el modelo podría ser aplicado en forma remota usando el modelo de comunicación basado en el paradigma de objetos distribuidos.

Referencias

[1] A. Ferrari, Alberto Sangiovanni-Vincentelli “System Design: Traditional Concepts and New Paradigms” ICCD99.

[2] "Platform-Based Design and Software Design Methodology for Embedded Systems" Alberto Sangiovanni Vincentelli, University of California - Grant Martin, Cadence Design Systems - IEEE D&TC (2001)

[3] Wander Cesario, Damien Lyonnard et al. “Multiprocessor SoC Platforms: A component-based Design Aproach” IEEE D&TC (2002)

[4]"Developing Architectural Platforms: A Disciplined Approach" Andrew Mihal, et al. University of California, Berkeley. IEEE D&TC (2002)

[5]“A design Chain for Embedded Systems” Grant Martin and Frank Schirrmeister-CADENCE. IEEE Computer Society, Computer March 2002, pp. 100-103.

[6]”FPGAs as Meta-platforms for Embedded Systems" Patrick Lysaght – Xilinx. IEEE International Conference in Field Programmable Technology (2002)

[7] K. Siozios, K. Tatas, G. Koutroumpezis, D. Soudris and A. Thanailakis- "An integrated framework for architecture level exploration of reconfigurable platform" IEEE (2005)

[8] Krishna Sekar, Kanishka Lahiri, Sujit Dey-UC San Diego, La Jolla, CA "Dynamic Platform Management for Configurable Platform-Based System on Chips" ICCAD (2003)

[9] Visvanathan Subramanian, et al "Design and Implementation of a Configurable Platform for Embedded Communication Systems" IPDPS (2003)

[10] Hidetomo Shibamura et. al. "Express1: A Dynamically Reconfigurable Platform using Embedded Processor FPGA" ICFPT (2004)

[11) K Ghali, O. Hammami I. Hermann “Multiobjective design of Embedded Processor in FPGA Platforms”
ICDCSW (2004)

[12] R.Henftling, A.Zinn, M.Bauer, W. Ecker, M. Zambaldi “ Platform-based testbench Generation”
DATE (2003)

[13] Gary Smith “Platform-based Design: Does it answer the entire SoC Challenge?” Proceedings of the 41st Design Automation Conference DAC-2004.

[14]Alberto Sangiovanni-Vincentelli, Luca Carloni, Fernando de Bernardinis Mauro Sgro “Benefits and Challenges for Platform-Based Design” DAC (2004)

[15)Pascal Nsame, Yvon Sacaria“A Customizable Embedded SoC Platform Architecture” IWSOC (2004)

[16] Wen Quan, Liu Bo, He Jianmin “Platform-based Síntesis Design Methodology for System-On-Chips”
ICEPT (2005)
[17] Zhihui Xiong, Sikun Li, Jihua Chen. - “Hardware Software Co-Design Using Hierarchical Platform-Based Design Method” - ASP-DAC (2005)
[18] Zhihui Xiong, Jihua Chen, Sikun Li - “Hardware/Software Partitioning for Platform-Based Design Method”
ASP-DAC (2005)
[19] Francesco Lertora, Mechele Borgatti. “Handling Different Computational Granularity by a Reconfigurable IC featuring Embedded FPGAs and a Network on Chip” IEEE(2005)
[20] Markus Visarius, Andre Meisel, Markus Scheithauer, Wolfram Hardt- “Dynamic Reconfiguration of IP based Systems” IEEE-CS(2005)
[21] Hans Jürgen brand, Steffen Rulke, Martin Radetzki “IPQ, IP Qualification for Efficient System Design” - IEEE-CS (2004)
[22] Qingxu deng, Hai Xu, Shuisheng Wei, Yu Han, Gen Yu. “An Embedded SOPC System Using Automation Design” ICPPW (2005)
[23]Mario Diaz Nava, Patric Blouet, Philippe Teninge, Marcello Coppola, Tarek Ben Ismail, Samuel Picchiottino, Robin Wilson, “An Open Platform for Developing Multiprocessor SoCs” IEEE-CS (2005)
[24) Marco Wehrmeister, Leandro Becker, Flavio Warner, Carlos Pereira- “An Object-Oriented Platform-Based Design Process for Embedded Real Time Systems” - ISORC (2005)
[25] Gabreil Lipsa, Andreas Herkersdorf, Wolfgang Rosentiel, O. Bringmann W. Stechele. “Towards a Framework and a Design Methodology for Automatic SoC” - ICAC (2005)
[26] Claudio Talarico, Aseem Gupta, Ebenezer Peter, Jerzy W. Rozenblit, “Embedded System Engineering Using C/C++ Based Design Methodologies” - ECBS(2005)
[27] Claudio Talarico, Esteban Rodriguez-Marek, Ming-sung Koh “Multi-objective Design Space Exploration Methodologies for Platform-bases SoCs” .ECBS(2006)
[28] Annti Pelkonen, Kostas Masselos, Miroslav Cupák “System-Level Modeling of Dinamicallly Reconfigurable Hardware with SystemC” IPDPS(2003)
[29] Yang Qu, Kari Tiensyrjä, Juha-Pekka Soininen - “SystemC-based Design Methodology for Reconfigurable System-on-chip” - DSD (2005)
[30]“Embedded UML: a merger of real-time UML and co-design” Grant Martin, Luciano Lavagno, Jean Louis Guerin – Cadence – CODES(2001)
[31] "Embedded system design using UML and platforms" Rong Chen, Marco Sgroi,

Luciano Lavagno, Grant Martin, Alberto SG-V, Jan Rabaey- Univ. Calif. At Berkeley-Cadence - FDL 2002

[32] Rong Chen, Marco Sgroi, Luciano Lavagno, Grant Martin, Alberto SG-V, Jan Rabaey "UML and platform-based design"- FDL 2002

[33] Jose Moya, Fco Moya, Fdo Rincon, J.C. Lopez- UCLM "Improving Embedded System Design by means of Hw-Sw Compilation on Reconfigurable Coprocessors" ISSS (2002)

[34] "An object Oriented Design Process for System-on-Chip using UML" Qiang Zhu, Akio Matsuda, Shinya Kuwamura, Tsuneo Nakata, Minoru Shoji-Fujitsu - ISSS 2002

[35] Li Li, Minglun Gao, Shuzhuan He. "A new Platform-Based Orthogonal SoC Design Methodology"

Institute of VLSI design, Nanjing University, China IEEE(2003)

[36] "The Modelling of Embedded Systems Using HASoC" P.N. Green M.D Edwards-UMIST, Manchester, UK DATE (2002)

[37] Nobuyuki Ohba, Koji Takano "An SoC Design Methodology Using FPGAs and Embedded Processors" - DAC (2004)

[40] "PowerPC 405 Processor Block Reference Guide" - UG018 (v2.1) July 20, 2005- Xilinx Corporation

[41] "Excalibur Device Overview" - 2002 Altera Corporation.

[42] http://www.cochrane.org/resources/handbook/

[43]Tim Tuan, Suet-Fei Li, Jan Rabaey "Reconfigurable Platform Design for Wireless Protocol Processors" ICASSP2001.

[44]Juha-Pekka Soininen, Axel Jantsch, Martti Forsell, Antti Pelkonen, Jari Kreku, and Shashi Kumar "Extending Platform-Based Design to Network on Chip Systems" VLSI 2003

[45]A. Balluchi, M. D. Di Benedetto, A. Ferrari et al, "Platform-based Design: Applications and Flows"
http://www.columbus.gr/documents/public/WPPBD/Columbus_DPBD2.pdf

[46] Jan Rabaey- Alberto Sangiovanni Vincentelli "System-on-a-Chip−A Platform Perspective"http://bwrc.eecs.berkeley.edu/people/faculty/jan/presentations/platformdesign.pdf -

[47]G Wang, W.R.Gong, R. Kastner, " A new approach for task level computational resource bi-partitioning" IEEE- ICPDCS-2003

[48] P. Brunet, C. Tanougast, Y. Berviller, S. Weber,"Hardware Partitioning Software for Dynamically Reconfigurable SoC Design" IEEE- IWSCRTA-2003

[49] Michael Ullmann, Wansheng Jin, Jürgen Becker- “Hardware Enhanced Function Allocation Management in Reconfigurable Systems” IEEE-IPDPS 2005

[50] Samarjit Chakraborty Simon Kunzli, Lothar Thiele “A General Framework for Analysing System Properties in Platform-Based Embedded System Designs” IEEE 2003

[51] “Procedures for Performing Sistematics Reviews”- Barbara Kitchenham – NICTA Technical Report- 2004

[52] Julie Glanville y Amanda Sowden Reporte CDR Número 4 Marzo de 2001 (2aEdición) NHS Centre for Reviews and Dissemination, University of York

[53] “Las revisiones sistemáticas, niveles de evidencia y grados de recomendación” Juan Antonio Guerra, Pedro Martín Muñoz, José Manuel Santos Lozano. Grupo MBE Sevilla, Red Temática de Investigación sobre Medicina Basada en la Evidencia

[54]”Systematic Reviews: Synthesis of Best Evidence for Clinical Decisions” Deborah J. Cook, MD, MSc; Cynthia D. Mulrow, MD, MSc; and R. Brian Haynes, MD, PhD- Annals of Internal Medicine March 1997 | Volume 126 Issue 5 | Pages 376-380

[56] “Cramming More Components Onto Integrated Circuits “ Gordon Moore, [Electronics Magazine, 1965]

[57] “65 Nanometer Technology”- Intel Corp. http://www.intel.com/technology/silicon/65nm_technology.htm

[58] http://embedded.eecs.berkeley.edu/mescal/

[59] http://www.semiconductors.philips.com/products/nexperia/index.html

[60] http://focus.ti.com/omap/docs/omaphomepage.tsp

[61] D.Morris, D.G.Evans, P.N Green and C. J Theaker “Object Oriented Computer System Engineering”. Springer-Verlag, 1996.

[62]http://www.coware.com/

[63] http://www.celoxica.com/

[64]http://www.equator.com/

[65]http://www.quicklogic.com/

[66]http://www.eetimes.com/news/design/columns/eda/showArticle.jhtml?articleID=17408109

[67]J. Resano D. Mozos F. Catthoor, D Verkest. “A Reconfiguration Manager for Dynamically Reconfigurable Hardware. IEEE Design and Test of Computers. September-October 2005. pp 452-460.

[68]P. Benoit, L. Torres, G. Sassatelli, M. Robert, G. Cambon, J. Becker. “Dynamic Hardware Multiplexing: Improving Adaptability with a Run Time Reconfiguration Manager”.

In Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI'06)

[69]E. Carvalho, N. Calazans, F. Moraes, D. Mesquita. "Reconfiguration Control for Dynamically Reconfigurable Systems"- pp405-410 – DCIS 2004.

[70]J. Resano, D. Mozos, D. Verkest , F. Catthoor, S. Vernalde "Specific Scheduling Support to Minimize the Reconfiguration Overhead of Dynamically Reconfigurable Hardware" pp 199- 124, in Proceedings of the 41st Design Automation Conference (DAC'04).

[71]P.R. Pukite, J. Pukite,Lt. D.S. Barnhart.Expert System for Redundancy and Reconfiguration Management' pp 233-240 – IEEE 1992

[72] P.G. Paulin, C. Pilkington, M. Langevin. E. Bensoudane, O. Benny, D. Lyonard, B.Lavigueur and D. Lo. Distributed object models for multi-processor SOC's, with application to low-power multimedia wireless systems. In *Proceedigns of the DATE'06 Conference*, Munich, Germany, 2006.

[73]R. Hecht, S. Kubish, H. Michelsen, E. Zeeb and D. Timermann. A distributed object system approach for dynamic reconfiguration. In *Reconfigurable Architectures Workshop (RAW 06)*, Rhodos, Greece, April 2006.

[74]M. Vuletic, L. Pozzi and P. Lenne. Seamless Hardware-Sofware Integration in Reconfigurable Computing Systems.In *IEEE Design and Test of Computers*, 2005.

[75]M. Hubner, C. Schuck, M. Kühnle, J. Becker. "New 2-Dimensional Partial Dynamic Reconfigurations Techniques for Real-Time Adaptive Microelectronic Circuits". In Proc. Emegin VLSI Technologies and Architectures – 2006 (ISVLSI'06)

[76]F. Ferrandi, M.Santambrogio, D. Sciuto. "A Design Methodology for Dynamic Reconfiguration. The Caronte Architecture". In Proc. of 19th IEEE International Parallel an Dristibuted Processing Symposium (IPDPS' 05)

[77]R. Pellizoni, M. Caccamo. "Adaptive Allocation of Software and Hardware Real-Time Tasks for FPGA-based Embedded Systems" In Proc. Of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS' 06)

[78]A. Sudarsanam, M. Srinivasan and S. Panchanathan "Resource Estimation and Task Scheduling for Multithreaded Reconfigurable Architectures" In proc. of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04)

[79]C. Steiger, H.Walder, M. Platzner and L. Thiele, "On Line Scheduling and Placement of Real-time Task to Partially Reconfigurable Devices. In Proc. IEEE International Real-Time Systems Symposium (RTSS) 2003

[80] Kiarash Bazargan, Ryan Kastner, Majid Sarrafzadeh. "Fast Template Placement for Reconfigurable Computing Systems". In IEEE Design and Test of Computers,-2000

volume 17, pages 68-83.

[81]Pedro Merino, Margarida Jacome, Juan Carlos Lopez. “A Methodology for Task Based Partitioning and Scheduling of Dynamically Reconfigurable Systems,” In Proc IEEE Symposium on FPGAs for Custom Computing Machines (FCCM) 1998- pages 324-325

[82]J-Y Mignolet, V, Nollet, P. Coene, D.Verkest, S.Vernalde, R. Lauwereins. “Infraestructure for Design and Management of Relocatable Tasks in a Heterogeneous Reconfigurable System-on-Chip.” In Proc. Design and Test in Europe Conference and Exhibition (DATE) 2003

[83]Xilinx. Early Access Partial Reconfiguration User Guide UG208 (v1.1), March, 2006.

[84] Xilinx, Development System Reference Guide 8.1

[85]Xilinx XUP Virtex II Pro Development System- Hardware Reference Manual- UG069(v1.0)March8, 2005.