Universidad Autónoma de Madrid

UAM

Diseño e implementación de arquitecturas dinámicamente reconfigurables basadas en microprocesador

TESIS DOCTORAL

Escuela Técnica Superior de Informática

Enero de 2001

Autor: Julio Manuel Faura Enríquez
Director: Eduardo Iván Boemo Scalvinoni

Resumen

Esta tesis estudia la metodología del diseño, implementación física y utilización de arquitecturas programables genéricas que permitirían la implementación rápida y barata de aplicaciones de tipo industrial de señal mixta basadas en microprocesador. Este tipo de arquitecturas, denominadas genéricamente FIPSOCs (*FIeld Programmable System on a Chip*), contendrían un substrato programable (una FPGA) para la parte digital del diseño, un conjunto de células analógicas programables para las tareas de interfaz con señales analógicas, y un microprocesador orientado a operaciones de control y computación de propósito general, y dispondrían de una metodología integrada consistente de diseño, programación y verificación.

En particular se estudian no sólo los aspectos arquitecturales de cada uno de estos tres bloques por separado sino los relativos a su integración con el objeto de maximizar la interacción entre ellos, además de los aspectos metodológicos derivados de esta fuerte interacción. De esta forma se explotan ideas tales como el acceso en tiempo real a las señales reales del circuito desde posiciones de memoria de datos del microprocesador, lo que permite una co-emulación conjunta *hardware-software* en tiempo pseudo-real integrada lo cual supone una nueva metodología de diseño y verificación; se propone además un esquema de memoria de configuración en el que cada *bit* de programación estaría precedido de una o varias células de memoria *buffer* accesibles como memoria normal desde el microprocesador, lo cual posibilita la reconfiguración dinámica multicontexto tanto parcial como total sin necesidad de detener el funcionamiento de la parte de circuito reconfigurada, y la reutilización de memoria para objetivos adicionales a la configuración del *chip*. Este tipo de substrato dinámicamente reconfigurable resulta ideal para explorar técnicas de *hardware virtual* que permitirían maximizar la utilización del silicio activo en cada momento.

Finalmente se discuten los aspectos relacionados con la implementación física de estos sistemas y se reportan los resultados obtenidos en silicio de una primera implementación de una arquitectura mixta programable de este tipo, a la vez que se apuntan sus posibles aplicaciones a problemas reales.

Abstract

In this thesis we introduce a methodology for the design, physical implementation and application of programmable architectures that would allow a fast and cost-effective implementation of microprocessor-based mixed signal solutions. This type of architectures, generically named FIPSOCs (*FIeld Programmable System on a Chip*), would include a programmable substrate (an FPGA) for the digital part, a set of configurable analog cells to interface with analog signals, and a microprocessor both for system control and general purpose computing; furthermore, a consistent integrated methodology could be provided for design, programming and verification.

In particular we study not only the architectural issues related to each of these three blocks but the ones related to their integration with the aim of maximizing the interaction between them, and all the methodological implications of such an strong interaction. We exploit new ideas such as providing real time access to user application signals as regular memory locations from the microprocessor, thus enabling the user to perform a hardware-software co-emulation in pseudo-real time, what constitutes a new design and verification methodology. We introduce also a new configuration memory scheme in which each programming bit would be backed up by one or several *buffer* memory cells mapped in the microprocessor address space, which enables the so-called multi-context dynamic reconfiguration, both partially and totally, without having to stop the application, and allows the user to re-use the memory for purposes other than configuration. This type of dynamically reconfigurable substrate becomes optimally suited for exploring the so-called *virtual hardware* techniques, which maximize the active silicon utilization.

Finally we discuss all the issues related to the physical implementation of this kind of systems, we report silicon results relative to the first implementation of a mixed architecture of this type, and we point out its typical applications.

Índice general

Capítulo 1: Arquitecturas programables en aplicaciones de señal mixta: análisis y limitaciones

1.1 Introducción
1.2 Motivación y objetivos generales de esta tesis
1.3 Precedentes y estado del arte
1.4 Organización de esta tesis
1.5 Resumen

Capítulo 2: Diseño de células programables digitales y analógicas

2.1 Introducción
2.2 Estructura general de una arquitectura de tipo FIPSOC
2.3 Diseño de FPGAs para arquitecturas de tipo FIPSOC
- 2.3.1 Caracaterísticas generales
- 2.3.2 Estructura general
- 2.3.3 El DMC
  - 2.3.3.1 Parte combinacional
    - 2.3.3.1.1 Estructura de las LUTs
    - 2.3.3.1.2 Estabilidad de las células de memoria
  - 2.3.3.2 Parte secuencial
    - 2.3.3.2.1 Diseño básico de *latches* y *flip-flops*
    - 2.3.3.2.2 Mecanismos de *reset* y escritura desde el microprocesador
    - 2.3.3.2.3 *Flip-flops* multicontexto
    - 2.3.3.2.4 Modos de funcionamiento
  - 2.3.3.3 Rutado interno
  - 2.3.3.4 Salidas del DMC
- 2.3.4 Células de entrada-salida
- 2.3.5 Recursos de rutado
  - 2.3.5.1 Diseño de arquitecturas de rutado
    - 2.3.5.1.1 Arquitecturas básicas
    - 2.3.5.1.2 Prestaciones
    - 2.3.5.1.3 Retardo medio
  - 2.3.5.2 Caso práctico de arquitectura de rutado
    - 2.3.5.2.1 Recursos de rutado de la matriz de DMCs
    - 2.3.5.2.2 Recursos de rutado de las células periféricas
    - 2.3.5.2.3 Rutado de reloj
    - 2.3.5.2.4 Análisis de rutabilidad

2.4 Bloques analógicos programables
- 2.4.1 Estructura general del CAB
- 2.4.2 Amplificadores
- 2.4.3 Comparadores
- 2.4.4 Conversores
- 2.4.5 Interconexión analógica

2.5 Configuración de arquitecturas programables
- 2.5.1 Reconfiguración multicontexto

2.5.1.1 Contextos *buffer*
2.5.1.2 Contextos *buffer* múltiples
2.5.1.3 Pilas de contextos
2.5.2 Mecanismos de reconfiguración de FIPSOC
2.5.2.1 Reconfiguración parcial y configuración múltiple
2.5.2.2 División de las LUTs en dos contextos
2.5.2.3 *Flip-flops* multicontexto precargables
2.5.2.4 Reutilización de memoria
2.5.2.5 El proceso de configuración
2.5.3 Configuración de células analógicas
2.6 Resumen

Capítulo 3: Interfaces entre microprocesadores y células programables en arquitecturas de tipo FIPSOC

3.1 Introducción
3.2 Interfaz entre el microprocesador y la lógica programable
3.2.1 El DMC a los ojos del interfaz con el microprocesador
3.2.2 Interacción entre mapas de memoria del microprocesador y células programables
3.2.3 Mapeo de memoria de configuración
3.2.4 Mapeo de señales
3.2.5 Mapeo de tablas de *look-up*.
3.2.6 Mapeo de registros de control
3.2.7 Interrupciones
3.2.8 Comunicaciones serie y modos de arranque
3.2.9 Generadores de reloj
3.2.10 *Breakpoints hardware* y *software*
3.2.11 Co-emulación y co-simulación *hardware-software*
3.3 Interfaz entre el microprocesador y las células analógicas programables
3.3.1 Control de células analógicas programables
3.3.2 Mapeo de puertos de células analógicas
3.3.3 Aplicaciones indirectas para emulación y desarrollo.
3.4 Interfaz entre la lógica programable y las células analógicas programables
3.5 Resumen

Capítulo 4: Implementación de arquitecturas tipo FIPSOC

4.1 Introducción
4.2 Metodologías seguidas
4.3 Implementación de las células lógicas programables
4.3.1 Parte combinacional
4.3.2 Registros programables
4.3.3 Memoria de configuración
4.3.4 Recursos de rutado
4.3.5 El DMC en su conjunto su viabilidad
4.3.6 Células de entrada-salida
4.3.7 IICs
4.3.8 Prestaciones
4.4 Implementación de células analógicas programables

4.4.1 Las células analógicas en sistemas mixtos
4.4.2 Amplificadores operaciones difereciales y balanceados
4.4.3 Secciones de amplificación
4.4.4 Convertidores
4.4.5 Comparadores
4.4.6 Rutado de señales analógicas
4.4.7 El CAB en su conjunto
4.5 Implementación del interfaz con el microprocesador
4.6 Test
4.7 Resumen

Capítulo 5: Aplicaciones

5.1 Introducción
5.2 Sistemas de desarrollo integrados
5.3 Sistemas monochip
5.4 Sistemas con *hardware* virtual
5.5 Sistemas adaptativos y reconfigurables
5.6 Coprocesadores reconfigurables para microprocesadores
5.7 Librerías para co-diseño *Hardware-Software*
5.8 Resumen

Capítulo 6: Conclusiones y trabajo futuro

6.1 Introducción
6.2 Principales aportaciones de esta tesis
6.3 Principales publicaciones generadas a partir de esta tesis
6.4 Líneas de trabajo futuro
6.4.1 Esquemas de rutado jerárquico
6.4.2 Arquitecturas analógicas programables avanzadas
6.4.3 Co-procesadores reconfigurables fuertemente acoplados
6.4.4 Herramientas y algoritmos de reconfiguración dinámica
6.4.5 Herramientas de co-diseño *Software-Hardware* reconfigurable
6.4.6 Co-simulación y co-emulación *hardware-software*
6.5 Resumen

Apéndice: Introducción a las arquitecturas programables: elementos constitutivos y parámetros definitorios

A.1 Introducción
A.2 Arquitecturas digitales programables
A.2.1 Elementos constitutivos de las FPGAs
A.2.1.1 Bloques programables combinacionales y secuenciales
A.2.1.2 Células de entrada-salida
A.2.1.3 Canales y recursos de rutado y matrices de conmutación
A.2.2 Parámetros definitorios de estructuras lógicas programables
A.2.2.1 Granularidad de la LUT
A.2.2.2 Granularidad del bloque programable

A.2.2.3 Tipo de bloque programable y de memoria de configuración
A.2.2.4 Rutabilidad externa
A.2.2.5 Rutabilidad interna
A.2.2.6 Accesibilidad
A.2.2.7 Evaluación de arquitecturas programables
A.2.3 Evolución histórica y estado del arte de las arquitecturas digitales programables
A.3 Arquitecturas analógicas programables
A.3.1 Estructura general de una célula analógica programable
A.3.2 Evolución histórica y estado del arte de las arquitecturas analógicas programables
A.4 Configuración de arquitecturas programables
A.4.1 Esquemas clásicos de configuración de células programables
A.4.2 Reconfiguración dinámica y parcial
A.4.3 *Hardware* virtual y procedimientos *hardware*
A.4.4 Evolución histórica de la reconfiguración de arquitecturas digitales programables

Lista de acrónimos

Bibliografía

AGRADECIMIENTOS

Tras todos estos años son muchas las personas a las que quisiera expresar mi agradecimiento por su apoyo en mi trayectoria profesional y académica.

Quisiera empezar por José María Insenser, director general de SIDSA y que como jefe y amigo ha sido una de las personas que más me ha ayudado en mi trayectoria profesional y que mejor me ha tratado; igualmente he de mencionar en lugar especial a Eduardo Boemo, quién además de mi director de tesis ha estado siempre apoyándome sin condiciones como amigo y mentor en el mundo académico. A los dos mi reconocimiento y mi agradecimiento por su ayuda, consejo y dirección, y por supuesto por las numerosas oportunidades que me han brindado en distintos momentos de mi vida.

Muchas gracias a todos mis compañeros de SIDSA, que tanto han me han ayudado durante mi trabajo con un sinfín de discusiones, ideas y sugerencias, y especialmente a Fernando, Juanma, Fede, Steve, Javier, Carlos, Miguel Company, Moisés, Miguel Riaza y Curro. Muy especialmente a Ignacio – cómo vas? –, haciendo justicia al reconocer que sin su impresionante esfuerzo FIPSOC no habría sido posible; igualmente debo agradecer haber colaborado tan estrechamente con todo el gran equipo FIPSOC, y entre sus estrellas con Manuel, Joan y Jordi de la UPC, y con Antonio, Miguel, Vicente y Leo de US-GTE; de igual forma no quiero olvidar a Javier Garrido y al resto de los profesores de la Universidad Autónoma de Madrid, a quien debo sin duda agradecer los buenos ratos pasados durante mis años de profesor y la oportunidad de presentar esta tesis.

Muchas gracias a Carlos López Barrio por su apoyo durante las etapas iniciales de este trabajo y por las ocasiones – numerosas – en las que me ha ayudado; y mi recuerdo a mis compañeros becarios de la UPM, no sin mencionar a mis amigos Miguel Angel – nice shoes! – y Angel Luis, por todos los buenos ratos de insomnio en el MBE de la E.T.S.I.T. durante el comienzo de mis estudios de doctorado. También quisiera no dejar de agradecer a Enrique Calleja su apoyo y la fe que puso en mí – no tengo sino palabras de agradecimiento y admiración para él.

Muchas gracias finalmente a mis padres y a mi hermano Pablo por sus consejos desinteresados, su apoyo incondicional, y el inmenso número de oportunidades que he recibido. Gracias de verdad por haber estado a mi lado en todo momento.

– Y más que a nadie gracias a Natalia – como puedes ver todo esto es para ti.

CAPÍTULO 1

ARQUITECTURAS PROGRAMABLES EN APLICACIONES INDUSTRIALES DE SEÑAL MIXTA: ANALISIS Y LIMITACIONES

1.1 Introducción

Esta tesis se desarrolla en el marco general del diseño y utilización de arquitecturas programables genéricas para aplicaciones industriales de señal mixta controladas por microprocesador. En particular se estudia el diseño, implementación e integración de arquitecturas programables universales de señal mixta que, junto con una metodología integrada para su utilización, resulten capaces de implementar aplicaciones industriales que típicamente incluyen una parte digital de control (puertas lógicas), una interfaz analógica, y un microprocesador que ejecuta un determinado programa de cálculo y control. Estas arquitecturas deberán incluir por tanto recursos programables para implementar cada una de estas tres partes de la aplicación típica: una FPGA para la parte digital, un conjunto de células analógicas configurables para la interfaz analógica, y un microprocesador, resultando de gran importancia la capacidad de interacción entre estos tres ámbitos. Además resultará necesario disponer de herramientas y metodologías adecuadas que garanticen la consistencia e integración de los flujos de diseño y verificación para cada uno de estos tres dominios así como del sistema global.

Más adelante la tesis presenta la primera realización práctica de una arquitectura de este tipo, desde sus detalles arquitecturales hasta su realización misma en silicio. En total este primer

circuito integrado tiene 163 pads y ocupa 65mm^2 (9.6x6.5 mm^2) en un proceso de 0.5μm y tres metales. Incluye más de 1.8 millones de transistores de los cuales 1.5 millones están implementados en *full custom* de forma manual, alcanzando densidades de integración superiores a 40,000 transistores/mm^2.

En este capítulo introductorio se expondrán las razones que han motivado el trabajo propuesto, y se identificarán las carencias de las soluciones existentes en la actualidad en este campo que el presente trabajo contribuye a paliar. A este respecto es importante hacer énfasis en el fuerte enfoque práctico que se ha adoptado para la realización de esta tesis, razón que ha obligado por ejemplo a hacer constantes comparaciones con manuales de usuario y libros de aplicaciones de arquitecturas programables comerciales que no pertenecen puramente al ámbito académico e investigador, lo cual no debería entrar en conflicto con el interés científico de las aportaciones de esta tesis al igual que no ocurre con la mayor parte de las publicaciones relacionadas con arquitecturas programables que se multiplican estos días por los congresos de circuitos integrados.

Finalmente, deberá disculparse el hecho de que una parte de la terminología utilizada se ha mantenido en su idioma original, esto es, inglés. Esta decisión se ha tomado con el objetivo de hacer prevalecer la claridad aún a costa de la corrección lingüística, si bien cada término no tradicional ha sido marcado en cursiva para distinguirlo. Igualmente se utiliza una serie de acrónimos que se clarifican en una lista adjunta.

1.2 Motivación y objetivos generales de esta tesis

Tradicionalmente, los diseños de señal mixta han sido acometidos en tres dominios bien diferenciados: *hardware* digital, *hardware* analógico, y *software* corriendo en un microprocesador o microcontrolador, tal y como se esquematiza en la figura 1.1. Las metodologías seguidas, las problemáticas que se presentan y las herramientas utilizadas suelen ser muy distintas para cada uno de estos tres ámbitos, siendo frecuentemente la comunicación entre ellos el cuello de botella a la hora del modelado y realización práctica del sistema: el diseñador utiliza una serie de componentes discretos para resolver el problema en cada uno de sus ámbitos que luego debe interconectar teniendo en cuenta la influencia de unas partes del sistema sobre otras. El uso de estas tres metodologías distintas presenta serias limitaciones, ya que no suelen ser compatibles, utilizan herramientas muy distintas que habitualmente son

distribuidas por proveedores diferentes, y su interacción suele ser difícil y poco fiable. Estos problemas metodológicos aumentan el ciclo de diseño, complican la verificación del sistema de cara a la fabricación y producción en serie y encarecen el diseño con el coste de tres conjuntos de herramientas.

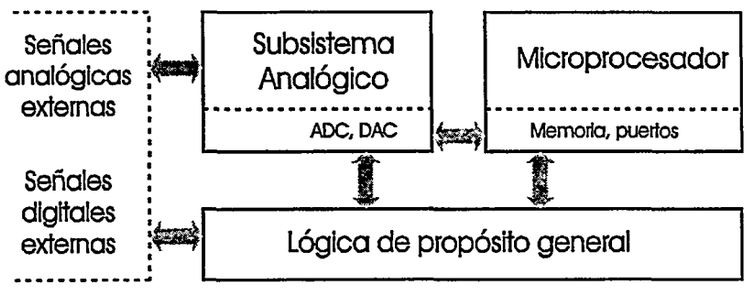

Fig. 1.1: Esquema de bloques genérico de una aplicación industrial típica de señal mixta

Si además el objetivo es realizar todas las funciones del sistema mediante un solo circuito integrado, el diseñador debe reunir todos estos elementos sobre el mismo substrato y verificar mediante simulaciones su correcto funcionamiento, si bien la dificultad a la hora de modelar la interacción entre bloques analógicos y digitales, y en especial la inyección de ruido en la parte analógica desde la parte digital a través del substrato y de las líneas de alimentación, hace que el riesgo al diseñar circuitos integrados de señal mixta sea elevado. Este riesgo, unido al alto coste de desarrollo de estos circuitos integrados de aplicación específica o ASICs (*Application Specific Integrated Circuit*), tanto en términos de fabricación de las máscaras como de tiempo de desarrollo, hace que sólo resulten rentables para grandes producciones, problema que se agrava día a día conforme las tecnologías se reducen de tamaño y aumentan en complejidad. El coste de prototipado es por tanto extremadamente alto, debido fundamentalmente al alargamiento del ciclo de diseño producido por la creciente complicación tecnológica, y al incremento en el coste de la fabricación de las máscaras conforme las tecnologías se hacen cada vez más complejas.

Este crecimiento en el coste del prototipado ha acelerado en los últimos años el gran desarrollo de los circuitos programables por el usuario. Entre ellos, el caso más espectacular es el de las FPGAs, utilizadas hoy día de forma rutinaria en la industria como herramienta de prototipado rápido y de validación previa antes de enviar un circuito integrado digital al proceso de fabricación [TRI94]. La ausencia de costes fijos de fabricación y la reducción en el ciclo de diseño hacen que incluso resulte rentable la utilización de esta tecnología no sólo para el prototipado sino para la misma fabricación en determinados casos, especialmente cuando se manejan volúmenes de producción medios y bajos o cuando se trata de productos para los que una rápida salida al mercado resulta vital.

Entre los inconvenientes que presentan las arquitecturas digitales programables se cuentan el mayor coste unitario de la pieza, debido al exceso de área de silicio que supone la programabilidad, y las peores prestaciones que se obtienen en comparación con las soluciones *ad hoc* tipo ASIC. No obstante, las modernas técnicas de reconfiguración dinámica capaces de soportar el llamado *hardware virtual* [LYS93], mediante el cual es posible emular el funcionamiento de un circuito grande a base de reconfigurar dinámicamente un substrato programable para realizar únicamente las funciones del sistema activas en cada momento, permiten aumentar drásticamente el rendimiento de los dispositivos programables, o lo que es lo mismo, aumentar el número de puertas efectivas soportadas por unidad de área hasta niveles comparables a los de los ASICs. Por otra parte, las aplicaciones industriales típicas de señal mixta no suelen requerir grandes prestaciones, razones que permiten afirmar que los dispositivos programables digitales, especialmente los que incorporan la capacidad de ser reprogramados dinámica y parcialmente, constituyen normalmente una buena alternativa para el desarrollo e incluso producción (en determinados casos) de aplicaciones industriales, o al menos las partes puramente digitales de éstas.

Estas ventajas de la programabilidad son también aplicables a los otros dos ámbitos de las aplicaciones de señal mixta de carácter industrial: Por una parte, los microprocesadores son elementos inherentemente programables, que incluso según Tredennick [TRE95] retrasaron la aparición de las FPGAs ya que eran la única alternativa viable en el estado del arte de la época para ofrecer programabilidad al usuario; y por otra parte, desde hace poco han empezado a aparecer circuitos analógicos programables por el usuario o FPAAs (*Field Programmable Analog Arrays*) que intentan ofrecer las mismas ventajas que las FPGAs brindan en el caso de aplicaciones digitales.

La posibilidad de disponer de dispositivos programables para prototipar o incluso producir en determinados casos circuitos analógicos es especialmente motivadora. Dentro de la industria microelectrónica de hoy se estima habitualmente que más del 90% de los fallos de los circuitos integrados de señal mixta se deben a problemas en la parte analógica, lo cual evidencia el mayor riesgo de este tipo de circuitos en comparación con los puramente digitales, especialmente si las células analógicas no han sido utilizadas anteriormente en otros diseños. Un diseño robusto sobre células programables largamente estudiadas y cuidadosamente modeladas, y fácilmente comprobable por el propio usuario sin tener que pasar por un largo y costoso proceso de fabricación, podría ser la solución a estos problemas.

En resumen, la conveniencia de utilizar elementos programables – FPGAs, FPAAs y microprocesadores – para el prototipado en incluso fabricación en determinados casos de aplicaciones de señal mixta de carácter industrial radica en la reducción drástica de costes fijos de fabricación y en la reducción del tiempo de desarrollo y prototipado de cada uno de los tres bloques constitutivos del sistema global. Sin embargo, cada uno de estos tres elementos programables sigue precisando herramientas y metodologías distintas y su comunicación sigue quedando a discreción del usuario.

Desde este punto de vista sería muy útil disponer de un circuito programable único que englobase elementos programables capaces de implementar cada uno de estos tres bloques. Dicho dispositivo, que constituiría lo que denominaríamos un sistema programable *monochip* o FIPSOC (*Field Programmable System-On-Chip*), mostrado esquemáticamente en la figura 1.1., incluiría una FPGA, una serie de elementos analógicos programables, y un microprocesador.

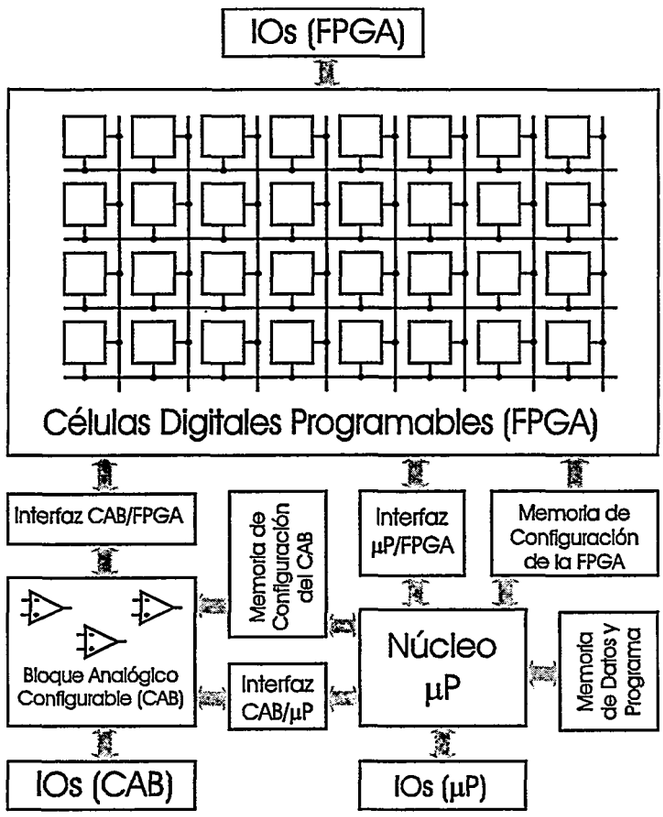

Fig. 1.2: Diagrama de bloques de un FIPSOC (*Field Programmable System-on.chip*)

Mediante este tipo de sistema se solucionaría no solamente el problema de los costes fijos de producción de los circuitos de aplicación específica gracias a la programabilidad del dispositivo, sino además el problema metodológico de integración de los tres subsistemas. Las interfaces entre los tres dominios estarían bien definidas y optimizadas, y cada uno de los tres bloques

programables estaría pensado *ad hoc* no sólo para su funcionamiento óptimo de forma aislada sino para su interacción con los demás. Las herramientas utilizadas para el diseño y programación en cada dominio serían consistentes entre sí y estarían integradas de forma natural, por lo que la metodología sería consistente y compacta. En particular, la comunicación entre la parte digital programable y el microprocesador se realizaría de forma clara y establecida, permitiendo implementar de forma sencilla y rápida periféricos y coprocesadores que asistan al microprocesador en sus tareas de control y cálculo, situación habitual en las aplicaciones industriales a pesar de la enjundia y el gasto de valiosas células de entrada-salida que implica. Más aún, esta comunicación permitiría al microprocesador controlar los procesos de reconfiguración dinámica y parcial que como antes se mencionó resultan necesarios para la viabilidad de las implementaciones basadas en FPGAs.

Dentro de este contexto esta tesis estudia el diseño e integración de arquitecturas programables capaces de ser utilizadas para la implementación de aplicaciones industriales de señal mixta, así como la metodología integrada de diseño y verificación sobre este tipo de arquitecturas. Se propone por tanto integrar arquitecturas programables y herramientas de diseño con los objetivos claros de eliminar los costes de fabricación y de reducir y simplificar el ciclo de desarrollo. Se estudian los aspectos relacionados con el diseño aislado de cada uno de estos tres dominios con el fin de optimizar su interacción subsiguiente, así como la problemática de su integración en un mismo substrato semiconductor. Finalmente, se exploran las técnicas de configuración y reconfiguración más beneficiosas tanto en términos de funcionalidad de la arquitectura como de integración y mejora de la metodología frente a las soluciones existentes en el actual estado del arte. La tesis se culmina con la introducción de la primera implementación práctica de un circuito de este tipo. En particular se estudian los aspectos relacionados con su implementación física y se reportan los resultados obtenidos en términos de área y prestaciones.

El sistema propuesto explota así la integración de los tres subsistemas con el doble fin ya mencionado: la integración y la programabilidad contribuyen a reducir área, problemas de calidad y tiempo de prorotipado, reduciendo dramáticamente el coste del sistema para prototipado y producción en determinadas condiciones; las mejoras metodológicas que se obtienen de la integración de los tres subsistemas se deben fundamentalmente al gran acoplamiento entre ellos:

- El microprocesador sirve como puente entre una estación de trabajo utilizada para el desarrollo y el resto del sistema en el chip, con lo cual todo el sistema de desarrollo podrá realizarse con una estación de trabajo (un PC típicamente) y el propio chip.

- El microprocesador controla la configuración de las células, por lo que permitirá realizar aplicaciones de *hardware virtual* como se ha descrito.
- El fuerte acoplamiento entre el microprocesador y las células programables permitirá una fuerte interacción entre el *hardware* y el *software*, creando un nuevo paradigma de soluciones puramente *hardware*, puramente *software*, o mixtas, pudiendo elegirse en cada momento la óptima.

En resumen, el tipo de sistemas monochip que se propone contribuye a incrementar la productividad de los diseñadores de aplicaciones industriales de señal mixta debido a la reducción de coste y ciclo de diseño y mejora de la metodología general.

1.3 Precedentes y estado del arte

Debido a la popularidad del uso de microprocesadores y microcontroladores en aplicaciones *hardware* de control industrial, adquisición de datos, comunicaciones, etc., es bastante habitual encontrar en la actualidad sistemas basados en microprocesadores y FPGAs. Las últimas se utilizan en estos sistemas para implementar la lógica de control que antes se construía mediante circuitos discretos, que a veces es controlable desde el mismo microprocesador en forma de dispositivo síncrono. Sin embargo, a pesar de la utilidad que tendría una FPGA con un microprocesador incorporado en el mismo *chip* [STA95], ninguna arquitectura de este tipo ha sido explotada comercialmente hasta el momento o ha sido reportada por ningún grupo de investigación.

Además de las típicas de control industrial, una de las aplicaciones más interesantes de la lógica dinámicamente reprogramable es la de implementar coprocesadores reconfigurables para microprocesadores en función de las necesidades de computación en cada momento. En general, los precedentes existentes sobre arquitecturas que incorporan microprocesadores con FPGAs han estado predominantemente dirigidos a este fin, más que a cubrir las necesidades habituales de los usuarios industriales que utilizan FPGAs y microprocesadores de propósito general para resolver aplicaciones de control, comunicaciones, etc. De hecho, este tipo de arquitecturas han sido desarrolladas en el entorno de los FCCMs más que en el entorno de las FPGAs de propósito general, con el propósito de aumentar la capacidad de computación de los microprocesadores a que complementan. En estos casos se dice que los microprocesadores y las FPGAs están fuertemente acoplados [DeH94] [DeH96A], de tal forma que los recursos programables (la

FPGA) pueden formar parte de la misma ruta de datos del microprocesador a que complementan e incluso modificar dinámicamente su *set* de instrucciones [ATH93] [WAZ93]. Para estudiar y clasificar estas arquitecturas y cualquier otra que contenga partes fijas y partes reconfigurables, la capacidad de computación fue utilizada como figura de mérito en función de parámetros tales como la capacidad de reconfiguración, granularidad, etc. en lo que André DeHon llamó el espacio RP de las arquitecturas dinámicamente reconfigurables [DeH96a].

Entre los trabajos destinados a integrar microprocesadores y FPGAs podemos destacar los siguientes:

- La familia 6200 de Xilinx incorpora un interfaz completo para microprocesador [CHU95]. Como ya se ha comentado, la memoria de configuración es accesible desde un bus controlable por un microprocesador externo, lo cual permitiría, al menos teóricamente, la incorporación de un microprocesador en el mismo chip. El interfaz de memoria de configuración es a su vez configurable permitiendo trasferencias de hasta 32 bits, lo que permite tiempos de reconfiguración del chip completo tan bajos como 100 μs. Además permite seleccionar varias células a la vez y escribir en ellas la misma configuración, lo que acelera la reconfiguración de aplicaciones regulares como *arrays* sistólicos o estructuras segmentadas, espcialmente del tipo utilizado en aplicaciones de video.
- La arquitectura PRISM (*Procesor Reconfiguration through Instruction-Set Metamorphosis*) implementada en la Universidad de Brown acoplaba una FPGA de Xilinx (XC3090) con un microprocesador 68010 de Motorola. Era capaz de procesar funciones lógicas de un bit de anchura 20 veces más rápido que un 68010 [ATH93]. El proyecto evolución a una segunda generación (PRISM II) que acoplaba una arquitectura de cuatro FPGAs al 68010 [WAZ93].
- En 1994 se comenzó a pensar en sistemas de microprocesadores fuertemente acoplados a lógica programable. Los ejemplos más claros son el trabajo de André DeHon que proponía integrar su DPGA [TAU95] con un microprocesador a su vez reconfigurable [DeH94], y la arquitectura PRISC (*Programmable Reduced Instruction Set Computer*) desarrollada en la Universidad de Harvard [RAZ94]. El trabajo de DeHon resulta interesante en tanto en cuanto identifica el ancho de banda y la latencia del interfaz entre la lógica programable y la de funcionalidad fija. La arquitectura PRISC, aunque sólo ha sido explorada en simulaciones, tiene una arquitectura claramente definida que extiende la ruta de datos de un miprocesador RISC a 200MHz mediante unidades funcionales programables (PFUs) en paralelo con las unidades funcionales (FU) de estructura fija. De esta manera la ruta de datos del procesador

puede adaptarse durante la ejecución de instrucciones a las necesidades computacionales de cada momento.

- Un caso de microprocesador con coprocesador reconfigurable integrado en un *chip* es el proyecto *OneChip* propuesto por la Universidad de Toronto en 1995 [WIT95]. El *chip* en cuestión no fue realmente fabricado sino emulado en la plataforma *Trasmografier I* desarrollada en la misma universidad. En este trabajo se estudia de forma clara el problema de la comunicación entre el microprocesador y el coprocesador programable. Se predijeron aumentos de la velocidad de ciertas aplicaciones de factores de hasta 40, con incrementos de área estimados en al menos un 350%.

- En 1996 se reportó un procesador de video implementado sobre FPGAs de la familia CLAy que se reconfiguraba dinámicamente en función del *software* que tenía que correr en cada momento [WIR96]. Las distintas configuraciones utilizadas en cada aplicación se guardaban en una arquitectura caché, lo que minimizaba el impacto del proceso de reconfiguración sobre la operación del circuito.

- Actualmente se está trabajando en la Universidad de Berkeley en el proyecto BRASS, que intenta crear microprocesadores autoadaptativos totalmente reconfigurables y automodificables en función del código. Aún no existen resultados concretos de este proyecto.

- En 2000 Atmel anunció las primeras muestras del llamado FPSLIC (Field Programmable System Level Integrated Circuit), un producto estándar que reúne un microprocesador AVR de 8 bits y un substrato programable de propósito general – con la arquitectura de su familia de FPGA AT6000 focalizada fundamentalmente en aplicaciones de procesamiento de señal. El producto dispone a su vez de una metodología de diseño y emulación que integra las herramientas anteriormente existentes para el microprocesador y para la FPGA.

- A finales de 2000 Altera anunció sus planes de desarrollar la nueva familia de FPGAs Excalibur, un chip que incluirá un microprocesador ARM7TDMI, una FPGA de la popular familia EP20K y una importante cantidad de memoria. El diseño está dirigido al mercado de la emulación y fabricación en pequeñas series de sistemas de comunicaciones típicamente basados en ARM.

Como puede verse, ninguna de estas arquitecturas está orientada a sistemas de control, y la metodología en todos los casos resulta demasiado complicada como para pensar en una aplicación practica de forma realista.

1.4 Organización de esta tesis

Esta tesis está organizada en seis capítulos. En este primer capítulo se introduce el trabajo que se va a presentar, las motivaciones que han llevado a hacerlo, y las líneas de investigación relacionadas con el tema propuesto que se han llevado o se están llevando a cabo en otros grupos de trabajo académicos e industriales. En el capítulo dos se analizará en detalle el diseño de células programables digitales desde un punto de vista estructural y funcional, y se discutirán detalladamente esquemas para almacenar los datos de configuración de los distintos elementos programables, dejando para el capítulo tres los aspectos relacionados con la interacción entre células programables y microprocesador. El capítulo cuatro trata de los problemas relacionados con la implementación física de los distintos subsistemas propuestos y reporta los resultados obtenidos en términos de área y caracterización de la primera realización práctica de un sistema de este tipo. El capítulo cinco está dedicado las aplicaciones típicas que se han propuesto para este tipo de sistemas, y finalmente el capítulo seis extrae las conclusiones de esta tesis y presenta las líneas de trabajo futuro, algunas de las cuales ya están empezando a acometerse. Un apéndice complementa el manuscrito para introducir de forma general las arquitecturas programables digitales o FPGAs y sus esquemas de configuración y reconfiguración, servir de referencia en cuanto a la terminología utilizada en el resto de la tesis, y presentar una breve introducción histórica a las arquitecturas programables con especial énfasis en los aspectos relevantes al tema de la tesis.

1.5 Resumen

En este primer capítulo hemos presentado el marco general sobre el que se desarrollará esta tesis, así como la motivación que subyace y los resultados que se buscan. La principal aportación que se propone consiste en el estudio del diseño e implementación práctica de dispositivos monolíticos que integran un microprocesador, una FPGA dinámicamente reconfigurable y un conjunto de células programables analógicas. Este concepto es totalmente novedoso a la vista del estado del arte actual en el campo de las arquitecturas programables, y supondrá un avance no sólo en términos de integración de sistemas sino de metodología, debido a las técnicas de diseño de *hardware* virtual y de co-diseño *hardware-software* que permite desarrollar.

CAPÍTULO 2

DISEÑO DE CÉLULAS PROGRAMABLES DIGITALES Y ANALÓGICAS

2.1 Introducción

Los criterios existentes para el análisis y diseño de arquitecturas digitales programables, tales como los utilizados por Rose y sus colaboradores [ROS90a] [ROS90b] [ROS91] [BRO92a] [SIN91] [SIN92] [CHU91] [KAV96] [BET97], se refieren a arquitecturas programables de propósito general, y han sido extraídos de forma experimental a partir de implementaciones de colecciones de *benchmarks* [MCN91] y estimaciones razonadas de las áreas y velocidades de los distintos bloques constitutivos de la célula programable.

En nuestro caso, el tipo de *arrays* programables que se proponen responden a un propósito más particular, y si bien se ha seguido en la medida de lo posible estos criterios, en este caso había nuevas variables a incluir en el análisis de coste de cada posible solución. Estas variables, de difícil evaluación y cuantificación, se refieren a aspectos tales como la capacidad de la célula programable de ser dinámicamente reconfigurada, la posibilidad de tratar su memoria de configuración como memoria de uso general para el microprocesador, o la facilidad y rapidez con que las señales de la célula programable pueden ser leídas y escritas por un microprocesador. En este sentido, el espacio de soluciones posibles para nuestro problema, como se mencionó en el capítulo anterior, se hace grande y de difícil cuantificación. No obstante, podemos considerar

que, de forma general, una característica básica perseguida en este diseño es la capacidad de interacción con un microprocesador incorporado al mismo circuito integrado.

En este capítulo se acomete en detalle el diseño de las células programables utilizadas en una arquitectura de tipo FIPSOC. Se explican los criterios seguidos durante el diseño de los bloques programables digitales y analógicos y se propone un nuevo esquema de configuración basado en contextos *buffer* especialmente pensado para facilitar la reconfiguración dinámica multicontexto y la reutilización de la memoria de configuración como memoria de propósito general. El capítulo se centra en los aspectos arquitecturales de los bloques programables digitales y analógicos y en sus recursos de configuración, dejando la comunicación entre estos bloques y el microprocesador y los aspectos relacionados con la implementación física para capítulos posteriores.

2.2 Estructura general de una arquitectura de tipo FIPSOC

La figura 2.1 muestra un esquema general de bloques de una arquitectura de tipo FIPSOC. La FPGA está rodeada de células programables de entrada/salida excepto por el lado por el que se conecta a la parte analógica programable y al microprocesador.

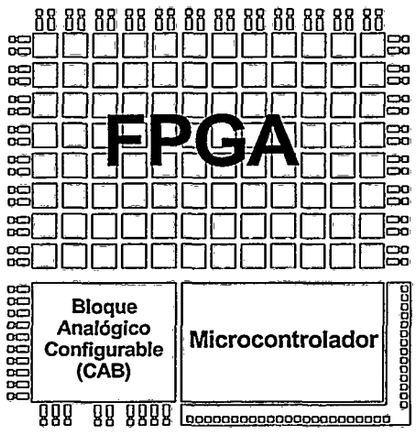

Fig. 2.1 Esquema general de una arquitectura de tipo FIPSOC

El esquema presentado arriba está particularizado para una arquitectura con un solo microprocesador, una FPGA y un solo bloque analógico configurable relativamente aislado del resto. En cualquier caso el esquema puede ser generalizado para configuraciones con varios de estos elementos. El tipo de FPGA a implementar debe ser elegido en función del tipo de

aplicaciones a que se dirige el *chip* en particular (de propósito general, para comunicaciones, para procesamiento de señal, etc.), si bien lo que en cualquier caso debe favorecerse es la capacidad de ser configurada y controlada desde un microprocesador.

2.3 Diseño de FPGAs para arquitecturas de tipo FIPSOC

2.3.1 Características generales

Como se mencionó en el capítulo introductorio, en principio nos centraremos en arquitecturas programables susceptibles de ser utilizadas en aplicaciones típicas de control industrial más que en aplicaciones de computación intensiva. La discusión girará en torno a ejemplos de realizaciones prácticas de elementos programables en vez de introducir la problemática del diseño de FPGAs desde una perspectiva totalmente general, lo cual irá dando pie a estudiar los aspectos más interesantes de su diseño que podrán ser generalizados a otros casos particulares de células programables susceptibles de ser utilizadas en arquitecturas de tipo FIPSOC basadas en microprocesador.

En términos generales, podemos afirmar que la FPGA que se describirá es de alta granularidad, basada en LUTs, implementada a partir de memoria RAM estática, dinámicamente reconfigurable, y más o menos orientada a operaciones con *nibbles* o pedazos de cuatro *bits*. Soporta operación multicontexto y reconfiguración dinámica y parcial. Esta configuración particular de FPGA resulta especialmente adecuada para aplicaciones típicas de control industrial que utilizan microcontroladores de 8 *bits* habituales tales como el i8051 o el MC68HC11.

El bloque básico programable se denomina DMC (*Digital Macro Cell*), y su granularidad puede considerarse alta ya que contiene cuatro LUTs de cuatro entradas y cuatro *filp-flops*, lo cual favorece la operación a nivel de *nibble*. No obstante, la relación e interconexión de estos elementos es lo suficientemente flexible y general como para permitir una operación eficiente a nivel de *bit*, y para poder usar de forma más o menos independiente las partes combinacional y secuencial. Se proveen una serie de modos de *macro* que permiten implementar circuitos de cuatro *bits* conectables en cascada tales como sumadores, contadores, registros de desplazamiento, etc. De esta forma resulta sencillo trabajar con anchuras mayores de bus. La

conveniencia de este tipo de arquitecturas para este rango de aplicaciones fue demostrada por Britton [BRI94] primero y generalizada para mayores complejidades por Singh [SIN97] después.

Desde el punto de vista de una arquitectura de tipo FIPSOC, lo crucial estriba en que las salidas combinacionales y secuenciales estén mapeadas en el espacio de direccionamiento de memoria del microprocesador para permitir una fuerte interacción entre *hardware* y *software*. Estas posiciones de memoria se utilizan así como puntos de comunicación con el *hardware* desde los programas del microprocesador, por ejemplo para implementar coprocesadores rápidos que complementen rutinas de cálculo numérico en algoritmos de procesamiento de señal, o como generalización del concepto de puerto de entrada/salida paralelo a todo el *hardware* programable.

La FPGA soporta reconfiguración dinámica multicontexto. Utilizando la terminología introducida por Lysaght [LYS93], más allá del concepto de reconfiguración dinámica la operación multicontexto se consigue duplicando (o repitiendo un número mayor de veces) la memoria que contiene la información de configuración de las células programables. Definimos un contexto como la información necesaria para configurar una célula programable, un conjunto de ellas, o la FPGA entera. Al menos dos ventajas se derivan de la duplicación de la memoria de configuración para disponer de más de un contexto de programación:

- El tiempo de reconfiguración se reduce drásticamente, normalmente a un ciclo de reloj o incluso al tiempo de retraso de una puerta lógica compleja, pues basta con multiplexar la información de reconfiguración entre los dos (o más) contextos. El área ocupada por el *hardware* virtual (como se define en el apéndice A) sería la de la memoria necesaria para albergar el contexto inactivo, permitiendo reaprovechar el área correspondiente a los canales de rutado y células programables en sí, que serían utilizadas siempre por el contexto activo en cada momento.

- El contexto que no está siendo usado en un cierto momento del tiempo puede reconfigurarse sin tener que parar la operación del circuito, que estaría funcionando utilizando la información de configuración contenida en el otro contexto. Esto permite paralelizar las tareas de reconfiguración al próximo contexto y operación real del contexto activo.

La rentabilidad de esta técnica se basa en el menor coste de los recursos de configuración respecto de los de rutado y de las células programables en sí, ya que si los primeros han de duplicarse el área que ocupan debe ser pequeña comparada con la de los recursos a compartir. El caso de las LUTs es distinto ya que una LUT es básicamente memoria, y por lo tanto duplicar la

memoria de configuración de una LUT equivale prácticamente a duplicar su área. En nuestro caso, cada LUT está internamente dividida en dos pedazos de igual tamaño, cada uno de los cuales se usa para implementar un contexto. Así, se puede elegir entre LUTs de tres entradas que soportan operación multicontexto, o LUTs de cuatro entradas que no la soportan.

También se soporta la reconfiguración parcial como se define en [LYS93]. La memoria de configuración de los DMCs puede manipularse desde el microprocesador a través de un banco de posiciones de memoria. Existen unos registros de máscara mediante los cuales pueden seleccionarse determinadas filas y columnas de DMCs para sobreescribir simultáneamente su memoria de configuración en cualquiera de sus dos contextos, o transferir el contexto activo solamente en los DMCs seleccionados por la máscara. Los conjuntos de DMCs seleccionables por la máscara son *rectángulos lógicos* o subconjuntos de DMCs {[x,y]} tales que si $[x_1,y_1]$ y $[x_2,y_2]$ están incluidos, entonces $[x_1,y_2]$ y $[x_2,y_1]$ también lo están. Como puede verse, este tipo de selección es adecuado para reconfigurar *arrays* sistólicos, ya implementados en FPGAs basadas en RAM [BOE96].

Dicho esto, resulta claro que la FPGA se configura por direccionamiento paralelo de memoria en vez de utilizar un *bitstream* serie clásico. En realidad, la configuración de la FPGA se realiza a través del microprocesador, por lo que las configuraciones se almacenan como programas en ensamblador o código máquina. De esta manera, en una arquitectura de tipo FIPSOC como la que se propone el microprocesador tiene una doble función: por una parte, se utiliza como elemento normal de diseño para ejecutar programas de usuario; por otra parte, se utiliza para controlar, configurar y reconfigurar dinámicamente el *hardware* programable.

2.3.2 Estructura general

En la figura 2.2 se muestra la estructura general de la FPGA incluida en una arquitectura de tipo FIPSOC. La arquitectura es modular para poder soportar *arrays* de distinto número de filas y columnas, si bien por motivos que se detallarán en secciones posteriores resulta especialmente conveniente que uno de estos dos números (*M* ó *N* en la figura) sea una potencia de dos (2, 4, 8, 16, etc.)

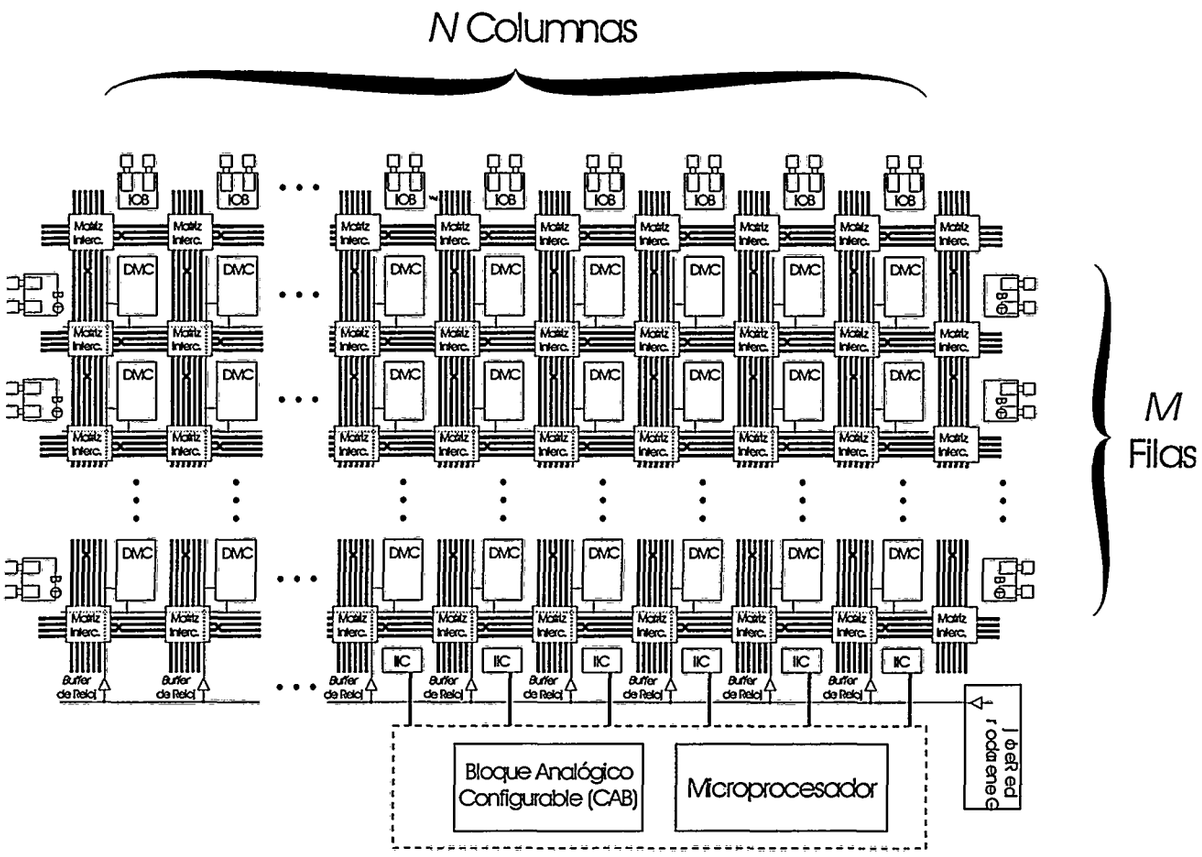

Fig. 2.2 Estructura genérica de la FPGA de FIPSOC

La parte superior y los laterales de la FPGA están rodeados de bloques de entrada-salida o IOBs (*Input-Output Blocks*), uno por fila o columna, cada uno de los cuales incorpora dos células programables de entrada salida o IOCs (*Input-Output Cells*) con sus correspondientes *bonding pads* (que se conectan directamente a las patillas externas del encapsulado del *chip*). La parte inferior está destinada a conexiones con el microprocesador y con el bloque analógico, a través de las células de interconexión interna o IICs (*Internal Interconnection Cells*). Este esquema es generalizable a cualquier otro caso particular de arquitectura FIPSOC, dado en cualquier caso la FPGA deberá estar rodeada por células programables de entrada salida (IOBs) y por células de interfaz con el microprocesador y con el bloque analógico.

A continuación se describen las arquitecturas propuestas para el bloque programable (DMC), el bloque de entrada salida (IOB), de los recursos de rutado de cada uno de los bloques, y de las conexiones con el microprocesador y con la lógica programable (IICs).

2.3.3 El DMC

La figura 2.3 muestra un ejemplo de diagrama de bloques de DMC o célula básica digital programable. Se compone de cuatro LUTs de cuatro entradas y cuatro *flip-flops* de dos entradas, además de los recursos internos de rutado necesarios para interconectar dichos elementos. En total se proveen doce entradas directas a las LUTs (compartidas de una determinada manera), cuatro entradas de datos (que pueden servir como entradas directas a los *flip-flops*), cuatro terminales de control y dos entradas auxiliares para la parte secuencial, y seis salidas, cuatro principales y dos auxiliares.

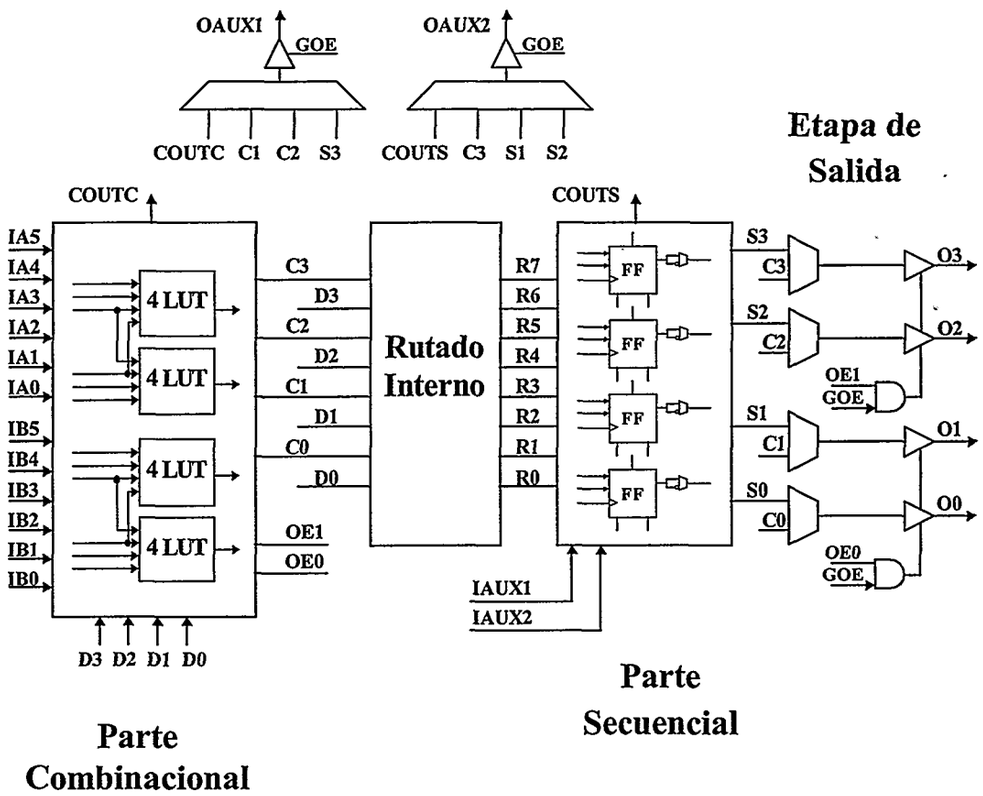

Fig. 2.3 Diagrama de bloques del DMC

La posibilidad de disponer de una parte combinacional y otra secuencial de granularidad similar conectables por recursos de rutado internos para poder usarlas de forma más o menos

independiente fue ya estudiada por Britton [BRI94] y Singh [SIN97] con excelentes resultados. Por otra parte, el tamaño ideal de cuatro entradas para las LUTs fue establecido por Brown y Rose [BRO92a] [ROS90a], mientras que Betz [BET97] demostró las ventajas que se obtienen compartiendo algunas de estas entradas (en particular dos para LUTs de cuatro entradas) entre pares de LUTs, en lo que llamó *clusters* o agrupaciones de dos a cuatro LUTs.

El bloque combinacional genera además dos señales internas **OE1** y **OE0** que controlan la operación tri-estado de los *buffers* de salida de la célula. Esta operación tri-estado está pensada para permitir la formación de bloques grandes de memoria RAM a partir de partes combinacionales de DMCs configuradas en modo memoria, como se explicará en el párrafo 2.3.3.1.1. La utilización de la capacidad tri-estado para la generación de funciones lógicas anchas cortocircuitando salidas (*wired-or*) está fuertemente desaconsejada por lo impredecible que resulta su modelización temporal, como ocurre en las FPGAs disponibles en el mercado en las que esto es posible.

Finalmente, existe una señal global **GOE** (*Global Output Enable*) que se conecta a todos los DMCs, IOBs e IICs, cuya misión es la de inhabilitar todas las salidas de todos los bloques funcionales de la FPGA. De esta forma es posible mantener todas las salidas inhabilitadas durante la secuencia de *reset* del microprocesador y durante la fase de programación inicial del dispositivo.

2.3.3.1 Parte combinacional

La parte combinacional del DMC se compone en realidad de 64 bits de memoria de doble puerto cuya organización es altamente flexible para soportar una serie de funciones combinacionales básicas. La organización de esta memoria se configura mediante multiplexores construidos con conmutadores NMOS y CMOS, que a su vez se gobiernan mediante *bits* de configuración o combinaciones de ellos.

Además se incluye una circuitería de *carry-look-ahead* fija [WES93] para permitir la implementación de funciones aritméticas básicas (sumadores y restadores) rápidas.

2.3.3.1.1 Estructura de las LUTs

Como se menciona en el apéndice A [ROS90a] [BRO92a], la granularidad óptima desde el punto de vista de aprovechamiento del área del *array* programable se consigue cuando las LUTs tienen entre tres y cuatro entradas, mientras que las mejores prestaciones (mínimo retardo crítico del circuito implementado) se obtienen cuando las LUTs tienen entre cuatro y cinco entradas. Por esto, parece adecuado basar la parte combinacional de la FPGA en LUTs de cuatro entradas de tal forma que puedan combinarse para formar LUTs de granularidades mayores, en particular de cinco e incluso de seis entradas.

La figura 2.4 muestra la estructura propuesta para la parte combinacional del DMC según el diagrama de bloques propuesto en la figura 2.3 . Puede verse que las líneas de selección **IA3** e **IB3** no entran directamente a las LUTs sino que pasan primeramente por un multiplexor configurable, de tal forma que las señales que verdaderamente se usa en las LUTs como líneas de selección, etiquetadas como **x_IA3** y **x_IB3**, son elegidas entre **IA2** e **IB2** ó **IA3** e **IB3**. En los modos estáticos este multiplexor está permanentemente configurado de tal forma que **x_IA3** y **x_IB3** son idénticamente **IA3** e **IB3**, mientras que en los modos dinámicos las señales **IA3** e **IB3** se utilizan para ejecutar operaciones de reconfiguración desde el propio *hardware*.

En general las LUTs se construyen a base de células de memoria y multiplexores que permiten acceder a ellas de distinta forma según el modo de configuración elegido en cada momento. En una arquitectura como la propuesta cuyo objetivo es el de ser utilizada junto a un microprocesador, el acceso a las células de memoria para su configuración está separado de su arquitectura funcional interna mostrada en la figura 2.4 . La interacción entre el microprocesador y los datos internos de las LUTs será explicada en profundidad en el capítulo siguiente.

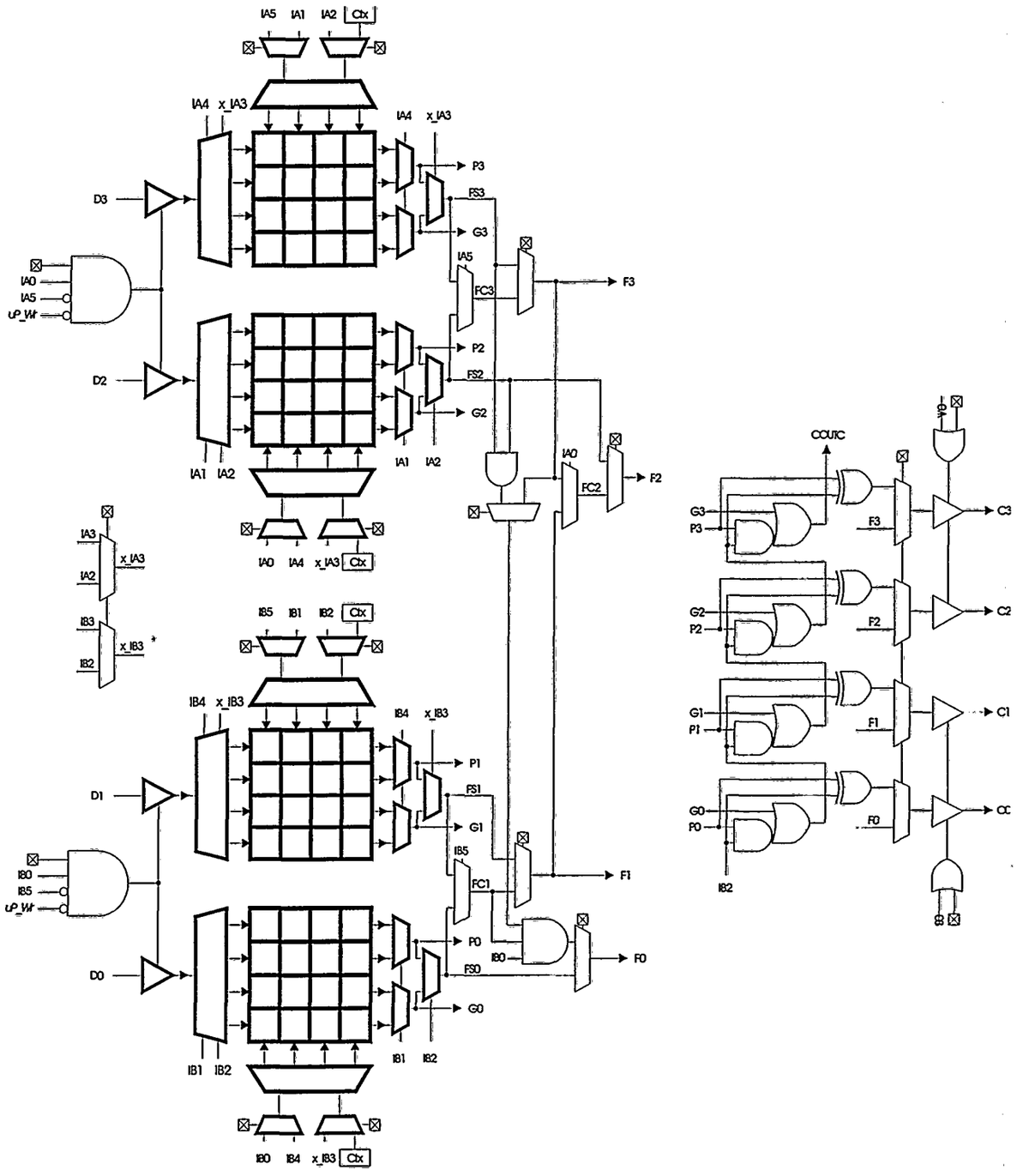

Fig. 2.4 Estructura de la parte combinacional del DMC

Cada una de las cuatro LUTs está formada por 16 células de memoria organizadas en forma de *array* de cuatro por cuatro (cada una de las matrices sombreadas de cuatro por cuatro en la figura 2.4 es una LUT). La célula elegida es de doble puerto, de tal forma que uno de los dos puertos está permanentemente conectado al microprocesador, gracias a lo cual pueden compartirse los datos almacenados en las células de memoria entre los programas del microprocesador y el

circuito programable propiamente dicho, permitiendo la interacción *hardware-software*. Además, esto permite al microprocesador reconfigurar total o parcialmente las LUTs sin tener que detener la operación normal del circuito, que se llevaría a cabo por el otro de los puertos. Las cuatro líneas de selección de cada memoria de 16x1 (2^4 x 1) constituyen las entradas a la LUT, que seleccionarán el resultado elegido para cada combinación de las entradas (es decir, el resultado de la función lógica implementada a partir de los valores de las variables de entrada). Estas líneas de selección, de las cuales dos direccionan la memoria de cuatro palabras de cuatro bits (en vertical en la figura 2.4) y otras dos controlan un multiplexor de cuatro a uno que elige uno de esos cuatro *bits* (en horizontal en la figura 2.4), pueden configurarse en función del modo de operación proyectado. En función del estado de los *bits* de programación (marcados con un cuadrado aspado) se obtienen los distintos modos de configuración. En el caso de la arquitectura combinacional mostrada en la figura 2.4, en total se soportan 33 modos de funcionamiento distinto que se resumen en la tabla 2.1 .

Modo de Funcionamiento	Características	Figura
Simple Estático	4 LUTs de 4 entradas (2 compartidas por cada 2 LUTs), no multicontexto	2.5
Complejo Estático	2 LUTs de 5 entradas (independientes), no multicontexto	2.6
Multiplexor Estático	2 multiplexores de 4 a 1 (independientes), no multicontexto	2.7
Memoria Estática	2 memorias de 16x2 (independientes) con salidas tri-estado, no multicontexto	2.8
Modos mixtos estáticos	12 combinaciones de los elementos anteriores en las dos mitades de la parte combinacional, sin operación multicontexto	2.9
Simple Dinámico	4 LUTs de 3 entradas (independientes), con multicontexto	2.11
Complejo Dinámico	2 LUTs de 4 entradas (independientes), con multicontexto	2.12
Multiplexor Dinámico	2 multiplexores de 4 a 1 (independientes), con multicontexto	2.13
Memoria Dinámica	2 memorias de 8x2 (independientes) con salidas tri-estado, con multicontexto	2.14
Modos mixtos dinámicos	12 combinaciones de los elementos anteriores en las dos mitades de la parte combinacional, con operación multicontexto	2.15
Modo sumador	Sumador de 4 *bits* con o sin multicontexto	2.16

Tabla 2.1 : Resumen de los modos de funcionamiento de la parte combinacional del DMC

Los modos estáticos se obtienen cuando el *bit* de cambio de contexto (marcado como **Ctx** en la figura 2.4) no interviene en la multiplexación de las señales internas de las LUTs. Las figuras 2.5 a 2.8 muestran los modos estáticos básicos tal y como se presentan a los ojos del usuario final. Combinando distintos modos en las mitades superior e inferior de la parte combinacional se obtienen los distintos modos mixtos (doce en total), de los que la figura 2.9 muestra un ejemplo.

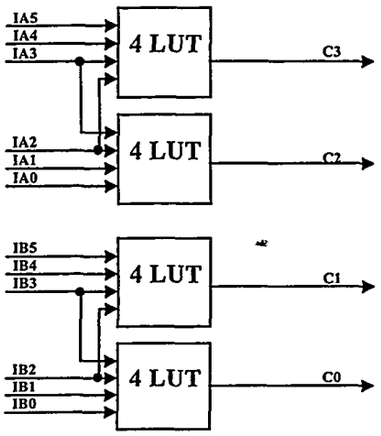

Fig. 2.5 Modo simple estático

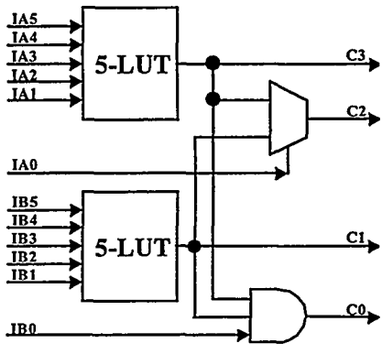

Fig. 2.6 Modo complejo estático

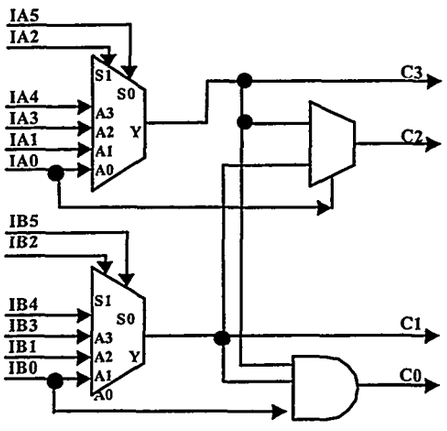

Fig. 2.7 Modo multiplexor estático

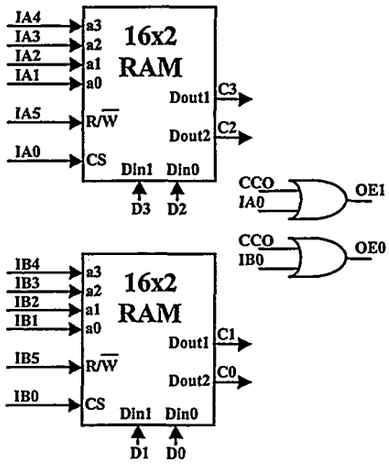

Fig. 2.8 Modo memoria estática

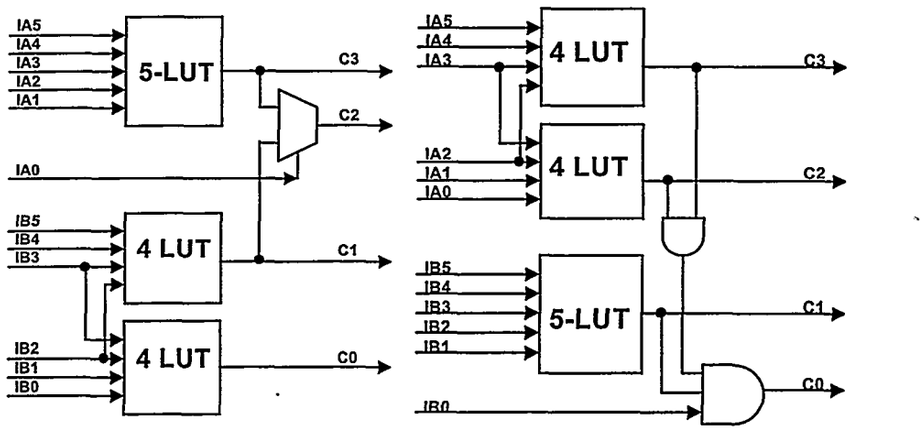

Fig. 2.9 Modos mixtos estáticos complejo-simple y simple-complejo

En los modos dinámicos cada LUT se divide en dos mitades destinadas cada una a un contexto, lo cual hace que las funciones implementables en estos modos tengan, en general, una entrada menos que en sus modos estáticos correspondientes. Como se muestra en la figura 2.4, en los modos dinámicos se usa un *bit* de configuración (marcado como **Ctx** en la figura) directamente conectado a una de las líneas de selección de las LUTs (en vertical en la figura 2.4), lo cual selecciona una de las dos mitades de cada LUT de cuatro entradas, lo que es equivalente a disponer de una LUT de tres entradas cuya configuración puede variarse rápidamente en función del *bit* de contexto. El esquema lógico del bloque se muestra en la figura 2.10: se dispone de dos versiones F_1 y F_0 de la función lógica de salida F_S, una para cada contexto. El contexto activo, almacenado en el *bit* de configuración **Ctx**, selecciona entre estas funciones.

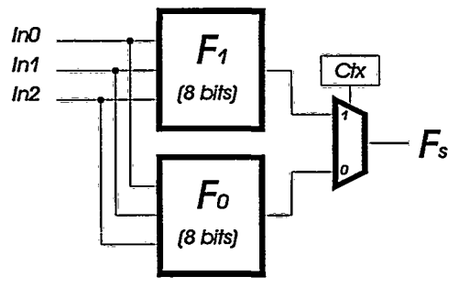

Fig. 2.10 Estructura lógica de una LUT bi-contexto de tres entradas

La figura 2.4 muestra como las líneas de selección son reordenadas en los modos dinámicos para poder aprovechar todas las entradas ya que ahora las funciones lógicas son más pequeñas. En particular, dado que se dispone de seis entradas en cada mitad de la parte combinacional, y que en el modo simple dinámico se tiene dos LUTs de tres entradas en cada mitad, se puede obtener dos funciones de tres entradas totalmente independientes en la mitad de una parte combinacional, como se muestra en la figura 2.11. Las figuras 2.11 a 2.15 muestran los correspondientes dinámicos a los modos explicados en las figuras 2.5 a 2.9, incluyendo los modos mixtos dinámicos con dos mitades configuradas en distintos modos dinámicos. Dado que existe un sólo *bit* para configurar toda la parte combinacional como estática o dinámica, no son posibles los modos mixtos con una mitad en un modo estático y la otra en un modo dinámico, si bien esta combinación es teóricamente posible y ha sido eliminada únicamente a efectos de la implementación práctica. Es interesante notar que existe también un modo multiplexor dinámico de 4 a 1 ya que basta con una LUT de tres entradas para implementar cada uno de los dos multiplexores de 2 a 1 de que se compone.

Finalmente, es importante considerar que al dividir en dos las LUTs, en el modo complejo y en el modo memoria sobra un *bit* de selección. Debido a esto se ha incluido un multiplexor que selecciona las señales **x_IA3** y **x_IB3** de entre **IA3** ó **IA2** e **IB3** ó **IB2**. En los modos simple y multiplexor se utilizaría **IA3** e **IB3** dado que todas las entradas son usadas, mientras que en el modo complejo y en el de memoria se usa **IA2** e **IB2** quedando **IA3** e **IB3** sin utilizar. En la sección 2.5 (Configuración de Arquitecturas Programables) se explicará otra situación en la que se usa **IA2** e **IB2** en vez de **IA3** e **IB3**, relativa a la reconfiguración dinámica (cambio de contexto) asistida por *hardware*.

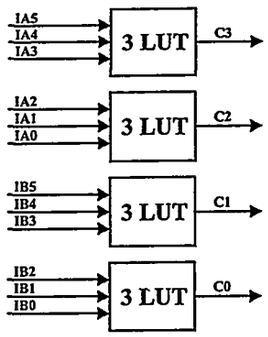

Fig. 2.11 Modo simple dinámico

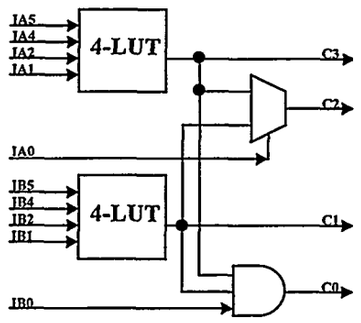

Fig. 2.12 Modo complejo dinámico

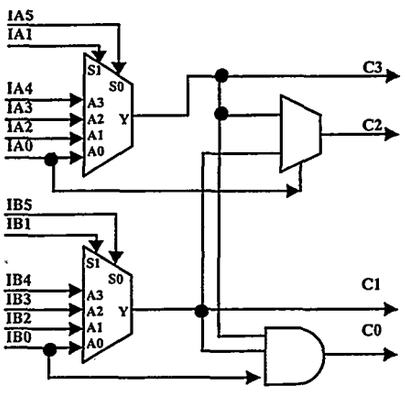

Fig. 2.13 Modo multiplexor dinámico

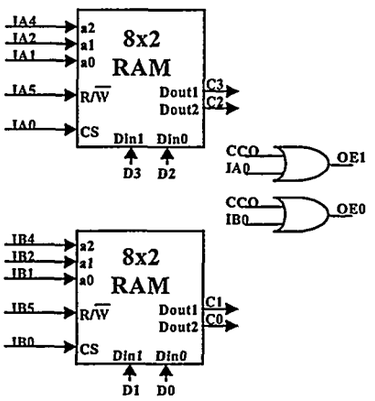

Fig. 2.14 Modo memoria dinámica

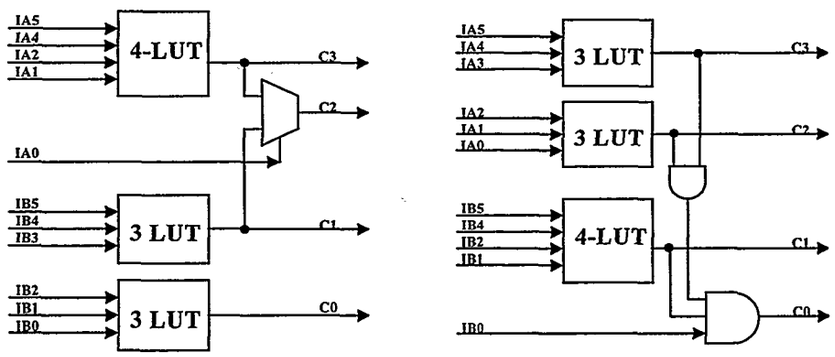

Fig. 2.15 Modos mixtos dinámicos complejo-simple y simple-complejo

Como muestra la figura 2.4, la parte combinacional del DMC dispone de una circuitería fija que implementa un sumador rápido del tipo *carry-look-ahead* como el descrito en [WES93]. La utilización de circuitería fija para apoyar las funciones computacionales de la lógica programable ya fue propuesta y utilizada en un gran número de FPGAs disponibles actualmente en el mercado [ALT99] [XIL99] [LUC97]. Las funciones de generación (G) y propagación (P) por cada *bit* tienen dos entradas que además son comunes, debido a lo cual es posible generar las dos funciones con una estructura de memoria de ocho *bits* dispuesta como bloque de cuatro por 2. Por tanto, basta con sacar dos salidas intermedias por cada LUT como se muestra en la figura 2.4. Como puede verse, el modo sumador funciona incluso en modo dinámico, ya que en este modo se dispone de ocho *bits* por LUT (en el modo simple las LUTs eran de tres entradas). Más aún, dado que las funciones P y G son programadas en los *bits* de las LUTs, es posible implementar variaciones de estas funciones para soportar otras operaciones aritméticas, por ejemplo la de restador sin más que considerar complementada una de las entradas y programar las funciones en consecuencia. La figura 2.16 muestra la parte combinacional del DMC configurada en modo sumador.

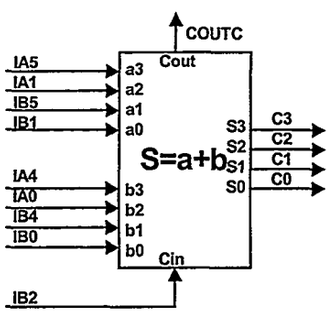

Fig. 2.16 Modo sumador

2.3.3.1.2 Estabilidad de las células de memoria

La figura 2.17 muestra la estructura utilizada para la célula básica de memoria de doble puerto. En principio todos los transistores utilizados son de la mínima longitud permitida por la tecnología (0.5μm en nuestro caso). En cambio, las anchuras deben ser cuidadosamente elegidas para asegurar la estabilidad de la célula en todas las situaciones.

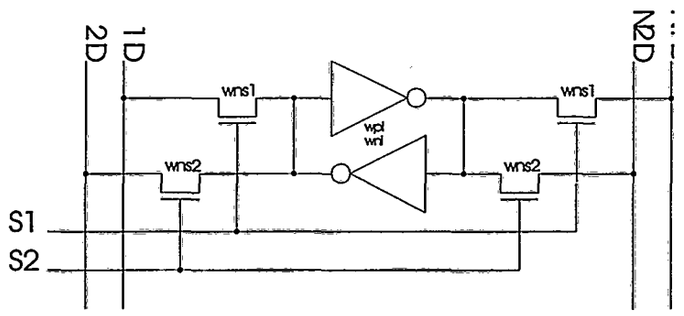

Fig. 2.17 Célula de memoria de doble puerto utilizada en las LUTs

Fundamentalmente, el problema del diseño de esta célula radica en el acceso totalmente asíncrono a los datos almacenados en las LUTs. Dado que las estructuras de memoria se utilizan para generar funciones combinacionales de propósito totalmente general, nada puede asegurarse sobre la temporización del acceso a la memoria, ya que el bus de direcciones, que constituye las entradas de la LUT (es decir, las variables de la función combinacional implementada), no va a mantenerse estable durante un intervalo garantizado de tiempo, y puede cambiar en cualquier momento, incluso en mitad de lo que sería un ciclo de lectura. A su vez, el otro puerto, aunque su temporización sí puede preveerse ya que está directamente conectado a un microprocesador, puede intentar accesos arbitrarios de lectura o escritura de forma totalmente independiente a los anteriores. Finalmente, dado que se permite que el mismo *hardware* programable escriba en las LUTs cuando se elige el modo memoria, varios accesos de igual o distinta naturaleza (lectura o escritura) podrían producirse al mismo tiempo, sin una temporización definida entre ellos.

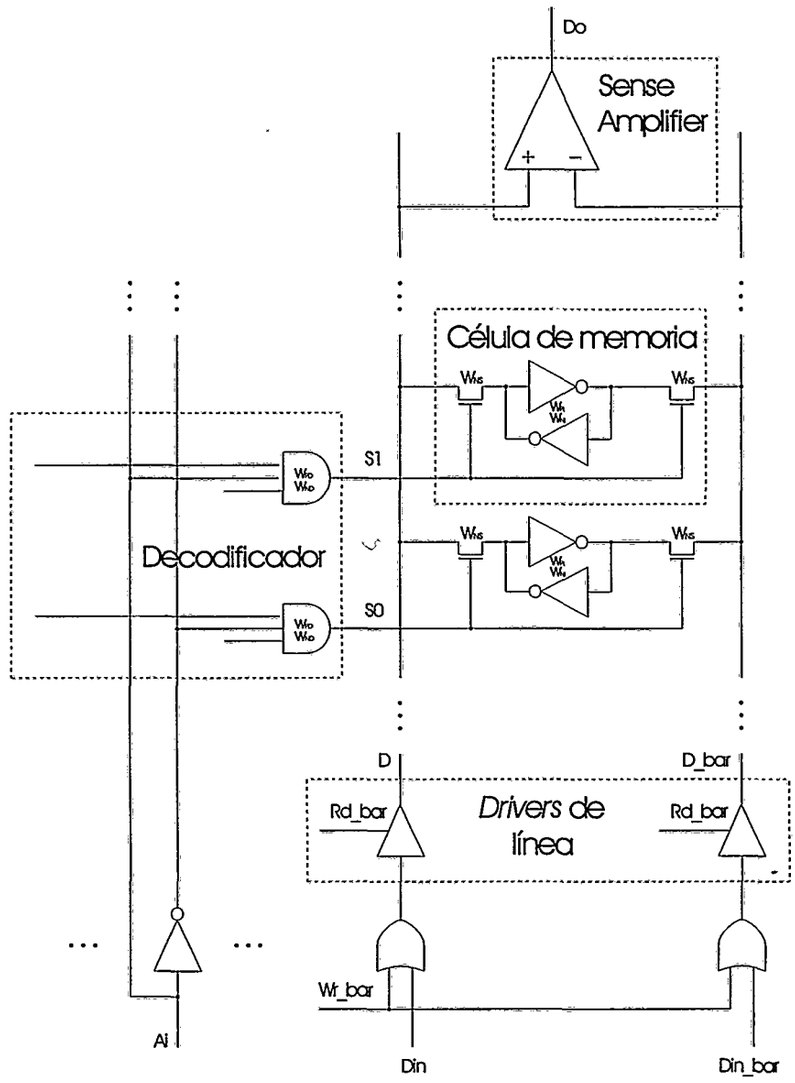

Fig. 2.18 Estructura general de un bloque de memoria

Para comprender la problemática del diseño de las LUTs es preciso tener presente los problemas que aparecen durante el diseño de una memoria de propósito general. La figura 2.18 muestra los distintos elementos que, de forma general, son necesarios para construir una memoria de un solo puerto (haría falta repetir cada uno de los elementos excepto los inversores realimentados de las células de memoria en sí para añadir más puertos). Los factores que intervienen en la estabilidad de estas estructuras son los siguientes:

- Tamaño de los inversores de la célula de memoria (W_{NI} y W_{PI}) : Se trata de la magnitud más crítica para la estabilidad de la célula. En general se utilizan tamaños pequeños, próximos a

los menores permitidos por la tecnología, pues el área de una célula de memoria suele estar minimizada ya que la célula se repite un gran número de veces y por tanto su influencia en el área del circuito es muy significativa, en especial para bloques grandes de memoria.

- Tamaño de los transistores de selección de las células de memoria (W_{NS}) : La anchura de estos transistores, cuya misión es la de comunicar las células de memoria en sí (los dos inversores realimentados) con las líneas de datos (D y D_bar en la figura 2.18), es de gran importancia en la estabilidad de la estructura. En los ciclos de escritura resulta beneficioso disponer de anchuras grandes para estos transistores, pues de esta forma permiten a los *drivers* de línea escribir rápidamente en las células de memoria debido a su baja resistencia, ya que deben sobreescribir los datos que las células de memoria intentan mantener. Sin embargo, las anchuras grandes son problemáticas en los ciclos de lectura, ya que abren un peligroso camino de baja resistencia entre los puntos estables de la célula de memoria, de poca fuerza, y las líneas de datos, de gran capacidad. Es importante recordar que estos transistores, al menos en el esquema propuesto, son de tipo NMOS, y por tanto sólo podrán forzar tensiones altas a V_{DD}-V_{TN} (siendo V_{TN} la tensión umbral de inducción del canal del NMOS) mediante canales relativamente debilitados (con estados de polarización próximos al estado de corte del transistor).

- Número de células de memoria apiladas en cada columna: Apilar células de memoria resulta normalmente rentable ya que de esta manera todas las células de la columna comparten el mismo par de *drivers* de línea utilizados para forzar datos en los ciclos de escritura y para precargar las líneas de datos (D y D_bar en la figura 2.18) antes de los ciclos de lectura, lo cual ahorra área de silicio. No obstante, las capacidades parásitas de los drenadores o fuentes de los transistores de selección (de anchura W_{NS}), junto con la capacidad de las mismas pistas de metal de estas líneas, hacen que la capacidad total vista por los *drivers* de línea sea mayor cuanto más células de memoria se apilen, lo cual hace más lentos y peligrosos los accesos de lectura a la vez que requiere el uso de *drivers* más potentes para los ciclos de escritura.

- Tiempos de subida y bajada de los selectores del decodificador: Estos tiempos están dominados por los transistores de salida de las líneas del decodificador (W_{ND} y W_{PD}), que se activan y desactivan conforme cambia el bus de direcciones (Ai en la figura, con i entre 0 y N-1). Lógicamente, mayores anchuras en los PMOS de salida (mayor W_{PD}) implican más rápidos tiempos de subida, e igualmente para los NMOS (W_{ND}) respecto de los tiempos de bajada.

En primer lugar, los transistores de selección de la célula deben ser suficientemente grandes como para que los *drivers* de línea sean capaces de sobreescribir las células de memoria durante los ciclos de escritura. La figura 2.19 ilustra el proceso de escritura de una célula de memoria en la que a partir de un estado inicial $V_1=V_{DD}$ y $V_2=0$, se pretende escribir el dato contrario a través de los transistores de selección NS1 y NS2.

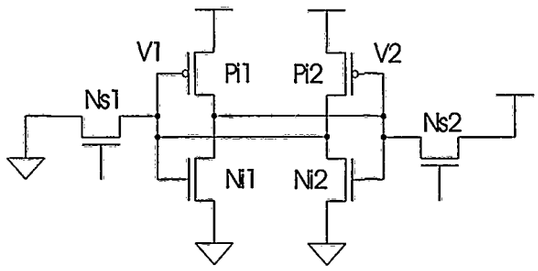

Fig. 2.19 Proceso de escritura de una célula de memoria

Suponiendo una fuerza infinita para los *drivers* de línea, las tensiones V1 y V2 de los nodos de la célula de memoria tomarían los valores expresados en la ecuación 2.1:

$$V1 = V_{DD} \cdot \frac{R_{Ns1}(V1) // R_{Ni2}(V2)}{R_{Ns1}(V1) // R_{Ni2}(V2) + R_{Pi2}(V2)}$$

$$V2 = V_{DD} \cdot \frac{R_{Ni1}(V1)}{R_{Ni1}(V1) + R_{Ns2}(V2) // R_{Pi1}(V1)}$$

(eq. 2.1)

Los factores que intervienen en esta ecuación son las resistencias equivalentes de los canales de los transistores a que se refieren: por ejemplo, $R_{Ni2}(V2)$ es la resistencia equivalente del canal del transistor Ni2, que depende fundamentalmente de su tensión de puerta V2. Las resistencias de los transistores Ns1 y Ns2 dependen de las tensiones V1 y V2 ya que si estas tensiones son más altas que V_{DD}-V_{TN} el canal se estrangularía dando lugar a valores distintos de resistencia efectiva debido a su no-linealidad. La expresión de todas estas resistencias en función de las tensiones de puerta de los transistores a que se refieren es complicada y no es lineal, aunque se corresponde con los tramos de corte, zona óhmica y saturación de los transistores MOSFET [MUL77].

La resolución de este sistema de dos ecuaciones con dos incógnitas (V1 y V2) determina el nuevo punto estable del sistema, que debe ser $V_1=0$ y $V_2=V_{DD}$, lo cual se consigue aumentando $R_{Pi1,2}$ (es decir, disminuyendo la anchura de los PMOS) o disminuyendo $R_{Ni1,2}$ (es decir,

aumentando relativamente la anchura de los NMOS), y en cualquier caso, disminuyendo $R_{Ns1,2}$ (aumentando la anchura del NMOS de selección).

Sin recurrir a métodos numéricos o simulaciones exactas, se puede utilizar la siguiente aproximación para obtener un punto estable de forma segura: en primer lugar, se fija una relación entre el PMOS y el NMOS de los inversores realimentados, y se calcula o se obtiene por simulación la tensión umbral del inversor así formado (siendo la tensión umbral la resultante de cortocircuitar la entrada con la salida); seguidamente se utiliza un NMOS para intentar forzar un nivel bajo en la salida de uno de estos inversores con su entrada puesta a cero, con lo cual la tensión resultante estará entre cero y V_{DD}; A continuación, se varía la anchura del transistor NMOS de paso hasta conseguir que la tensión a la salida del inversor (cuya entrada está a cero) esté por debajo de la tensión umbral, lo cual provocaría que el otro inversor conmutase hacia el nuevo estado estable fijando la nueva tensión a la entrada del inversor simulado en V_{DD}, provocando a su vez la conmutación total de este último. El algoritmo se puede repetir sobre inversores de partida con transistores NMOS mayores para conseguir transistores NMOS de paso más pequeños en el caso de que los valores resultantes no sean prácticos.

En principio este algoritmo puede repetirse para las distintas condiciones de tensión, temperatura y variaciones de proceso que deseen contemplarse. Las variaciones de temperatura y tensión de alimentación influyen notablemente en las resistencias que intervienen en la ecuación 2.1, si bien no en la relación entre ellas ya que aunque los coeficientes térmicos de los canales NMOS y PMOS no son idénticos al menos su diferencia no es significativa para nuestro análisis. Sin embargo, las variaciones de proceso no son proporcionales en los transistores NMOS y PMOS, ya que en principio puede coexistir un PMOS óptimo con un NMOS pésimo, o viceversa. La figura 2.20 muestra lo que se conoce como el esquema de las esquinas del proceso, que incluye todas las condiciones de simulación posibles a partir de los modelos (típico, lento y rápido) suministrados por la fábrica.

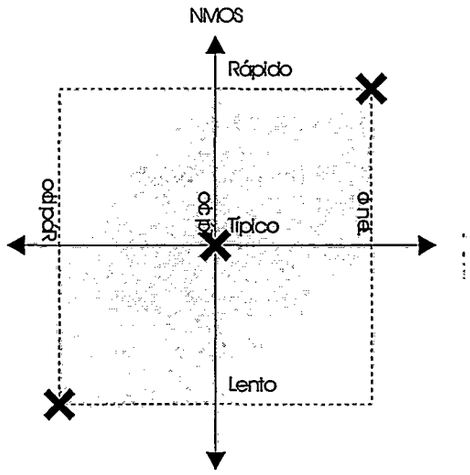

Fig. 2.20 Esquema de esquinas del proceso

El área delimitada por el cuadrado a trazos encerraría todos los casos posibles de combinaciones entre PMOS y NMOS. No obstante, al menos teóricamente es imposible que coexistan un PMOS óptimo con un NMOS pésimo o viceversa, debido a la correlación entre las distintas variables que influyen en el resultado del proceso: por ejemplo, la calidad del polisilicio influye de la misma manera en ambos tipos de transistores y por lo tanto hace tender a ambos hacia la misma esquina de proceso (rápido o lento); por otra parte, un mayor dopaje en el substrato de base tipo P- provocaría un mayor dopaje final en las zonas activas P+ y menor en las N+, haciendo tender las conductividades de los canales NMOS y PMOS hacia esquinas opuestas de proceso (uno rápido y otro lento); finalmente, el proceso de aleado tras la implantación del pozo N para el transistor PMOS no tendría ningún efecto en los transistores NMOS pero sí en los PMOS. Vemos así que hay variables que influyen en la misma dirección, otras que influyen en la dirección opuesta, y otras que influyen de forma independiente dentro del proceso para los transistores NMOS y PMOS, y que además algunas de estas variables están correladas entre sí. Por ello es imposible que coexistan las condiciones óptimas para uno de ellos y pésimas para el otro, por lo que las esquinas rápido/lento y lento/rápido no son realistas. En general, el área sombreada en la figura 2.20 resulta un espacio más realista, delimitado por las condiciones típico/lento, típico/rápido, lento/típico y rápido/típico[1]. Finalmente, las condiciones marcadas con una cruz

[1] Para tener una idea exacta del espacio de casos posibles, sería necesario disponer de una información cuantitativa exacta de las variaciones de los parámetros de los modelos de simulación de los transistores y las expresiones matemáticas de sus correlaciones. En su lugar es habitual utilizar una aproximación experimental basada en análisis de Montecarlo en la que se hace variar de forma aleatoria los parámetros del modelo dentro de los límites definidos por casos los extremos.

(lento/lento, típico/típico y rápido/rápido) son las únicas que se comprueban rutinariamente en circuitos digitales, y normalmente van dirigidas a medir retardos y predecir frecuencias máximas de operación. Sin embargo, para asegurar la estabilidad de la célula de memoria se suele comprobar su funcionamiento en todas las esquinas del proceso (zona delimitada por línea a trazos), debido a lo crítico de su naturaleza.

Una vez resuelto el problema de la escritura de las células de memoria desde un punto de vista estático, cabe preguntarse por su comportamiento dinámico. Es importante recordar que la capacidad de las líneas de datos (D y D_bar en la figura 2.18) es mayor cuanto mayor es el número de células de memoria que se apilan en cada columna, pudiendo llegar a varios picofaradios en memorias de varios *kilobits*.

Una capacidad tan grande puede considerarse como un *buffer* de fuerza infinita para una célula de memoria de pequeño tamaño, ya que el condensador es capaz de suministrar corrientes comparativamente grandes sin apenas variar su tensión, o lo que es lo mismo, será capaz de mantener la tensión durante un intervalo de tiempo grande comparado con el tiempo de conmutación de los transistores de la célula de memoria.

Por esta razón resulta vital precargar las líneas de datos en el diseño de memorias grandes y medianas. Esta precarga debe llevarse a cabo con anterioridad al ciclo de lectura durante el tiempo suficiente como para fijar ambas líneas a la misma tensión. De esta forma, al conectar los transistores de selección la célula de memoria en cuestión no sufre ninguna asimetría, por lo cual no se pierde el dato almacenado. A partir de este momento se inhabilitan los *buffers* precargadores, por lo que las líneas de datos comienzan lentamente a evolucionar hacia un futuro estado estable, movidas por los pequeños inversores de las células de memoria a través de los transistores de selección. Dado que el estado estable tardaría mucho en alcanzarse debido a la alta capacidad de las líneas y al pequeño tamaño de los transistores de las células de memoria, tras la precarga se activa el *sense amplifier* o amplificador diferencial que mide las diferencias de tensión entre las líneas de datos para registrar una predicción de lo que será el estado estable, que constituye el dato de salida, sin tener que esperar a que éste se alcance. Una vez obtenido el dato de salida, es habitual deseleccionar la célula direccionada y reactivar el proceso de precarga, para así preparar la memoria para un próximo ciclo de lectura.

La importancia de la precarga es enorme en memorias grandes y medianas. Si las líneas de datos llegaran a estar estables a 0 y V_{DD} alguna vez, la sola selección de una célula de memoria provocaría la sobreescritura del dato almacenado en ella. Esto implica que el bus de direcciones

nunca debe cambiar ni tener *glitches* o inestabilidades pasajeras en ciclos de lectura si se permite que las líneas de datos lleguen a un estado estable asimétrico.

En memorias pequeñas es posible evitar las estructuras de precarga siempre que las líneas de datos tengan unas capacidades parásitas suficientemente pequeñas de tal forma que no sean capaces de suministrar la corriente necesaria como para sobreescribir los datos almacenados en las células de memoria. Para garantizar una operación segura sin precarga suele ser necesario ajustar la tensión umbral de los inversores de la célula, aumentar el tamaño de los transistores de estos inversores, disminuir el tamaño de los transistores de selección y, en general, no apilar demasiadas células de memoria en cada columna de tal forma que la capacidad parásita de estas líneas debida a las pistas de metal y a las capacidades de los drenadores o fuentes de los transistores de selección del resto de las células de memoria no se haga demasiado grande.

En el caso de las LUTs no es posible utilizar estructuras de precarga debido a la falta de temporización de sus señales de entrada, que constituyen las líneas de dirección de la memoria. A su vez es importante evitar los solapamientos de las señales de selección de distintas células de memoria, como se muestra en la figura 2.21.

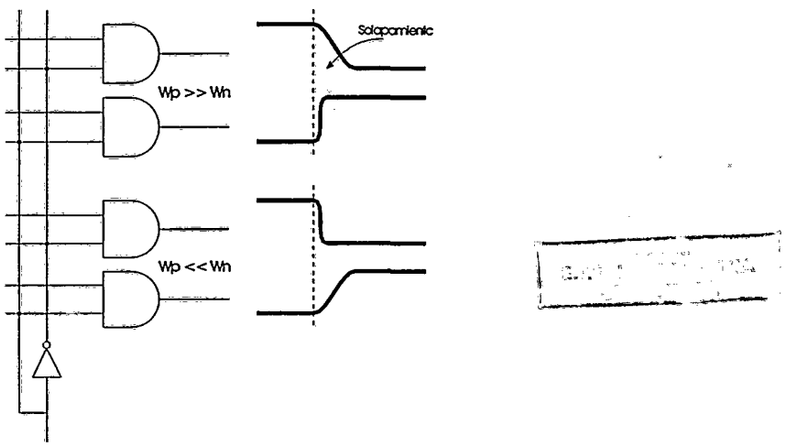

Fig. 2.21 Solapamiento entre líneas de selección de células de memoria distintas

La figura 2.21 muestra dos situaciones distintas en las que los tiempos de subida y bajada de los transistores de salida de los selectores (puertas *AND*) se han asimetrizado de forma contraria. Si los transistores de *pull-up* (los PMOS) se favorecen frente a los de *pull-down* (los NMOS), se reducirá el tiempo de subida frente al de bajada, pudiendo existir un solapamiento en las líneas de selección de distintas células. Por esto suele resultar beneficioso mantener los tiempos de subida

bajos y aumentar los de bajada sin más que favorecer los *pull-downs* (NMOS) frente a los *pull-ups* (PMOS).

Los efectos producidos por este solapamiento sólo se manifiestan cuando se comparan simulaciones en las esquinas opuestas rápido-lento y lento-rápido, e incluso en los puntos rápido-típico, típico-rápido, lento-típico y típico-lento. El problema radica en que el solapamiento de líneas de selección comunica transitoriamente dos células de memoria a través de los transistores de selección, haciendo que una de las células sobreescriba a la otra si el intervalo de solapamiento es suficientemente grande. El tamaño de los transistores de selección es crítico en este caso pues anchuras demasiado grandes permitirían este mecanismo de sobreescritura, mientras que anchuras demasiado pequeñas no permitirían forzar datos durante los ciclos normales de escritura.

La anchura de transistores de selección debe ser suficientemente grande como para que en la esquina PMOS rápido / NMOS lento sea posible vencer la fuerza de los transistores PMOS que en este caso actúan como un poderoso *pull-up*. Tras fijar esta anchura, la esquina contraria (PMOS lento / NMOS rápido) reduce la tensión umbral de los inversores de las células y la resistencia de los transistores de selección, por lo cual resulta sencillo que un inversor de una célula de memoria que esté fijando establemente un nivel bajo mediante un buen *pull-down* (NMOS rápido) transmita este nivel bajo a otra célula de memoria a través del transistor de selección, ahora altamente conductivo, encontrando solamente la oposición del ahora débil transistor PMOS.

Para evitar estos problemas es importante reducir al máximo el solapamiento en la activación de líneas de selección distintas, reduciendo para ello el tiempo de subida y aumentando el de bajada en estos nodos. Además es habitual utilizar inversores fuertemente asimétricos en las células de memoria, por ejemplo con el transistor NMOS considerablemente más grande que el PMOS, en vez de usar la proporción habitual en la que el PMOS es entre dos y tres veces más grande que el NMOS debido a la diferencia de movilidad entre electrones y huecos en silicio; con ello se persigue reducir la tensión umbral de los inversores de forma que la diferencia absoluta de esta tensión umbral entre las esquinas rápido-lento y lento-rápido sea más pequeña que si está centrada en el punto típico-típico.

El diseño de las LUTs de la parte combinacional del DMC se acometió teniendo en cuenta todas estas consideraciones. Se encontró que resultaba posible apilar hasta ocho células de memoria de simple puerto de las dimensiones seleccionadas para la implementación física seguida, si bien por

seguridad la estructura final elegida se compone de dos pilas de cuatro filas de células de memoria de doble puerto de tal forma que para el puerto de usuario (el que no está conectado al microprocesador) sólo se apilan cuatro filas de células y para el otro se apilan las ocho filas. El esquema lógico de este apilamiento se muestra en la figura 2.22

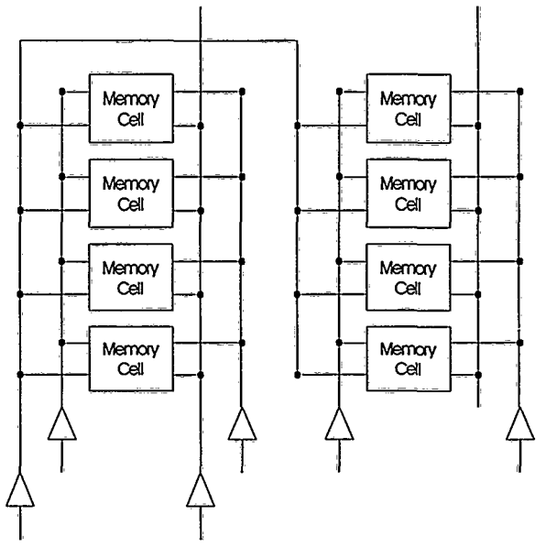

Fig. 2.22 Apilamiento de células de memoria de doble puerto

2.3.3.2 Parte secuencial

Desde los primeros días de las FPGAs [BRO92a] [TRI94] se comenzó a incluir células secuenciales junto a las LUTs como alternativa a las primeras arquitecturas introducidas por El Gammal [GAM89] (que luego serían comercializadas bajo la firma Actel y posteriormente Texas Instruments [ACT99]) en las cuales los elementos secuenciales se construían a base de combinar los mismos recursos programables (en concreto multiplexores) que se usaban para realizar los elementos combinacionales; de esta forma los *latches* y FFs se construían como básculas RS-NOR o esquemas parecidos cuya estabilidad dependía de la implementación y cuya economía no era óptima ya que se consumía una gran cantidad de recursos de rutado al realizar este tipo de construcciones.

En nuestro caso, el uso de FFs dedicados es crucial ya que éstos constituyen el punto primordial de comunicación entre el *software* y la lógica programable, ya que se encuentran mapeados dentro del espacio lógico de direccionamiento del microprocesador. Esta característica es

novedosa y resulta fundamental para cualquier arquitectura de tipo FIPSOC en la cual se quiera explotar la cercana interacción entre el hardware y el *software*.

Para complementar la parte combinacional del DMC que se explicó anteriormente, la parte secuencial del DMC está compuesta por cuatro *Flip-flops* (FFs) configurables que pueden usarse de forma más o menos independiente para operación a nivel de *bit*, o de forma combinada dentro de modos combinados o de *macro secuencial*, esto es, como contadores *up-down* y como registros de desplazamiento, ambos de cuatro *bits*. La conveniencia de este tipo de funcionalidad para distintas aplicaciones fue ya estudiada por diversos autores [JHI93] [HAR94], y la existencia de este tipo de modos *macro* es un hecho en la mayoría de las arquitecturas de FPGAs disponibles en el mercado actual.

En los siguientes párrafos se describirá métodos de implementación de la parte secuencial de una célula programable o DMC, en particular cómo implementar los distintos mecanismos de *reset* síncrono y asíncrono y cuál es la estructura interna del FF que puede soportar la reconfiguración dinámica multicontexto de forma útil.

2.3:3.2.1 Diseño básico de *latches* y *flip-flops*

Un *Flip-flop* se compone de dos *latches* conectados en cascada y controlados por fases contrarias de reloj. Los tamaños de los inversores realimentados que forman el *latch*, así como el transistor que permite su escritura (controlado por la señal de reloj), deben ser cuidadosamente seleccionados para asegurar la estabilidad de la célula en todas las condiciones de proceso. En particular el inversor que comunica los datos de izquierda a derecha debe ser más fuerte que el de dirección contraria para así poder comunicar el dato desde el primer *latch* al segundo. Las relaciones entre los transistores NMOS y PMOS de estos inversores también debe optimizarse desde este punto de vista teniendo en cuenta las consideraciones apuntadas en el párrafo 2.3.3.1.2 referente a la estabilidad de las células de memoria.

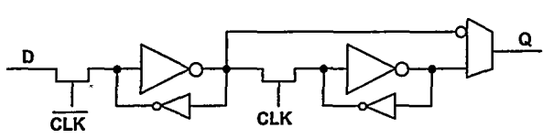

Fig. 2.23 Estructura básica del *latch / flip-flop*

Para evitar problemas de consumo y de estabilidad es habitual abrir el lazo de realimentación utilizando otro transistor controlado por la fase contraria de reloj en cada *latch*. De esta forma el inversor de salida del primer *latch* y el de entrada del segundo no tendrían conectadas sus salidas en ningún momento, lo cual disminuye drásticamente el consumo dinámico. No obstante, si se utiliza un transistor NMOS para abrir el lazo, la célula consumiría en condiciones estáticas en el estado en el que el inversor que realimenta intente forzar un nivel alto en el inversor de salida, ya que un NMOS introduce una caída equivalente a la tensión umbral al trasmitir los niveles altos [MUL77] lo cual situaría al transistor PMOS en un cierto estado de conducción débil con consumo estático. Por esto es necesario utilizar un conmutador CMOS para abrir el lazo de realimentación si el consumo estático no es admisible.

Los elementos secuenciales introducidos en el DMC pueden ser configurables de tal forma que puedan ser usados como *latches* o como *flip-flops*. En el caso de ser usados como *latches*, sólo la mitad de la célula es utilizada como se muestra en la figura 2.23, por lo cual la utilización de *latches* no es más económica que la de FFs como ocurre en las implementaciones con ASICs. No obstante, soportar la configuración de *latches* resulta interesante si la FPGA va a usarse para emular un circuito que posteriormente será fabricado masivamente como ASIC.

2.3.3.2.2 Mecanismos de *reset* y escritura desde el microprocesador

Los mecanismos de *reset* implementados sobre la estructura básica de la figura 2.23 se muestran en la figura 2.24 . Para generar un *reset* o *set* síncrono, se incorpora un multiplexor a la entrada, de tal forma que se fuerza un "0" o un "1" lógico en función de que se trate de un *reset* o de un *set* síncrono. En el caso de un *reset* o *set* asíncrono, la escritura en el primer *latch* es independiente de la fase de reloj activa, a la vez que es necesario producir una propagación inmediata desde el primer *latch* al segundo, utilizando otro conmutador en paralelo con el controlado por la señal de reloj. En la figura 2.24, la señal **SSR** implementa un *reset* o *set* (seleccionable con el *bit* de programación marcado como **SR1**) síncrono, mientras que **ASR** produce un *reset* o *set* (seleccionable con **SR2**) asíncrono.

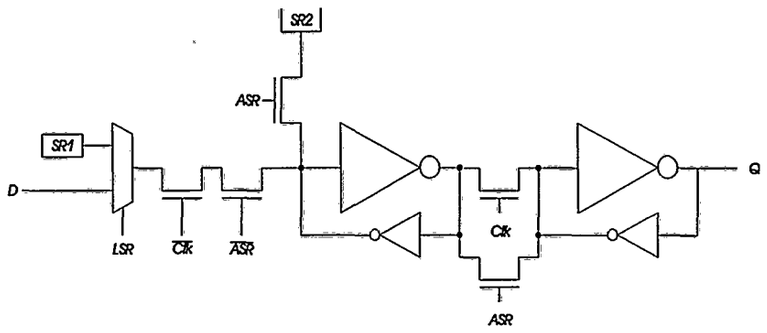

Fig. 2.24 Mecanismos básicos de *reset*

En la estructura de la figura 2.24 puede verse que el *reset* asíncrono **ASR** tiene prioridad sobre el síncrono **SSR**. En el caso de implementar varios mecanismos de *reset* simultáneos es necesario disponer de un mecanismo de resolución de prioridades entre *resets* y *sets* síncronos y asíncronos, aunque es habitual que los mecanismos asíncronos tengan prioridad sobre los síncronos.

2.3.3.2.3 *Flip-flops* multicontexto

Para soportar la operación bicontexto es necesario duplicar internamente la estructura del *flip-flop*, como se muestra esquemáticamente en la figura 2.25 .

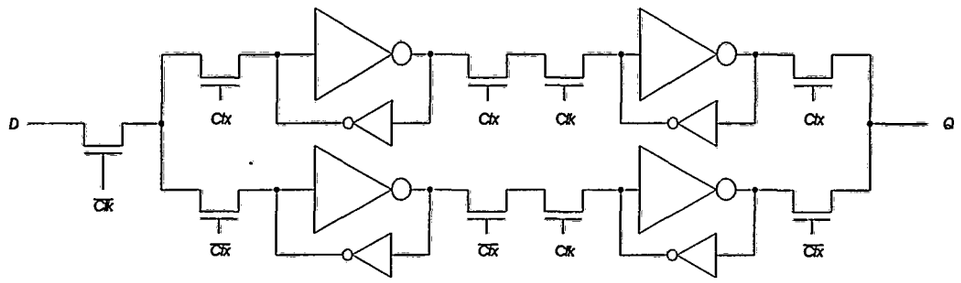

Fig. 2.25 *Flip-flop* bicontexto

El contexto activo en cada momento se selecciona mediante la señal **Ctx**, que a su vez puede estar o no gobernada por el mismo *bit* de configuración que programa el contexto activo en las LUTs del mismo DMC. Activando este control se consigue que los datos almacenados en los FFs sean distintos en cada contexto, de tal manera que una tarea interrumpida y salvada en un

contexto puede ser re-arrancada después pues sus datos seguirían intactos. Desactivando este control, y por tanto utilizando en ambos contextos el mismo FF (es decir, manteniendo la señal **Ctx** independiente del contexto activo en las LUTs), se consigue comunicar datos entre contextos que operan en distintos momentos del tiempo (otra forma más lenta de hacer esto es leer y escribir datos con el microprocesador). Finalmente, se puede escribir en el contexto inactivo desde el microprocesador, lo cual resulta útil para fijar las condiciones iniciales en que el bloque reconfigurado empezará a operar.

2.3.3.2.4 Modos de funcionamiento

Combinando estos tres elementos (selección entre *latch* o *flip-flop*, diferentes mecanismos de *reset* configurables incluyendo la escritura asíncrona desde el microprocesador, y la operación multicontexto) se puede construir cualquier estructura final del FF básico tipo D a utilizar. Sobre él se puede construir el resto de la circuitería de configuración funcional a base de multiplexores y células de memoria de programación, de una forma parecida a como se construyeron las LUTs.

Como ejemplo, la figura 2.26 muestra un diseño de FF complejo programable mediante seis señales de configuración que en realidad pueden derivarse de un número menor de *bits* de configuración. Este diseño incluye en total 52 transistores, y dispone así de una entrada de datos **D**, una de reloj **CLK**, una de *reset* asíncrono **GSRN**, otra de *reset* configurable como síncrono o asíncrono **LSR**, una señal de escritura desde el microprocesador (implementada como otro *reset* asíncrono), y un mecanismo de cambio de contexto. Como puede verse, cada FF básico tipo D se encapsula dentro de un FF de dos entradas como se muestra en la figura 2.26. La parte secuencial del DMC incluiría así cuatro FFs de dos entradas como se muestra en la figura 2.27.

Dado que se dispondría de cuatro FFs en cada DMC, las señales de control que gobiernan los mecanismos de reconfiguración, *reset* síncrono y asíncrono y su gestión de prioridades, y escritura desde el microprocesador, son comunes para los cuatro. El gobierno de estos mecanismos está basado en las técnicas explicadas en los puntos anteriores aunque su implementación final resulta algo más compleja debido a la intrincada combinación de todas las posibilidades.

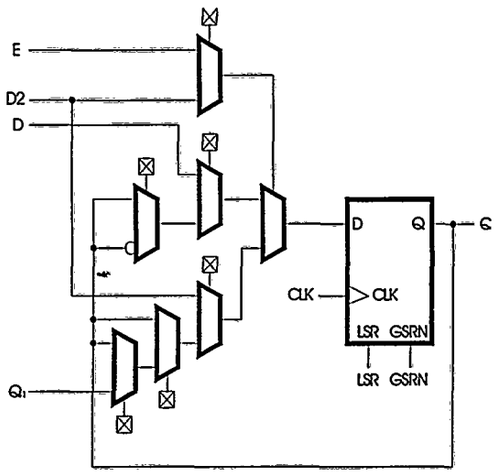

Fig. 2.26 *Flip-flop* complejo de dos entradas

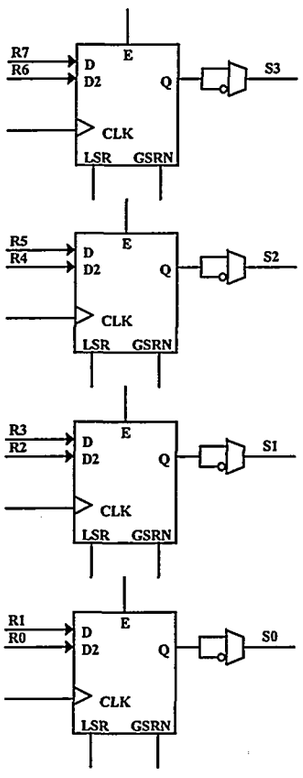

Fig. 2.27 Parte secuencial de un DMC de 4 FFs

Las señales de control (La señal de reloj **CLK**, Las de *reset* **LSR** y **GSRN**, y la señal de selección **E**) son comunes a los cuatro FFs de la parte secuencial del mismo DMC. Realizando distintas combinaciones de las señales de configuración (marcadas con un cuadrado aspado en la figura 2.26), que en realidad se pueden forzar mediante tres *bits* de configuración, se obtienen los

distintos tipos de FFs soportados. Las figuras 2.28, 2.29, 2.30 y 2.31 muestran ejemplos de modos de configuración de la arquitectura programable de la figura 2.26 con valores particulares de las señales de configuración, tal y como aparecerían a los ojos del usuario. Los casos mostrados responden a un FF tipo multiplexor, un FF tipo D con *reset* síncrono local, un FF tipo D con *enable*, y un FF tipo D con *enable* local, respectivamente. Aquí el término *local* se refiere a que la señal en particular no se comparte con los otros FFs del DMC, como suele ser habitual en gran parte de las arquitecturas programables disponibles en el mercado.

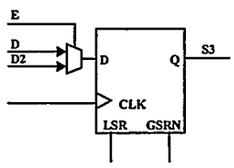

Fig. 2.28 FF tipo multiplexor

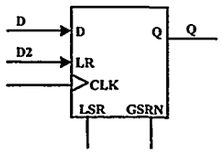

Fig. 2.29 FF tipo D con *reset* síncrono local

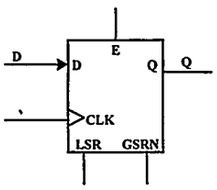

Fig. 2.30 FF tipo multiplexor

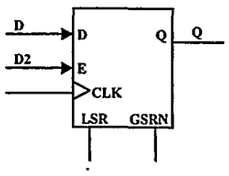

Fig. 2.31 FF tipo D con *enable* local

Analizando cuidadosamente la arquitectura propuesta en la figura 2.26 se puede comprobar que además ésta resulta adecuada para encadenar varios de estos FFs en cascada para realizar funciones de *macro* complejas (el dato del FF anterior entra por la entrada $\mathbf{Q_{-1}}$ y los acarreos se comunican a través de las señales auxiliares). La figura 2.32 muestra dos de estos modos de *macro* secuencial tal y como aparecerían a los ojos de un usuario.

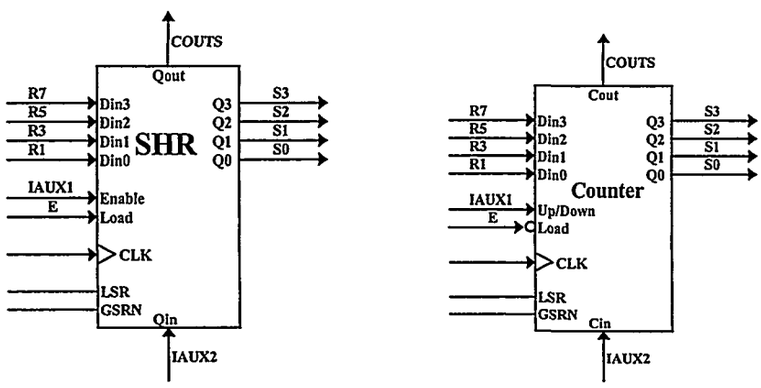

Fig. 2.32 Parte secuencial del DMC como registro de desplazamiento (izquierda) y contador *up-down* (derecha)

2.3.3.3 Rutado interno

La estructura de *rutado interno* es el bloque responsable de comunicar la parte combinacional y la secuencial dentro de un mismo DMC. La arquitectura de este bloque es crítica para la flexibilidad del DMC, ya que debe ser capaz de permitir que, por una parte, ambos bloques puedan ser utilizados de forma independiente, mientras que en otras ocasiones el DMC completo pueda ser usado como bloque compacto con una parte combinacional y otra secuencial integradas.

En la figura 2.33 se muestra el bloque de rutado interno con una granularidad de cuatro *bits* para utilizarse como interfaz entre el diseño de las partes combinacional y secuencial del DMC propuesto en la sección anterior. El bloque está formado por *intercambiadores* o o bloques que selectivamente conectan las salidas a las entradas de forma directa (**I1** a **O1** e **I0** a **O0**) o intercambiada (**I1** a **O0** e **I0** a **O1**). La estructura de estos intercambiadores es tan sencilla como dos multiplexores de dos a uno con la misma señal de control conectada a un *bit* de configuración y las entradas intercambiadas.

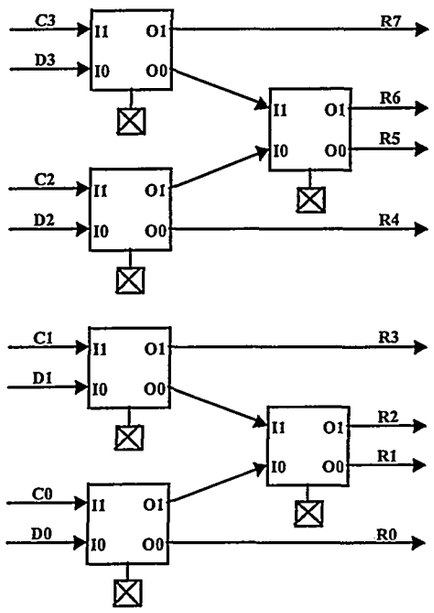

Fig. 2.33 Estructura del bloque de rutado interno

El interés de este bloque radica en su gran flexibilidad, equivalente a la de soluciones para granularidades similares como la propuesta por Britton [BRI94] y Singh [SIN97], a pesar de utilizar tan sólo seis *bits* de configuración (simbolizados por los cuadrados aspados).

2.3.3.4 Salidas del DMC

En general los autores de diseños de FPGAs de distintas clases de granularidad media y grande [CAR86] [BRI94] [SIN97] [HAR94] han utilizado un número de salidas por DMC mayor al de su granularidad, con el doble objetivo de favorecer la utilización de las partes combinacional y secuencial de forma independiente, y de permitir expandir las posibilidades de conexión del DMC en modos de función específica tipo *macro*.

La figura 2.34 muestra el esquema propuesto para las señales de salida del DMC introducido en las secciones anteriores. Las señales auxiliares sirven a salidas combinacionales y secuenciales y a las salidas especiales de acarreo de los modos de *macro* secuencial, esto es, contador y registro de desplazamiento.

Las salidas "oficiales" permiten además el control de alta impedancia de los *buffers* de salida para la implementación de bloques de memoria mayores de lo que cabe en un DMC. En cualquier caso, todas las señales de salida (en realidad, todas las salidas de todos los DMCs) se pueden inhabilitar con una señal global (**GOE**), especialmente durante la rutina de inicialización (*reset*) del circuito.

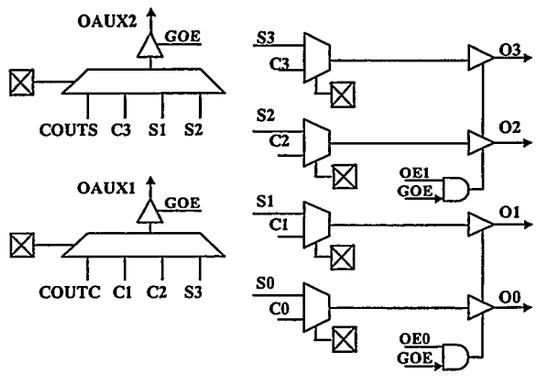

Fig. 2.34 Estructura de la etapa de salida del DMC

2.3.4 Células de entrada-salida

Como se explicó en el párrafo 2.3.2, la figura 2.2 muestra cómo rodeando a la matriz de DMCs se encuentran los bloques de entrada salida o IOBs, que contienen células de entrada-salida o

IOCs funcionalmente equivalentes de forma independiente a su posición relativa en el *chip*. Sin pérdida de generalidad consideramos que existe un IOB por cada fila y columna, dispuestos en los laterales y en la parte superior del *chip* ya que la parte inferior está dedicada al interfaz con la parte analógica y con el microprocesador. Cada IOB contiene un cierto número de IOCs, que en general variará en función de la granularidad de la célula básica y del énfasis que quiera darse a la FPGA en general en cuanto al número de entradas/salidas por puerta útil.

La figura 2.35 muestra la estructura de un IOB típico con dos IOCs.

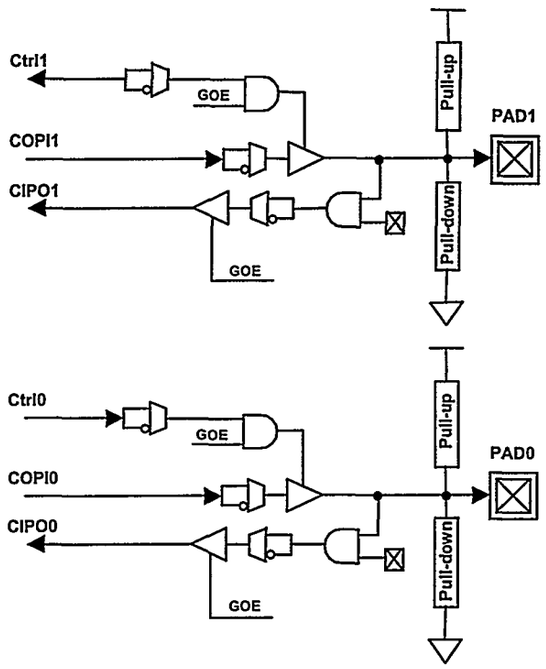

Fig. 2.35 Célula de entrada-salida

En general los diseños normales de IOCs disponibles en la literatura técnica de FPGAs permiten algunas o todas estas posibilidades (algunas arquitecturas incluyen además un registro en cada una de las señales de salida). Cada IOC puede configurarse de forma independiente como entrada, salida, o *buffer* tri-estado. Por tanto cada IOC tiene tres señales funcionales que se conectan mediante recursos de rutado de propósito general como el resto de las entradas y salidas de los DMCs y de los IICs: la entrada **COPI** (*Core Out Pad In*) para el modo de *pad* de salida, la salida **CIPO** (*Core In Pad Out*) para el modo de *pad* de entrada, y la entrada **Ctrl** de control de alta impedancia de los *buffers* de salida para el modo de salida tri-estado. Cada una de estas señales puede invertirse y además la salida **CIPO** puede inhabilitarse mediante un *bit* de

configuración, de tal forma que se puede fijar valores constantes tanto en la configuración de salida como en la de entrada, lo cual resulta especialmente útil en aplicaciones de emulación de sistema.

Para los tamaños más grandes de FPGA es importante poder limitar la corriente entregada por los *pads* de salida debido al alto número de ellos que habría en el *chip*. Con esta idea se ha implementado un control de fuerza en los *buffers* de salida como se muestra en la figura 2.36. En total se utilizan seis *bits* de configuración (marcados como **Pu<2:0>** y **Pd<3:0>** en la figura) para elegir uno de ocho posibles tamaños efectivos tanto para los transistores PMOS de *pull-up* como para los NMOS de *pull-down* de forma independiente. Los valores '000' corresponden a la inhabilitación de la etapa correspondiente (*pull-up* o *pull-down*), gracias a lo cual es posible implementar salidas en drenador abierto (PMOS superior inhabilitado) o fuente abierta (NMOS inferior inhabilitado). En cualquier caso, además se proveen unas resistencias de valor fijo (en torno a 30KΩ) conectables de forma seleccionable como *pull-up* y como *pull-down*, para las configuraciones de entrada en que los *pads* puedan quedarse sin conectar.

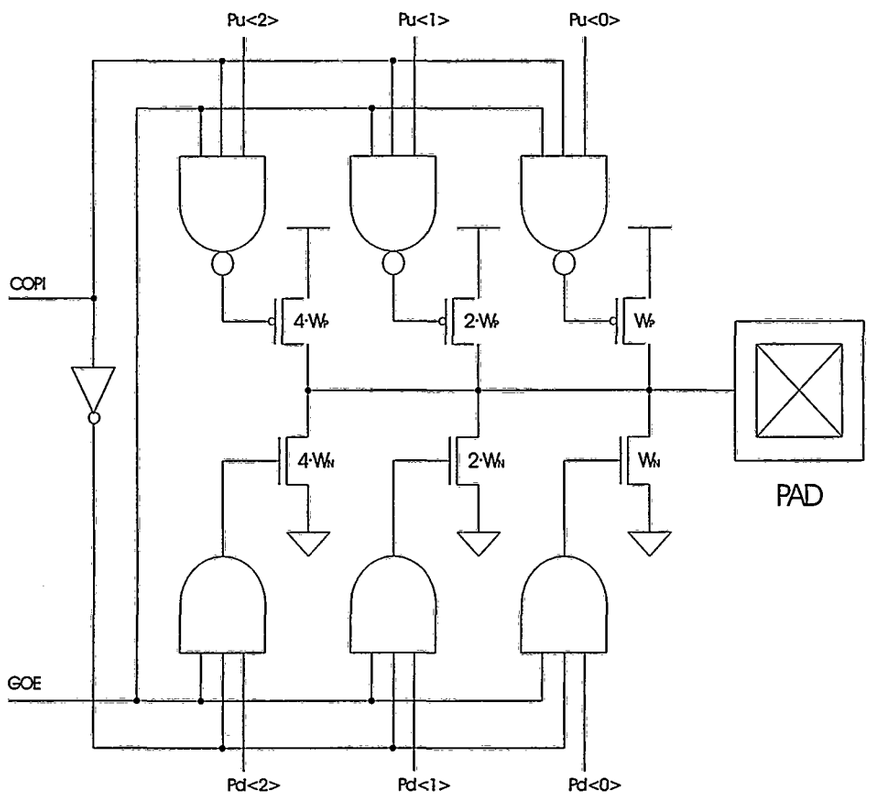

Fig. 2.36 *Buffers* de salida con fuerza programable

2.3.5 Recursos de rutado

En contraposición a los recursos de *rutado interno* tratados en el párrafo 2.3.3.3, los recursos de *rutado externo* comprenden la circuitería de conexión entre los DMCs de la FPGA. El diseño de una buena arquitectura de rutado es una de las claves para la viabilidad económica de una FPGA, dado que la mayor parte del área de los DMCs corresponde a los recursos de rutado y por tanto un buen aprovechamiento de los mismos es crucial.

Los estudios de rutabilidad de estas arquitecturas fueron consolidados por Brown y Rose [BRO92a] [ROS90b] [ROS91], y sus resultados son comúnmente aceptados para el diseño de esquemas de rutado. Gran parte de los diseños de FPGAs comerciales disponibles en el mercado en la actualidad incorporan además segmentos de diversas longitudes, aunque sin llegar a utilizar arquitecturas puramente jerárquicas como la propuesta en [WAN94]. Las pistas cortas son necesarias para permitir mayor flexibilidad en la interconexión, mientras que las pistas más largas consiguen menores retardos en interconexiones distantes, ya que no necesitan conmutadores que introducen resistencia parásita y por tanto retardo debido a la carga de las capacidades parásitas de los propios canales de rutado y de las entradas que a ellos se conectan.

Seguidamente se discutirán los criterios de rutabilidad y prestaciones desarrollados para el diseño de las estructuras de rutado de la FPGA a incluir en una arquitectura de tipo FIPSOC; finalmente se presentará un ejemplo completo de arquitectura de rutado diseñada utilizando estos criterios.

2.3.5.1 Diseño de arquitecturas de rutado

Las figuras de mérito a utilizar a la hora del diseño de una arquitectura de interconexión programable como la de una FPGA son la rutabilidad (probabilidad de éxito a la hora de implementar un circuito típico sobre la FPGA), el retardo máximo, y el área consumida. Lógicamente un mayor número de canales de rutado y una mayor flexibilidad en la conexión de las entradas a ellos consiguen una rutabilidad mayor, aunque también requieren una mayor área para su implementación. Por otra parte, utilizar conmutadores independientes para las interconexiones y aumentar su tamaño reducen el retardo de cada conexión, pero a la vez necesitan más *bits* de programación y por tanto mayor tamaño de silicio.

Los trabajos de Brown y Rose [BRO92a] [ROS90b] [ROS91] demostraron de forma más o menos sistemática y genérica que las matrices de conmutación necesitan tener la flexibilidad suficiente para permitir conectar cada terminal a uno de al menos otros tres terminales, y que el porcentaje de canales a los que las entradas de los bloques lógicos deben poder conectarse es al menos del 70% . Se ha contemplado estos criterios para el diseño de la arquitectura de rutado de FIPSOC.

2.3.5.1.1 Arquitecturas básicas

La arquitectura de rutado diseñada utiliza básicamente tres tipos de interconexión programable:

- Conmutadores CMOS: Existe un solo par CMOS que une el terminal en cuestión del DMC con el canal de rutado. El retardo y el área consumida por la estructura de interconexión propiamente dicha son minimizados [KHE94], aunque es necesario un bit de configuración para cada posible conexión.

- Multiplexores construidos a partir de transistores de paso CMOS: Varios terminales del DMC pueden unirse de forma selectiva (sólo uno de ellos puede estar conectado en cada momento) mediante un multiplexor bidireccional construido a partir de pares CMOS. El retardo es mayor que si se utilizan conmutadores CMOS únicos y el área se incrementa en algo menos del doble, pero en este caso el número de bits de configuración es reducido a $log_2(N)$, donde N es el número de terminales a conectar al canal de rutado en cuestión.

- Multiplexores construidos a partir de transistores de paso NMOS: Funcionalmente es similar al caso anterior, salvo que se utilizan transistores NMOS para implementar los multiplexores en lugar de pares CMOS. El ahorro de área es del orden de un factor de cuatro ya que no son necesarios los transistores PMOS cuyo tamaño debe ser entre 2 y 3 veces superior al de los NMOS para compensar la diferencia de movilidad entre los electrones (canales NMOS) y huecos (canales PMOS), y no son necesarios los espacios de separación de las zonas activas N+ de los transistores NMOS con los pozos N- para los transistores PMOS. Como contrapartida se obtiene un peor retardo de interconexión debido a la mala transmisión de los niveles altos por los transistores NMOS, ya que en los niveles altos los canales de estos transistores están al borde de la estrangulación (*pinch-off*) y por tanto resultan menos conductivos, si bien es cierto que las transiciones al nivel bajo resultarían más rápidas ya que

al minimizar el área se minimiza también la capacidad parásita. Además este tipo de arquitecturas produce consumo estático en las puertas CMOS a las que atacan debido a que los niveles altos se ven reducidos a $V_{DD} - V_{TN}$, donde V_{TN} es la tensión umbral del transistor NMOS.

2.3.5.1.2 Prestaciones

Una vez que se realiza una conexión entre una salida de un bloque programable y una entrada de otro, se obtiene una sucesión de pistas metálicas de baja resistencia y de capacidad parásita apreciable unidas por los elementos de interconexión (típicamente conmutadores CMOS o NMOS) que introducen una cierta resistencia parásita en serie. Esta situación se esquematiza en la figura 2.37, que muestra el banco de trabajo sobre el que realizar simulaciones físicas para estudiar este tipo de interconexiones.

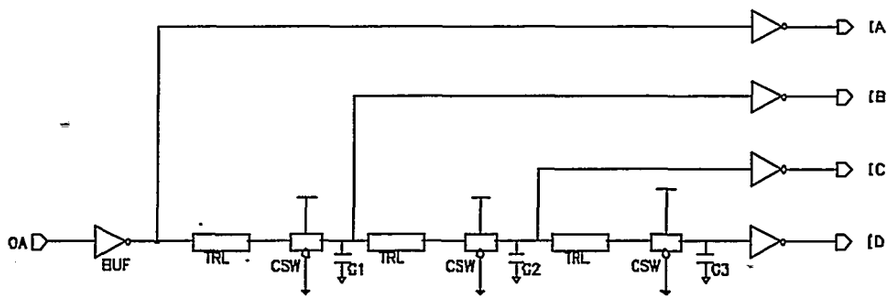

Fig. 2.37 Banco de trabajo para la caracterización de retardos en interconexiones programables

El esquema de la figura 2.37 modela la conexión en cuatro tramos desde la salida **A** de un bloque programable hasta la entrada **ID** de otro a través de tres conmutadores **CSW** conectados por pistas de metal **TRL** con una capacidad parásita **C1**, **C2** y **C3**. Las capacidades antes de cada conmutador pueden ser distintas ya que no sólamente dependen de la capacidad parásita de la pista y del conmutador en sí, sino además de la capacidad de las entradas de los bloques programables que se conectan a la red multipunto. Con los puntos intermedios **IA**, **IB**, **IC**, e **ID** se pueden medir los retardos hasta las entradas de los bloques programables conectados a la misma red a distintas distancias de la entrada.

Un estudio básico aproximado [WES93] demuestra que el retardo global de una red RC en escalera, en esencia equivalente a la presentada en la figura 2.37 si las capacidades son iguales y si los conmutadores son modelados por resistencias puras constantes, responde a la expresión

$$t_n \approx \frac{RCn(n+1)}{2}; \quad \text{(eq. 2.2)}$$

, en donde **R** y **C** son la resistencia y capacidad de cada sección, y **n** es el número de secciones conectadas en cascada. Como puede verse el retardo global aumenta cuadráticamente con el número de secciones, lo que hace poco recomendable comunicar bloques programables distantes a base de interconexiones cortas. Por este motivo suele ser útil disponer de canales de rutado que se extiendan no sólo de DMC en DMC sino cada dos, cuatro o más bloques programables.

Para el caso de la estructura de la figura 2.37, y considerando todas las capacidades iguales y el *buffer* inicial de retardo despreciable, se obtiene la siguiente expresión para el retardo entre la salida incial **OA** y la entrada final **ID**:

$$t_{r,f} \approx 3(R_{Path} + \frac{R_{Unit}}{W})(C_{Load} + C_{Unit}W); \quad \text{(eq. 2.3)}$$

, en la que $\mathbf{R_{Path}}$ es la resistencia de cada pista de metal **TRL**, **W** es la anchura equivalente del transistor conmutador, $\mathbf{R_{Unit}}$ es la resistencia equivalente por unidad de anchura del canal del transistor conmutador, $\mathbf{C_{Unit}}$ es la capacidad parásita equivalente por unidad de anchura de puerta del drenador y fuente del transistor conmutador, y $\mathbf{C_{Load}}$ es la carga capacitiva de cada tramos producida por las entradas de los bloques programables conectados y por la capacidad parásita de la pista metálica.

Para el cálculo de la resistencia y capacidad parásitas equivalentes de los conmutadores CMOS se ecualiza en primer lugar el retardo debido a cada uno de ellos (mostrado en la figura 2.38) para así poder trabajar con una sola magnitud, por ejemplo la anchura del transistor NMOS (las longitudes se suponen ambas las mínimas permitidas por la tecnología, 0,5μm en nuestro caso), ya que uno de ellos va a estar habilitado en función de que se transmita un nivel alto (el PMOS) o bajo (el NMOS). Utilizar pares CMOS de transistores cuya relación de anchuras no responda a la gráfica de la figura 2.38 implica que uno de los dos está sobredimensionado y que por tanto consume más área de la necesaria sin reducir el retardo global del conmutador que en principio está dominado por el más lento de los dos.

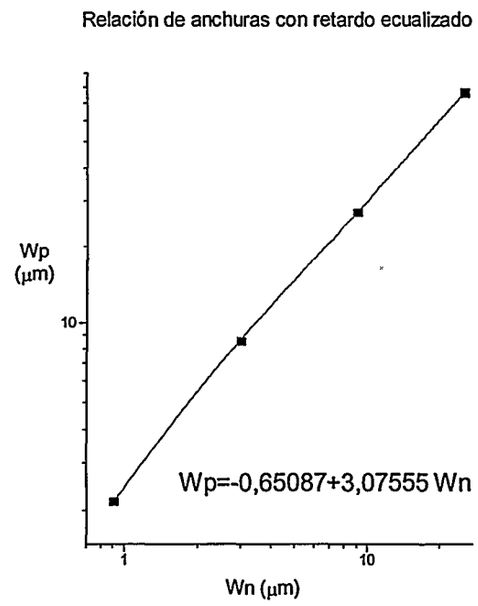

Fig. 2.38 Relación de anchuras entre transistores PMOS y NMOS con retardo ecualizado

La relación presentada en la figura 2.38 ha sido obtenida a base de simulaciones físicas con HSPICE™, si bien es cierto que debido a los distintos valores de la capacidad parásita de las zonas activas de fuente o drenador para cada uno de los transistores, esta curva depende ligeramente de la carga aplicada. La relación de anchuras óptima entre el PMOS y el NMOS se estabiliza en torno a tres aunque es sensiblemente diferente en torno a las dimensiones mínimas de los transistores debido a los distintos comportamientos de los transistores en presencia de modulación de las dimensiones del canal y otros efectos submicrónicos [MUL77].

La ecuación 2.3 implica que la variación de la anchura equivalente del conmutador produce dos tendencias contrapuestas, una de disminución del retardo debido a la disminución de la resistencia equivalente, y otra de aumento debido a la mayor capacidad parásita de sus drenadores y fuentes. En realidad, la tendencia de crecimiento no debería manifestarse más que en caso de usar conmutadores muy grandes para los cuales la capacidad parásita de sus drenadores y fuentes dominase sobre las cargas capacitivas de pistas y entradas de bloques programables, lo cual no es habitual. Por otra parte, el retardo global también depende del tamaño equivalente del *buffer* de salida que ataca a toda la interconexión (marcado como **BUF** en la figura 2.37).

La figura 2.39 muestra resultados experimentales de simulación con HSPICE™ en los que puede confirmarse la presencia de las dos tendencias antes mencionadas al medir los retardos en función del tamaño del conmutador CMOS. Las simulaciones corresponden a un tamaño equivalente del *buffer* de 2,5μm y una carga de 200fF.

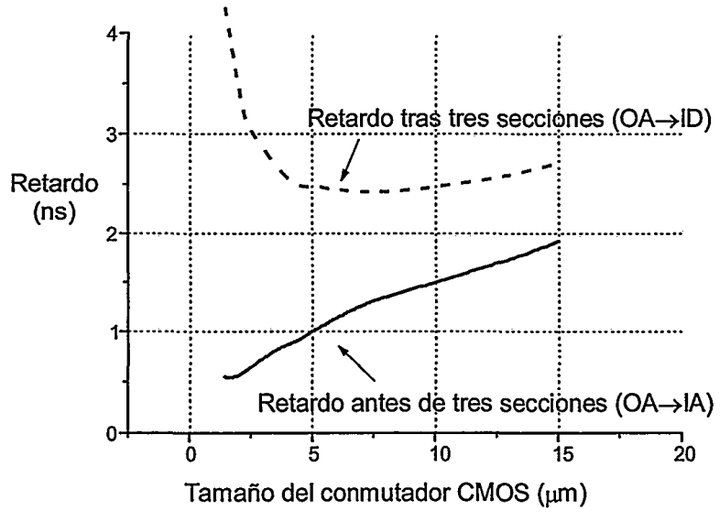

Fig. 2.39 Retardo de interconexión en función del tamaño equivalente del conmutador

La figura 2.39 muestra cómo aunque el retardo global (de **OA** a **ID** en la figura 2.37) disminuye al aumentar el tamaño del conmutador, al menos hasta el punto en que la capacidad parásita del conmutador empieza a dominar sobre la carga capacitiva constante, el retardo local en la primera interconexión, que sería el retardo visto por el bloque programable conectado al primer nodo de la red multipunto (el punto **IA** de la figura 2.37), se incrementa considerablemente y de forma más o menos lineal conforme aumenta el tamaño del conmutador. Esto se debe a que al reducir la resistencia del conmutador aumenta la carga capacitiva vista por los puntos cercanos al *buffer* de salida. Por tanto puede afirmarse que aumentando la anchura de los conmutadores (hasta ciertos límites) se disminuye el retardo a las conexiones lejanas al *buffer* pero se aumenta el retardo de las conexiones cercanas, lo cual no siempre es una garantía para minimizar el retardo crítico de un circuito, como se explicará más adelante.

La curva del retardo global (o lejano al *buffer*) presenta una pendiente mucho mayor en valores bajos de la anchura del conmutador, como corresponde al termino en $\mathbf{W}^{-1}$ de la eq. 2.3, por lo que no parece adecuado escoger el punto de mínimo retardo para el que la anchura del conmutador resulta de **W** = 8μm cuando si se admite un aumento en el retardo del 5% la anchura se reduce a

W = 3.6μm, es decir, un 55% . Esta reducción no sólo es preferible por motivos de área consumida, sino por la reducción que también implica para el retardo local (cercano al *buffer*), que resulta del 40 % .

Si además se varía el tamaño equivalente del *buffer* de salida a la vez que el del conmutador, se obtiene las distribuciones tridimensionales de la figura 2.40, para las que las cargas y las condiciones de simulación se han mantenido.

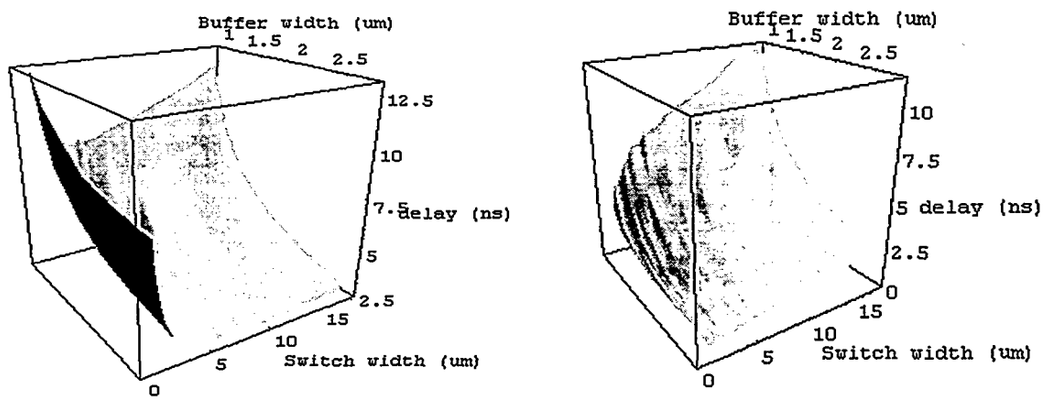

Fig. 2.40 Retardo de interconexión en función del tamaño equivalente del conmutador y del tamaño del *buffer* para la interconexión lejana (izquierda) y cercana (derecha)

Sobre la superficie del retardo global (a la izquierda en la figura 2.40) puede trazarse una línea siguiendo el criterio mencionado del rutado del 5% del mínimo para cada tamaño del *buffer* y del conmutador. La expresión de esta línea para el caso de nuestra tecnología, obtenida a partir de los datos de simulación física utilizados en la figura 2.40, resulta

$$W_{Switch} \approx 1.14 + W_{Buffer} \; ; \quad \text{(eq. 2.3)}$$

, donde $\mathbf{W_{Switch}}$ y $\mathbf{W_{Buffer}}$ son las anchuras del conmutador y del *buffer* respectivamente. Esta expresión relaciona el tamaño del conmutador y del *buffer* de salida que se ha utilizado para el diseño de la estructura programable de rutado de FIPSOC, que aunque no minimiza totalmente el retardo global de la conexión (resulta del orden del 5% superior) reduce significativamente el retardo de las interconexiones locales cercanas a los *buffers* de salida de las células programables.

2.3.5.1.3 Retardo medio

Las conexiones de los circuitos lógicos a implementar en la FPGA no tienen por qué ser redes monopunto, es decir, con un *driver* y una sola entrada atacada más o menos distante. Resulta normal el caso de la red que se conecta a distintos puntos como en el modelo esquematizado en la figura 2.37. En estas situaciones el retardo crítico del circuito no siempre depende de las conexiones más lejanas de estas redes multipunto, como se ilustra en la figura 2.41.

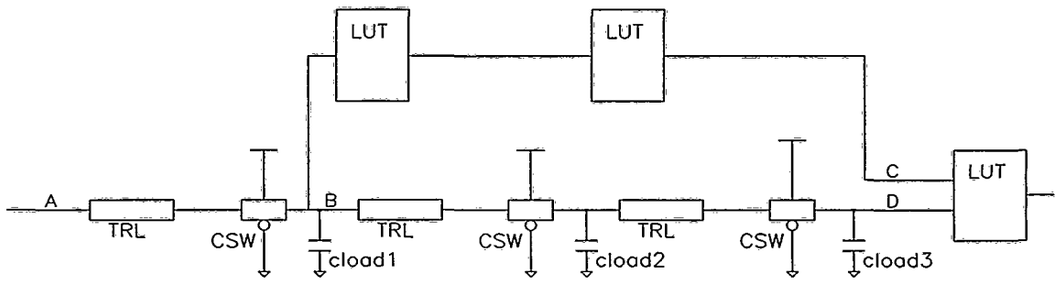

Fig. 2.41 Retardo crítico dominado por interconexión cercana

En el circuito de la figura 2.41 el retardo crítico puede ser el medido entre los puntos **A** y **C** o entre **A** y **D**, dependiendo de la velocidad de las LUTs y del resto de la interconexión. Si el camino **A - C** resulta ser el crítico, sería beneficioso minimizar el retardo de interconexión local al punto **B**, que aumentaba al sobredimensionar los conmutadores de paso como se explicó en el punto anterior. Por esto, y especialmente para estructuras de rutado de FPGAs de alta granularidad como en nuestro caso, resulta beneficioso reducir el retardo medio de las interconexiones punto a punto, y no sólo el retardo global de los puntos más alejados en una conexión multipunto.

A nivel de implementación de la arquitectura es posible reducir este retardo medio utilizando relaciones entre *buffers* y conmutadores algo distintas de las que producen el retardo global mínimo, ya que como se explicó en el punto anterior un aumento del 5% en el retardo global permite reducir considerablemente el retardo local (un 40% en el ejemplo discutido en el punto anterior). A nivel de utilización de la FPGA, es posible utilizar la asimetría de la carga como variable para optimizar el retardo medio.

Para comprender la influencia de la asimetría de la carga consideremos el banco de trabajo de la figura 2.37, en el que las cargas capacitivas, en vez de ser iguales, tienen un valor expresado por un actor de asimetría a:

$$C_1 = \tfrac{2}{3} C_T \cdot a$$
$$C_2 = \tfrac{1}{3} C_T \qquad \text{(eq. 2.4)}$$
$$C_3 = \tfrac{2}{3} C_T \cdot (1-a)$$

, donde $\mathbf{C_T}$ es la carga total a conectar. Los resultados de simulación mostrados en la figura 2.42 muestran cómo el retardo global disminuye y el local aumenta al aumentar el factor de asimetría, debido a las distintas cargas vistas por cada uno de los puntos hacia el *buffer*. El algoritmo de rutado a implementar en el flujo de diseño físico podría tener en cuenta este hecho para minimizar uno u otro retardo en función del circuito implementado en cada momento. Para ello debería analizar los retardos en los distintos nodos de la red lógica multipunto y asimetrizar las cargas en consecuencia.

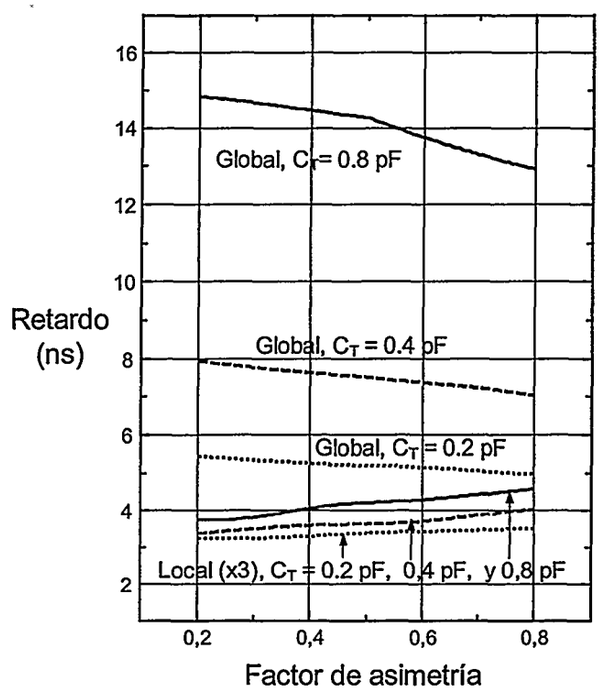

Fig. 2.42 Efecto de la asimetría de la carga en el retardo

2.3.5.2 Caso práctico de arquitectura de rutado

Esta sección presenta un ejemplo de arquitectura de rutado susceptible de ser utilizada para interconectar una matriz de DMCs como los propuestos en las secciones anteriores. En general esta arqutiectura refleja la granularidad general del DMC propuesto, por lo que las pistas se agrupan en buses de cuatro. La estructura es direccional, favoreciendo el flujo de datos de izquierda a derecha y de abajo a arriba, por lo que las entradas se agrupan abajo y a la izquierda del DMC y las salidas arriba y a la derecha, preferentemente.

La estructura se basa enteramente en multiplexores debido a las fuertes restricciones en cuanto a número de *bits* de configuración disponibles. Los selectores para los terminales de entrada están implementados a base de multiplexores NMOS relativamente anchos, típicamente de 32 a 1 (algunas entradas de propósito especial tienen menos flexibilidad), rematados con inversores con una tensión umbral de conmutación relativamente baja (es decir, con una relación entre tamaños de PMOS a NMOS más pequeña que la recomendada en la gráfica de la figura 2.38) para compensar la mala transmisión de los niveles altos por los transistores NMOS. Estos inversores "conformadores" son pequeños para evitar un excesivo consumo estático cuando a la entrada existen niveles altos mal transmitidos.

Las salidas de los bloques programables se implementan igualmente con multiplexores, aunque construidos con conmutadores CMOS para evitar el excesivo retardo que se produciría de forma acumulativa al transmitir valores altos. Se proveen varios de estos multiplexores independientes para cada salida, de tal forma que sea posible realizar dos conexiones independientes hacia orientaciones distintas del mismo DMC, lo cual sirve para paliar en cierta medida la reducción de flexibilidad en la conexión de las salidas debido al uso de salidas multiplexadas en vez de independientes.

Las matrices de conmutación están igualmente implementadas a base de multiplexores CMOS bidireccionales de alta conductividad, con lo cual también existe alguna limitación en cuanto a las conexiones factibles a través de una de estas matrices.

Los *bits* de configuración utilizados para programar cada uno de estos multiplexores están mapeados en el espacio de memoria de uno u otro DMC. La distribución lógica de esta memoria no es inmediata ya que está supeditada a la implementación física final. Por ejemplo, los multiplexores de salida a la derecha de un DMC están en realidad mapeados en el espacio lógico

de memoria del DMC de la derecha en vez de en el suyo propio. Mediante estos cambios se consigue un mapa de memoria compacto y denso para disponer de bloques de memoria grandes y sin huecos para uso general como se explicará en la sección 2.5 . En las figuras 2.43 a 2.47, que muestran la arquitectura de rutado cercana a cada uno de los bloques programables, se ha utilizado una línea quebrada para delimitar los espacios lógicos a que pertenece cada uno de los elementos de rutado programable.

Primeramente se presentarán los recursos de rutado que rodean al DMC en sí y que por tanto se repiten en las dos direcciones de la matriz bidimensional, continuando con la estructura de rutado de las regiones periféricas del *array* y del árbol de reloj, para terminar exponiendo una breve justificación de los patrones elegidos.

2.3.5.2.1 Recursos de rutado de la matriz de DMCs

La figura 2.43 muestra los recursos de rutado correspondientes a un DMC y su periferia. Toda la estructura se repite a lo largo y ancho de la matriz con la periodicidad de un DMC hasta llegar a las zonas periféricas de la FPGA que serán explicadas en el punto 2.3.5.2.2

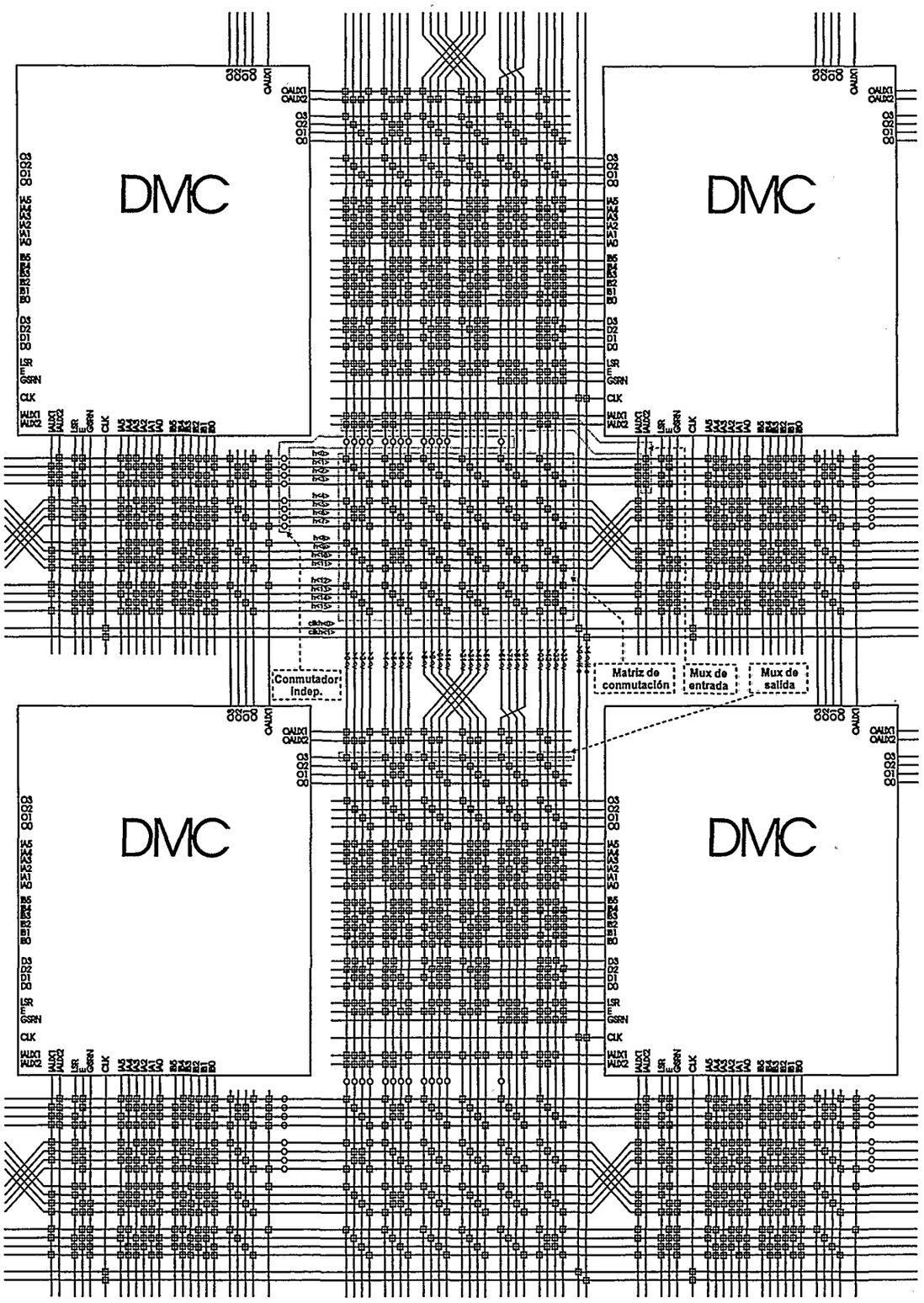

Fig. 2.43 Estructura básica de rutado del DMC

La estructura dispone de 24 canales verticales por columna y 16 horizontales por fila, además de dos verticales y dos horizontales adicionales por columna y fila para el rutado especial del reloj. Los canales de rutado están agrupados en buses de cuatro *bits*, y tienen distintas longitudes para permitir la conexión optimizada de DMCs a distintas distancias. En total hay ocho canales verticales (**v<0>** a **v<7>**) y cuatro horizontales (**h<0>** a **h<3>**) que ocupan lo que un DMC, ocho verticales (**v<8>** a **v<15>**) y ocho horizontales (**h<4>** a **h<11>**) que se extienden dos DMCs, cuatro canales verticales (**v<16>** a **v<19>**) cuya longitud es la de cuatro DMCs, y cuatro líneas largas verticales (**v<20>** a **v<23>**) y otras cuatro horizontales (**v<12>** a **v<15>**) que se extienden toda la altura y anchura respectivamente del *array*. Las líneas largas están pensadas fundamentalmente para permitir la comunicación de señales globales tales como *resets*, señales de *enable* globales, líneas de selecciones de decodificadores o multiplexores anchos, aunque se pueden usar para interconexiones de propósito general entre DMCs distantes. El resto de las líneas se segmentan mediante grandes conmutadores CMOS controlados independientemente con *bits* de configuración propios. Existen por tanto trece conmutadores independientes para los canales verticales y ocho para los horizontales, además de otros dos englobados en la matriz de conmutación cuya misión es conectar las líneas horizontales y verticales de reloj.

Por cada terminal de entrada del DMC sólo se dispone de un multiplexor NMOS, por lo que sólo puede conectarse una entrada, además de un '1' ó '0' lógico constante.

Por cada terminal de salida **O0** a **O3** se dispone en cambio de dos multiplexores CMOS independientes, uno para realizar una conexión a la derecha y otro para realizarla arriba o a la izquierda. Las salidas auxiliares sólo cuentan con un multiplexor CMOS. Todos los multiplexores de salida pueden ponerse en cualquier caso en circuito abierto.

Las matrices de conmutación proveen un multiplexor bidireccional independiente por cada línea horizontal. De esta forma, una línea vertical podría conectarse a varias horizontales, pero dos verticales no pueden conectarse a la misma línea horizontal. Además, cada uno de estos multiplexores pueden dejarse en circuito abierto para no realizar ninguna conexión si así se requiere.

Como se explicó anteriormente, los multiplexores de salida de un DMC se controlan con *bits* de configuración mapeados en el espacio lógico del DMC de la derecha por motivos de optimización del *layout* físico que implementa la arquitectura. El espacio lógico que engloba los

recursos de rutado de los bloques programables está delimitado por la línea a trazos en las figuras 2.43 a 2.47.

2.3.5.2.2 Recursos de rutado de las células periféricas

La estructura de rutado que rodea las células periféricas extiende la estructura de la matriz de DMCs con el objeto de poder recolocar cualquier bloque funcional pre-implementado en cualquier lugar de la matriz, sin importar que se trate de un borde de la FPGA. El espacio lógico de memoria de configuración (delimitado por la línea a trazos en las figuras 2.43 a 2.47) de las células de entrada-salida, así como su estructura particular de rutado, resulta por tanto bastante diferente en función del lado del *array* en el que se encuentre, aunque las células en sí sean funcionalmente equivalentes. En cualquier caso, los mapas de memoria son compatibles en el sentido de que las palabras de configuración equivalentes de distintos bloques se encuentran en la misma dirección relativa de mapas lógicos diferentes.

Las figuras 2.44 a 2.47 muestran las cuatro esquinas de la FPGA. Las líneas a trazos delimitan los espacios lógicos de memoria de configuración de cada uno de los bloques funcionales programables. Las salidas CIPO de las células de entrada-salida disponen de varios multiplexores CMOS de salida (dos o tres según el caso) para permitir mayor flexibilidad.

Los IICs (*Internal Interconnection Cells*) se encuentran en el lado inferior del *array* hacia la esquina inferior derecha. Estos bloques no son más que multiplexores de rutado de entrada y de salida que permiten conexiones entre las pistas del *array* y las señales de control de microprocesador y del CAB (*Configurable Analog Block*). En la realización particular propuesta, cada uno de los IICs tiene nueve entradas (salidas de la lógica programable) y cuatro salidas (entradas para la lógica programable). Ejemplos de entradas al IIC (salidas de la lógica programable) serían las entradas de datos para los DACs del CAB o las entradas de interrupción para el microprocesador; ejemplos de salidas del IIC (entradas para la lógica programable) serían las salidas de los comparadores del CAB o las salidas del ADC.

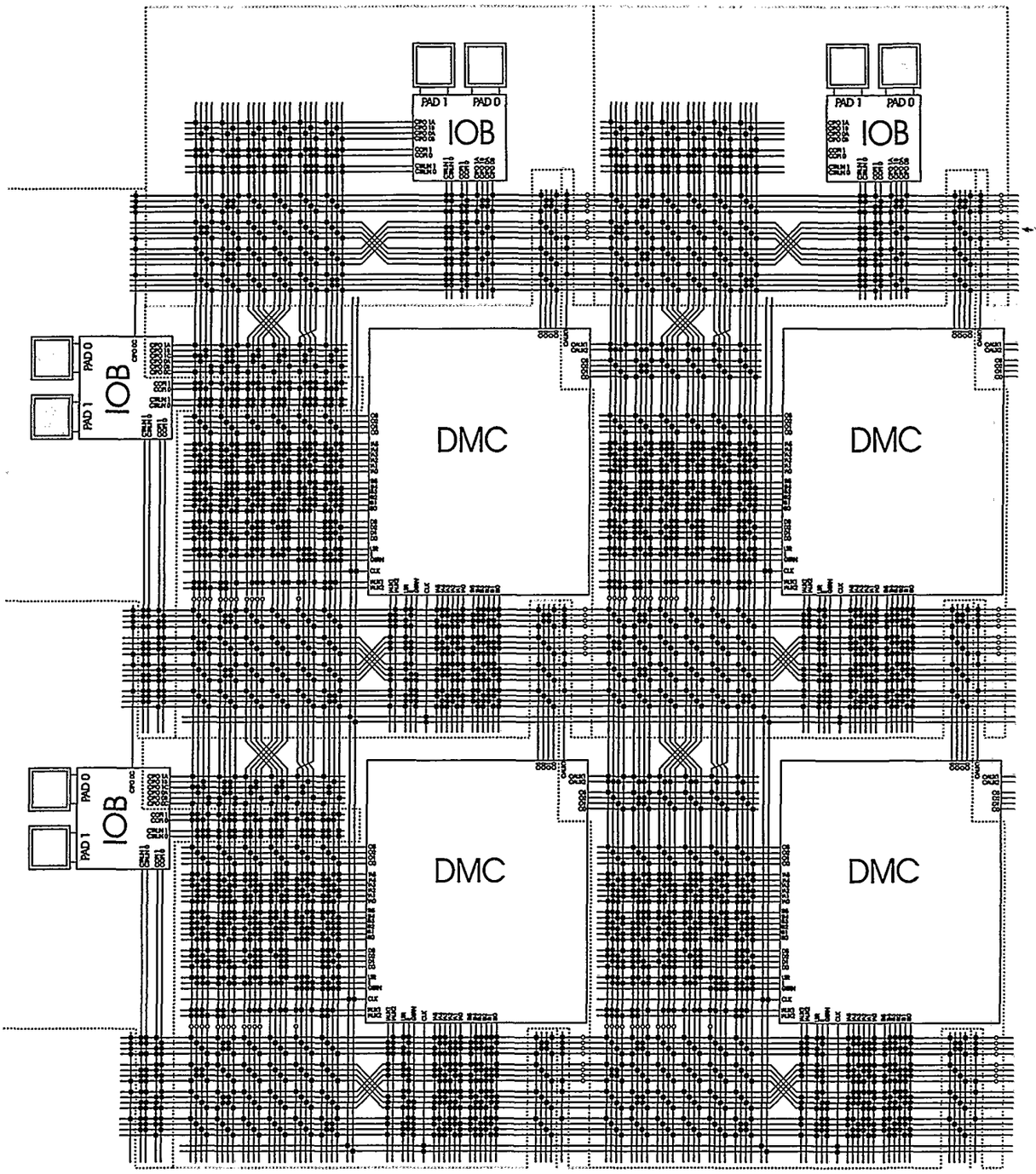

Fig. 2.44 Esquina superior izquierda de la FPGA

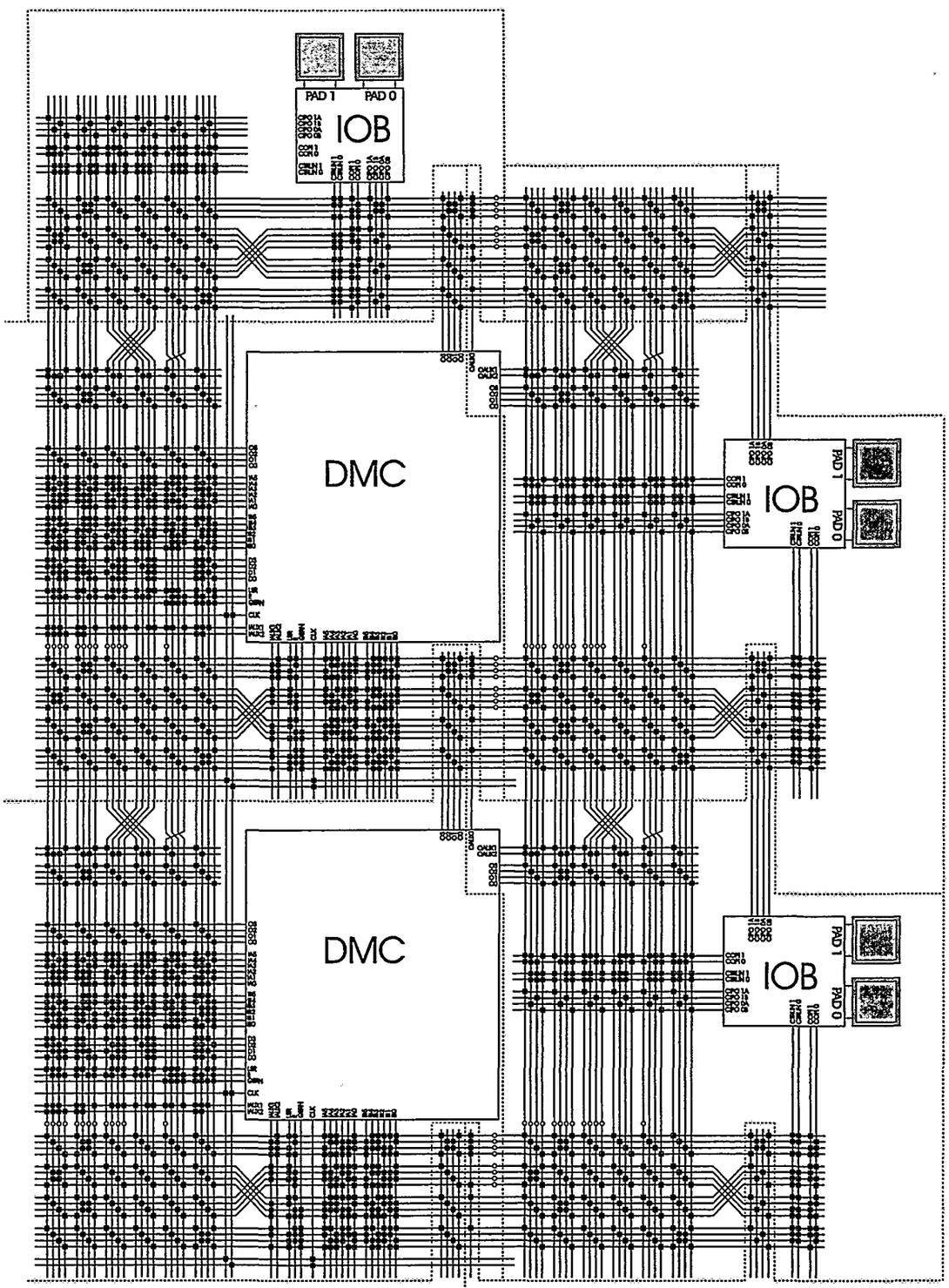

Fig. 2.45 Esquina superior derecha de la FPGA

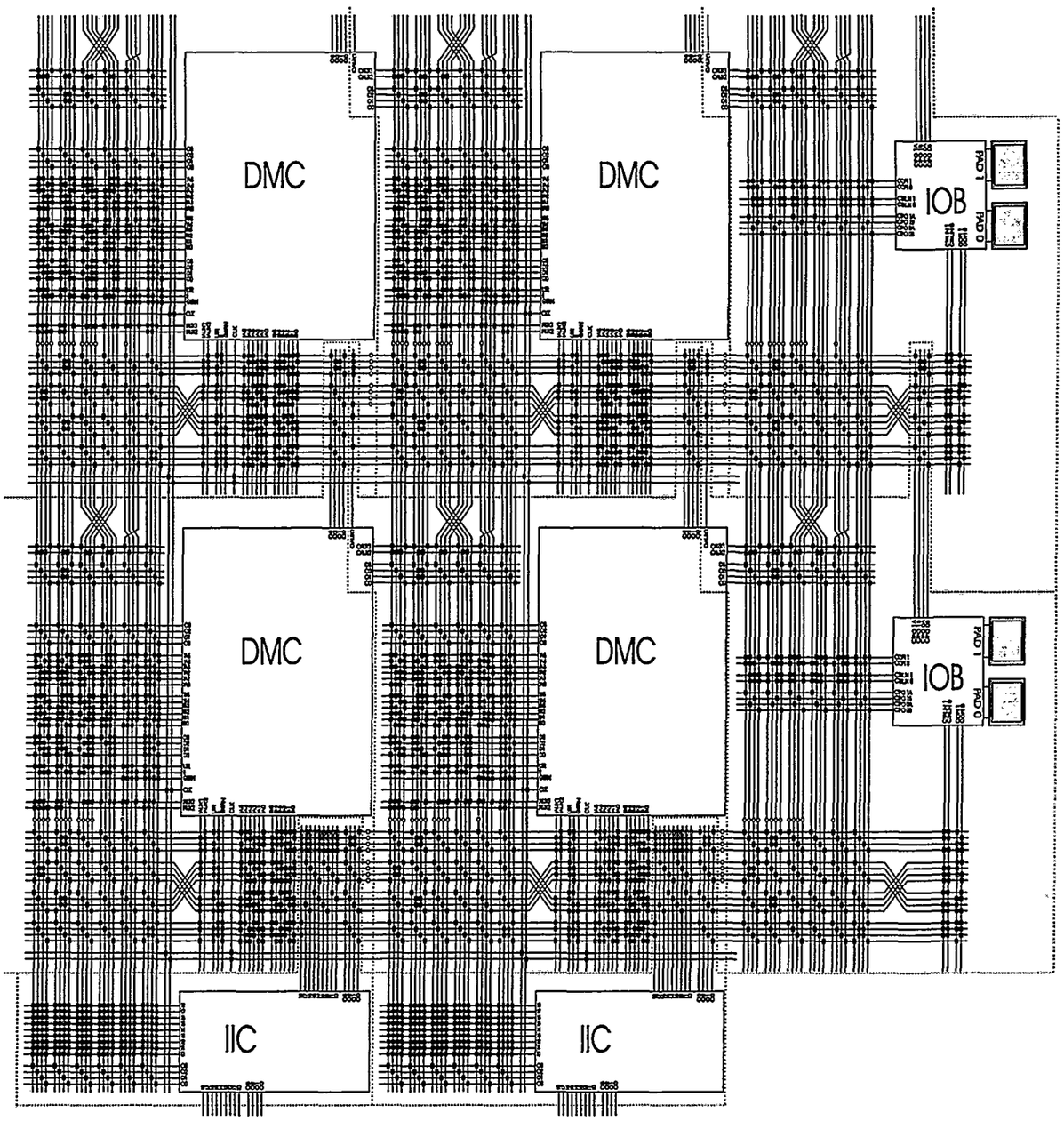

Fig. 2.46 Esquina inferior derecha de la FPGA

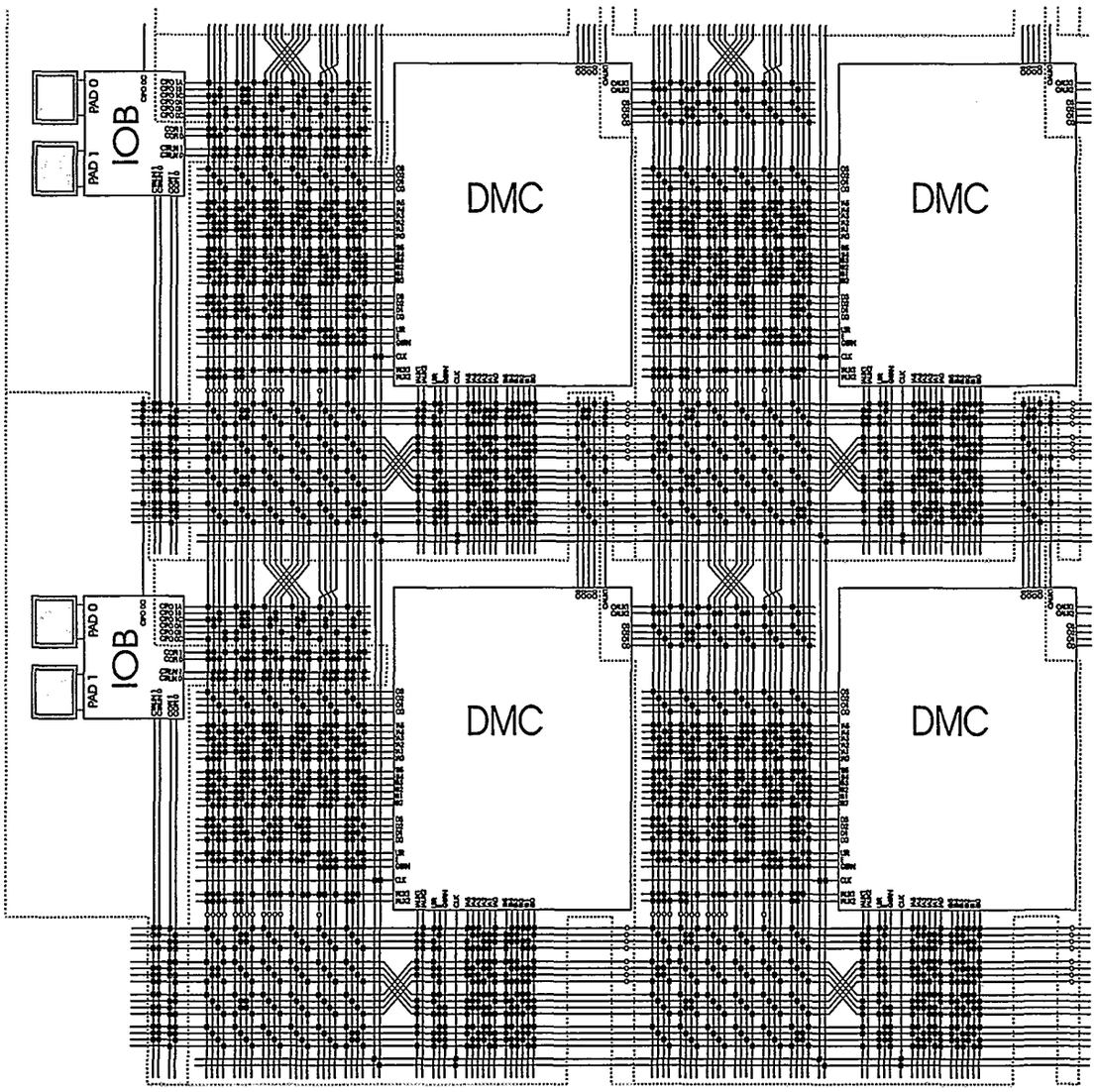

Fig. 2.47 Esquina inferior izquierda de la FPGA

2.3.5.2.3 Rutado de reloj

En la práctica totalidad de las implementaciones de FPGAs de granularidad media y alta, las redes de reloj se tratan de forma separada al resto de las redes del circuito para minimizar los efectos de *skew* o retardos diferenciales de la misma señal de reloj en puntos distintos del *array* (las FPGAs de granularidad más baja tratan las redes de reloj como cualquier otra señal, incluso implementando los elementos secuenciales a base de células lógicas combinacionales [BRO92a]).

En el ejemplo propuesto puede verse cómo las redes de reloj son independientes del resto de los recursos de rutado. La figura 2.48 muestra cómo se seleccionaría el reloj mediante multiplexores (*clock selectors*) de entre una serie de señales de reloj disponibles. Estas señales de reloj se generarían mediante un bloque especial controlado por el microprocesador como se explicará en el capítulo 3 (El Interfaz entre el Microprocesador y las Células Programables en FIPSOC), aunque algunas de estas señales (**customCLK1** y **customCLK2** en la figura 2.48) podrían entrar directamente a través de un IIC, siendo esta una manera segura de generar relojes mediante lógica programable para la lógica programable.

Los relojes se generarían por tanto verticalmente en las columnas y podrían conectarse a las filas en geometría de árbol. Las entradas de reloj de los DMCs podrían conectarse a las líneas horizontales o a las verticales.

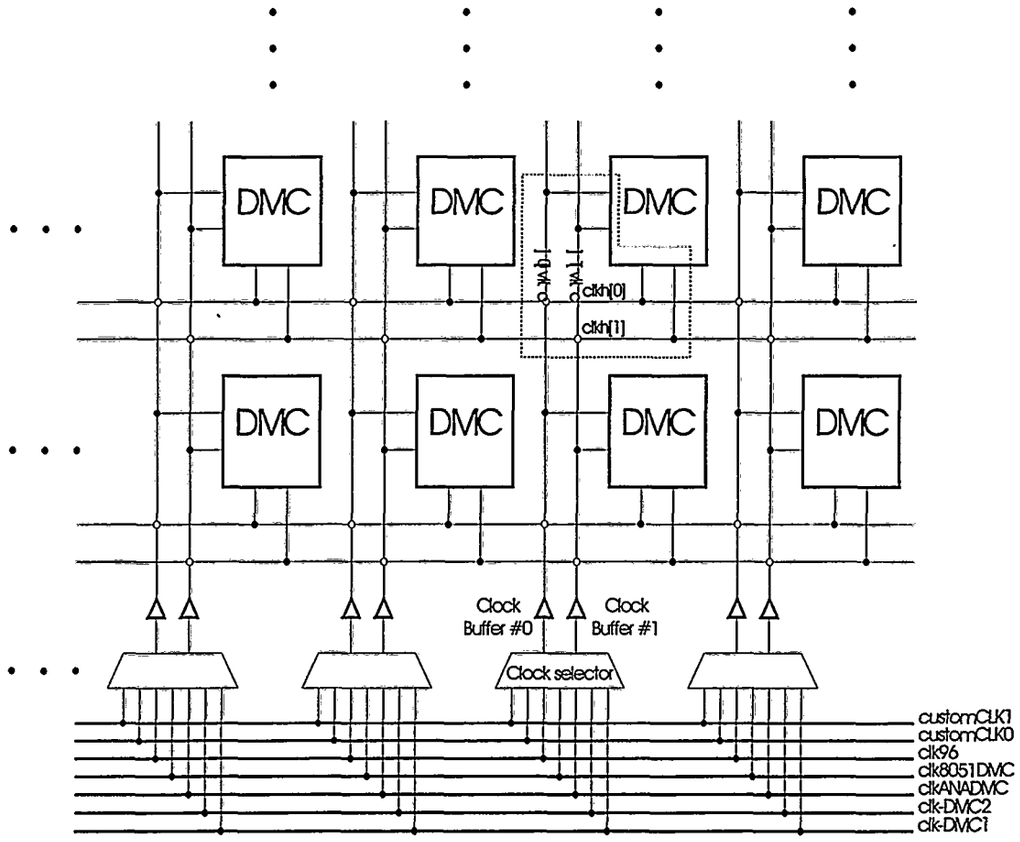

Fig. 2.48 Arquitectura de rutado de redes de reloj.

La figura 2.49 muestra cómo se realizaría la selección de reloj dentro de un DMC. Como puede verse, se intercalan *buffers* antes del multiplexor que realiza la selección del reloj, por lo cual las

líneas de reloj están siempre igual de cargadas independientemente del número de DMCs que estén lógicamente conectados a cada una. De esta forma, si no se conecta ninguna línea horizontal a una vertical, las líneas verticales tienen siempre la misma carga, al igual que las horizontales. Las únicas variaciones de carga se producen al conectar una o varias líneas horizontales a la misma línea vertical, caso en el cual será necesario comprobar que dos DMCs no estén conectados a la misma red de reloj desde una línea horizontal y una vertical sin cargar, ya que una situación así produciría retardo diferencial peligroso entre las mismas señales lógicas de reloj. La misma técnica se utiliza en la selección de reloj de los *buffers* de la figura 2.48, de tal forma que las señales originales de reloj que vienen del bloque generador de reloj tienen siempre la misma carga, por lo cual no influye el número de columnas conectadas a una red y se preserva con seguridad la fase diferencial de las señales de reloj distintas.

Con simulaciones físicas se puede comprobar que el retardo diferencial que así se obtiene entre puntos distintos de la misma red de reloj es en todo caso menor que el retardo mínimo producido en un DMC (tanto el retardo de propagación de la LUT como el del FF), gracias a lo cual puede se trabajar a salvo de problemas derivados del *skew* de la señal de reloj.

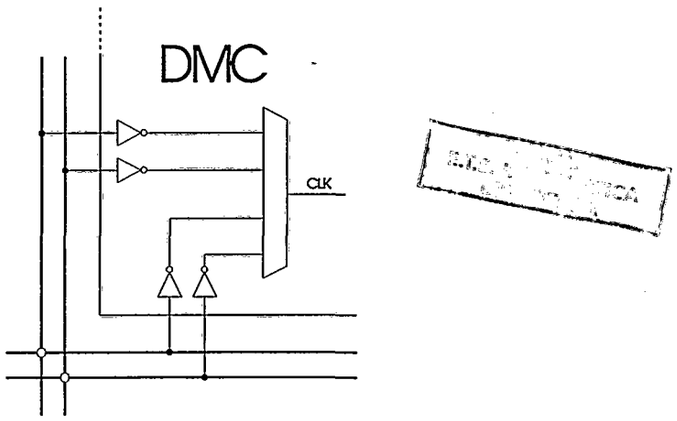

Fig. 2.49 Selección de la señal de reloj dentro de un DMC

2.3.5.2.4 Análisis de rutabilidad

En el Apéndice A se resumen las conclusiones básicas del estudio clásico de Brown y Rose [BRO92a] sobre los parámetros que influyen en la rutabilidad de arquitecturas programables y sus valores típicamente recomendables. En líneas generales estos criterios han sido respetados en la arquitectura de rutado propuesta aunque teniendo en cuenta consideraciones de orden práctico.

Por ejemplo, las entradas al DMC de uso completamente general, como las entradas a las LUTs, deberían tener rutabilidades en general mayores a las de las entradas de control de propósito especial como por ejemplo la entrada de *reset* global **GSRN** cuya función típica es la de proveer un mecanismo de *reset* global a todos los FFs del circuito en tiempo de inicialización del *chip*. Además, los patrones de rutado no pueden ser escogidos de forma arbitraria sin más que cumplir el criterio de $F_{RR} \geq 0{,}7$, ya que deben permitir todas las conexiones posibles de forma eficaz e igualitaria para todos los terminales del DMC. Por ejemplo, los modos de propósito no general de la parte combinacional del DMC (como memoria de cuatro *bits* o como sumador) o de cada mitad (como multiplexor de cuatro entradas) utilizan buses de cuatro *bits* (la granularidad general del DMC) que típicamente podrán estar conectados a las cuatro salidas de un DMC cercano o lejano, por lo cual es importante asegurar que será posible realizar una conexión sencilla a base de buses de cuatro *bits* para estos modos.

Como se muestra en las figuras 2.43 a 2.47, la estructura de rutado se basa en buses de cuatro *bits* debido a la granularidad del DMC en sí (hay cuatro LUTs y cuatro FFs con cuatro salidas principales en un DMC). En vez de dotar de la misma flexibilidad a las entradas y a las salidas, lo cual implicaría conectividades redundantes, la mayor flexibilidad se ha dado a las entradas, que en general resultan más económicas ya que pueden implementarse mediante conmutadores NMOS en vez de CMOS. Las matrices de conmutación permiten al menos reflejar lateralmente estos buses, de forma que en principio la conexión entre una salida y una entrada cualquiera de dos DMCs más o menos distantes debe proyectarse en función de cuál de las cuatro pistas de un bus va a usar esa salida, ya que esa posición relativa (de cero a tres) va en principio a manterse independientemente del rutado detallado final elegido. De esta forma la interconexión puede empezar a proyectarse en tiempo de *technology mapping*, sin tener que llegar a la fase de rutado detallado para descubrir que una conexión es imposible. En ocasiones resulta útil sin embargo poder cambiar una señal de posición relativa en el bus, sobre todo si se trata de una red multipunto que se debe conectar a varias entradas de distintos DMCs. Para ello las matrices de conmutación permiten reflejar los buses de forma directa (esto es, conectar un bus vertical **A<0:3>** a uno horizontal **B<0:3>**) además de de forma inversa (**A<0:3>** a **B<3:0>**). También es posible conectar una salida de un DMC a una posición relativa en el bus distinta de su posición preferencial (las salidas **O0** a **O3** se conectan normalmente a posiciones relativas del bus **0** a **3** respectivamente), y además es habitual que no sólo el programa de *technology mapping* sino que el mismo programa de rutado detallado sea capaz de intercambiar entradas funcionalmente equivalentes (por ejemplo las entradas a la LUT) o incluso salidas para optimizar la

compatibilidad de la conectividad (esto es, elegir la posición relativa más conveniente en el bus de rutado).

Teniendo esto cuenta, una vez fijadas las directrices básicas de los patrones preferenciales de salida (esto es, que las salidas **O0** a **O3** se conectan normalmente a posiciones relativas del bus **0** a **3** respectivamente), los patrones de las entradas a las LUTs se han elegido de tal forma que se cumplieran las premisas expresadas a continuación. Se considera que hay ocho *canales* de cuatro *pistas* cada uno (cada pista tiene una posición relativa de cero a tres dentro del canal), pues los patrones de las líneas largas se diseñan separadamente:

- Dos entradas cualesquiera deben poder conectarse al menos por cuatro canales distintos en la misma posición relativa (es decir, deben compartir al menos cuatro pistas de la misma posición relativa dentro del bus).
- Tres entradas cualesquiera deben poder conectarse al menos por tres canales distintos en la misma posición relativa (es decir, deben compartir al menos tres pistas de la misma posición relativa dentro del bus).
- Cuatro entradas cualesquiera deben poder conectarse al menos por un canal en la misma posición relativa (es decir, deben compartir al menos una pista).
- En el modo sumador y en el modo mlutiplexor debe existir la posibilidad de conectar directamente las cuatro pistas de tantos canales como sea posible a ambas entradas de datos del sumador y los buses de datos del multiplexor de ambas mitades de la parte combinacional del DMC. Esto debe cumplirse tanto para buses directos (con posiciones relativas dentro del canal de cero a tres) como reflejados (de tres a cero).
- En el modo memoria debe poder conectarse directamente las cuatro pistas de tantos canales como sea posible al bus de direcciones del bloque de memoria (el bus de datos se conecta a las entradas de datos **D0** a **D3**).

Los tres primeros criterios aseguran que cualquier conexión multipunto desde el mismo *driver* hasta dos, tres o cuatro entradas puede realizarse, pues existen conexiones posibles con la misma posición relativa (y por tanto conectables a la misma salida) que siempre podrán llegar a base de transmisiones y giros por matrices de conmutación. En tiempo de *technology mapping* puede entonces preveerse la conexión y reordenar las salidas o entradas del DMC origen para asegurar la compatibilidad con las entradas a las que debe conectarse. En tiempo de rutado detallado en

principio sólo se reordenan las entradas que son funcionalmente equivalentes (hay que comprobar por tanto el modo de funcionamiento de la parte combinacional del DMC para asegurar que las entradas son intercambiables; por ejemplo, en los modos combinacionales simples y complejos, las entradas a las LUTs de cuatro o cinco entradas son normalmente intercambiables siempre que las comunes lo sigan siendo, mientras que en los modos de multiplexor o de memoria este intercambio está mucho más limitado, a la vez que en el modo sumador es prácticamente imposible).

El siguiente programa se escribió para comprobar el primer criterio. Para comprobar los dos criterios siguientes se modificó ligeramente (se anidaron uno y dos bucles más sobre el principal para disponer de más entradas que comprobar a la vez), mientras que la conectividad de los modos especiales (memoria, sumador, etc.) se comprobó manualmente. Estudios parecidos se realizaron con el resto de las entradas al DMC, en particular con las entradas de datos **D0** a **D3** y con las entradas de control y auxiliares para la parte secuencial. En el caso de los bloques periféricos se realizó un estudio análogo inmediato puesto que se reaprovecharon patrones de rutado de entradas y salidas del DMC para estas células.

```
#define num_pistas 4        /* Pistas por canal */
#define num_canales 8       /* Canales entre horizontales y verticales */
#define num_entradas 12     /* Número de entradas al DMC comprobadas */
#define umbral 4            /* Mínimo número requerido de pistas comunes */

struct patt_line {int data[pistas*canales]};
typedef struct patt_line patt;

void main(void)
{
  int in1, in2, pista, canal, cuenta, fallos;

  static patt patron[inputs] = /* Estructura de datos de los patrones */
  {
    {1,1,0,1,   1,0,1,1,   1,1,1,0,   0,1,1,1},
    {1,0,1,1,   1,1,0,1,   1,1,1,0,   0,1,1,1},
    .
    .
    .
    {0,1,1,1,   1,0,1,1,   1,1,0,1,   1,1,1,0}
  }

  fallos=0;
  for(in1=1; in1<num_entradas; in1++)
    for(in2=0; in2<in1; in2++)
      for(pista=1; pista<num_pistas; pist++)
```

```
      {
        cuenta=0;
        for(canal=0; canal<num_canales; canal++)
          if(patron[in1].data[pista+canal*num_pistas]&&
             patron[in2].data[pista+canal*num_pistas])
             cuenta++;
        if(cuenta<umbral)
        {
          printf("Pista %d se conecta a entradas %d y %d por %d puntos
(<%d)\n",
                  pista, in1, in2, cuenta, umbral);
          fallos++;
        }
      }
  printf("\nFallos = %d\n", fallos);
}
```

Este algoritmo puede utilizarse sobre cualquier arquitectura de rutado de granularidad media o alta suscpetible de ser utilizada en aplicaciones en las que se necesiten transferencia y operaciones de datos en paralelo (por ejemplo, en aplicaciones de tipo *datapath*).

Con respecto a los criterios presentados en el Apéndice A sobre rutabilidad externa de arquitecturas programables, las recomendaciones presentadas en [BRO92a] han sido observadas en general, aunque debe notarse que la estructura propuesta es relativamente distinta a sus precedentes y ni siquiera responde fielmente al modelo de FPGA sobre los que estos criterios fueron extraídos. En cualquier caso, las siguientes conclusiones son aplicables:

- Dado que se utilizan multiplexores de 32 a uno, y dos de estas 32 entradas son fijas ('1' y '0' lógicos), se dispone de 30 entradas para 40 canales entre horizontales y verticales, lo cual supone un factor de flexibilidad F_{RR} de 0,75, mayor del 0,7 recomendado en [BRO92a] (este factor se explica en el Apéndice A).

- El número de lados T (explicado en el Apéndice A) a los que cada entrada puede conectarse es 2, lo cual coincide con lo recomendado en [BRO92a]. No obstante, no debe olvidarse que la arquitectura es fuertemente direccional, por lo cual el flujo de datos debe ser de izquierda a derecha y de abajo a arriba de forma preferencial. Por supuesto que es posible comunicar datos de derecha a izquierda (por ejemplo utilizando las salidas de realimentación a la izquierda del DMC, muy útiles para la implementación de máquinas de estados y otros circuitos secuenciales) o de arriba a abajo, pero su implementación no resultaría óptima ya que se requeriría un mayor número de recursos de rutado.

- Finalmente, las matrices de conmutación, que conceptualmente engloban a los conmutadores independientes que segmentan los canales, permiten de esta forma conectar cada canal a otros tres (el canal de abajo mediante el conmutador independiente y los canales de los lados mediante el multiplexor), lo cual supone una flexibilidad F_{SM} de 3 (según se explica en el Apéndice A) como se recomienda en [BRO92a].

Los patrones para el resto de las señales se eligieron de forma análoga, estudiando su rutabilidad y manteniendo estos factores en los límites recomendados cuando era posible. Las entradas de propósito menos general, en particular la entrada de *reset* global **GSRN** y la entrada auxiliar **IAUX2**, tienen menos flexibilidad que las entradas más comúnmente usadas para distintas funciones, en particular las entradas a las LUTs y las salidas principales (no auxiliares) del DMC.

Finalmente, es importante notar que el factor que más ha influido en los recortes de flexibilidad de estas señales, e incluso en el uso de multiplexores en terminales de salida, ha sido la economía de *bits* de configuración. En particular es justo decir que el hecho de que la memoria de configuración debía ser compacta para crear grandes bloques continuos de memoria (por motivos que se introducirán en la sección 2.5) ha limitado a un número exacto y concreto (una potencia de 2) el número de *bits* disponibles, lo cual ha mediatizado en cierta medida la cantidad de recursos de rutado disponibles y la flexibilidad impartida a la estructura programable.

En [BAE99] puede encontrarse un estudio más detallado que confirma que la utilización de multiplexores en los recursos de rutado de FIPSOC tiene efectos prácticamente despreciables en la rutabilidad.

2.4 Bloques analógicos programables

El primer paso a la hora de diseñar un subsistema analógico programable consiste en decidir cuál va a ser el ámbito de aplicación del mismo, esto es, cuál va a ser el compromiso entre flexibilidad (o capacidad de ser utilizado en una gran variedad de aplicaciones distintas) y optimización (para ser utilizado en aplicaciones concretas). En general los subsistemas analógicos programables de propósito verdaderamente general, que en general proponen una baja granularidad para maximizar su versatilidad, no han dado buenos resultados y no han sido aceptados por el mercado para su aplicación en productos reales. En particular debemos mencionar entre estos las arquitecturas de baja granularidad basadas en arquitecturas de capacidades conmutadas

propuestas (y ya retiradas del mercado) por Pilkington primero y Motorola después [CHO95] [BRA96] [ZHA96] [AND97].

En general parece claro que el concepto de un sistema analógico programable de propósito general, equivalente a la FPGA en el caso digital, resulta cuestionable y a menudo impracticable. Resulta por tanto necesario definir cuál será el rango de aplicaciones a que se dedicará el circuito antes de intentar definir su arquitectura; y en el caso de tratarse de un sistema programable de señal mixta, esta decisión influirá igualmente en la arquitectura interna a elegir para la FPGA, como se explicó en la sección anterior.

Como se mencionó al principio de este capítulo, nuestro interés se centra en arquitecturas de sistemas programables para aplicaciones de señal mixta. Es por ello que los bloques analógicos programables (o CABs) que se proponen contienen células configurables analógicas de funcionalidad restringida, en principio valederas para funciones de acondicionamiento y conversión de señal típicamente utilizadas en *front-ends* de adquisición de datos para aplicaciones industriales. La programabilidad se realizará en cualquier caso utilizando conmutadores CMOS tanto para configurar las propias células como para interconectar (rutar) los bloques constitutivos entre sí. El control de impedancia de las ramas significativas se realiza mediante escalado de conmutadores de programación (como se detallará en el punto 2.4.2) en vez de utilizar tensiones analógicas variables como propuso LEE [LEE92] [LEE95].

2.4.1 Estructura general del CAB

La figura 2.50 muestra esquemáticamente la estructura de un bloque analógico programable o CAB. Las entradas se conectan directamente a una primera etapa de ganancia que las acondiciona antes de entrar en los conversores analógico-digitales del bloque de conversión de señal, que además produce las salidas del CAB mediante conversores digital-analógicos. Ambos bloques utilizan señales de referencia provenientes del bloque de referencia, y además se dispone de comparadores que permiten detectar la polaridad de las señales acondicionadas respecto a umbrales de referencia. Esta arquitectura permite acondicionar señales, compararlas, y realizar conversiones analógico-digitales y digital-analógicas, que en principio cubren las necesidades de aplicaciones industriales típicas.

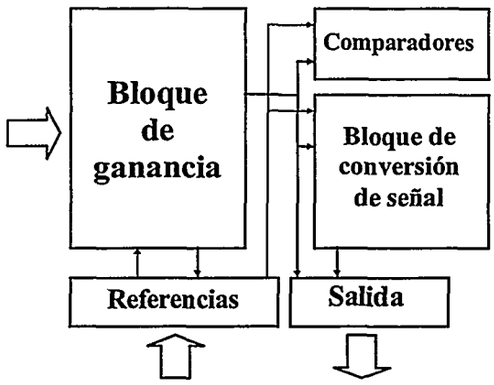

Fig. 2.50 Diagrama de bloques de la estructura del CAB

En la figura 2.51 se muestra en detalle un ejemplo de realización práctica de CAB susceptible de ser utilizado en una arquitectura de tipo FIPSOC. Se dispone de cuatro canales diferenciales de tres amplificadores programables cada uno que en total permiten una ganancia teórica combinada de hasta 70,5 dB. La señal se mantiene diferencial a lo largo de toda la amplificación para minimizar el ruido en modo común que pudiera filtrarse por el substrato, tensión de alimentación, etc., y sólo se convierte en unipolar en el momento de la comparación (cada canal dispone de un comparador con umbral programable) o de la conversión a digital. El bloque de conversión dispone de cuatro conversores digital a analógico de ocho *bits* con cuatro comparadores de muestreo (*sample and hold*) y cuatro registros de aproximaciones sucesivas, de tal forma que se tiene cuatro canales de conversión de ocho *bits* que pueden usarse como DACs o como ADCs. Cada dos de estos conversores de ocho *bits* pueden asociarse para formar uno de nueve, o los cuatro pueden formar uno de diez, tanto en modo DAC como ADC. También es posible utilizar dos o cuatro ADCs de ocho *bits* para formar una estructura de conversión segmentada (conversor *pipeline*) que divide entre dos o entre cuatro respectivamente el tiempo de conversión.

Las señales normales de referencia se generan mediante un divisor resistivo, aunque es posible utilizar las mismas señales de salida de los canales de amplificación, e incluso las señales de salida de los DACs, como referencias. Una aplicación inmediata de esta posibilidad consiste en utilizar dos DACs para generar las referencias del DAC o ADC restante, de tal forma que las referencias pueden ser programadas dinámicamente aumentando el número aparente de *bits* de precisión si la conversión se realiza en dos etapas, una de ajuste de rango y otra de conversión en el subrango elegido.

Los canales de amplificación son totalmente diferenciales al estar construidos a base de amplificadores totalmente diferenciales en los cuales el modo común de las salidas diferenciales se programa con una señal de referencia. Como referencia puede utilizarse cualquier señal del divisor resistivo u otras provenientes de los DACs o incluso de las propias señales de otros canales de amplificación. De esta forma, puede imponerse dinámicamente el modo común a la salida de los canales diferenciales de amplificación, lo cual permite realizar sumas (con las señales positivas) y restas (con las negativas) ponderadas (mediante los factores programables de amplificación) con las fases diferenciales de los canales de amplificación.

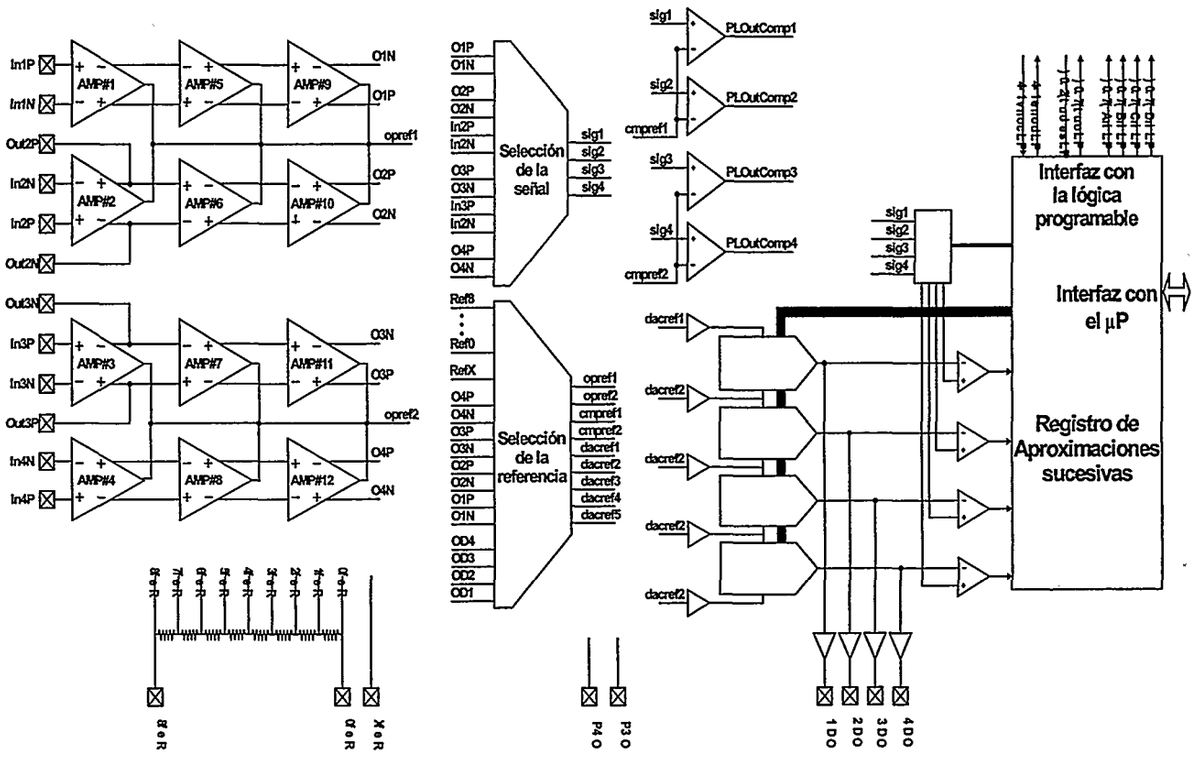

Fig. 2.51 Estructura interna del CAB

2.4.2 Amplificadores

Cada canal del bloque de amplificación del CAB se compone de tres secciones de amplificación programables. La estructura interna de estas secciones de amplificación se muestra en la figura 2.52.

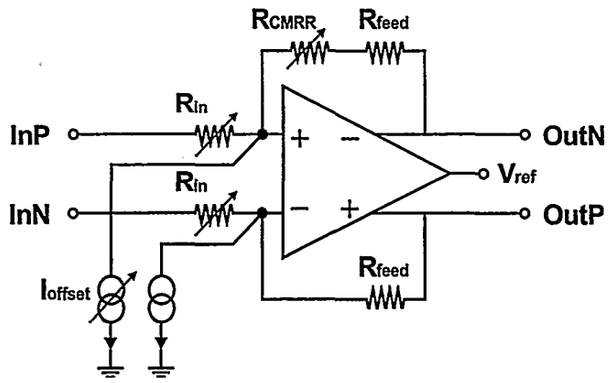

Fig. 2.52 Sección de amplificación

La sección de amplificación está basada en un amplificador operacional con entrada y salida diferencial. El modo común a la salida se establece mediante un tercer terminal **V_{ref}**, mientras que la diferencia de las señales de salida **OutP** y **OutN** sigue a la diferencia (amplificada según lo programado) de las señales de entrada **InP** e **InN** independientemente del modo común a la entrada. Este funcionamiento puede apreciarse en la figura 2.53

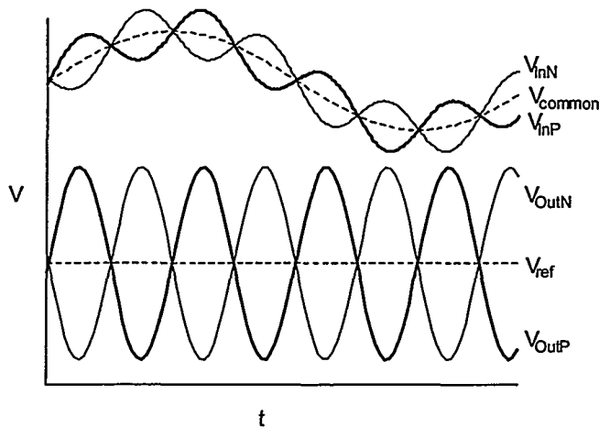

Fig. 2.53 Amplificación diferencial con rechazo al modo común

Este tipo de sección de ganancia programable resulta especialmente indicado para las aplicaciones a las que las arquitecturas de tipo FIPSOC están dedicadas. Su capacidad de ser controlada digitalmente y la programabilidad del *offset* y CMRR hacen que sea fácilmente configurable y compensable de forma automática dentro de sistemas tipo ASIC de mayor

complejidad, especialmente si estos últimos están basados en microprocesador o microcontrolador.

La estructura de bloques del amplificador operacional diferencial propuesto se muestra en la figura 2.54. Se dispone de dos pares diferenciales que actúan sobre una carga simétrica en configuración *folded cascode* [LAK94], de tal forma que uno de ellos actúa sobre el modo común y otro sobre el modo diferencial. Además se incluye una etapa diferencial de salida (dos salidas clase AB replicadas), circuitería de polarización, y una etapa de control de trasconductancia constante [HOG96]. Los pares diferenciales se realimentan de forma independiente, el de modo diferencial mediante lazos de realimentación resistivos clásicos como los mostrados en la figura 2.52 (cada salida realimenta a la entrada de polaridad inversa), y el de modo común midiendo el modo común a la salida mediante un simple divisor resistivo como se muestra en la figura 2.54.

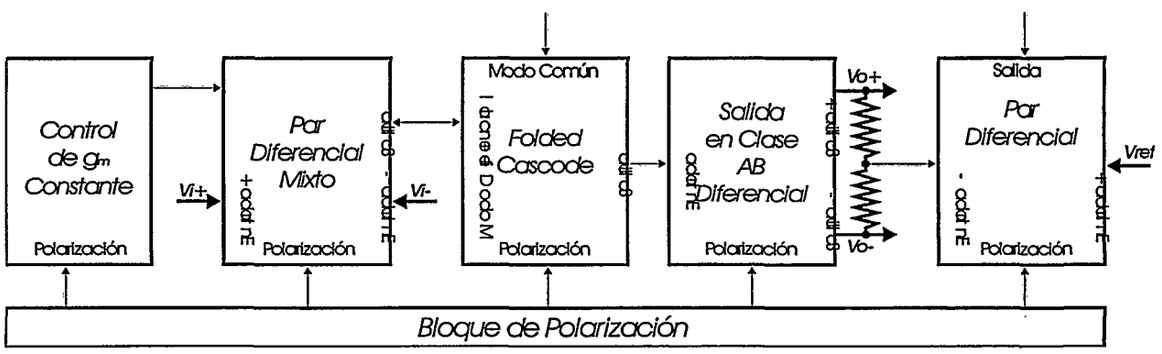

Fig. 2.54 Estructura interna del amplificador operacional diferencial

Mediante esta estructura se consigue un alto rechazo al modo común de entrada, y en particular al ruido en modo común en todos los puntos del circuito. El coeficiente de rechazo al modo común o CMRR (*Common Mode Rejection Ratio*) de la sección de amplificación depende de esta forma del CMRR interno del amplificador, que no es infinito debido al *mismatch* o desapareamiento entre los transistores de la carga cascodo, par diferencial, etapa de salida, etc. Sin embargo, lo que habitualmente limita el CMRR de este tipo de secciones es el desapareamiento de las resistencias de realimentación en modo diferencial. La diferencia entre los cocientes de resistencia de realimentación a resistencia de entrada ($\mathbf{R_{feed}}$ / $\mathbf{R_{in}}$ en la figura 2.52) para cada una de las ramas influye directamente en el CMRR. Para compensar en cierta medida el desapareamiento debido a las diferencias en los valores de las resistencias, se puede modular muy ligeramente la resistencia de una de las ramas de realimentación (mediante la resistencia variable

R_{CMRR} de la figura 2.52) de forma programable, teniendo en cuenta los desapareamientos esperados en función de las limitaciones tecnológicas de la implementación física.

La figura 2.55 muestra la estructura interna de la sección de amplificación completa esquematizada previamente en la figura 2.52. La ganancia de la sección de amplificación se programa variando la resistencia de entrada mientras se mantiene fija la de realimentación, si bien esta última se modula asimétricamente mediante cuatro conmutadores de distinta resistencia equivalente. Modulando la resistencia de entrada en vez de la de realimentación se consiguen factores de amplificación enteros y seguidos (como se explicará seguidamente), además de limitar el consumo de la sección en ausencia de saturación independientemente del factor de ganancia seleccionado.

El *offset* se programa mediante dos fuentes de corriente, una de ellas fija y la otra programable mediante ocho *bits*. De esta forma, la programación completa de cada sección de amplificación precisa 16 *bits*: cuatro para la ganancia, ocho para el *offset* y cuatro para la compensación de CMRR. El *offset* se programa así mediante un valor entre cero y 256, siendo el valor 128 el necesario en condiciones ideales para las que el par diferencial de entrada no tiene *offset* (con este valor se inyecta la misma corriente a cada rama de entrada). El efecto producido por esta inyección de corriente es una tensión diferencial que se suma (o resta) a la tensión diferencial a la salida independientemente del punto de polarización del circuito de entrada. Los escalones de tensión de esta compensación sólo dependen del valor final de la corriente (que a su vez depende de una resistencia de polarización) y de las resistencias de inyección de corriente y de realimentación de la sección, por lo cual toman unos valores relativamente controlados (su indeterminación se mueve en torno al 25% en función de las variaciones de proceso únicamente) que no dependen de la temperatura ni del estado de polarización de la señal de entrada. La corrección es monótona según las simulaciones, si bien por motivos tecnológicos no es esperable una monotonicidad mayor de seis *bits*.

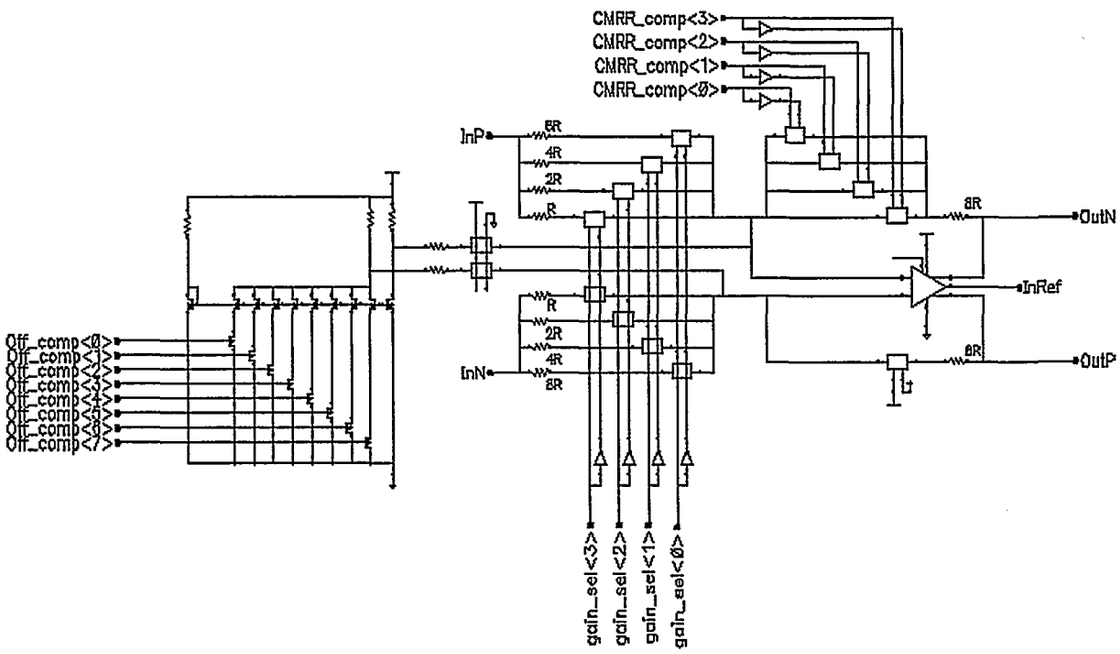

Fig. 2.55 Estructura interna de la sección de amplificación

Para programar la ganancia se dispone en la rama de entrada de cuatro resistencias de valores $\mathbf{R_{feed}}$, $\mathbf{R_{feed}}/2$ $\mathbf{R_{feed}}/4$ y $\mathbf{R_{feed}}/8$ conmutables de forma independiente mediante cuatro *bits* de configuración. Los valores posibles para la ganancia, expresados en la ecuación 2.5, son enteros seguidos entre cero y quince.

$$\frac{V_{Out}}{V_{In}} = \frac{R_{feed}}{\frac{R_{feed}}{b_0} // \frac{R_{feed}}{2 \cdot b_1} // \frac{R_{feed}}{4 \cdot b_2} // \frac{R_{feed}}{8 \cdot b_3}} = b_0 + 2 \cdot b_1 + 4 \cdot b_2 + 8 \cdot b_3 \quad \text{(eq. 2.5)}$$

Los conmutadores CMOS que conectan las resistencias de entrada para seleccionar la ganancia introducen un término no despreciable en la ganancia total de la rama, por lo cual tienen que estar escalados proporcionalmente al valor de cada resistencia que conmutan. De igual forma, las resistencias de realimentación, aunque fijas, se conectan mediante un conmutador CMOS similar siempre activado. La situación se esquematiza en la figura 2.56, que representa un lazo de realimentación en un amplificador unipolar programable. Para este amplificador, la ganancia puede expresarse según la ecuación 2.6, en la cual $\mathbf{R_{CMOS1}}$ y $\mathbf{R_{CMOS2}}$ simbolizan las resistencias equivalentes de los conmutadores $\mathbf{S_1}$ y $\mathbf{S_2}$.

$$G = \frac{R_{feed} + R_{CMOS2}}{R_{inp} + R_{CMOS1}} \quad \text{(eq. 2.6)}$$

La resistencia equivalente de estos conmutadores depende de la temperatura, del proceso y del estado de polarización del conmutador, ya que en tensiones altas será fundamentalmente la parte PMOS la que domine mientras que en tensiones bajas será la NMOS la importante. En cualquier caso, los conmutadores son diseñados para presentar en todas las condiciones una impedancia significativamente menor que las resistencias que conmutan, de tal forma que la caída óhmica a través de uno de ellos es significativamente menor que a través de la resistencia conectada. Por ello, el estado de polarización de ambos conmutadores $\mathbf{S_1}$ y $\mathbf{S_2}$ es aproximadamente el mismo, ya que todos los drenadores y fuentes están a una tensión aproximadamente igual al valor de referencia (marcado como masa de señal en la figura 2.56). De esta manera, dado que los conmutadores están programados (activados o no) por señales digitales (masa o alimentación) provenientes de *bits* de configuración, la relación entre las resistencias equivalentes de los conmutadores sólo depende de la relación entre sus tamaños, o específicamente de la relación entre las razones **W/L** de uno y otro. Haciendo que esta relación sea la misma a la relación entre las resistencias que conmutan, se consigue que la expresión de la eq. 2.6 sea constante e independiente de la temperatura, condiciones de proceso, tensión de alimentación, y estado de polarización (valor actual de la tensión de referencia del amplificador).

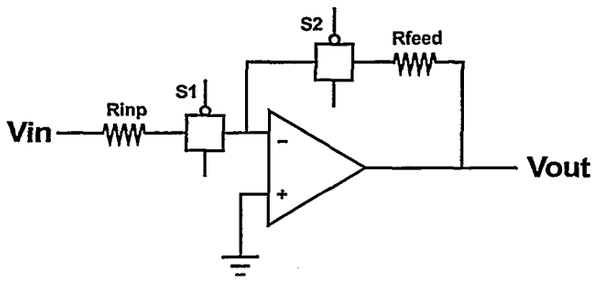

Fig. 2.56 Principio de funcionamiento de la sección de amplificación

Esta técnica se ha extendido al amplificador diferencial como se muestra en la figura 2.55, de tal forma que se mantiene siempre la relación entre el tamaño del transistor NMOS, el tamaño del transistor PMOS, y el valor de la resistencia conmutada, para cualquier resistencia de entrada o de realimentación utilizada en la sección de amplificación (incluyendo las resistencias de inyección de corriente de las fuentes de corrección de *offset*), de manera que cualquier factor de amplificación resulta constante en cualquier condición. En la rama de realimentación superior se dispone además de tres conmutadores CMOS adicionales de tamaño sensiblemente menor al

nominal (usado en ambas ramas) conectados en paralelo a éste para así poder asimetrizar ligeramente la resistencia de las ramas y compensar parcialmente los efectos del desapareamiento de las reistencias de realimentación que como antes de mencionó limitan el CMRR de la sección.

Finalmente, es interesante notar que el primer amplificador de cada una de las dos secciones intermedias es directamente accesible desde el exterior, de forma que pueden desconectarse todas las resistencias conectadas a él, tanto las de entrada (programando una ganancia nula) como las de realimentación (mediante los conmutadores de compensación). De esta manera el usuario dispone de un amplificador diferencial con salida diferencial y *offset* programable en tiempo continuo con el que puede realizar cualquier montaje de propósito general (filtros, amplificadores con resistencias de precisión para maximizar el CMRR, rectificadores, amplificadores logarítmicos, etc.).

2.4.3 Comparadores

Como se muestra en la figura 2.51, se dispone de cuatro (uno por canal) comparadores clásicos basados en amplificadores de transconductancia u OTAs (*Operational Transconductance Amplifiers*) de tres etapas, con un par diferencial de entrada mixto como el utilizado en el diseño del amplificador operacional diferencial de la sección de amplificación. Cada dos comparadores comparten el nivel umbral con el que comparan, que a su vez se elige entre las tensiones de referencia, salidas del DAC, o las mismas salidas de los canales de amplificación.

2.4.4 Conversores

El bloque de conversión incluido en el CAB se compone de cuatro DACs de ocho *bits* y cuatro registros de aproximaciones sucesivas, de tal forma que cada DAC puede utilizarse como ADC de forma independiente al resto. Con este ejemplo quiere ilustrarse las distintas posibilidades en cuanto a programabilidad y flexibilidad general de arquitecturas de conversión susceptibles de ser usadas en cualquier otro rango de aplicaciones del sistema tipo FIPSOC particular.

Cada uno de los DACs está compuesto por un divisor resistivo de 256 resistencias distribuidas en forma de matriz, de tal forma que las entradas digitales al DAC seleccionan la señal entre 256

puntos de este divisor. Cada 32 resistencias unitarias se conecta una resistencia de preescalado, como muestra la figura 2.57. De esta forma se consigue establecer de forma rápida el primer rango de conversión evitando que las capacidades de las líneas de selección y de la carga de salida lleven a las pequeñas resistencias unitarias fuera de su rango habitual de polarización, además de minimizar los efectos de desapareamiento acumulativo del gran divisor resistivo de 256 etapas.

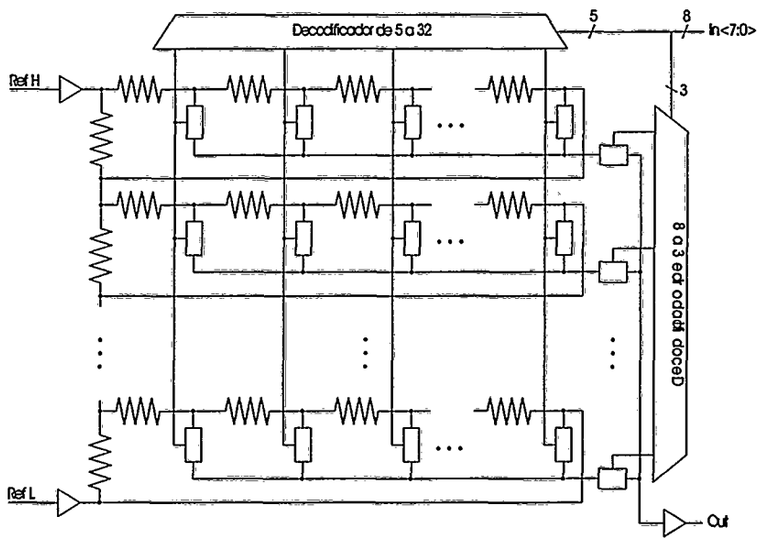

Fig. 2.57 Esquema de un DAC de 8 *bits*

Los cuatro DACs de ocho *bits* se conectan en cascada como se esquematiza en la figura 2.58, de tal forma que cada dos DACs adyacentes comparten una referencia. Las referencias son impuestas a través de *buffers* seguidores de tensión que pueden ser inhabilitados para combinar varios DACs: Si se inhabilita el *buffer* común a dos DACs, se obtiene un divisor resistivo de 512 resistencias con una referencia alta (la referencia alta del DAC superior) y otra baja (la baja del DAC inferior), consiguiendo así un DAC de nueve *bits* sin más que cortocircuitar las entradas de control digital de ambos DACs. Inhabilitando los tres *buffers* compartidos se configura todo el bloque como un único DAC de 10 *bits*.

En cualquier caso se proveen multiplexores que permiten usar el primer ADC con la salida de cualquiera de los canales de amplificación, por lo cual es posible multiplexar el mismo DAC en la máxima resolución (diez *bits*) para cuatro canales distintos (a una velocidad de conversión cuatro veces menor, lógicamente).

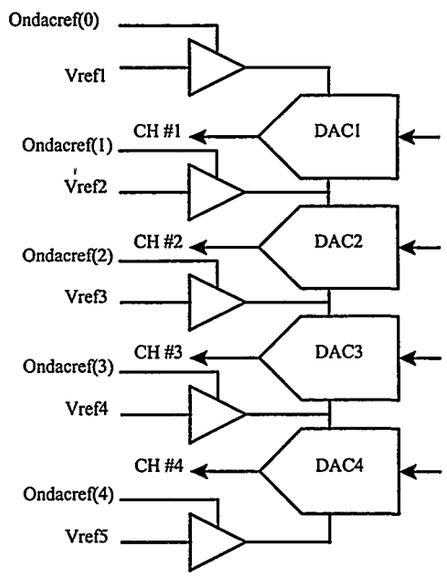

Fig. 2.58 Estructura de los DACs del bloque de conversión

Al lado de cada DAC se encuentra un registro de aproximaciones sucesivas o SAR (*Successive Approximation Register*) como el mostrado en la figura 2.59. Este circuito permite ejecutar un algoritmo clásico de aproximaciones sucesivas por búsqueda binaria para utilizar el DAC como ADC. Cada SAR tiene capacidad para ejecutar el algoritmo con ocho, nueve o diez *bits* para poder utilizar las distintas combinaciones de DACs de ocho *bits* básicos.

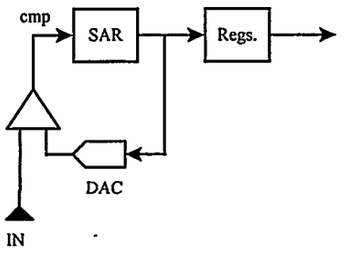

Fig. 2.59 Esquema del registro de aproximaciones sucesivas de cada DAC

Los *buffers* de las referencias son en realidad amplificadores operacionales unipolares con par de entrada mixto para permitir una operación *rail-to-rail*, de tal forma que los niveles de referencia pueden estar entre masa y alimentación. La misma salida de un DAC puede seleccionarse como

señal de entrada a uno de estos *buffers*, con lo cual pueden crearse subrangos programables dinámicamente en los cuales se realiza la conversión.

Aunque normalmente en los modos combinados de nueve y diez *bits* los *buffers* intermedios se inhabilitan, también es posible habilitarlos para así crear rangos de conversión lineales a tramos. Por ejemplo, en un ADC de nueve *bits*, podría programarse la referencia baja a masa, la alta a alimentación, y la intermedia a un valor bajo (menor que el punto medio) para así obtener escalones cuánticos más pequeños cuando la señal es en valor absoluto más pequeña, de tal forma que se homogeneizaría el error relativo.

Finalmente, los SARs pueden ser controlados automáticamente para trabajar en modo *pipeline*. En este modo, la misma señal se ruta a dos o cuatro conversores de ocho *bits* que comienzan a convertir de forma escalonada. El registro final va multiplexando las salidas de los distintos ADCs para así producir datos dos o cuatro veces más rápido que un sólo conversor.

Mediante toda esta programabilidad el usuario puede elegir el mejor compromiso de precisión (de ocho a diez *bits* más subrangos), velocidad de conversión (desde un cuarto de la velocidad nominal para los modos multiplexados hasta cuatro veces más para los modos de *pipeline*), y número de canales (de uno a cuatro).

2.4.5 Interconexión analógica

Como se ha mencionado, la estructura elegida no es la de una matriz analógica regular de baja granularidad, sino la de un *front-end* analógico adecuado para acondicionamiento y conversión de señal para adquisición de datos. La interconexión posible entre estos bloques es por tanto limitada y se realiza en cualquier caso a base de multiplexores analógicos (hechos con conmutadores CMOS) programados por *bits* de configuración provenientes de memorias digitales. En general la interconectividad necesaria para un bloque programable analógico particular depende del rango de aplicaciones a que se dedique la arquitectura de tipo FIPSOC global.

Los recursos de interconexión analógica incluidos en el CAB permiten utilizar las mismas salidas del bloque de amplificación y de los DACs como referencias en cualquier punto del circuito, en particular como referencia de modo común para los amplificadores diferenciales, como tensión

umbral para los comparadores, y como referencias de tensión para los DACs y ADCs. Gracias a ello es posible programar dinámicamente tanto digital (mediante DACs) como analógicamente (mediante salidas de los canales de amplificación) las distintas tensiones de referencia del CAB.

2.5 Configuración de arquitecturas programables

El diseño de mecanismos de configuración de arquitecturas programables basadas en microprocesador presenta nuevas problemáticas frente al caso de las arquitecturas programables clásicas, que como se mencionó anteriormente normalmente se realiza a base de *bits* aislados configurados mediante un *bit-stream* o cadena de configuración serie. Al disponer de un microprocesador resulta interesante poder realizar la configuración mediante accesos normales a memoria, para de esta manera poder configurar selectivamente partes del *array* y además poder hacerlo de forma dinámica. Cuanto mayor sea el número de *bits* transferidos en cada escritura del microprocesador, menor será el tiempo de configuración empleado, lo que permitirá reutilizar una misma parte del *array* para funciones distintas más eficientemente. Por ello resultaría adecuado utilizar *bits* de palabra ancha de datos, o permitir la transferencia simultánea a varios bloques programables o a varias posiciones de memoria.

Además de aumentar el ancho de palabra del microprocesador o la frecuencia de transmisión de datos de configuración, es posible además utilizar otras técnicas de reconfiguración parcial y dinámica que permitan explotar al máximo la interacción entre los recursos programables y el microprocesador, ya que éste resulta prácticamente imprescindible de cara a planificar y gestionar el *hardware* virtual soportable mediante estructuras dinámicamente reconfigurables.

Mediante una cuidada planificación de gestión de contextos activos se consigue reducir área real de la aplicación dado que sólo las partes activas estarían mapeadas en el *array* en cada momento. Gracias a la técnica propuesta de contextos *buffer* que se introducirá a continuación es posible cargar nuevas configuraciones mientras el *chip* sigue funcionando normalmente, y producir cambios de contexto en decenas de nanosegundos, lo cual no afectaría la temporización de la aplicación en un gran número de casos prácticos.

2.5.1 Reconfiguración multicontexto

Como se explica en el apéndice A, se entenderá por *contexto* la configuración de un DMC o de un grupo de ellos. La reconfiguración multicontexto consiste en disponer de varios contextos desde los que obtener la configuración real de los recursos programables, de tal forma que si se precargan los datos de configuración en los contextos múltiples se puede producir un cambio de configuración mediante un simple cambio en la selección de contexto.

Nuestro esquema de configuración se basa en producir transferencias entre estos contextos y las células reales de configuración en vez de una mera selección. Esto permite reutilizar posteriormente la memoria del contexto *buffer* previo para uso general o cargar un nuevo contexto mientras el circuito funciona normalmente con la configuración real salvada.

2.5.1.1 Contextos buffer

La ventaja de mapear la memoria de configuración en el espacio lógico de direccionamiento del microprocesador radica en el preciso control que éste adquiere sobre el proceso de configuración del *hardware*, lo que permite la reconfiguración dinámica y parcial a la vez que facilita enormemente el sistema de desarrollo que se implementaría mediante un ordenador personal comunicándose con el microprocesador. Sin embargo, podemos apuntar al menos dos inconvenientes a esta idea: En primer lugar, las células de memoria son débiles ya que deben ser pequeñas por cuestión de optimización de área y para poder escribirlas a través de líneas largas, debido a lo cual resulta peligroso usarlas directamente para comunicar la configuración de la estructura. En segundo lugar, una gran parte del mapa de memoria del microprocesador estaría dedicado a la configuración, lo que hace ineficiente el uso del microprocesador para programas de propósito general, ya que un usuario que no necesite cambiar la configuración sobre la marcha tendría una gran parte del mapa de memoria ocupada innecesariamente.

Sobre todo debido a la primera causa, resulta así aconsejable separar la célula de configuración "real" de la memoria mapeada en el espacio lógico del microprocesador, como se muestra en la estructura propuesta en la figura 2.60. El *bit* de configuración "real" está aislado por el conmutador controlado por la orden de carga de contexto. La célula de memoria de la derecha

constituye el contexto *buffer* y está mapeada en el espacio lógico de direccionamiento del microprocesador.

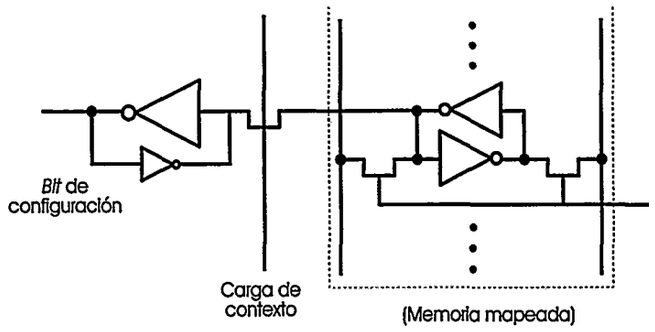

Fig. 2.60 Contexto *buffer*

La orden de carga de contexto puede realizarse tras decodificar una escritura del microprocesador en memoria, o bien mediante el mismo *hardware* programable si se quiere un circuito que se modifique a sí mismo. Mediante el uso de esta estructura se obtienen varios beneficios:

- La memoria mapeada (los contextos *buffer*) puede ser usada como memoria RAM de propósito general para correr programas de usuario y guardar datos. Tras la configuración, provocada por la orden de transferencia de contexto, la memoria queda libre para cargar una nueva configuración (sin tener que interrumpir al circuito mientras se carga) o para cualquier otro uso.

- Las células de memoria mapeadas en el espacio del microprocesador, débiles y pequeñas, están aisladas de las células reales de configuración, lo cual las hace más seguras ya que su salida no tiene que escribir directamente las señales de configuración. Si consideramos que la mayoría de los *bits* de configuración se dedican a los recursos de rutado, y que estos están implementados a base de multiplexores anchos, las señales de configuración y sus negadas, usadas como líneas de selección en los multiplexores, se cruzan con las señales 'reales' del circuito que se conectan por esas pistas (incluidas las señales de reloj), lo que obliga a que los *bits* de configuración estén almacenados en células de memoria más "fuertes" que las pequeñas células de una memoria normal.

- Dado que estos multiplexores anchos se han implementado usando conmutadores NMOS, resulta interesante poder separar la alimentación de la célula de memoria de configuración "real" de la mapeada para así poder utilizar tensiones mayores ($V_{DD} + V_{TN}$) que aseguren una buena conductividad de los transistores NMOS (ya que transmitirían bien los niveles altos).

Los retardos de interconexión obtenidos de esta forma estarían minimizados y serían incluso menores que en estructuras CMOS ya que la capacidad parásita de los conmutadores NMOS sería del orden de un tercio de la de los CMOS (puesto que anchura del PMOS debe ser más del doble que la del NMOS para compensar la diferencia de movilidad entre huecos y electrones).

2.5.1.2 Contextos buffer *múltiples*

Aunque la reconfiguración dinámica es posible con la estructura de la figura 2.60, y no es necesario detener el normal funcionamiento del circuito a reconfigurar ya que se puede cargar el nuevo contexto mientras el circuito se encuentra funcionando con la última configuración transferida, el tiempo de vida del contexto debe ser igual o superior al de carga del nuevo contexto, ya que si no el circuito tendría que esperar a que el nuevo contexto estuviera cargado para poder ser transferido.

Para evitar este inconveniente es posible disponer de varios contextos *buffer* como se muestra en la figura 2.61. Existen entonces tantas órdenes de carga de contexto como contextos *buffer* previos. Mediante esta estructura es posible disponer de dos o más configuraciones precargadas que pueden cambiarse muy rápidamente sin tener que recargar el contexto *buffer* cada vez.

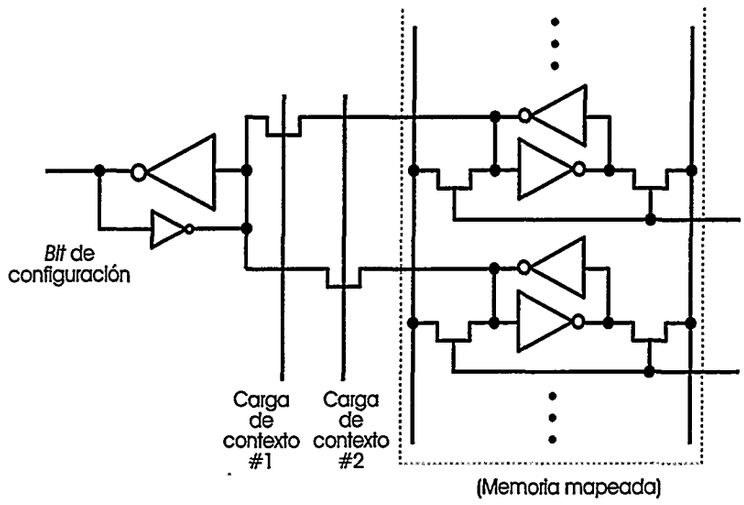

Fig. 2.61 Contextos *buffer* múltiples

2.5.1.3 Pilas de contextos

Cuando el número de contextos se eleva la estructura de la figura 2.61 se complica, ya que los decodificadores para la memoria crecen y además es necesario disponer de más transistores de transferencia de contexto que deben estar todos conectados a un mismo punto (la entrada de la célula "real" de configuración), lo cual complica la implementación física.

En su lugar es posible utilizar pilas de contextos como la mostrada en la figura 2.62. En la pila existe un mayor número de células *buffer* de las que sólo una es accesible desde el microprocesador. En lugar de las órdenes de transferencia de contexto existen las órdenes de *push* o introducción en la pila (activando las señales **SHL** de la figura 2.62) y *pop* o recuperación de la pila (activando las señales **SHR** de la figura 2.62). Una aplicación dada se descompondría jerárquicamente en subsistemas no solapantes en el tiempo que se tratarían como las subrutinas *hardware* propuestas en [HAS90] o incluso como objetos *hardware* [CAS95] instanciados y comunicados mediante un lenguaje de programación de alto nivel. Estas subrutinas u objetos pueden ser jeráquicos y a su vez instanciar a otras subrutinas y objetos en lugares cada vez más bajos de la jerarquía, siendo el número de niveles máximo igual al número de contextos *buffer* de la pila.

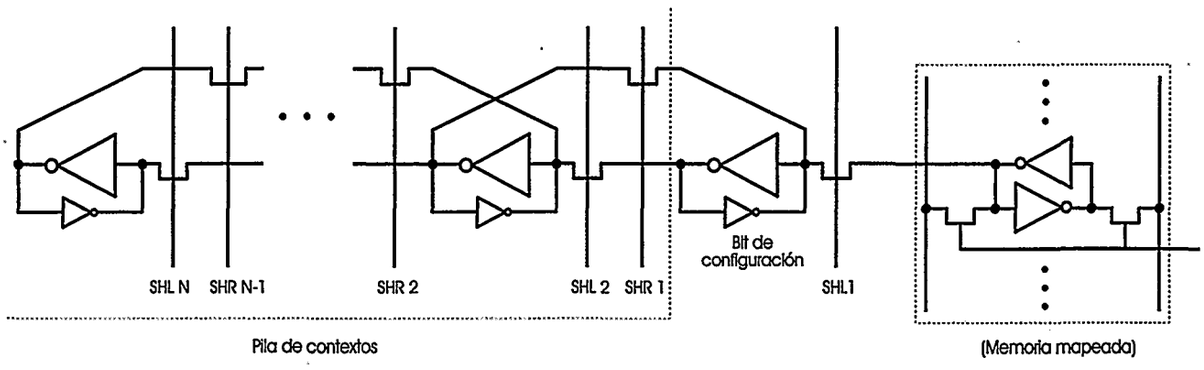

Fig. 2.62 Pila de contextos

Cada vez que se instancia una subrutina sobre un subconjunto de DMCs de la FPGA, la configuración almacenada en la memoria mapeada se transfiere las células de configuración real a la vez que la pila entera rota hacia la izquierda. Cada vez que una subrutina termina, la pila rota a la derecha y la tarea que fue interrumpida vuelve a estar activa. Si además se dispone de una estructura similar para los *flip-flops* de usuario en vez de la simple duplicación propuesta en el apartado 2.3.3.2.3 (*Flip-flops* multicontexto), las tareas pueden ser interrumpidas y reanudadas a

voluntad, sin que se pierdan los datos almacenados en cada nivel jerárquico aunque todos los niveles funcionen sobre el mismo substrato programable. Los *flip-flops* utilizados en modo multicontexto sirven así para implementar variables locales a los procedimientos, mientras que utilizados en modo normal actúan como variables globales y sirven para comunicar datos entre los distintos contextos. Además es posible imponer los valores iniciales de estas variables para que el procedimiento se inicialice correctamente al conmutar al nuevo contexto.

Finalmente se puede conectar el último contexto de la pila a la célula de configuración, de tal forma que una vez precargada la pila de contextos desde la memoria mapeada sería posible rotar hacia los dos lados la estructura circular de configuraciones. Mediante una planificación cuidada se sintetizaría una máquina de estados en la que sólo se permitirían las transiciones a los estados adyacentes a los que se podría pasar mediante las órdenes de rotación de configuración a un lado o a otro, o bien se dispondría de varios contextos independientes cuya selección implicaría no un sólo ciclo de escritura sino varios.

La ventaja de esta aproximación respecto a la de los contextos múltiples radica en que los decodificadores de la memoria mapeada no crecen, si bien no es posible reaprovechar la memoria de configuración para uso general. Además, el número de órdenes de cambio de contexto, y con él el número de transistores de transferencia, no crecen al crecer el número de contextos, lo cual no imposibilita la implementación física como se explicará en el capítulo 4. El éxito de esta estructura depende críticamente de la disponibilidad de herramientas de síntesis arquitectural y de planificación de contextos, pues la planificación manual resulta habitualmente impracticable. La herramienta debería realizar una síntesis desde una especificación en un lenguaje de descripción *hardware* organizado jerárquicamente en procedimientos y procesos teniendo en cuenta la compatibilidad temporal y la criticalidad de los procesos en cada momento.

2.5.2 Mecanismos de reconfiguración

En esta sección se proponen varios mecanismos de reconfiguración que hacen uso de las estructuras multicontexto introducidas en la sección anterior. En el capítulo 4 se describirá la implementación física de las arquitecturas propuestas en este capítulo, y finalmente se estudiará, a la vista de los resultados, la rentabilidad de la estructura multicontexto en términos de área dedicada a cada elemento de la arquitectura programable.

2.5.2.1 Reconfiguración parcial y configuración múltiple

Para la configuración y reconfiguración parcial de la arquitectura es necesario disponer de dos registros que definan una máscara de filas y columnas que designen los DMCs a los que se accede en cada momento como se esquematiza en la figura 2.63 . Este esquema es similar al propuesto por Chrucher y Kean en [CHU95].

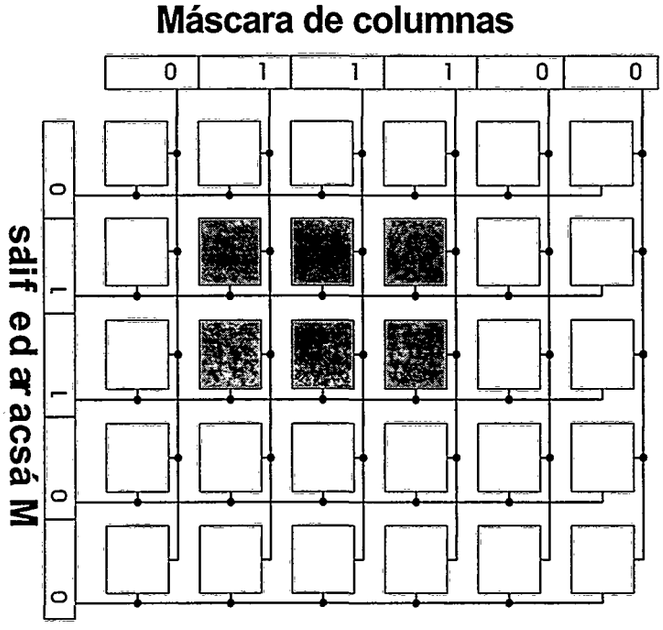

Fig. 2.63 Reconfiguración parcial

Las máscaras seleccionan un *rectángulo lógico* de DMCs, entendiendo por tal un conjunto de elementos {[X,Y]} tales que si los elementos $[X_1,Y_1]$ y $[X_2,Y_2]$ pertenecen a él, entonces los elementos $[X_1,Y_2]$ y $[X_2,Y_1]$ también pertenecen. El microprocesador es capaz de provocar el proceso de reconfiguración sobre este subconjunto de DMCs en vez de sobre todo el *array*, soportando así la reconfiguración parcial.

Otra posibilidad consiste en que el microprocesador permita un tipo de acceso especial (que será descrito en mayor detalle en el capítulo 3) a través de un *buffer* del tamaño del espacio de configuración de un DMC, de tal forma que cualquier escritura sobre este *buffer* escribe sobre la misma posición relativa de todos los DMCs seleccionados por la máscara. De esta forma podría

programarse un rectángulo lógico de DMCs con la misma configuración, lo que resultaría útil en el caso de circuitos de procesamiento paralelo tipo *array* sistólico [BOE96], para inicializar el *array* a una cierta configuración en la que por ejemplo todas las salidas estén inhabilitadas (típica configuración tras *reset*), o para el *test* del circuito como se explicará en el capítulo 4.

2.5.2.2 División de LUTs en dos contextos

El interés de las estructuras multicontexto radica en que el área adicional necesaria para su soporte es significativamente menor que el área del recurso programable al que configuran. Esta premisa resulta especialmente cierta en el caso de los recursos de rutado como se demostrará al estudiar la implementación física en el capítulo 4. Sin embargo, el caso de las LUTs es conceptualmente distinto al del resto de los recursos programables del DMC, ya que una LUT está constituida fundamentalmente por memoria de configuración, y una duplicación de esta memoria supone una duplicación directa del área que consume. Dado que el área consumida por las LUTs es significativa (del orden del 20% en el caso de la implementación práctica del DMC que será descrita en el capítulo 4), el hecho de duplicarla afecta significativamente al consumo de área del DMC; en el apartado 2.3.3.1.1 (figura 2.10) se explicó cómo las LUTs de los DMCs pueden ser divididas en dos mitades para soportar reconfiguración multicontexto. De esta forma la granularidad de las LUTs en los modos dinámicos es de un *bit* menos que en sus correspondientes estáticos, pero la flexibilidad es mayor ya que se dispone de un mayor número de entradas independientes ya que no se comparten en los modos estáticos.

La operación multicontexto es posible igualmente ya que el microprocesador puede leer y escribir de forma independiente en cada contexto de la LUT. Es posible por tanto reconfigurar un contexto mientras el otro está siendo usado como ocurre con el resto de la memoria de configuración, o precargar las dos mitades con dos configuraciones y luego conmutar entre uno y otro, para lo cual es necesario provocar un cambio de contexto en el DMC completo ya que la señal que multiplexa entre las dos mitades está gobernada directamente por un *bit* de configuración. Utilizando LUTs en modo memoria RAM de usuario es posible comunicar datos entre contextos si éstas se utilizan en modo estático, o bien salvar los datos internos a cada proceso si se usan en modo dinámico, en forma análoga a como pueden utilizarse los *flip-flops*.

2.5.2.3 Flip-flops *multicontexto precargables*

Como se describió en la sección 2.3.3.2.3, los *flip-flops* pueden estar internamente duplicados para apoyar la reconfiguración dinámica multicontexto (fig. 2.25). Si además el microprocesador puede escribir asíncronamente en cualquiera de estos dos contextos en cualquier momento, resultaría posible precargar datos en el nuevo contexto de los *flip-flops* que se van a usar para una determinada subrutina *hardware* [HAS90] [CAS95] para que en el momento de su activación las "variables" utilizadas en ella dispongan de los valores iniciales que se especificaron.

Los *flip-flops* utilizados en modo multicontexto servirían así de variables locales al procedimiento, de tal forma que si éste es interrumpido por otra tarea de mayor prioridad, o si instancia otra subrutina jerárquicamente inferior, los datos se salvarían y se recuperarían al volver al contexto salvado. Los *flip-flops* pueden usarse en modo estático normal en los puntos de comunicación entre contextos, haciendo las veces de variables globales a la aplicación (o en determinados niveles de la jerarquía de procesos) o de parámetros del proceso en sí.

2.5.2.4 Reutilización de memoria

Con el objeto de que la implementación práctica de las estructuras de reconfiguración multicontexto propuestas resulte económica, resulta crucial que los contextos *buffer* estén mapeados en el espacio lógico de direccionamiento del microprocesador, y que además sea posible utilizarlos como memoria de propósito general para correr programas o guardar datos de usuario en ellos. De esta manera, si el usuario no necesita hacer uso de la reconfiguración multicontexto en parte o en todo el *array*, la memoria siempre puede ser reutilizada por el microprocesador. Así, puede considerarse que la memoria que de todas formas el microprocesador necesita para las operaciones de usuario se ha distribuido a lo largo y ancho del *array* para poder ser reutilizada como memoria *buffer* de configuración, en vez de ser implementada junto al microprocesador como un simple bloque monolítico con una sola función.

Si se utiliza el esquema propuesto con dos contextos *buffer*, resulta especialmente adecuado organizar lógicamente la memoria en dos bloques que mapeen de forma continua (sin huecos) todas las posiciones de memoria de todos los DMCs, de tal forma que cada mitad de este bloque pertenezca a un contexto. De esta forma se puede utilizar selectivamente toda la memoria, una

mitad o las dos mitades como memoria de configuración en función de las necesidades reales de reconfiguración dinámica: si no es necesaria en absoluto, la configuración se transfiere una vez provocando una orden de transferencia desde cualquiera de los dos contextos, dejando después toda la memoria libre para uso general; si es necesario hacer cambios aislados de configuración, las nuevas programaciones se van cargando en un contexto *buffer* que se iría transfiriendo en momentos seleccionados, mientras que la otra mitad de la memoria (la correspondiente al otro contexto) se utilizaría normalmente para uso general; y si es necesario intercambiar dos configuraciones de forma rápida, o se necesita un mayor número de configuraciones para lo cual es necesario disponer de los dos contextos *buffer*, toda la memoria se utilizaría para configuración y la aplicación del usuario deberá disponer de otras memorias externas o incluidas en el circuito integrado (se dispone de otro bloque de memoria auxiliar para apoyar al microprocesador en estas situaciones).

2.5.2.5 El proceso de configuración

El proceso de configuración de los DMCs de una arquitectura de tipo FIPSOC que utilice células de configuración de dos contextos, esquematizado en la figura 2.64, se planificaría en función de las necesidades de reconfiguración dinámica de la aplicación en particular tal y como se explicó en el apartado anterior.

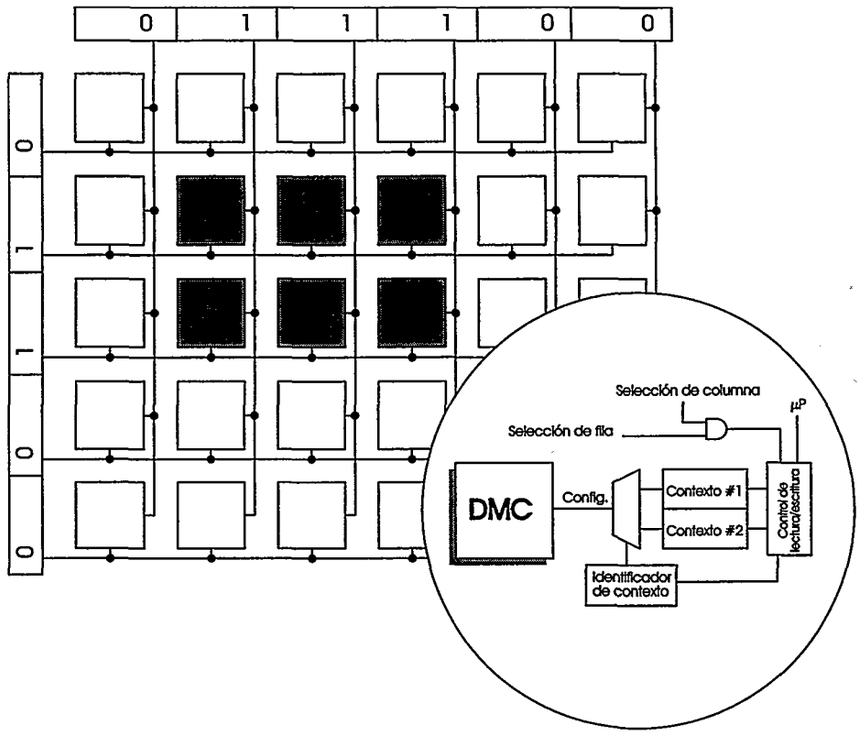

Fig. 2.64 El proceso de reconfiguración

En cualquier caso es necesario precargar la configuración en un contexto *buffer*, imponer una máscara de filas y columnas para especificar el conjunto de DMCs a configurar, y finalmente provocar la transferencia de configuración desde el contexto seleccionado. El microprocesador es, en general, el único que impone la máscara y que precarga los contextos *buffer* con las nuevas configuraciones. El proceso de transferencia en sí podría en cambio ser disparado no sólo por el microprocesador sino además por el propio *hardware* programable. Mediante el uso de este modo el "motor" de la reconfiguración podría independizarse del microprocesador (excepto por el hecho de que si son necesarias más de dos configuraciones por DMC el microprocesador debe usarse para cargar los contextos *buffer*) e implementarse mediante los propios DMCs que podrían además modificar a otros DMCs e incluso a sí mismos, permitiendo la creación de circuitos lógicos no basados en microprocesador capaces de modificarse a sí mismos.

2.5.3 Configuración de células analógicas

Las características programables de las células analógicas, tales como los factores de ganancia de los amplificadores, su *offset* o su CMRR, el rutado de las referencias o los modos de

funcionamiento de los conversores, se programan en general mediante *bits* de memoria estática convencional, de forma análoga a la propuesta por Klein [KLE96] o Bratt y Macbeth [BRA96]. En una arquitectura de tipo FIPSOC, esta memoria estará mapeada en un cierto lugar del espacio lógico de direccionamiento del microprocesador aunque en general no dispondrá de la estructura de contextos *buffer* descrita hasta ahora, ya que las células analógicas funcionan a velocidades considerablemente menores (en torno al MHz) y además su tiempo de establecimiento (decenas de microsegundos) es significativamente grande comparado con los tiempos de reconfiguración manejados hasta el momento (nanosegundos).

Los *bits* de configuración no deberán provenir directamente de la memoria de configuración analógica sino que se deberá interponer unos *buffers* intermedios para conseguir un mejor aislamiento en ambos sentidos: por una parte se filtraría el ruido desde la parte digital hacia la analógica dado que estos *buffers* intermedios se conectarían a la alimentación analógica en vez de a la digital como el resto de la memoria; por otra, se aislaría a las células de memoria, pequeñas y débiles, de las señales reales de usuario para asegurar su estabilidad.

Este aislamiento es crítico a la vista de la estructura interna del CAB descrita anteriormente. Si las señales digitales de configuración actuasen directamente sobre los terminales de control de los multiplexores de las referencias, inyectarían ruido en estas últimas debido a las capacidades parásitas entre puerta y drenador o fuente. Este efecto resultaría especialmente crítico en el caso de la conmutación de las distintas resistencias que configuran la ganancia de los amplificadores programables, así como en las fuentes de corriente correctoras del *offset*, ya que el ruido se inyectaría directamente en las entradas de los amplificadores operacionales diferenciales.

2.6 Resumen

En este capítulo se ha discutido en detalle la metodología a seguir para el diseño de los distintos bloques programables que integran un arquitectura de tipo FIPSOC, a la vez que se ha ilustrado mediante ejemplos prácticos basados en las estructuras concretas que conforman la primera realización práctica de arquitectura de este tipo cuya implementación práctica se detallará en el capítulo 4. Será posible en cualquier caso generalizar los casos más concretos referidos hasta el momento a la hora de acometer otro caso distinto de arquitectura de tipo FIPSOC, en especial si el interés principal no se centra en aplicaciones de control industrial como ocurre en nuestro caso.

De entre todos los criterios utilizados al tomar las decisiones arquitecturales, es importante no olvidar que el que verdaderamente importa cuando se trata de FPGAs para ser utilizadas junto a un microprocesador es su capacidad de interacción con él, lo que permitrá soportar técnicas novedosas de co-diseño *hardware-software* así como de reconfiguración dinámica multicontexto avanzada, mediante la cual pueden obtenerse las ventajas propias del *hardware* virtual.

En [LYS93] Lysaght se refería a una serie de características deseables para poder trabajar con técnicas de reconfiguración dinámica en las FPGAs que la soportaran. Entre estas peticiones destacan la posibilidad de acceso a la configuración y su control desde el mismo dispositivo, el acceso individualizado a las células del *array* y la posibilidad de reconfigurarlas individualmente. Es interesante comprobar que las soluciones propuestas cumplen todas estas características mencionadas en el artículo.

CAPÍTULO 3

INTERFACES ENTRE MICROPROCESADORES Y CÉLULAS PROGRAMABLES EN ARQUITECTURAS DE TIPO FIPSOC

3.1 Introducción

Para conseguir explotar todas las posibilidades que brinda una arquitectura de tipo FIPSOC, es de vital importancia proveer de la máxima flexibilidad a los distintos interfaces entre cada uno de los bloques que integran el sistema. Sin esta flexibilidad añadida el *chip* no sería conceptualmente distinto a un ASIC de señal mixta, en el que es necesario construir los mecanismos necesarios de comunicación entre los bloques para cada aplicación concreta.

Más aún, la metodología para el diseño de células programables que se ha introducido en el capítulo anterior se basa principalmente en esta capacidad de interacción con el objetivo de diseñar FPGAs controladas por microprocesador de forma inherente: su configuración se realiza mediante escrituras en memoria desde el microprocesador, el cual puede además acceder a los valores de las señales combinacionales y secuenciales de cada DMC en tiempo real, escribir en los *flip-flops* en cualquiera de sus dos contextos, compartir datos con la lógica programable a través de las LUTs (y de los *flip-flops*), configurar y operar directamente el CAB, detener y

relanzar a voluntad los relojes utilizados en el resto del *chip*, o utilizar señales reales de la FPGA como fuentes de interrupción o para detener los propios relojes. Estas características hacen que las estructuras de FPGA introducidas en el capítulo anterior puedan considerarse como las primeras FPGAs basadas en microprocesador.

Esta relación debe ser relativamente biunívoca, es decir, el microcontrolador a incluir en sistemas de tipo FIPSOC debe estar especialmente diseñado para trabajar junto a la FPGA: sobre la estructura inicial del microcontrolador a utilizar como base se puede construir un sistema de periféricos que permita el acceso a la configuración y a las señales de la FPGA, la generación de frecuencias de reloj que incluya la posibilidad de detener los relojes de forma controlada, un controlador de interrupciones para gestionar de forma automática las distintas interrupciones propias de la FPGA, un sistema de comunicaciones serie utilizado entre otros momentos durante la descarga de la configuración desde memorias serie, o una circuitería de depuración dentro del *chip* que incluya detección de *breakpoints* o captura de accesos a memoria de datos.

Fundamentalmente son dos las ventajas de este fuerte acoplamiento entre la lógica programable y el microprocesador: por una parte, constituye un poderoso sistema de desarrollo ya que el microprocesador es capaz de monitorizar en todo momento el estado de la lógica programable y es capaz de ejecutar en tiempo pseudo-real ciclos aislados de reloj o programar una parada tras un cierto número de ciclos, gracias a lo cual se puede llevar a cabo un emulación en tiempo pseudo-real que puede sustituir fidedignamente (exceptuando ciertas diferencias en la temporización detallada) a la simulación; por otra parte, la interacción entre el *hardware* y el *software* es muy fuerte al estar mapeadas todas las señales *hardware* en el espacio de direccionamiento del microprocesador, por lo que la metodología a seguir en las aplicaciones de FIPSOC debe ser radicalmente distinta a la seguida en sistemas convencionales de compuestos por una FPGA y un microprocesador ya que las rutinas más críticas pueden ser dinámicamente ejecutadas sobre el *hardware* programable consiguiendo así mejores prestaciones que sus correspondientes versiones *software*.

A continuación se tratará la interacción entre cada dos de los tres subsistemas que componen un sistema de tipo FIPSOC: la FPGA, la parte analógica (el CAB) y el microprocesador, explorando las posibilidades que ofrecen este tipo de *chips* de cara a una fuerte interacción *hardware-software* que permita una co-emulación consistente y una metodología de co-diseño novedosa.

De igual manera que en el capítulo anterior se acometió el estudio del diseño de células programables mediante ejemplos de arquitecturas más o menos concretos, en este capítulo también se utilizarán arquitecturas concretas desde las que será posible generalizar los aspectos

más importantes del diseño de cada parte del interfaz. Nuestro caso particular utiliza un microprocesador de 8 *bits*, el popular i8051, debido fundamentalmente a que como se mencionó en el capítulo anterior nuestro interés se centra primordialmente en sistemas programables para ser utilizados en aplicaciones de control industrial. En cualquier caso los aspectos arquitecturales del interfaz que se explicarán en el capítulo son directamente generalizables para cualquier otro microprocesador.

3.2 Interfaz entre el microprocesador y la lógica programable

El interfaz entre el microprocesador y la lógica programable es el más complicado de los tres y el que en principio más posibilidades ofrece. Básicamente, la interacción entre ambos bloques se puede producir a través de accesos a memoria o mediante entradas de interrupción. A través del espacio de direccionamiento lógico del microprocesador sería posible acceder a las memorias *buffer* de configuración, a los datos de las LUTs (que pueden ser datos de usuario compartidos si se usan en modo memoria), y a los valores reales en cada momento de las señales combinacionales y secuenciales de cada DMC, además de poder escribir de forma asíncrona e incondicional en los *flip-flops* en cualquiera de sus dos contextos (si se implementan) y poder provocar transferencias de configuración desde los contextos *buffer* a la memoria de configuración "real". Por otra parte, el microprocesador podría disponer de entradas de interrupción directamente procedentes de la lógica programable, además de otras entradas mediante las que puede provocarse una parada del sistema de generación de relojes para la propia lógica programable (lo cual puede a su vez provocar una interrupción). Estas últimas entradas permitirían la implementación de lo que se denominará *breakpoints hardware* en contrapartida a los clásicos *breakpoints software* que también pueden soportarse. Además de ser una poderosa herramienta para la implementación de aplicaciones mixtas, estas características simplificarían enormemente la construcción de un sistema completo de desarrollo.

En esta sección se explica la metodología general a seguir para implementar todos estos mecanismos de comunicación y control, y se analiza en profundidad un ejemplo particular de diseño de interfaz entre el microprocesador y la lógica programable a partir del cual puede generalizarse los aspectos más importantes de este tipo de diseños.

3.2.1 El DMC a los ojos del interfaz con el microprocesador

En general el método más rápido y efectivo de comunicación entre células programables y microprocesadores es a través de accesos a memoria. Los DMCs serían vistos por el microprocesador como pequeños bloques de memoria, y las escrituras y lecturas de datos producirían comandos y transferencias de datos como en cualquier otro periférico de microprocesador clásico.

La figura 3.1 muestra un ejemplo de modelo de DMC visto por el interfaz con el microprocesador. El diseño particular del DMC utilizado está formado por los distintos bloques combinacionales y secuenciales introducidos en el capítulo anterior. En este caso, en cada DMC existen cuatro bancos de memoria distintos y seleccionables de forma independiente por el microprocesador a través del interfaz.

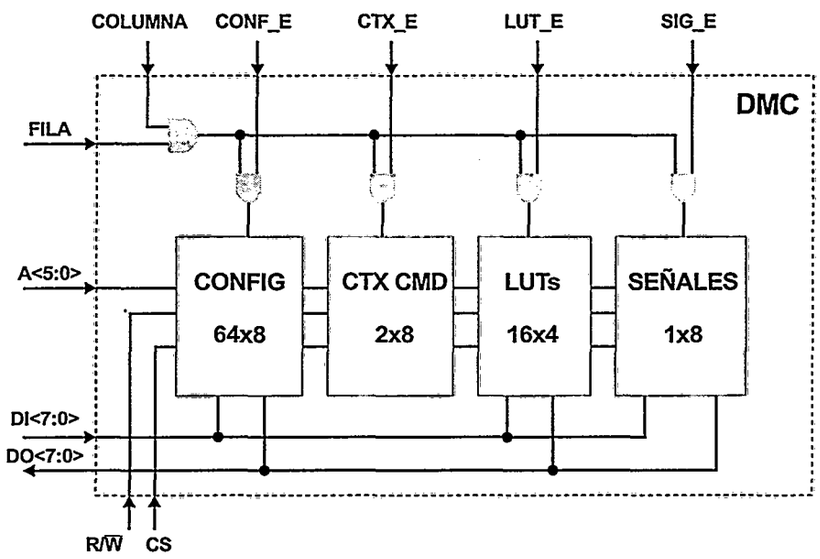

Fig 3.1: El DMC a los ojos del interfaz con el microprocesador

Estos cuatro bancos corresponden a las memorias *buffer* de configuración, los comandos de transferencia de contexto desde las memorias *buffer* a la memoria "real" de configuración, la información de las LUTs, y las señales de las partes combinacional y secuencial del DMC. La selección de cualquiera de estos bancos debe estar validada con las señales de marcado de fila y columna, que pueden estar impuestas directamente desde el microprocesador a través del registro de máscara de selección, o desde el interfaz para permitir un acceso secuencial a la memoria como se explicará más adelante.

Cada contexto *buffer* de un DMC necesita 32 *bytes* (256 *bits*), por lo que al existir dos de ellos se dispone de 64 *bytes* de memoria dedicados a este fin.

Las órdenes de transferencia de configuración se provocan desde el microprocesador al realizar una escritura en una de dos posibles posiciones de memoria según cuál sea el contexto que se quiere transferir. Las operaciones de lectura en estas posiciones de memoria no son significativas.

Las LUTs representan un espacio de memoria de 16x4 a través del cual se puede configurar las funciones lógicas a implementar por ellas o bien compartir datos con la aplicación *hardware* de usuario (especialmente si las LUTs permiten un modo de funcionamiento tipo memoria RAM de usuario, para lo cual las LUTs podrían implementarse como pequeñas memorias de doble puerto, como se explicó en el capítulo anterior).

Las señales de salida combinacional (generadas a partir de la información programada en las LUTs) y secuencial (obtenida a partir de los datos contenidos en los FFs) pueden leerse en tiempo real a través del cuarto bloque de memoria del DMC. Las escrituras en las posiciones de memoria correspondientes a las señales secuenciales producen cargas directas de datos en los FFs.

El papel del interfaz con el microprocesador es el de generar las señales de filas y columnas y las de habilitación de cada uno de los cuatro bloques en función de las posiciones de memoria absolutas accedidas por el microprocesador en cada momento y de los registros de configuración del sistema. En principio la arquitectura que se detallará a continuación es una extensión de la filosofía mostrada por Churcher y Kean [CHU95], aunque más allá del simple mapeo de memoria de configuración.

3.2.2 Interacción entre mapas de memoria del microprocesador y células programables

El diseño del interfaz entre el microprocesador y las células programables del sistema tipo FIPSOC deberá acometerse teniendo en cuenta la arquitectura de mapas de memoria del microprocesador particular a utilizar, cuya elección en principio dependerá del tipo de aplicaciones que se deseen cubrir. Si, como en nuestro caso, el objetivo primordial de la arquitectura es servir para aplicaciones de control industrial, parece adecuado elegir un microprocesador de ocho *bits* como el i8051 de Intel, el 68HC11 de Motorola o uno de los PICs de Microchip. En particular el 8051 dispone de un mapa de memoria especialmente complicado

(a la vez que versátil), por lo que resulta un buen ejemplo del tipo de complejidad con que el diseñador de arquitecturas de tipo FIPSOC puede encontrarse.

En concreto el 8051 dispone realmente de cuatro mapas de memoria a los ojos del usuario: la memoria de código (64 Kb), la memoria externa de datos (64 Kb), la memoria interna de datos (256 *bytes*) y los registros de función especial o SFRs (*Special Function Registers*, 128 *bytes*). El acceso a los primeros 128 *bytes* de la memoria interna de datos y a los SFRs se realiza mediante direccionamiento directo, y resulta especialmente rápido y cómodo para el usuario. El acceso a los 128 *bytes* más altos (y también a los primeros 128 *bytes*) de la memoria interna de datos, así como a la memoria externa de datos, se realiza mediante direccionamiento indirecto (ambos modos de direccionamiento se realizan a su vez mediante distintos registros de índice). A la hora de mapear los distintos registros de control del sistema es importante no comprometer todos estos recursos que el usuario normal puede necesitar para el desarrollo de sus aplicaciones *software*.

En general en gran parte de los microprocesadores disponibles en el mercado los mecanismos de control de periféricos se implementan a base de registros de función especial o SFRs iguales o análogos a los disponibles en el 8051. La programación de los relojes, la configuración y habilitación de *breakpoints*, la programación de interrupciones y sus vectores, la interacción con el bloque analógico o la transferencia de contextos se realizan mediante SFRs no utilizados en las versiones comerciales del 8051 (sólo unos cuantos SFRs son utilizados en las versiones estándar de 8051 disponibles en el mercado, estando el resto de las posiciones de memoria de ese banco inhabilitadas e incluso prohibidas por los ensambladores comerciales).

Normalmente el acceso a los registros especiales suele resultar particularmente rápido y conveniente, pues gran parte del control del sistema e incluso el mismo *software* de usuario suelen estar basados en ellos. En el caso del 8051, el acceso a los SFRs sólo puede ser directo, lo cual agiliza tremendamente su operación (es el más rápido de todos los modos permitidos en el 8051) aunque su flexibilidad es reducida ya que no se permite la indirección, lo que aconseja su uso para funciones especiales de gestión del sistema y no para propósito general.

Los bloques de memoria más grandes, como por ejemplo los contextos *buffer* de configuración de un DMC, o incluso las salidas combinacionales y secuenciales de todos los DMCs de una cierta zona, no pueden direccionarse a través de SFRs debido a su gran extensión (cada contexto de configuración ocupa 32 *bytes*). Para garantizar un rápido acceso a estos bloques de memoria se ha definido un "visor" o *buffer* de 64 *bytes* sobre la memoria interna baja (entre las posiciones $30 y $6F) al que puede accederse mediante direccionamiento directo e indirecto, lo cual permite un rápido a la vez que flexible sistema de acceso. Mediante su control con SFRs se puede mapear

distintos bloques de memoria de distintos DMCs en el visor, e incluso producir escrituras simultáneas en varios DMCs mediante la misma instrucción de escritura en memoria. La figura 3.2 muestra esquemáticamente la organización de la memoria interna en la realización particular del interfaz de ejemplo que se ha diseñado.

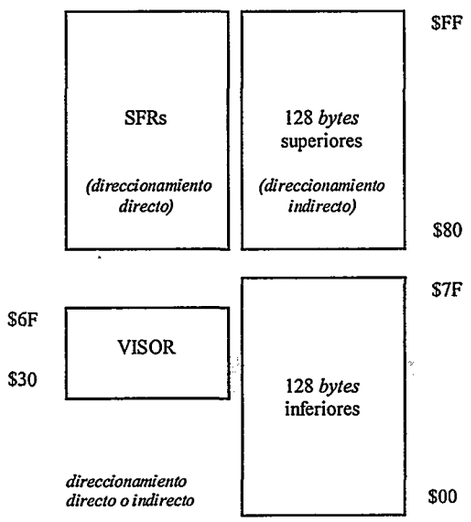

Fig 3.2: Organización de la memoria interna del 8051 de FIPSOC

Finalmente, los recursos de memoria más extensos, en este caso los contextos *buffer*, son susceptibles de ser utilizados como memoria masiva, bien de datos como memoria externa direccionable de forma indirecta, o bien de programa. Por ello resulta útil permitir hacer coincidir las memorias masivas de datos (externa) y de programa, de tal forma que los contextos *buffer* puedan ser utilizados como memoria "normal" por el sistema.

3.2.3 Mapeo de memoria de configuración

Tras los primeros diseños en los que se permitía mapear la memoria de configuración para ser accedida en paralelo mediante un microprocesador introducidos por Chucher y Kean [CHU95] y por DeHon [DeH96a] [TAU95], nuestra realización es la primera que permite reutilizar la misma memoria para aplicaciones de reconfiguración dinámica y para guardar datos o correr programas de usuario de propósito general (se pueden mapear tantos contextos *buffer* reutilizables como resulte económico). Sin esta posibilidad la utilidad de mapear la memoria de configuración es limitada ya que se gasta espacio lógico de direccionamiento de microprocesador en el caso en el que no se desee trabajar técnicas de reconfiguración dinámica avanzadas con o sin multicontexto.

Sin embargo, es posible que la organización lógica de esta memoria optimice ambas funciones. En nuestra realización particular sobre el DMC con el mapa lógico presentado al principio del capítulo, un primer acceso presenta dos bloques de memoria correspondientes cada uno a un contexto *buffer* de todos los DMCs del *array* de forma seguida y sin huecos. Esto permite poder utilizar la memoria *buffer* de cada contexto como un gran banco de memoria de uso general una vez realizada la configuración del *array*. La separación en dos bancos para cada contexto permite trabajar sólo con memoria de uso general, o bien con un contexto como memoria de uso general y otro para reconfiguración dinámica, o bien con los dos para configuración dinámica. En este modo de acceso las señales de selección de fila y columna son generadas mediante el interfaz en función de la dirección de memoria impuesta por el microprocesador y de la configuración del sistema.

La otra forma de acceder a la memoria de los dos contextos *buffer* de los DMCs es a través del visor. En este modo la memoria de configuración del DMC seleccionado mediante la máscara (ambos contextos seguidos) queda mapeada en el visor. Si se selecciona un conjunto de DMCs indicando varias filas y columnas en la máscara (un rectángulo lógico como se definió en el capítulo anterior), las escrituras sobre el visor se realizarán sobre la memoria de configuración de los DMCs o IICs seleccionados. Si se realiza una lectura, la circuitería de prioridades del interfaz seleccionará uno de ellos.

Mediante el acceso a través del visor es posible realizar una configuración múltiple, lo cual es especialmente interesante para aplicaciones de tipo *array* sistólico, para programar células de entrada-salida con la misma configuración o para pre-configurar todo el *array* o parte de él con una configuración inactiva inicial.

Finalmente, mediante un *bit* de configuración es posible utilizar la memoria externa como memoria de programa. En este modo es por tanto posible correr programas sobre los bancos de memoria de configuración de los DMCs.

3.2.4 Mapeo de señales

Mapear las señales digitales "reales" de las aplicaciones de usuario como posiciones de memoria legibles y escribibles desde el microprocesador facilita enormemente las posibilidades de interacción *hardware-software* del sistema. En nuestro ejemplo, el mapeo de las señales de salida del DMC se puede realizar tanto a través del visor (memoria interna) como a través de la

memoria externa de datos. En el caso de utilizar el visor, mediante SFRs se selecciona un cuadrante activo de 8x8 DMCs cuyas salidas combinacionales y secuenciales estarán visibles en el visor.

Dado que el DMC está fundamentalmente orientado a operación a nivel de *nibble* o unidad de datos de cuatro *bits*, se permite varios modos de acceso a los datos para poder aprovechar el hecho de que el microprocesador tiene un bus de datos de ocho *bits*. Así, en cada *byte* puede mapearse las salidas combinacionales y secuenciales (cuatro de cada) del mismo DMC de forma simultánea para así poder obtener el estado completo de un DMC con un solo acceso, o bien formar cada *byte* con las salidas combinacionales de DMCs adyacentes tanto vertical como horizontalmente, e igualmente con las secuenciales. Estos accesos a parejas de DMCs se esquematizan en la figura 3.3.

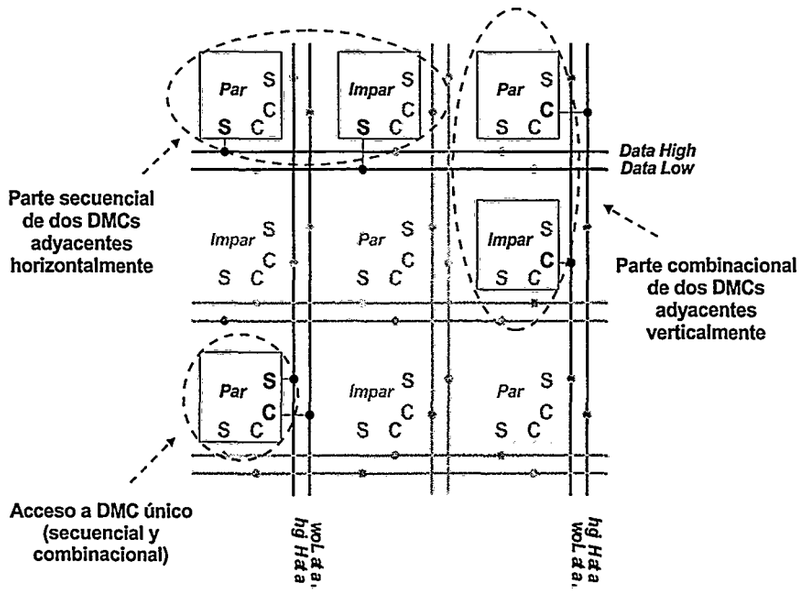

Fig 3.3: Acceso a señales de los DMCs.

Los DMCs quedan organizados así en pares e impares según tengan las salidas secuenciales o combinacionales en la parte alta y baja del bus de datos, o a la inversa. El interfaz es el que realiza el reordenamiento de los datos de forma dinámica en función de la configuración y del tipo de acceso en cada momento, ofreciendo al microprocesador un bloque compacto de memoria con todas las señales de salida de los DMCs ordenadas de distintas maneras.

Esta técnica es generalizable a DMCs de otras anchuras y a microprocesadores con mayores tamaños de palabra.

3.2.5 Mapeo de tablas de look-up.

Las LUTs constituyen otro interesante punto de interacción entre el *hardware* y el *software*, especialmente si pueden utilizarse como pequeños bloques de memoria utilizables por el usuario en sus aplicaciones *hardware* y a la vez estar mapeados en el espacio de direccionamiento lógico del microprocesador. En nuestra realización particular de interfaz, la configuración de las LUTs se realiza a través del visor de forma análoga a la configuración del resto de los recursos programables del DMC. Las dos LUTs de cada una de las dos mitades del DMC pueden accederse de forma separada (de cuatro en cuatro *bits*) para permitir independencia entre las dos mitades, o de forma conjunta (de *byte* en *byte*) para agilizar el proceso de configuración del DMC entero. De nuevo, las escrituras se realizan en todas las LUTs de todos los DMCs seleccionados por el rectángulo lógico marcado por la máscara, mientras que la lectura se resuelve mediante un sistema de prioridades.

3.2.6 Mapeo de registros de control

Los registros de control son quizás la parte del interfaz más dependiente del microprocesador particular elegido. En nuestro ejemplo, el interfaz del microprocesador intercepta los accesos directos a memoria en las direcciones $80 a $FF, zona conocida como banco de registros de función especial o SFRs en el 8051. De estas 128 posiciones de memoria, el 8051 estándar utiliza 21 para diversas tareas de gestión del sistema, tales como el control de los puertos de entrada/salida, del interfaz de comunicaciones serie, o para mapear el acumulador o el puntero de indirección de la memoria externa. Además de estos 21, el microprocesador implementado en FIPSOC incorpora otros 66 SFRs adicionales mediante los cuales se gestionan todos los periféricos del microprocesador y se controla la FPGA y el bloque analógico.

Es interesante notar que a los SFRs sólo puede accederse mediante direccionamiento directo, ya que los accesos indirectos a las posiciones $80 a $FF se realizan sobre la memoria interna y no sobre el banco de SFRs. Este hecho dificulta en cierta medida el diseño de sistemas de desarrollo en los que el usuario desea controlar el valor de los SFRs desde un interfaz remoto, ya que al especificar el SFR al que se quiere acceder es necesaria una indirección para llegar a él. Por ello, en el sistema de desarrollo diseñado se dispone además de una memoria RAM de código sobre la

que se codifica dinámicamente (el microprocesador escribe el código máquina correspondiente) el acceso directo a realizar, para así emular el acceso indirecto sobre la memoria directa.

3.2.7 Interrupciones

La utilización de interrupciones como mecanismo de comunicación entre la lógica programable y el microprocesador resulta especialmente interesante en un gran número de aplicaciones, en particular en el caso de utilizar la lógica programable para implementar co-procesadores o extensiones de la capacidad computacional del microprocesador. La implementación del sistema de control de interrupciones y su interfaz con el microprocesador dependerá fuertemente del microprocesador elegido, siendo normalmente necesario respetar criterios prefijados de construcción de periféricos, protocolos de comunicaciones, o incluso modificar o adaptar arquitecturas existentes de controladores de interrupciones de propósito general ya disponibles en la familia de bloques funcionales del microprocesador.

En el caso del 8051, la versión estándar dispone de dos entradas de interrupción externas de propósito general y otras tres internas de propósito específico, cada una de las cuales dispone de un vector de interrupción distinto. Debido a que en nuestro caso existían subsistemas adicionales, tales como el interfaz de comunicaciones serie síncronas o un bloque generador de reloj, existen nuevas fuentes de interrupción que deben atenderse de forma separada. La figura 3.4 muestra esquemáticamente las distintas interrupciones posibles en nuestra realización particular de FIPSOC. El bloque EIC (*External Interrupt Controller*) es el controlador de las interrupciones adicionales producidas por los distintos subsistemas de la arquitectura.

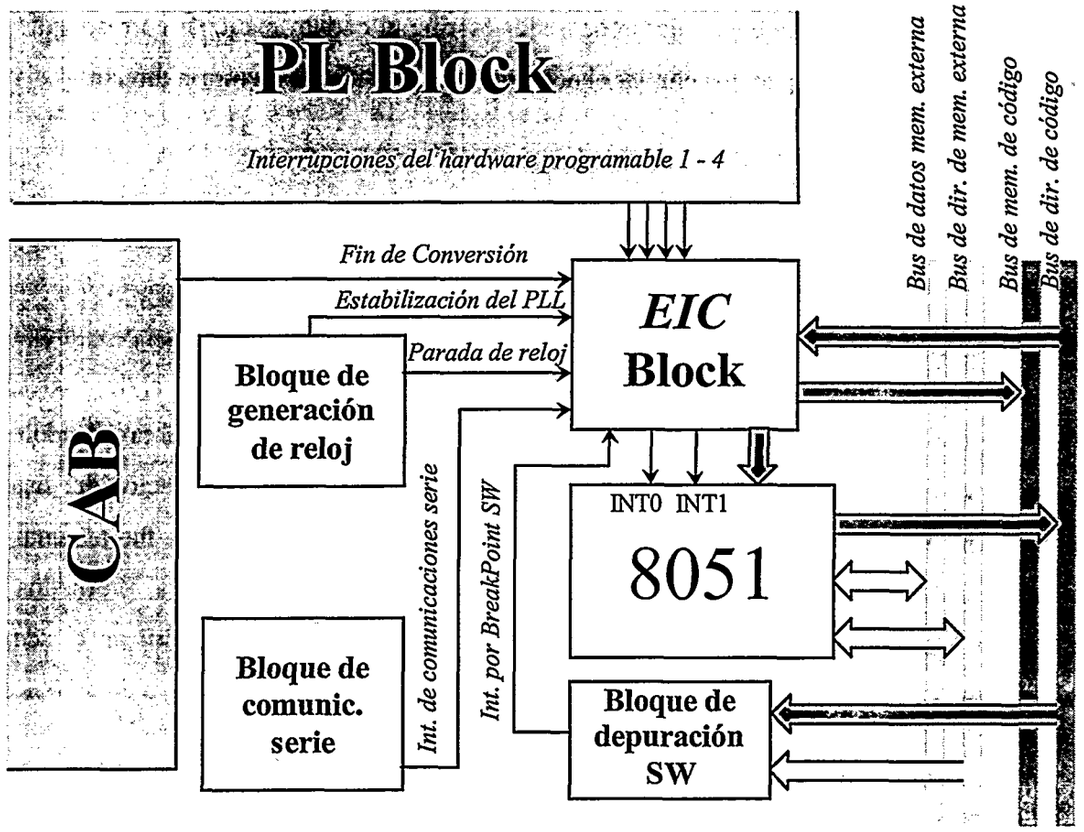

Fig 3.4: El controlador de interrupciones de FIPSOC

Para atender a las distintas causas de interrupción utilizando sólo las dos entradas disponibles en el 8051 existen varias posibilidades. Una de ellas es la de compartir el vector de interrupción. De esta forma, varias causas distintas podrían activar la misma entrada de interrupción (mediante una función *or*) y la rutina de servicio sería la encargada de discernir entre las posibles causas consultando un registro de estado. Esta solución complica las rutinas de servicio ya que estas deben atender a varias causas que no siempre están relacionadas unas con otras, lo cual puede producir efectos laterales de difícil caracterización.

En su lugar, nuestro interfaz incorpora una circuitería dedicada que intercepta en bajo nivel los accesos a los vectores de interrupción de las dos causas externas, y teniendo en cuenta la temporización de la memoria de código inyecta en el bus de datos los valores previamente almacenados en ciertos SFRs, haciendo que el microprocesador salte sobre las direcciones almacenadas en los SFRs en vez de las direcciones habituales de las causas de interrupción externa estándar. Mediante esta técnica se consigue extender el número de interrupciones de forma transparente al usuario, y disponer de un vector efectivo independiente (almacenado en un

par de SFRs) para cada causa. Para esto se utilizan las entradas de interrupción externas del 8051 estándar.

En total, Las causas de interrupción que incorpora nuestro interfaz son las siguientes:

- Estabilización del PLL.- Como se explicará más adelante, el generador de relojes utiliza un PLL interno a partir del cual se obtienen las frecuencias de base con las que generar los relojes de forma síncrona para los distintos subsistemas del *chip*. Dado que el PLL necesita un tiempo relativamente grande para estabilizarse (del orden de milisegundos), se puede generar una interrupción cuando el PLL esté finalmente estabilizado, para así utilizarlo para generar relojes sólo cuando estos tengan garantías de funcionar correctamente.
- Bloque de comunicaciones serie síncronas.- En el siguiente párrafo se describen las distintas posibilidades de comunicación serie que incorpora el *chip*. En particular se ha implementado un interfaz completo de comunicaciones serie síncronas que siguen los protocolos I^2C y SPI. Los distintos eventos producidos durante la comunicación, tales como el final de la transmisión de una palabra, la recepción de un dato, o la detección de un error en el protocolo, pueden producir una interrupción.
- Parada de reloj.- Como se describe más adelante, el bloque generador de relojes incluido en la arquitectura permite programar paradas de reloj tras un cierto número de ciclos. Coincidiendo con este evento puede programarse una interrupción.
- *Breakpoints software*.- El chip incorpora toda una circuitería de *breakpoints* para emulación *on-chip*, que permiten producir una interrupción cuando se alcanza una determinada línea de código o cuando se accede a una determinada posición de la memoria de datos.
- Final de conversión de un ADC.- Se puede programar individualmente una máscara de interrupción para cada uno de los conversores analógico-digitales del CAB.
- Interrupciones *hardware* generales.- Se dispone de cuatro entradas de interrupción de propósito general, que se conectan desde los mismos recursos de rutado de la FPGA a través de los IICs. Mediante la FPGA y estas entradas se puede de esta manera implementar cualquier tipo de periférico y habilitar una interrupción de la naturaleza que se desee.
- Interrupciones internas del 8051.- Además de las descritas, que son las añadidas por el interfaz, el 8051 estándar posee otras tres causas de interrupción totalmente independientes, con un vector de interrupción distinto. Estas interrupciones son las debidas a cada uno de los dos temporizadores y la correspondiente al puerto serie asíncrono (RS232).

El interfaz no sólo implementa la circuitería especial que permite independizar las causas de interrupción "extendidas" con vectores distintos y autónomos, sino que además incorpora máscaras independientes y un sistema de prioridades. En particular, la circuitería de las interrupciones extendidas está duplicada para las dos interrupciones "externas" del 8051 de base que en realidad se usan. De esta manera es siempre posible hacer que una causa de interrupción tenga una prioridad relativa mayor que otra, pues cualquiera de las causas extendidas puede habilitarse sobre cada una de las dos entradas de interrupción externa del 8051 de base, de las cuales una tiene mayor prioridad que otra.

3.2.8 Comunicaciones serie y modos de arranque

Otro aspecto importante a la hora de diseñar un sistema basado en microprocesador, y especialmente cuando se trata de un sistema programable, es el de sus mecanismos de inicialización y arranque. Típicamente el sistema se arrancará desde un PC, desde una memoria serie al estilo de las FPGAs clásicas o desde una memoria paralelo al estilo de los microcontroladores. Para dar consistencia a los distintos modos de arranque, se ha incorporado una circuitería especial de inicialización inspirada en el modo *special bootstrap* soportado por el microcontrolador 68HC11 de Motorola [MOT92], en el cual el micro inicializa el puerto serie asíncrono (RS232) y se queda esperando un conjunto predefinido de *bytes* que va colocando de forma secuencial a partir de una posición fija de la memoria RAM, tras lo cual ejecuta un salto a esa posición. De esta manera se consigue que el microprocesador pueda arrancarse mediante cualquier interfaz serie (típicamente desde un PC a través del puerto serie RS232) sin tener que grabar previamente un programa en una memoria no volátil (típicamente una EPROM).

En lugar que utilizar un esquema tan rígido como el utilizado por Motorola, nuestro interfaz dispone de un modo de arranque altamente versátil mediante el cual no sólo pueden transferirse programas de incialización sino que además pueden modificarse registros, ejecutar subrutinas, o incluso realizar lecturas de memoria y de registros. El programa de arranque inicializa primeramente el interfaz de comunicaciones y después se mantiene interpretando órdenes de control codificadas como una extensión de los campos HEX utilizados por Intel para programar microprocesadores. Los campos (o *records*) HEX extendidos que se usan tienen un formato similar al siguiente:

```
:  <numero_de_bytes>  <direccion_alta>  <direccion_baja>  <tipo_de_record>
{<byte_de_datos>} <checksum>
```

, es decir, comienzan con un carácter de dos puntos (";"), siguen con el número de *bytes* que serán transmitidos, después se especifica la dirección de destino (64Kb), luego el tipo de *record*, después la secuencia de datos a transmitir, y finalmente el *checksum* de comprobación de consistencia de datos, definido de tal forma que la suma de todos los *bytes* transmitidos, incluido el *checksum*, sea igual a cero despreciando el acarreo.

Mediante distintos tipos de *record* se pueden ejecutar diversos comandos, como leer o escribir memoria de datos, código o interna, o ejecutar saltos de programa o de subrutina. Es posible por tanto no sólo programar e inicializar el *chip* sino además controlarlo desde un terminal remoto cpaz de generar estos *records* e interpretar las respuestas (en los comandos de lectura). El programa de interpretación en tiempo real de estos *records* está almacenado en la ROM de inicialización del *chip*, y solamente ocupa unos cientos de *bytes*.

Más importante todavía, el interfaz de comunicaciones para transmitir estos *records* es también seleccionable mediante ciertos terminales del *chip* controlados desde el exterior. En total son tres los interfaces de comunicaciones desde los que puede transmitirse *records* HEX de control: el puerto serie asíncrono (RS232), el puerto síncrono SPI, y el puerto síncrono I^2C.

El puerto serie asíncrono está incluido incluso en las versiones comerciales del 8051, y puede considerarse como parte integrante fundamental del microcontrolador. Los interfaces síncronos fueron añadidos para disponer de canales estandarizados de comunicación con memorias serie y con otros FIPSOCs, con el objetivo de poder inicializar todo el sistema desde una memoria serie de forma análoga a las FPGAs existentes en el mercado. Lo interesante de esta metodología radica en que el programa de inicialización que interpreta los *records* es el mismo independientemente del canal serie empleado, por lo cual la inicialización se lleva a cabo de igual forma desde una memoria serie que desde el PC. Es posible por tanto desarrollar una aplicación desde un PC y posteriormente "congelar" la configuración del sistema sin más que escribir en la memoria serie los mismos comandos que se escribieron desde el PC, salvados típicamente en un fichero de historial (*log*).

Los protocolos síncronos utilizados son SPI e I^2C. Ambos protocolos permiten comunicar simultáneamente un dispositivo "maestro" (*master*) y varios "esclavos" (*slaves*), de tal forma que es el maestro el que genera el reloj de comunicaciones. Dado que un FIPSOC puede ser

configurado para arrancar desde cualquiera de estos dos interfaces (además del puerto serie asíncrono) en modo maestro o esclavo, es posible hacer que un conjunto de FIPSOC arranquen desde una misma memoria serie o desde un banco de ellas, compartiendo todos los dispositivos (FIPSOCs, memorias o cualquier otro dispositivo que cumpla el protocolo) el mismo bus. Además de inicializarlos, también es posible controlar cualquiera de los dispositivos de la cadena mediante uno de estos interfaces, por ejemplo para crear un sistema de desarrollo de aplicaciones distribuidas entre varios FIPSOCs.

Finalmente, las memorias serie comerciales existentes controladas mediante estos protocolos síncronos, típicamente producidas en tecnología *flash* o *EEPROM*, suelen permitir su escritura desde el mismo bus de comunicaciones. Por tanto es posible programar la memoria desde el mismo *chip*, cambiar su configuración para arranques subsiguientes, o incluso utilizar partes de esta memoria de configuración inicial para usos de propósito general (guardar datos importantes de la aplicación). La aplicación puede por tanto evolucionar y adaptarse a las condiciones de contorno particulares en cada momento, guardar parámetros de autocalibración, o incluso autodestruirse sin más que borrar los datos de inicialización de la memoria serie para hacer imposible el arranque del dispositivo si se detecta una mala operación o evidencia de espionaje industrial.

La utilidad de esta posibilidad de escritura desde el *chip* es aún más importante para el diseño de un sistema de desarrollo, ya que la información de configuración puede transferirse a la memoria desde el mismo *chip* sin ser necesario ningún *hardware* adicional. La aplicación puede por tanto desarrollarse en el PC, probarse en el *chip*, y finalmente transferirse a la memoria, todo ello utilizando tan sólo el PC, el *chip*, y un cable serie RS232.

3.2.9 Generadores de reloj

Para explotar el acoplamiento entre el microprocesador y la FPGA es importante asegurar las transacciones entre el microprocesador y los registros (FFs) de los DMCs. Para conseguir que las transferencias de datos entre ellos sean síncronas y por tanto seguras, el interfaz diseñado incorpora un bloque que genera los relojes que se utilizan en todo el dispositivo, incluyendo el microprocesador y la FPGA. La figura 3.5 muestra esquemáticamente este bloque.

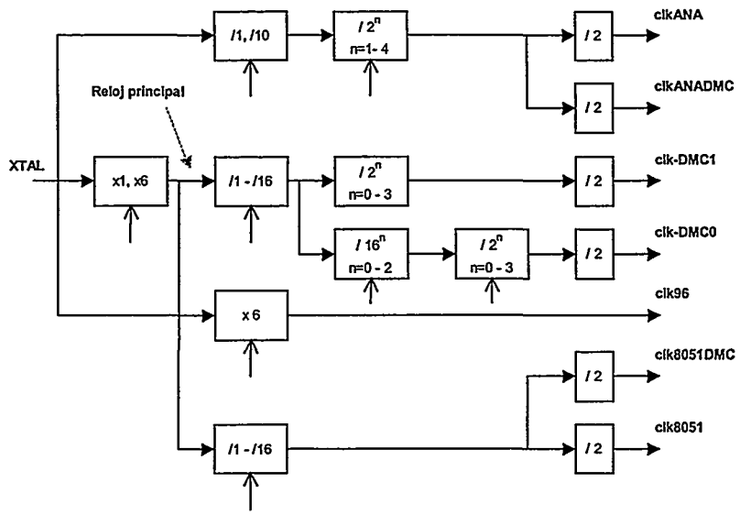

Fig 3.5: Bloque generador de reloj

El generador de relojes deriva todas las señales de reloj para los distintos subsistemas mediante divisores (contadores) a partir de un reloj principal rápido que a su vez puede venir directamente de un oscilador de cristal o de un PLL que multiplica la frecuencia del oscilador por seis. De esta forma, todas las señales de reloj generadas a partir de este bloque tienen una fase relativa conocida y solidaria, con lo cual pueden garantizarse las transferencias síncronas entre relojes de distintas frecuencias, y en particular entre los programas del microprocesador y los registros de los DMCs sin necesidad de parar ningún reloj.

Otra interesante ventaja de generar los relojes de forma coherente desde una misma fuente es la posibilidad de programar paradas de relojes sin que se pierda el sincronismo entre ellos. Este tipo de operación se gobierna en nuestro caso mediante SFRs: primeramente se designa uno de los relojes como "referencia", y después se especifica el número de ciclos del reloj de referencia tras el que debe producirse la parada. El hecho de que todos los relojes se generen desde el mismo reloj principal hace que en ningún caso se pierdan ciclos independientemente del instante de parada o del reloj de referencia, pues todo el "engranaje" de la máquina de reloj se mueve solidariamente como un conjunto de ruedas dentadas en un reloj mecánico.

Si el número de ciclos de reloj a esperar para cada parada se hace igual a uno, se obtiene un modo de ejecución "flanco a flanco", en el que el microprocesador tiene el control (recibe una interrupción tras cada parada) al final de cada ciclo de reloj. De esta forma, el microprocesador puede leer y escribir datos en los *flip-flops* mientras la máquina de reloj está parada, por ejemplo para inicializar los registros de un determinado subsistema mapeado en la lógica programable.

Finalmente, en lugar de programarla es posible disparar la parada de la máquina de reloj mediante una señal proveniente de la misma FPGA. Para ello se dispone de una serie de entradas de parada de reloj en ciertos IICs.

3.2.10 Breakpoints hardware *y* software

Mediante la circuitería descrita en el párrafo anterior es posible implementar lo que se ha llamado *breakpoints hardware* o puntos de parada programada del reloj del sistema, como analogía a los *breakpoints software* conocidos en los emuladores y otras herramientas de desarrollo de programas. No sólamente es posible programar paradas del reloj en un cierto instante de tiempo, sino que además se puede utilizar una LUT (o cualquier combinación de *hardware* programable) para generar una función de parada, por ejemplo para detectar una determinada condición de operación incorrecta y así poder examinar detenidamente el estado del sistema en el momento de la parada.

Además de estos *breakpoints hardware* nuestro sistema dispone de una circuitería dedicada para la programación de *breakpoints software* clásicos, muy útiles para la construcción de emuladores y sistemas de desarrollo en general. Los *breakpoints* se programan mediante SFRs para producir una interrupción cuando el contador de programa del microprocesador coincide con alguno de los valores programados en los registros correspondientes (*breakpoints* de programa), o bien cuando se detecta un acceso de lectura o de escritura en una posición de la memoria de datos que coincide con una de las programadas en los registros correspondientes (*breakpoints* de lectura o de escritura de datos).

3.2.11 Co-emulación y co-simulación hardware-software

Como puede verse, la conjunción de todos estos elementos que se han presentado en los últimos párrafos resulta de gran utilidad para la implementación de sistemas de desarrollo.

En primer lugar, la posibilidad de disponer de *breakpoints hardware* y *software* permite una emulación integrada: dado que se conoce el número de ciclos del reloj de referencia que se avanzará durante cada instrucción del microprocesador, es posible mantener ambos dominios ejecutándose concurrentemente. De esta forma, cada vez que se ejecutase una instrucción del

microprocesador se programaría una parada de reloj tras el número correspondiente de ciclos, o bien se avanzaría este número de ciclos flanco a flanco. Para conseguir resultados más exactos es posible avanzar el número indicado de ciclos hasta el sub-ciclo dentro de la temporización de la instrucción particular en el que los datos se muestreen o se escriban.

Si se mantiene una ejecución sincronizada paso a paso es posible además muestrear las señales de interés (marcadas con puntas de prueba en el mismo sistema de desarrollo sobre los propios esquemas de la aplicación) tras cada ciclo de reloj, y presentar los resultados leídos en el interfaz gráfico en un visualizador de formas de onda análogo a los utilizados por los simuladores o los analizadores de estados. En el fondo el esquema propuesto emula hasta cierto punto un analizador de estados, con la salvedad de que se para el reloj al final de cada ciclo y por lo tanto la temporización no es completamente realista aunque sí es posible validar la aplicación hasta una cierta frecuencia de funcionamiento a la que se perdería la sincronía.

El interfaz gráfico mostraría de esta forma un sistema de co-emulación integrada, en el que las instrucciones del microprocesador y las formas de onda irían evolucionando de forma sincronizada. Este tipo de esquema puede incluso sustituir a un simulador, pues las funciones lógicas, más que simularse, se ejecutarían realmente en la propia lógica programable de forma mucho más rápida a la velocidad de cálculo de los simuladores actuales. Los estímulos pueden ser impuestos desde FFs a base de escrituras desde el microprocesador, o directamente desde células de entrada-salida aislándoles la entrada real y cambiando su polaridad, pudiéndose así incluso emular el comportamiento de un sistema externo ideal al imponer los estímulos que éste generaría.

No obstante, para la emulación de sistemas grandes que desborden la capacidad del *chip* resulta interesante disponer de la posibilidad de simular partes del sistema, o incluso de mezclar la simulación con la co-emulación integrada. Para ello bastaría con utilizar el mismo protocolo de comunicaciones entre la actualización de la cola de eventos del simulador y el controlador del co-emulador. De esta forma se podría llevar a cabo una co-emulación co-simulación integrada en la que partes *hardware* y *software* de la aplicación se ejecutarían parcialmente en el *hardware* programable y el microprocesador (parte emulada), y otras partes se simularían en el PC.

Es interesante notar que todo el control de la emulación, presentación gráfica de resultados, y gestión general del *chip* se lleva a cabo desde un terminal conectado a él mediante un canal serie síncrono o asíncrono, típicamente desde un PC a través de un puerto RS232. En la sección siguiente se comentará las posibilidades que la parte analógica incorpora de cara a una emulación extendida de señal mixta.

3.3 Interfaz entre el microprocesador y las células analógicas programables

La interacción entre el microprocesador y el CAB dependerá en general de la granularidad elegida para la célula analógica. En nuestro caso, en el que el CAB es un bloque programable de funcionalidad fija orientado a un determinado tipo de aplicaciones, el interfaz responde a un esquema relativamente clásico en el cual las partes digitales de las células analógicas se controlan desde el microprocesador mediante SFRs. Fundamentalmente la interacción se centra en la programación del subsistema analógico y en las entradas y salidas de datos digitales de los conversores y comparadores. De forma análoga a la existente entre el microprocesador y la FPGA, esta interacción ofrece interesantes posibilidades hacia la integración de aplicaciones *software* y el *hardware* de señal mixta involucrado, además de facilitar el diseño de sistemas de desarrollo e incluso emulación analógica.

3.3.1 Control de células analógicas programables

En primer lugar, el interfaz del microprocesador diseñado mapea en memoria de datos los registros de control del CAB. Estos registros permiten controlar de forma digital los distintos aspectos programables de los bloques constitutivos del CAB, que en nuestra realización particular introducida en el capítulo anterior son la configuración de las secciones de amplificación (ganancia, *offset*, CMRR), el rutado de las señales a acondicionar y a convertir, el rutado de las señales de referencia, la habilitación de las secciones de amplificación y de los *buffers* de los conversores, la configuración del modo de funcionamiento del bloque de conversión, el acceso directo a los amplificaciones operacionales diferenciales y balanceados de las secciones de amplificación, y la gestión del modo de calibración para prefijar los *offsets* de los distintos amplificadores.

Las estructuras de memoria de configuración de estos aspectos programables se basan en células de memoria RAM clásica y no en memorias multicontexto como las descritas en el capítulo anterior, ya que los tiempos de estabilización de las células programables es en principio grande comparado a los tiempos de ejecución y reconfiguración dinámica. No obstante es posible cambiar de forma dinámica la programación de estas células, por ejemplo para cambiar los

factores de ganancia de los canales de amplificación y producir así modulaciones en amplitud controladas por el microprocesador, si bien la utilidad de este tipo de aplicaciones es limitada.

3.3.2 Mapeo de puertos de células analógicas

Los puertos digitales de las células mixtas se acceden desde el microprocesador como posiciones de memoria. Entre estos puertos se encuentran las salidas (digitales) de los comparadores, y los registros de control del DAC/ADC (palabra a convertir en el DAC, palabra convertida y órdenes de conversión del ADC). Este tipo de interacción es el habitual en microcontroladores que incorporan bloques de conversión analógico-digital.

3.3.3 Aplicaciones indirectas para emulación y desarrollo.

Extendiendo el concepto de co-emulación *hardware-software* de sistema en tiempo pseudo-real como se introdujo en la sección anterior, sería interesante permitir una co-emulación de señal mixta. Con una herramienta así podría co-emularse en tiempo real un mayor número de aplicaciones en las que la interacción entre el dominio analógico y el digital no es evidente. No obstante, la naturaleza continua de las señales analógicas hacen que esto sea muy difícil de conseguir incluso con una utilidad limitada.

En cualquier caso, puede proponerse el siguiente esquema de funcionamiento: el microprocesador, o el *hardware* programable en su lugar, podrían tomar muestras de una cierta señal analógica a tratar por la aplicación (o esta señal podría ser generada en un fichero de muestras desde un programa o cual otro elemento externo). Esta señal "grabada" irá reproduciéndose de forma segmentada en tiempo discreto según avance el tiempo de emulación. Para ello se configuran los canales de amplificación en modo de calibración (en el cual sus entradas se cortocircuitan a una referencia conocida) y se varían los *offsets* y los factores de amplificación para conseguir los valores deseados (dentro de la precisión permitida por los conversores A/D, ya que es mediante estos como se miden los valores particulares de la señal "emulada"). En cualquier momento, y por definición, una conversión resultaría en el valor impuesto en cada momento en el fichero de muestras de la señal a emular. Y más importante todavía, los comparadores funcionarían como es debido en función del valor real de la señal en

cada momento, produciendo eventos digitales que pueden procesarse con el resto de las señales digitales en la lógica programable en tiempo segmentado de emulación.

Como puede verse el sistema resulta relativamente limitado y de difícil uso, pero al menos permite ciertas posibilidades interesantes si se trata de desarrollar aplicaciones en las que la sincronización entre los comparadores o los conversores y la lógica programable o el programa del microprocesador es crítica. Es importante notar que aunque los relojes de los registros de aproximaciones sucesivas que implementan los conversores analógico-digitales se generan a partir de la misma máquina de reloj, no es posible detenerlos y volver a ponerlos en marcha ya que los valores almacenados en los condensadores de muestreo (*sample & hold*) de los comparadores perderían su valor debido a sus corrientes de fugas relativamente significativas. En cualquier caso, como el emulador conoce la temporización relativa del algoritmo de aproximaciones sucesivas es por tanto posible realizar la conversión A/D con el resto de los relojes parados justo en el ciclo en el que se esperaba el resultado, aparentando a todos los efectos que el A/D está perfectamente sincronizado con el resto de la aplicación.

De forma más sencilla de utilizar y más habitual, la conversión A/D se utiliza para emular un osciloscopio digital desde el PC, el cual presentaría los datos convertidos a intervalos regulares (el ADC soporta un modo de conversión continua) sobre una escala de tiempos. En general se puede utilizar dos DACs adicionales para fijar las referencias del ADC usado para las conversiones, y de esta manera disponer de un mando virtual de offset (que sumaría o restaría la misma constante a ambos DACs de referencia) y factor de amplificación (que variaría la diferencia entre los valores de los DACs de referencia). El osciloscopio digital emulado forma parte del emulador integrado, y sirve para monitorizar señales o para grabar señales a emular posteriormente.

3.4 Interfaz entre la lógica programable y las células analógicas programables

La interacción entre los bloques programables analógicos y digitales se centra naturalmente en la parte digital de las células analógicas. En particular resulta de gran utilidad disponer de una conexión entre los canales de rutado general y las salidas de los comparadores y los terminales de control de las estructuras de conversión, para así obtener una total flexibilidad de cara a la construcción de aplicaciones de señal mixta. Además, generalmente, sólo mediante la lógica

programable es posible explotar todas las posibilidades del CAB en términos de velocidad de operación. En nuestra realización particular, la parte digital de los conversores y comparadores puede conectarse directamente a la lógica programable a través de los IICs, independizando de esta manera su operación del microprocesador.

Para apoyar el concepto de emulador integrado presentado en las secciones anteriores, que disponía de un co-emulador paso a paso *hardware-software* y de un osciloscopio o incluso un analizador de estados integrados, resulta interesante disponer de otros instrumentos de laboratorio emulados en el chip. Entre ellos podemos contar el generador de formas de onda, que puede utilizarse tanto como señal de base para una modulación como para generar una salida determinada al mundo exterior. Un generador de forma de onda puede generarse en nuestra arquitectura sin más que introducir las muestras necesarias en un conjunto de DMCs configurados como banco de memoria RAM cuyas salidas de datos se conectan a un DAC y cuyas entradas de direcciones se conectan a un contador también implementado a base de DMCs. La frecuencia de salida se selecciona variando la frecuencia de operación del contador, y su amplitud cambiando las referencias del DAC utilizado o variando los valores máximo y mínimo de las muestras utilizadas. Si bien la salida desde los DACs al exterior no es directa ya que es necesario un *buffer*, el cual limita en cierta medida el ancho de banda de la señal de salida (el amplificador utilizado tiene típicamente un ancho de banda a ganancia unidad de 3MHz), sí es cierto que mediante este esquema se puede conseguir la máxima frecuencia para la señal de salida analógica si la comparamos con la generada por el microprocesador, la cual estaría limitada por la velocidad del programa que escribiría las muestras en el puerto de entrada del DAC.

Con la conversión analógico-digital ocurre algo parecido, ya que en los modos de velocidad de conversión más rápida el microprocesador no es capaz de leer las muestras de salida al mismo ritmo. En particular en el modo *pipeline* se puede llegar a generar muestras a ritmo de una cada microsegundo, tiempo en el que el micro sólo podría ejecutar tres operaciones simples utilizando la velocidad de funcionamiento más rápida. En su lugar, es posible utilizar DMCs configurados como bloques de memoria RAM de usuario para almacenar las muestras provenientes del ADC, en las que el bus de direcciones estaría conectado de nuevo a un contador implementado igualmente con DMCs. Tras la etapa de conversión, el microprocesador podría leer todas las muestras seguidas desde las LUTs, y presentarlas en pantalla o realizar cualquier otro tratamiento sobre ellas.

Finalmente, la lógica programable puede utilizarse para hacer variar de forma más o menos rápida el valor de un DAC utilizado como referencia para las secciones de amplificación o para los comparadores. Por ejemplo sería posible implementar de esta manera un control de histéresis programable para los comparadores sin más que cambiar la referencia mediante la lógica programable al detectar una transición en la salida del comparador.

3.5 Resumen

En este capítulo se ha ilustrado la metodología a seguir en el diseño del interfaz *hardware* entre los tres subsistemas presentes en una arquitectura de tipo FIPSOC – FPGA, CAB y microcontrolador. Se ha hecho especial énfasis en la cercana interacción entre el microprocesador y los DMCs que permite una fácil y potente reconfiguración dinámica y parcial así como un acceso en tiempo real a los estados de las señales de la FPGA a base de simples accesos a memoria. Se ha descrito la funcionalidad de los distintos periféricos incorporados al microprocesador para su utilización como supervisor de la FPGA y como interfaz con un PC o cualquier otro elemento externo de interacción con el usuario. Además se ha descrito los mecanismos básicos de depuración *hardware* y *software* que pueden ser incorporados al mismo *chip*, aspecto que simplifica la creación de un sistema de desarrollo y emulación integrado como se explicará en el capítulo cinco. Finalmente se han considerado los distintos interfaces con las células analógicas, tanto para su control desde el *hardware* programable o el microprocesador como para su configuración desde posiciones de memoria de datos.

CAPÍTULO 4

IMPLEMENTACIÓN DE ARQUITECTURAS TIPO FIPSOC

4.1 Introducción

En los capítulos anteriores se ha descrito de forma más o menos exhaustiva la metodología a seguir para el diseño de arquitecturas programables tipo FIPSOC desde un punto de vista teórico y funcional, sin entrar en detalles sobre su implementación práctica. Sin embargo, los aspectos prácticos del diseño físico de la estructura son de gran importancia para su viabilidad, pues en gran medida influyen en las prestaciones, el área consumida, y su estabilidad.

Como demostrador de la metodología explicada en los capítulos anteriores, se ha implementado un primer ejemplo de arquitectura tipo FIPSOC. Este *chip*, probado con éxito en laboratorio, integra los bloques presentados como ejemplos concretos en los capítulos anteriores, y por tanto demuestra su viabilidad técnica. En este capítulo se estudiará el área consumida por cada subsistema para así poder conocer el coste relativo de las soluciones arquitecturales propuestas. La figura 4.1 muestra una microfotografía de la primera arquitectura FIPSOC implementada.

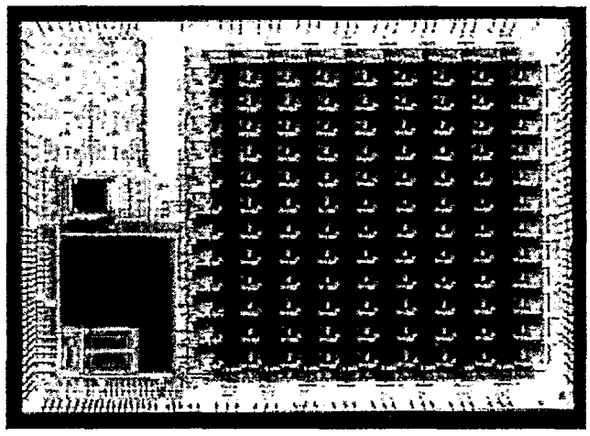

Fig 4.1: Microfotografía del primer demostrador de arquitectura de tipo FIPSOC

El circuito integrado de la figura 4.1 incluye los tres subsistemas estudiados en los capítulos anteriores. La parte derecha, especialmente regular incluso a simple vista, es la FPGA rodeada de las células programables de entrada-salida. La esquina superior izquierda está ocupada por el bloque analógico programable (CAB), que incluye su propia alimentación (separada de las partes digitales) y células de entrada-salida de funcionalidad fija. El resto del *chip*, situado en la esquina inferior izquierda, incluye el microprocesador y su interfaz, implementado mediante células estándar tras una síntesis lógica, y las memorias (dos RAM y una ROM).

En total el circuito tiene 163 pads y ocupa 65mm^2 (9.6x6.5 mm^2). Incluye más de 1.8 millones de transistores, de los cuales 1.5 millones están implementados en *full custom* de forma manual. La densidad de integración es por tanto mayor de 27.000 transistores/mm^2, llegando a 40000 transistores/mm^2 en los DMCs e incluso a más en las distintas áreas de memoria. El circuito fue montado para su prueba sobre un encapsulado PGA cerámico de 225 pines.

Previamente a la implementación de este primer FIPSOC completo se fabricaron dos *chips* de prueba para comprobar las ideas propuestas de mayor riesgo, especialmente la viabilidad de la reconfiguración dinámica y la calidad de las células analógicas. El primero de estos circuitos de prueba incluía una implementación preliminar de un *array* de cuatro DMCs y un primer *layout* del bloque conversor digital-analógico. El segundo *testchip* incluyó los amplificadores operacionales utilizados como *buffers* para las señales de entrada, de referencia y de salida del DAC, el amplificador operacional diferencial balanceado, y la sección de amplificación completa basada en este último.

En las secciones siguientes se analizará los aspectos prácticos de la implementación de los distintos subsistemas que integran este primer FIPSOC, y se presentarán resultados de caracterización medidos sobre silicio real. A partir de los resultados obtenidos en cuanto a área consumida y prestaciones se pueden sacar conclusiones sobre la viabilidad de la estructura propuesta.

4.2 Metodologías seguidas

Debido a la diferente naturaleza de las distintas partes que integran el *chip*, la metodología seguida para el diseño y verificación de cada una de ellas y de su interacción resulta de gran importancia. El flujo de diseño seguido resulta extenso y complicado, pues incorpora un gran número de herramientas diferentes que no están bien preparadas para trabajar de forma conjunta.

La FPGA se diseñó a nivel de esquemas utilizado la herramienta Composer™ (Cadence™) Las simulaciones se llevaron a cabo a nivel físico utilizando el simulador HSPICE™ (Avant!™) El diseño físico (*layout*) se realizó mediante la herramienta Virtuoso™ (Cadence™), y se utilizó Diva (Cadence™) para las extracciones de *layout* y para las comprobaciones físicas de reglas de diseño (DRC, *Design Rules Check*) y de *layout* contra esquema (LVS, *Layout Vs. Schematic*). Se volvió a utilizar HSPICE™ para la simulación física *post-layout*.

La parte analógica siguió un flujo de diseño igual al de la FPGA, aunque las simulaciones tuvieron en cuenta las posibles dispersiones de parámetros que podían afectar críticamente a las prestaciones de la célula (*offset*, margen dinámico, etc.), y muy especialmente al *matching* o apareamiento de los transistores y de las resistencias de polisicio.

El microprocesador y su interfaz fueron implementados mediante una librería de células estándar proveída por el fabricante, Atmel-ES2. El microprocesador era una *netlist* arquitectural a nivel de puertas lógicas, mientras que el interfaz fue escrito y simulado en Verilog™ (Cadence™) y sintetizado con Synopsys™ sobre la librería de células estándar. Tras la síntesis se obtuvo una *netlist* completa de células estándar desde la que se hizo un *placement & routing* usando Silicon Ensemble™ (Cadence™) y su generador de árbol de reloj. Finalmente se importó el *layout* resultante y se calcularon retardos de interconexión a partir de resistencias y capacidades parásitas extraídas en formato SDF (*Standard Delay Format*), que luegon fueron introducidas (*backannotation*) sobre los esquemas originales para llevar a cabo una simulación *post-layout*.

Las células *custom*, tanto analógicas como digitales, fueron modeladas mediante VHDL y Verilog™ para realizar verificaciones funcionales a nivel de sistema. La temporización de las células digitales *custom* fue descrita en TLF (*Timing Library Format*) y compilada en vistas de temporización o *timing views* que luego el calculador central de retardos (CDC, *Central Delay Calculator*) de Cadence™ podría usar para los cálculos de temporización globales del circuito y así mezclar ambos tipos de células (estándar y *custom*) en las simulaciones digitales a nivel de sistema.

Finalmente se llevó a cabo el diseño del *bonding* o conexión del *chip* a su encapsulado mediante la herramienta FAST suministrada por la misma fábrica.

4.3 Implementación de las células lógicas programables

La FPGA fue implementada en *full custom* siguiendo una metodología *bottom-up*, es decir, diseñando bloques pequeños iniciales y siguiendo hacia arriba según el orden de la jerarquía. En primer lugar se diseñaron los bloques constitutivos del DMC para luego unirlos y después replicar el resultado bidimensionalmente para crear una matriz (aunque en realidad se trata de dos sub-mallas de DMCs pares e impares). Posteriormente se crearon las células de entrada-salida y los IICs, para los cuales se reaprovecharon las estructuras de memoria y los multiplexores de la arquitectura de rutado del DMC. Finalmente se unió todo para dar lugar a la matriz que puede observarse en la figura 4.1.

En los siguientes párrafos se comentan los aspectos más significativos de la implementación de cada uno de los bloques constitutivos del DMC, así como del DMC en su conjunto. A continuación se presentan las células de entrada-salida y los IICs, para terminar realizando un balance sobre la viabilidad de la estructura de configuración multicontexto propuesta.

4.3.1 Parte combinacional

La figura 4.2 muestra el *layout* de la parte combinacional del DMC del primer FIPSOC. A la izquierda puede apreciarse la estructura regular de células de memoria que configura las cuatro LUTs agrupadas en dos mitades. Arriba a la derecha se observa la circuitería dedicada a las

funciones de propagación y generación utilizadas en el modo aritmético (sumas y restas fundamentalmente).

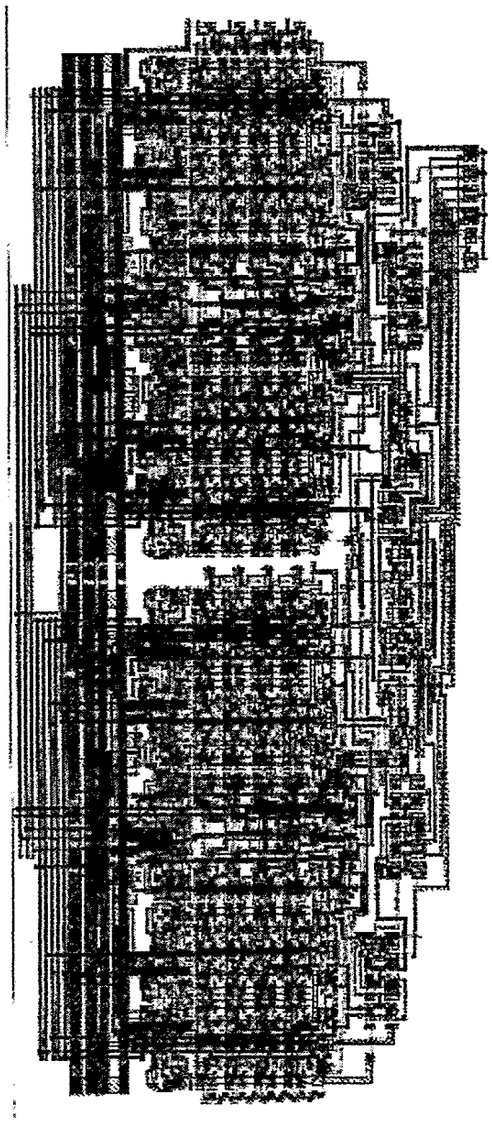

Fig 4.2: *Layout* de la parte combinacional del DMC

Como se explicó en el capítulo 2, el diseño de las estructuras de memoria utilizadas en las LUTs no es evidente ya que se trata de memoria de doble puerto en el que uno de ellos no tiene una temporización definida. Además de esta consideración que puede tenerse en cuenta a nivel de esquemas, la mayor preocupación a la hora de realizar el *layout* de estas células es la capacidad parásita adicional que existe debido a las líneas que conectan salidas intermedias y finales de las

LUTs con la circuitería aritmética y la parte secuencial. Estas capacidades se deben a los cruces de estas líneas, trazadas en el metal superior, con las líneas de datos de las memorias, trazadas en el metal intermedio, y las difusiones mismas de los inversores realimentados que conforman las células de memoria. Es preciso tener en cuenta la magnitud de estas capacidades y simular su comportamiento sobre la estabilidad del efecto memoria durante los cambios bruscos de las entradas a las LUTs.

La figura 4.3 muestra una microfotografía de la parte combinacional de un DMC del primer prototipo completo de FIPSOC. Las LUTs y la parte aritmética pueden identificarse por paralelismo desde el *layout*.

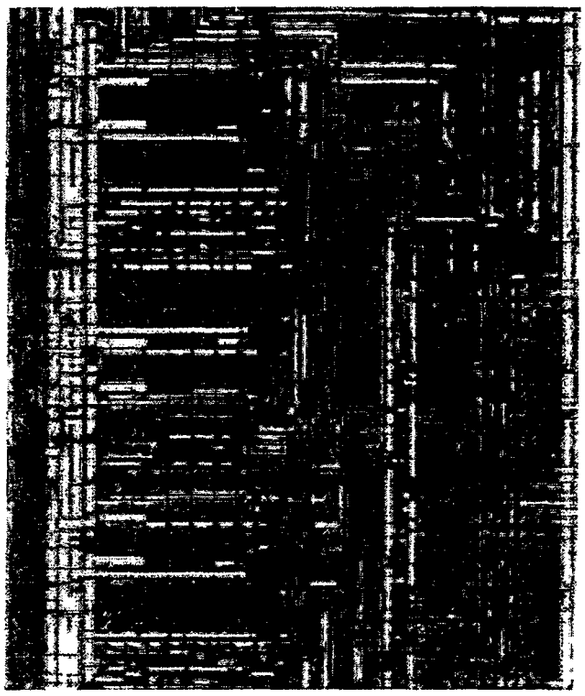

Fig 4.3: Microfotografía de la parte combinacional del DMC

4.3.2 Registros programables

La implementación física de los registros del DMC precisó de un bloque especial dedicado a generar una serie de funciones intermedias para los distintos mecanismos de *reset* y *set* asíncrono necesarios para soportar todas las funciones de los FFs. La figura 4.4 muestra la estructura completa de la parte secuencial del DMC, donde en particular puede apreciarse los cuatro FFs básicos más el bloque de generación de *resets* en la parte derecha.

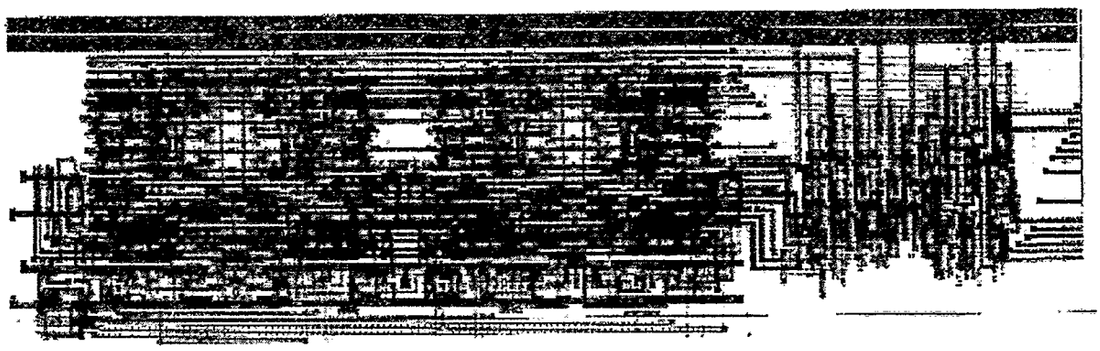

Fig 4.4: *Layout* de la parte secuencial del DMC

En la parte inferior derecha de la microfotografía mostrada en la figura 4.3 puede identificarse también este bloque.

4.3.3 Memoria de configuración

La memoria de configuración constituye el reto más importante dentro del problema general de la implementación física del DMC. Como se ha descrito en los capítulos precedentes, la memoria de configuración de los DMCs está mapeada en el espacio lógico de direccionamiento del microprocesador. De esta forma, la memoria total disponible por el microprocesador se encuentra distribuida por todo el *array* en vez de estar concentrada en un bloque como en los esquemas clásicos de microprocesadores con memoria.

La distribución de esta memoria a lo largo y ancho de la FPGA presenta varios problemas de implementación que deben ser considerados en detalle. Debe tenerse en cuenta que este primer miembro de la familia, incluso siendo uno de los pequeños, dispone de un *array* de doce por ocho DMCs, lo cual supone una área de unos 5.4mm por 5.6mm. Como cada DMC incluye dos contextos de 256 *bits* de configuración, en total el *chip* dispone de 6 *Kbytes* de memoria distribuidos en una área más o menos cuadrada del orden de $30mm^2$, lo cual supone más de 20 veces más área de la necesaria para una memoria RAM estática de la misma capacidad.

El primer problema a tener en cuenta es la resistencia parásita de las líneas de datos, ya que éstas se extienden todo lo alto de la FPGA. Los *drivers* de línea y los *sense amplifiers* se sitúan en la parte inferior de las columnas, cerca del interfaz con el microprocesador. El problema surge

durante las operaciones de escritura, en las que la resistencia parásita de las líneas se encuentra conectada en serie con los transistores NMOS de selección de las células de memoria. Es preciso asegurar que el efecto resistivo del conjunto es suficientemente pequeño como para permitir la sobreescritura de un dato en la célula de memoria, independientemente de la posición de la célula en el *array* y de la esquina del proceso considerada.

Más peligroso aún, es importante tener en cuenta la posible interacción entre las líneas de datos y los canales de rutado de la FPGA. Los canales verticales podrían producir interferencias (*crosstalk*) sobre las líneas verticales de datos de la memoria, así como los canales horizontales podrían interferir a través de las capacidades parásitas entre los dos metales debidas al cruce de pistas. Una comprobación exhaustiva pasaría por verificar que incluso cuando todos los canales de rutado que cruzan cambian su valor a la vez la memoria no pierde sus datos aunque se encuentre en mitad de un ciclo de escritura o – más peligroso – de lectura. Sin embargo, dado que la implementación de la memoria es diferencial (esto es, existen dos líneas de datos con las dos polaridades), un cruce de este tipo no debería, en principio, modificar la diferencia de tensión que evoluciona en las líneas de datos, por lo cual el resultado leído en el *sense amplifier* no debería cambiar a la postre, aunque quizás sí su temporización.

Finalmente, resulta vital dimensionar como es debido los buses de alimentación. En principio, la memoria de configuración y la circuitería activa del DMC comparten su alimentación, por lo cual debe considerarse con cuidado la cantidad de corriente necesaria en cada momento para ambas partes, y dimensionar las pistas de metal de alimentación convenientemente para evitar hacer pasar corrientes excesivas por estas pistas que podrían producir efectos no deseados de electromigración, resistencia parásita, o calentamiento excesivo del dispositivo. Especialmente debe tenerse en cuenta el momento de la transferencia de contexto, en el que todos los 256 *bits* de configuración del DMC son escritos desde la memoria, produciendo un pico de consumo dinámico muy significativo. Si además consideramos que la máscara de filas y columnas podría seleccionar todo el *array* de DMCs a la vez, el número de *bits* transferidos se multiplicaría en consonancia al igual que el pico de corriente necesario. Una mala planificación de los buses de alimentación podría impedir la correcta transferencia de información desde la memoria, podría hacer descender la tensión de alimentación instantánea por debajo de la tensión de garantía de la retención de datos en la memoria de configuración, o podría producir daños irreversibles en los buses de alimentación de forma gradual o instantánea.

En la microfotografía de la figura 4.5 puede apreciarse los distintos bloques de la estructura de configuración: a la derecha se están los decodificadores alimentados por las líneas verticales de

direcciones. En el centro se disponen los *bits* de memoria, agrupados 8 por palabra, alternándose *bytes* de uno y otro contexto. En vertical pueden verse las líneas de datos y en horizontal se aprecian las líneas de decodificación de palabras (debajo de las líneas de datos) y unas líneas horizontales en los metales superiores que llevan los datos individuales de cada *bit* a la estructura de transferencia de contexto, situada a la izquierda. Esta estructura toma como entradas cada uno de los dos *bits* en la misma posición en dos *bytes* de contextos diferentes (ya que están uno junto a otro). La orden de transferencia de contexto, así como las señales de alimentación, son comunes para todo el DMC, y se propagan en dirección vertical. A la izquierda de la fotografía, aunque fuera de ella, se distribuirían los recursos de rutado.

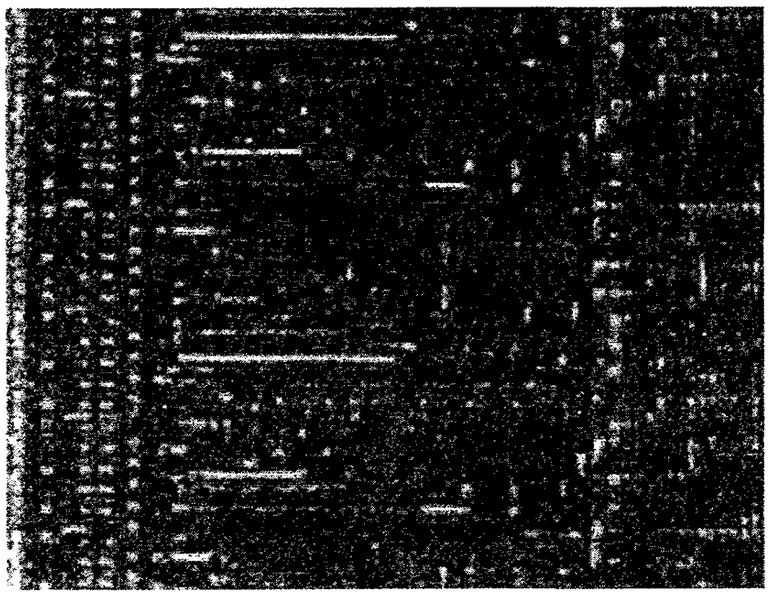

Fig 4.5: Microfotografía de la memoria de configuración y los recursos de rutado

4.3.4 Recursos de rutado

Para el diseño físico de los canales de rutado es preciso tener en cuenta los efectos parásitos en las pistas de metal. Cada pista de rutado presenta una capacidad parásita equivalente del orden de medio pF, dato importante para el cálculo incluso aproximado de los retardos de interconexión que se quieran garantizar. Según se expuso en el capítulo dos, es importante dimensionar los conmutadores que conectan canales de rutado y los *drivers* que los excitan, y la capacidad parásita de estas pistas modifica de forma fundamental este cálculo.

Además, es preciso tener en cuenta las dimensiones mínimas de los canales de rutado para garantizar su integridad durante los picos máximos de corriente dinámica en los momentos de conmutación a la máxima frecuencia de operación, pues podrían aparecer problemas de electromigración que darían lugar a fallos degenerativos del dispositivo.

Otro aspecto a considerar es la posible interacción entre canales de rutado (*crosstalk*) que debido al acoplamiento capacitivo entre líneas paralelas largas podría provocar picos de señal en líneas que deberían mantenerse estables durante la conmutación rápida de otras. Para evitar este tipo de problemas es necesario mantener la razón entre la capacidad de interacción entre líneas y la capacidad parásita con el substrato por debajo de un cierto límite que dependerá de la máxima pendiente de subida (*slew rate*) de la señal que provoca la interferencia. Naturalmente, esta relación disminuye al aumentar la distancia entre las líneas. Debe notarse aquí que este efecto no es problemático en el caso de las líneas de datos o de direcciones de la memoria de configuración, ya que aunque su acoplamiento es mucho más significativo debido a que se encuentran incluso a menor distancia unas de otras (las líneas de direcciones se encuentran dos a dos a la distancia mínima permitida por la tecnología) y a que su longitud es enorme (las líneas de direcciones recorren todas las columnas de la FPGA de abajo a arriba), su temporización es única, por lo que sus transiciones se producen al unísono y llegan a la estabilidad en los mismos momentos. Los canales de rutado, en cambio, transmiten señales cuya temporización relativa es arbitraria.

La figura 4.6 muestra el *layout* de las entradas a las LUTs. A la derecha pueden verse las estructuras de memoria mapeada y la arquitectura multicontexto como se comentó en el punto anterior, y a la izquierda pueden apreciarse los canales de rutado. Dado que los recursos de rutado utilizan multiplexores de forma exhaustiva, las señales horizontales son copiadas a pistas verticales de longitud igual a la altura del DMC. De esta forma, la multiplexión se realiza sobre canales verticales únicamente, lo que lleva a una implementación más compacta.

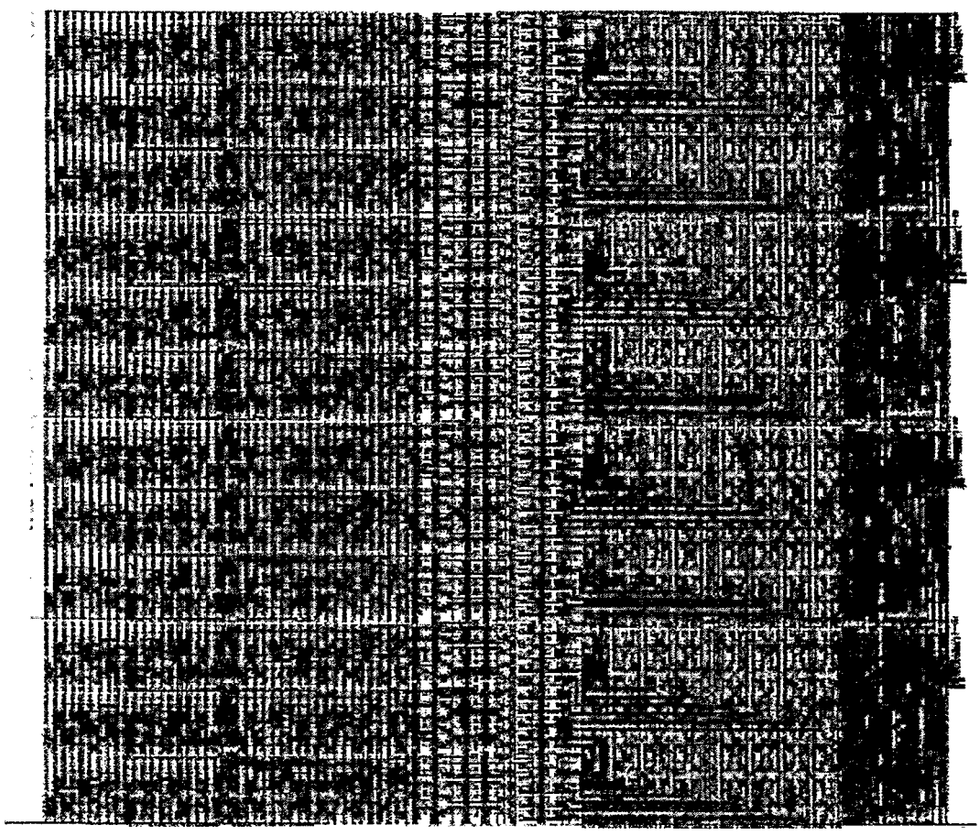

Fig 4.6: *Layout* de los recursos de rutado de las entradas a las LUTs

Las estructuras de rutado fueron finalmente implementadas mediante multiplexores realizados a base de transistores de paso, y configuradas con *bits* multicontexto de dos células de memoria *buffer*. En la implementación preliminar del DMC incluida en el primer *chip* de prueba, mostrada en la figura 4.8, la alimentación de la estructura multicontexto se separó del resto, y las puertas de los transistores que la constituían se dimensionaron por encima del mínimo permitido por la tecnología para así hacerlas resistentes a tensiones de alimentación mayores de la nominal (al estar drenador y fuente más separados el campo eléctrico entre ellos disminuye y con ello la probabilidad de que ocurran problemas de *punchtrough* o de perforación del canal, existiendo así sólamente los límites de perforación del óxido fino de puerta y de rotura de la unión drenador-substrato). Todos los multiplexores, tanto de entrada como de salida, fueron construidos a base de transistores NMOS de paso, directamente configurados por estos *bits* de memoria multicontexto. Los resultados experimentales corroboraron que al alimentar la estructura de memoria (sólo la circuitería multicontexto una vez transferida la información) a tensiones superiores a la de alimentación, típicamente a 5V con una alimentación nominal en el resto del

circuito de 3.3V, los retardos de interconexión obtenidos eran entre dos y tres veces menores que al utilizar la misma alimentación (3.3V). Esto se debe a que al habilitar las puertas de los transistores NMOS con tensiones superiores a $V_{DD}+V_{TN}$, donde V_{TN} es la tensión umbral del NMOS, las tensiones altas son transmitidas perfectamente, al igual que las tensiones bajas.

Sin embargo, esta técnica presenta inconvenientes de índole práctica, ya que es necesario hacer coexistir dos tensiones distintas en el *chip*. Cada vez que se quisiera transferir un contexto sería necesario bajar la tensión de alimentación de la estructura multicontexto a la tensión nominal ya que si no las débiles células de memoria "clásica" alimentadas a 3.3V no serían capaz de escribir datos sobre las células multicontexto alimentadas a 5V. De esta forma, durante el tiempo de la transferencia la temporización sería distinta ya que los retardos de interconexión dependen críticamente de la tensión aplicada en las puertas de los transistores NMOS de los multiplexores. Por otra parte, sería importante que el usuario no tuviera que imponer dos tensiones de alimentación distintas desde el exterior, para lo cual se precisaría una bomba de carga para aumentar la tensión de alimentación en las células seleccionadas, lo cual complicaría el diseño y posiblemente introduciría ruido adicional que afectaría negativamente a la parte analógica.

Como se explicó en el capítulo 2, en el *chip* final se ha utilizado multiplexores NMOS para los recursos de entrada y CMOS para los de salida debido a la alta dependencia del retardo de interconexión en multiplexores NMOS con respecto de la carga conmutada.

4.3.5 El DMC en conjunto y su viabilidad

La figura 4.7 muestra el *layout* completo del DMC. Aproximadamente la mitad izquierda corresponde a los recursos de rutado y la memoria que los configura, y la mitad derecha a la parte funcional y su memoria de configuración. La implementación física del DMC precisó un total de 12000 transistores distribuidos sobre una área de 450 μm por 700 μm, lo cual supone una densidad de integración cercana a los 40000 transitores/mm^2.

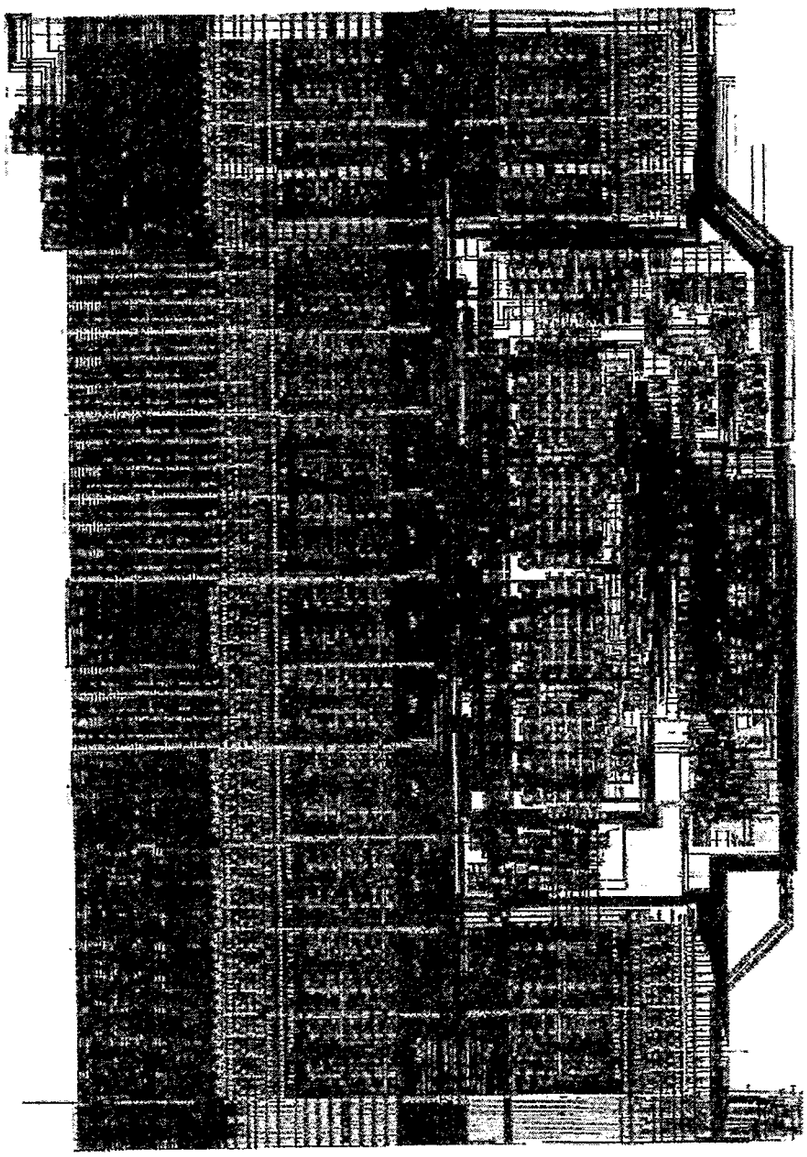

Fig 4.7: *Layout* completo del DMC

La figura 4.8 muestra una fotografía de la implementación preliminar del DMC incluida en el primer *chip* de prueba para comprobar la viabilidad de la reconfiguración dinámica multicontexto y la reutilización de memoria para propósito general.

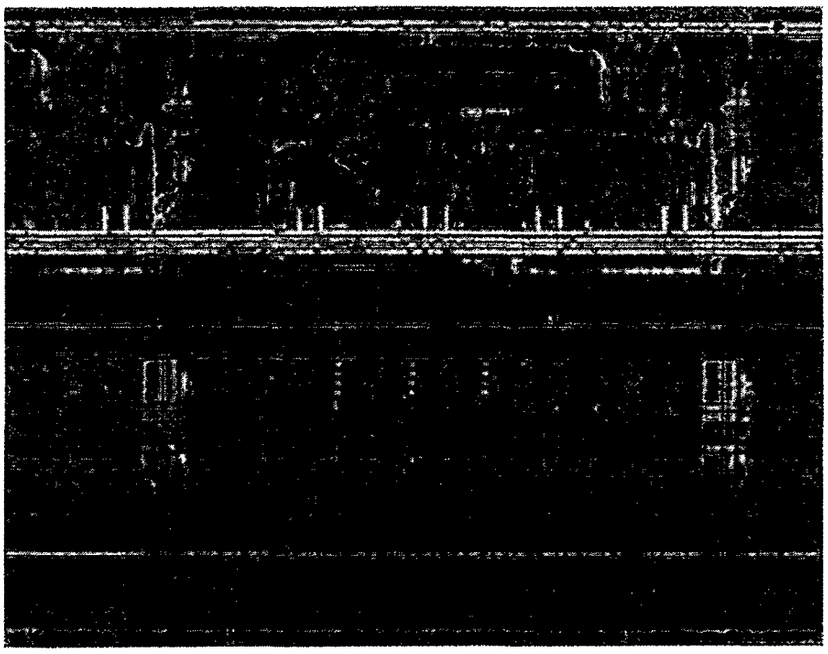

Fig 4.8: Microfotografía de la implementación preliminar del DMC

Ante la gran densidad de integración obtenida, del orden de 40000 transitores/mm^2, es preciso tener en cuenta la potencia disipada por el *chip* y por el DMC en particular, pues podrían aparecer problemas de evacuación de calor que podrían hacer subir la temperatura de la unión a valores excesivos para la integridad del dispositivo, además de afectar muy negativamente a las prestaciones. Sin embargo, el DMC está fundamentalmente frío, ya que en esencia se trata de multiplexores anchos (de hasta 32 a 1) construidos a base de transistores de paso en los cuales sólo una entrada está activa en cada momento, y de células de memoria RAM estática de las cuales sólo una será leída o escrita en cada momento. Un cálculo más sistemático, comprobado posteriormente con simulaciones físicas (HSPICE), corrobora esta apreciación.

Para validar la viabilidad de la estructura multicontexto propuesta es necesario estudiar las proporciones de área empleadas en cada parte del DMC a fin de determinar qué penalización de área supone la introducción de un nuevo contexto *buffer* como se ha propuesto. Si, por ejemplo, el área dedicada a un contexto *buffer* fuera tan grande como el resto del DMC, el interés por esta técnica quedaría en entredicho ya que sería necesaria tanta área como la ocupada por un DMC para poder utilizar el mismo DMC en otro contexto temporalmente incompatible con el actual; en este caso sería más económico gastar esta área en implementar un nuevo DMC monocontexto.

La tabla 4.1 muestra la distribución de áreas obtenida en la implementación física de un DMC.

Función	Area (%)
Estructura de rutado (multiplexores)	34
Bits de configuración "reales"	9
Contexto *buffer* mapeado con su interfaz de memoria	14
Contexto *buffer* extra	9
Cuatro 4-LUTs	20
FFs y su control	4
Interfaz con el microprocesador	3
Conexiones internas y área no utilizada	7

Tabla 4.1: Distribución por áreas de los distintos elementos constitutivos de un DMC

Como era previsible, gran parte del área consumida se gasta en los multiplexores de la estructura de rutado, así como en la memoria de configuración. Las LUTs representan así mismo una fracción importante del área consumida, mientras que los registros programables, con todo su complicado control asíncrono de mecanismos de *set* y *reset*, ocupan un área significativamente pequeña.

Una interesante conclusión que puede sacarse de este análisis es que cada contexto *buffer* adicional representa solamente un 9% del DMC. El contexto mapeado es en realidad un contexto *buffer* que además incluye los decodificadores de memoria, razón por la cual resulta más grande que la memoria de configuración real y que el otro contexto *buffer*. Pero en cualquier caso estos decodificadores eran necesarios para permitir la configuración de la estructura desde el microprocesador, ya que si no habría sido necesario algún otro tipo de interfaz con el microprocesador que permitiera los accesos de escritura de alguna forma.

La consecuencia principal es que aumentando el área del DMC del orden del 10% se obtiene un nuevo contexto que podría duplicar el área efectiva del *hardware* virtual. Por supuesto esta proposición es altamente dependiente de la aplicación: en aplicaciones totalmente paralelas en las que todos los bloques deben funcionar a la vez y a la máxima frecuencia de reloj, esta técnica no resulta rentable; para aplicaciones en las que las distintas operaciones deben ejecutarse en momentos secuenciales de tiempo, el *hardware* virtual podría llegar a ser varias veces mayor que el substrato reconfigurable sobre el que se implementa.

En cualquier caso, si integramos las áreas dedicadas a memorias *buffer* en todos los DMCs del *chip*, el área resultante resulta equivalente a lo que ocuparía un bloque de memoria RAM de

librería de dimensiones equivalentes, de donde se deduce que no es costoso en términos de área distribuir la memoria sobre el *array* de DMCs frente a concentrarla en un bloque únicamente conectado al microprocesador. De esta forma, sin aumento de coste los DMCs pueden disponer de la memoria de configuración para su reconfiguración dinámica y el microprocesador para correr sus programas o guardar sus datos, según sea necesario en cada caso y en cada momento.

4.3.6 Células de entrada-salida

La figura 4.9 muestra una microfotografía de una célula de entrada-salida situada a la izquierda del *array*. A la derecha de esta célula puede pareciarse un DMC completo.

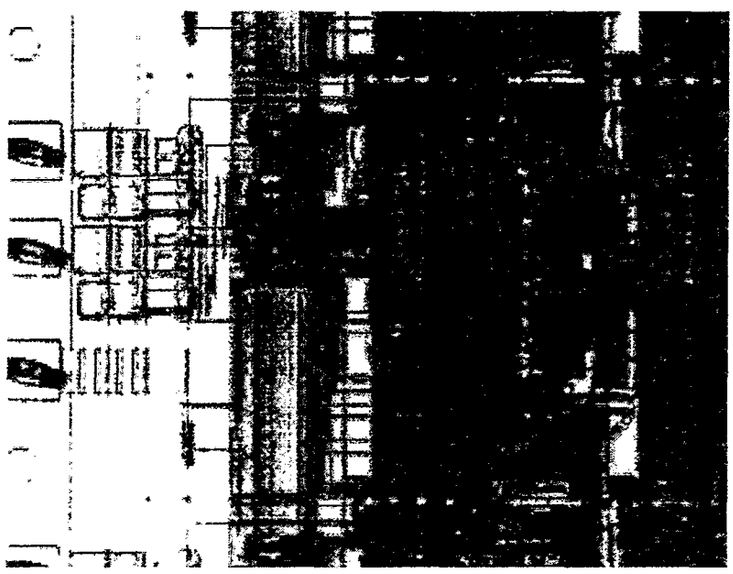

Fig 4.9: Microfotografía de un IOB con un DMC

Las células de entrada-salida incluyen una parte funcional, su estructura de rutado, la memoria que las configura, el *bonding pad* propiamente dicho, y una circuitería dedicada a prevenir problemas de *latch-up* [TRO86] y a minimizar los efectos nocivos de las sobretensiones y las descargas electrostáticas. La microfotografía de la figura 4.10 muestra un detalle de la circuitería de protección del pad.

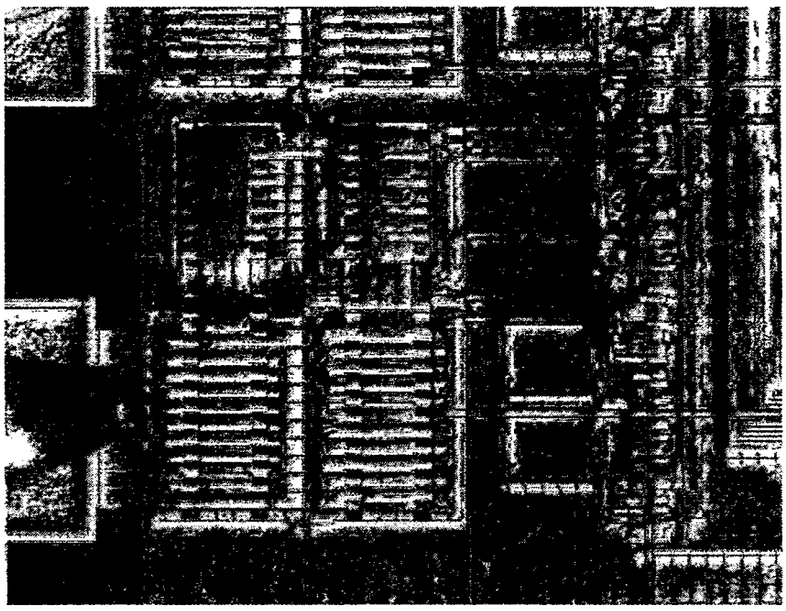

Fig 4.10: Microfotografía del detalle de un pad

4.3.7 IICs

Tanto para los IOBs como para los IICs se aprovecharon bloques de rutado previamente utilizados en el DMC, así como la estructura de memoria asociada. La figura 4.11 muestra el *layout* de un IIC, que básicamente presenta el mismo aspecto que un IOB al que se le ha quitado el *bonding pad*. El anillo de alimentación se cierra a través de los IICs para distribuir convenientemente la alimentación a todo el *chip*.

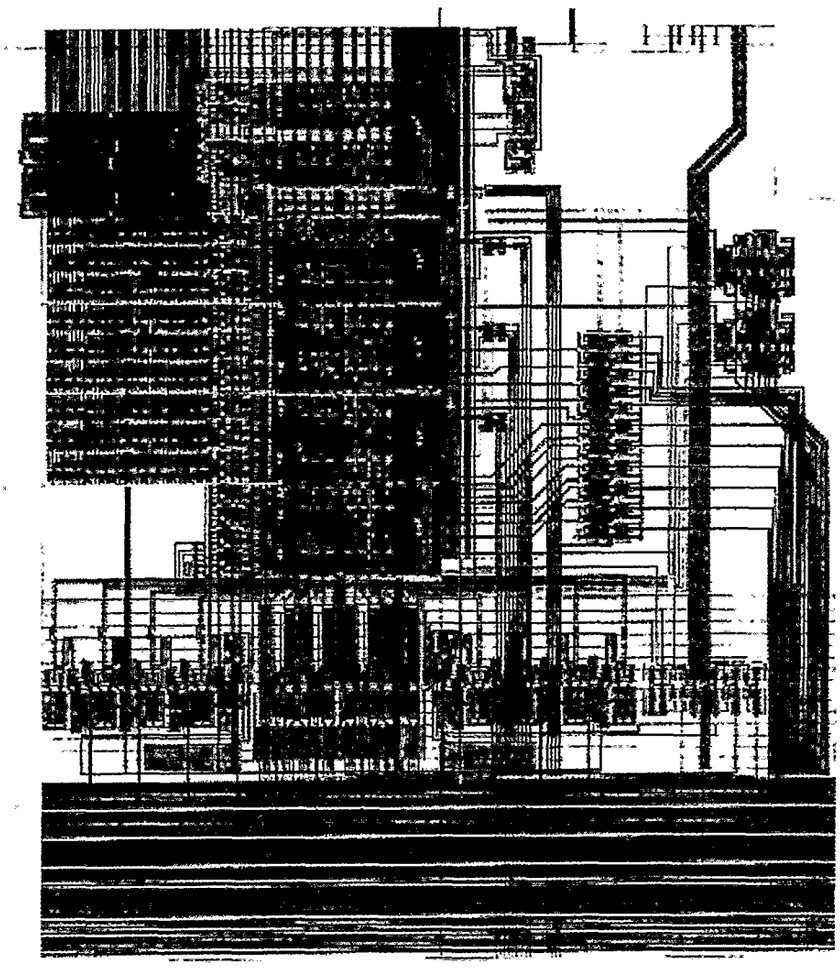

Fig 4.11: *Layout* de un IIC

4.3.8 Prestaciones

Las primeras aplicaciones que ya se han acometido utilizando el *chip* diseñado funcionan a frecuencias de reloj del orden de 20 a 40 MHz típicamente, si bien es posible hacer funcionar estructuras concretas (contadores, registros de desplazamiento) a más de 100 MHz en zonas locales del *chip*. Los retrasos de interconexión varían naturalmente con la carga y la longitud de pista, si bien para interconexiones locales se reducen a unos pocos (entre 2 y 4) nanosegundos. Los retardos típicos de las LUTs se encuentran en el orden de 3 a 5 ns. Las prestaciones son por tanto comparables a las FPGAs coetáneas a nuestra arquitectura fabricadas en tecnologías de 0.5μm y 0.35μm [ALT98] [XIL98] [LUC98].

4.4 Implementación de células analógicas programables

Aunque la implementación física de las células digitales programables no resulta sencilla como se ha puesto de relieve en la sección anterior, su complicación y su impacto en la calidad del dispositivo es en cualquier caso menor que en el caso de las células programables analógicas. Para estas células la implementación física resulta igual de importante que el mismo diseño funcional a base de transistores, y resulta vital tener en cuenta la implementación física en el momento del diseño mismo de sus bloques constitutivos. Más importante aún, gran parte de los efectos producidos por una mala implementación física, tales como los problemas de apareamiento de dispositivos (*matching*) o las figuras de ruido, no se aprecian en las simulaciones normales en las distintas esquinas del proceso, y resulta necesario realizar simulaciones especiales y consideraciones teóricas para cuantificarlos debidamente.

La implementación física influye fundamentalmente en el desapareamiento (*mismatching*) de dispositivos y en las figuras de ruido. El desapareamiento es una medida del parecido entre dos transistores, resistencias o condensadores que por diseño deben ser iguales. Este parecido resulta de vital importancia en casos como el par diferencial de entrada o las cargas activas en cascodo de un amplificador operacional, ya que de él depende de forma crítica el *offset* del amplificador. Análogamente, la variación en el valor de las resistencias en el divisor resistivo de un DAC determina la precisión del mismo. Así mismo, en los circuitos de capacidades conmutadas suele ser de vital importancia que los valores de los condensadores utilizados en secciones diferenciales o bicuadráticas coincidan perfectamente.

La magnitud del desapareamiento de dispositivos suele ser de difícil cuantificación. Normalmente la fábrica de semiconductores suele realizar un estudio estadístico sobre una muestra de pares de dispositivos de distintos tamaños dibujados de manera idéntica. El apareamiento óptimo suele conseguirse para anchuras de polisilicio de unas diez veces la anchura mínima, aunque es muy dependiente del proceso y del tipo de dispositivo. En cualquier los dispositivos de mínima dimensión, tanto transistores como resistencias, suelen presentar un desapareamiento significativamente mayor (típicamente más de un orden de magnitud) que los diez veces mayores, debido a que las irregularidades de los bordes, que suelen ser las más importantes, son más significativas cuanto menor es la anchura. La relación entre anchura y bordes debe mantenerse alta para que el error relativo sea pequeño, aunque existe una anchura óptima ya que longitudes demasiado grandes introducirían grandes distancias entre los dos dispositivos, y los gradientes del proceso producirían diferencias significativas entre ambos.

Casi todas las magnitudes a las que afecta el desapareamiento suelen variar en cualquier caso de forma gradual en alguna de las direcciones del plano de la oblea. Sea cual sea la función bidimensional que expresa el valor de una cierta magnitud en un determinado punto, es habitual considerar únicamente el primer término de su desarrollo en serie de Taylor, lo que equivale a considerar que el gradiente de la magnitud en cuestión es lineal. Esta suposición suele ser bastante aproximada en la mayoría de los casos, sobre todo si el área total ocupada por los dispositivos considerados es relativamente pequeña (en un radio de unas 100 μm). Por eso es habitual descomponer los conjuntos de dispositivos críticos en varios trozos que se intercalarían unos con otros, para de esta hacer que el gradiente afecte de forma equivalente a los distintos elementos del conjunto. Más aún, si se hace que el centroide de los distintos pedazos en que se descompone uno de los dispositivos coincida con el del otro, los efectos de un gradiente lineal se cancelarían perfectamente y los dos dispositivos globales serían idénticos. A esta técnica se la conoce como *centroide común* [MAL94], y suele ser de gran importancia para la implementación de *arrays* de condensadores en circuitos diferenciales de capacidades conmutadas, en pares diferenciales, o en DACs realizados a base de fuentes de corrientes.

Mediante la descomposición de dispositivos en varios pedazos se consigue no sólo una minimización de los efectos de los gradientes de proceso, sino una mayor tolerancia a los errores más aleatorios puntualmente tales como cargas en el óxido de puerta de los transistores o grumos en el polisilicio de las resistencias. El efecto de este tipo de problemas de naturaleza aleatoria se suele dividir por el número de secciones en las que se divide el dispositivo en cuestión.

El segundo problema para el que una buena implementación física es crítica es el ruido. Por una parte, el tamaño concreto del dispositivo, incluyendo los tamaños de las zonas de drenador y fuente en los transistores, determina su ancho de banda de ruido [MOT93], que en el caso de un par diferencial de un amplificador influye directamente en las figuras de ruido del dispositivo. Es importante dimensionar el ancho de banda de ruido en función del ancho de banda proyectado para la arquitectura, y en general es conveniente no utilizar bloques constitutivos que admitan un ancho de banda mucho mayor que el que se pretende soportar para la arquitectura global. Así por ejemplo, en un típico amplificador operacional con compensación por polo dominante implementada con un condensador en algún punto de la etapa de salida, mantener el ancho de banda del par diferencial de entrada órdenes de magnitud por encima del ancho de banda proyectado para el amplificador global no hace sino aumentar la magnitud del ruido obtenido a la salida, debido fundamentalmente al ruido térmico generado en el par de entrada.

Aparte de las consideraciones sobre ruido intrínseco debido a efectos térmicos, la implementación física debe minimizar los problemas de inyección de ruido desde la parte digital o desde otros bloques analógicos. Para ello suele ser necesario interponer grandes anillos o barreras de guarda (largas zonas de contactos al substrato) que colecten las corrientes espúreas que puedan venir por el substrato provenientes de otros bloques. También es habitual utilizar grandes zonas de metal conectadas a masa para apantallar estructuras especialmente sensibles al ruido, especialmente si se dispone de tres metales como en nuestro caso: el metal inferior se utiliza para implementar los bloques constitutivos de la célula, el metal intermedio para apantallarlos, y el superior para interconectarlos.

La implementación física debe cuidar así mismo del *crosstalk* o interferencia entre señales no relacionadas, interponiendo los apantallamientos necesarios entre líneas largas de señales críticas.

Finalmente, es importante tener en cuenta los posibles problemas de *latchup* que pueden aparecer en geometrías especialmente grandes y más concretamente en las etapas de potencia de los amplificadores. Por eso y por consideraciones de ruido suele resultar de gran importancia encerrar estos dispositivos en anillos de guarda polarizados a las tensiones de alimentación.

4.4.1 Las células analógicas en sistemas mixtos

Los sistemas en que deben coexistir células analógicas con células digitales presentan problemas adicionales al del diseño general de células analógicas puras. Estos problemas tienen que ver con la tecnología, que habitualmente está optimizada para arquitecturas digitales, y con el ruido inyectado desde la parte digital.

Nuestro mayor problema de cara a diseñar un dispositivo de tipo FIPSOC radica en que fue necesario utilizar un proceso CMOS de bajo coste optimizado para operación digital. Aunque existen tecnologías que disponen de pozos gemelos, capas epitaxiales enterradas, dispositivos bipolares, o una segunda capa de polisilicio que permite implementar condensadores de alta linealidad y gran valor, su coste es significativamente más alto que el de las tecnologías CMOS digitales equivalentes.

Al no disponer de una segunda capa de polisilicio, la implementación de condensadores de alto valor (en el rango de picofaradios) se convierte en un problema. Las posibles alternativas son: a) utilizar capas de metal intercaladas, que aunque producen condensadores de gran linealidad

requieren mucha área debido al bajo valor de la capacidad parásita entre metales; además, debido a que se trata de una capacidad parásita, su valor no suele estar garantizado y menos aun su apareamiento; y b) utilizar puertas de transistores, que aunque su valor está mucho más controlado y suele ser relativamente alto, su linealidad se mala ya que depende del punto de polarización al tratarse de una capacidad MOS. La conclusión es que es preferible utilizar condensadores MOS a base de puertas de transistores siempre que sea posible por cuestiones de optimización de área, aunque sólo en aplicaciones en las que la linealidad no sea un problema. Por ejemplo, es posible utilizar este tipo de condensadores para la compensación por polo dominante de la etapa de salida de un amplificador o para la circuitería de muestreo de un comparador *sample & hold*, aunque no para un filtro construido a base de secciones gm-C.

Otro problema de la utilización de un proceso no optimizado para la implementación de circuitos analógicos es la gran dispersión de los valores absolutos de las magnitudes del proceso. Por ejemplo, la resistividad del polisilicio, que influye directamente sobre el valor absoluto de las resistencias implementadas con este material, tiene una dispersión mayor al ±33%. De la misma forma existen grandes variaciones (dentro de este orden) en las resistividades equivalentes de los transistores MOS, con el agravante de que no suele ser posible garantizar ningún tipo de correlación entre el PMOS y el NMOS. Por ejemplo, si se utiliza una resistencia de polisilicio en una fuente de corriente que polarice un amplificador, la corriente de polarización podría tener una variación mayor al ±30%, lo que dificulta el diseño pues debe verificarse su funcionamiento en todo el rango de corrientes.

Finalmente, en sistemas mixtos resulta de gran importancia garantizar un buen aislamiento entre las arquitecturas digitales y las analógicas. Para ello es habitual separar las alimentaciones de ambos subsistemas, disponiendo de anillos separados con *pads* de alimentación distintos. También suele ser beneficioso interponer grandes anillos de guarda entre ambos dominios para minimizar en la medida de lo posible la inyección de ruido por el substrato, efecto de gran importancia en las tecnologías submicrónicas conforme la longitud de puerta y la tensión de alimentación se reducen a la vez que la frecuencia de operación de la parte digital aumenta [GHA94].

Un último punto de inyección de ruido desde la parte digital es la entrada digital a células de señal mixta tales como convertidores o células programables. Si estas señales digitales se conectan directamente a partes sensibles de la estructura analógica las capacidades parásitas entre puerta y drenador de los transistores analógicos podrían inyectar ruido en las señales de salida, especialmente si el subsistema digital funciona a alta frecuencia o si las señales digitales

presentan una gran pendiente (*slew rate*) de transición. Para minimizar este efecto es posible interponer inversores alimentados desde la parte analógica que filtren el ruido existente en las señales digitales antes de alcanzar los transistores analógicos. No debe olvidarse que las células programables analógicas deben considerarse como puramente mixtas ya que se configuran mediante señales digitales que normalmente provienen de puertos o de estructuras de memoria conectadas al bus de direcciones y datos del microprocesador que habitualmente produce una gran cantidad de ruido de alta frecuencia.

4.4.2 Amplificadores operaciones diferenciales y balanceados

Para el trazado del amplificador operacional diferencial balanceado propuesto en el capítulo 2 debe ponerse el mayor cuidado en la implementación de los pares diferenciales, especialmente el que actúa sobre el modo diferencial, pues de él depende el *offset* en modo diferencial a la salida. El par diferencial usado para la realimentación en modo común también debe cuidarse, aunque el *offset* en modo común a la salida no es tan importante ya que será eliminado en la siguiente etapa de amplificación, ya que éstas rechazan el modo común. Los transistores de los pares diferenciales se han troceado en varios pedazos para distribuir los efectos graduales del proceso, y así minimizar sus efectos en cuanto a la diferencia entre ambos dispositivos. Lo mismo se ha hecho con la estructura en *folded cascode*, en la que el apareamiento entre los dos transistores que constituyen las cargas activas es igualmente crítico.

A partir de los datos de caracterización de la tecnología proveídos por la fábrica se concluye que el apareamiento se maximiza cuando se utilizan longitudes de puerta sensiblemente mayores a la longitud mínima. Los elementos más críticos, como los pares diferenciales, utilizan longitudes de puerta grandes, normalmente hasta ocho o diez veces la longitud mínima. Los elementos menos críticos, como la circuitería de inhabilitación del amplificador para cuando éste se apague (modo de *power-down*), se realizan con la mínima longitud de puerta.

4.4.3 Secciones de amplificación

La figura 4.12 muestra el *layout* detallado de la sección de amplificación completa, donde además del amplificador operacional diferencial se pueden observar las estructuras de

resistencias apareadas, el control de *offset* en tiempo continuo y la circuitería de corrección de CMRR. Es posible apreciar cómo la estructura está implementada fundamentalmente con el metal inferior (en azul oscuro) y apantallada con el metal intermedio (azul claro), mientras que las conexiones entre los bloques y con las señales de los *bits* digitales de programación se realizan con el metal superior (en gris azulado) siempre sobre zonas apantalladas. La capa intermedia sirve de esta forma de jaula de Faraday para proteger los elementos más sensibles al ruido dentro de la estructura.

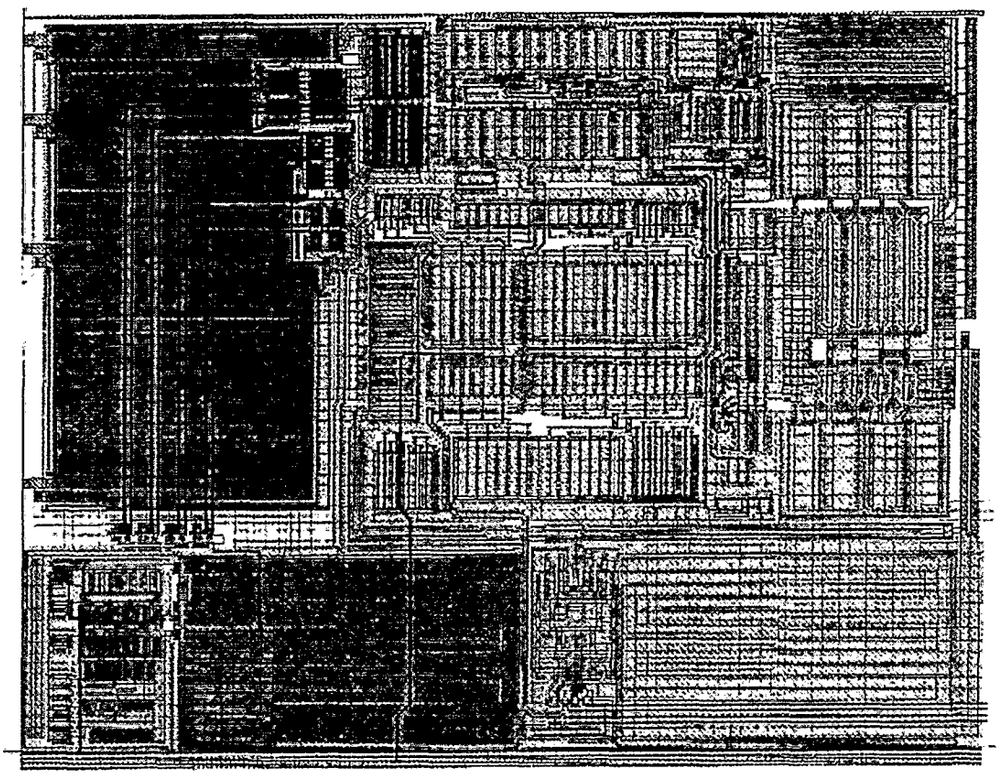

Fig 4.12: *Layout* de la sección de amplificación completa.

En la figura 4.13 se puede apreciar una fotografía de la sección de amplificación basada en el amplificador diferencial balanceado. La fotografía corresponde a la implementación preliminar probada en el segundo *chip* de prueba, que luego fue utilizada tal cual en el primer FIPSOC completo.

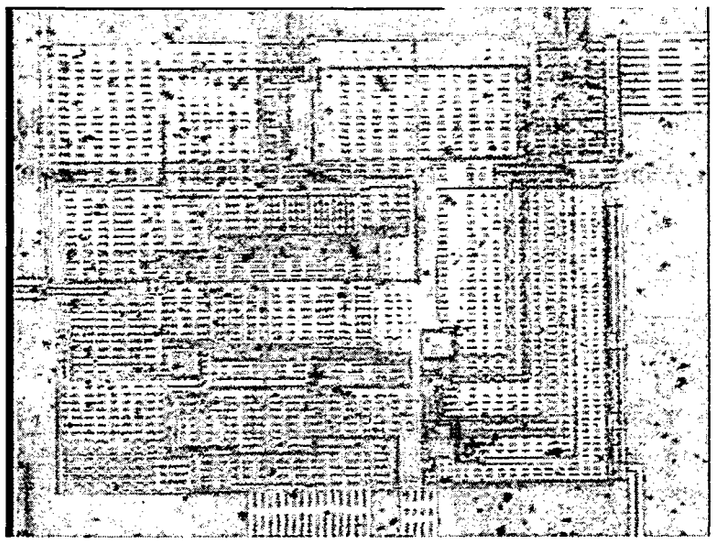

Fig 4.13: Microfotografía de la sección de amplificación diferencial completa

Finalmente, la figura 4.14 muestra resultados reales de caracterización medidos sobre la sección de amplificación completa. Las distintas formas de onda representadas en la figura corresponden a los siguientes casos:

a) Señal diferencial sinusoidal de entrada con ruido superpuesto en modo común de alta y baja frecuencia. En la parte inferior se aprecia esta señal de entrada en azul y rojo, mientras que la salida sin ruido se muestra en las trazas superiores verde y negra. El factor de amplificación elegido es 1. La salida distorsionada que puede distinguirse en el tercer cuadro a la derecha del origen se debe a que una de las entradas diferenciales baja por debajo de cero. Es interesante observar que aunque la salida está distorsionada la tensión común a la salida se mantiene debido al funcionamiento del bucle de realimentación de modo común.

b) Este caso es análogo al anterior con un factor de amplificación de 15. También se ha probado a conectar varias secciones en cascada para obtener mayores amplificaciones.

c) En este caso se utiliza la entrada de modo común a la salida para inyectar una señal dinámicamente en vez de fijarla a un nivel estable. La traza roja inferior muestra una de las entradas diferenciales, a las que se aplica una sinusoide (el ruido que puede apreciarse se rechaza al ser común a ambas entradas). Las salidas, representadas con las trazas superiores verde y negra siguen la suma de la sinusoide de entrada amplificada y la otra sinusoide más lenta inyectada por la entrada de realimentación de modo común, representada con la traza azul.

d) En este caso la entrada diferencial es la misma sinusoide y el nivel de modo común a la salida está fijo a una tensión media entre alimentación y masa. El microprocesador se usa para producir un barrido en el *byte* que controla el nivel de *offset* diferencial a la salida. De esta manera, la tensión diferencial obtenida es la suma de un diente de sierra (producido por el barrido lineal del control de *offset*) y la sinusoide amplificada desde la entrada.

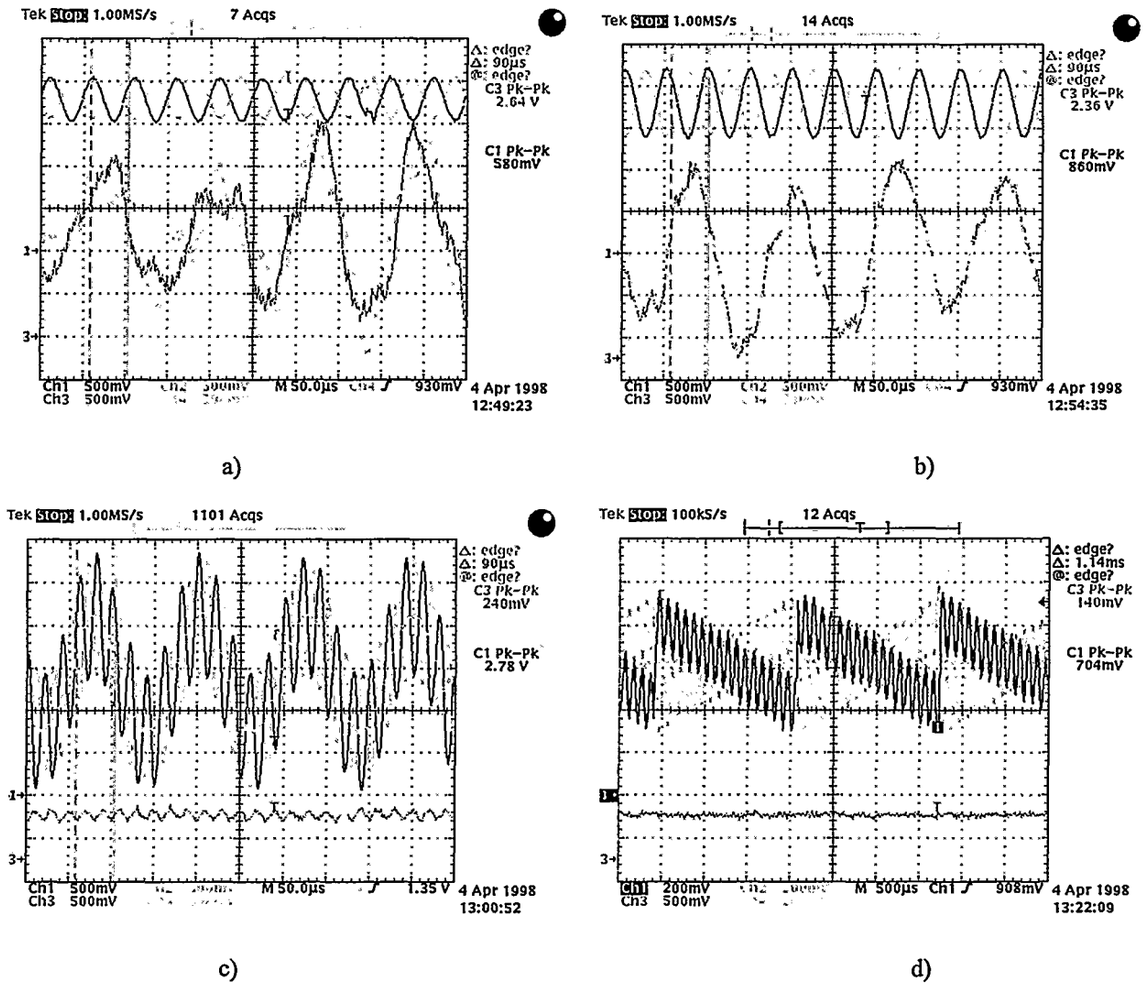

Fig 4.14: Resultados de caracterización de la sección diferencial de amplificación

4.4.4 Convertidores

La figura 4.15 muestra una microfotografía de la implementación preliminar de la estructura de convertidores digital-analógico que sirve de base al bloque de conversión del CAB. El *layout*

final utilizado en el primer FIPSOC, mostrado en la figura 4.16, fue ligeramente distinto para optimizar las condiciones de apareamiento de los tramos del divisor resistivo, con el objetivo de mejorar la linealidad del conversor.

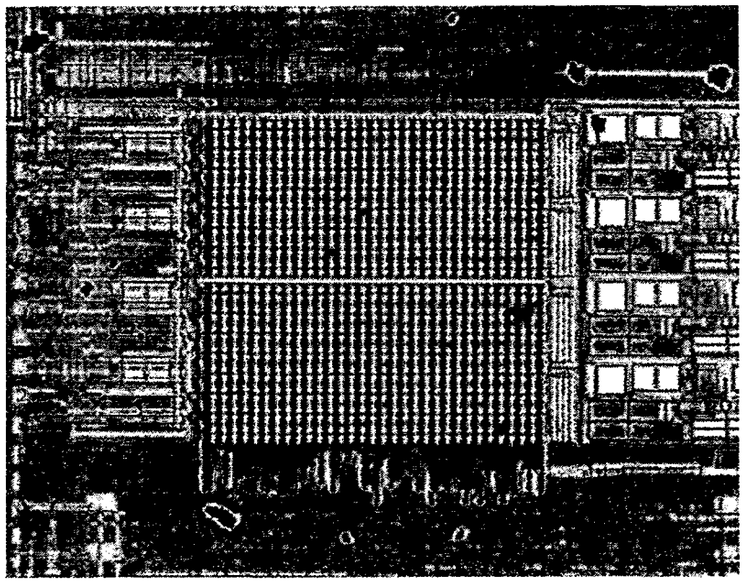

Fig 4.15: Microfotografía de la implementación preliminar del DAC del primer *chip* de prueba

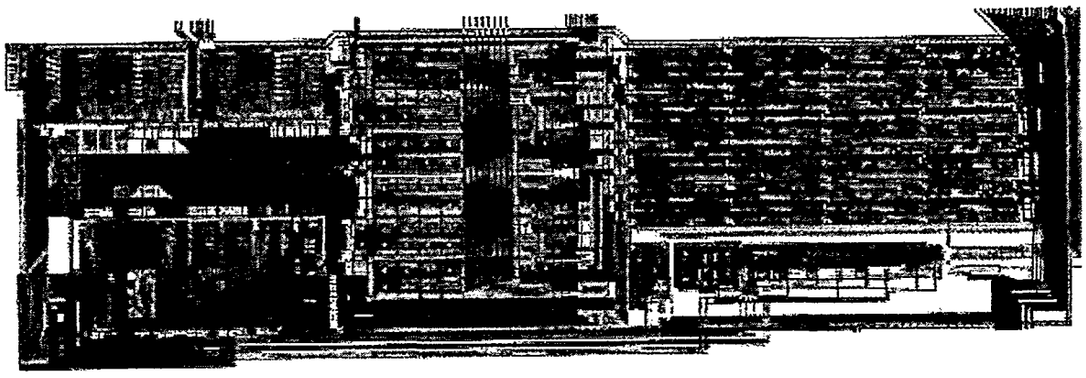

Fig 4.16: *Layout* de la parte *full custom* del bloque de conversión

El bloque de conversión se implementó de forma mixta: el *array* resistivo, sus decodificadores, los comparadores de muestreo (*sample & hold*), y ciertas partes del control digital tales como multiplexores anchos de bus, registros, etc., fueron implementados en *full custom* a nivel de transistor. El resto de la lógica de control, especialmente la dedicada a la máquina de estados que

controla el algoritmo de aproximaciones sucesivas de los ADCs, fue implementada mediante células estándar de librería. La validación del sistema se realizó a base de simulaciones digitales utilizando modelos (digitales) de comportamiento de los bloques analógicos.

4.4.5 Comparadores

Los comparadores de tiempo continuo pueden apreciarse en la parte inferior de la figura 18. Para su implementación se debieron hacer las mismas consideraciones en cuanto a los pares diferenciales y cargas activas que para el caso de los amplificadores, ya que su estructura se basa en amplificadores operacionales de transconductancia típicos (OTAs) [LAK94].

4.4.6 Rutado de señales analógicas

Como ya se comentó en el capítulo dos, el objetivo de la parte analógica de nuestro FIPSOC ejemplo no era proveer un substrato reconfigurable análogo a una FPGA en el que pudiera implementarse cualquier tipo de aplicación analógica, sino más bien el permitir al usuario disponer de un *front-end* analógico con el que poder acondicionar y convertir señales. Partiendo de esta filosofía, el rutado de las señales analógicas no constituye un problema de la magnitud de su equivalente digital, ya que la rutabilidad no es realmente una de las preocupaciones en el diseño.

En cualquier caso, el CAB permite una cierta flexibilidad a la hora de interconectar señales de referencia o incluso al seleccionar las señales activas a convertir. Estas selecciones se realizan mediante multiplexores analógicos (construidos a base de conmutadores CMOS) a cuyas entradas se encuentra un gran número de señales en principio no relacionadas unas con otras. Recordemos que las señales activas de salida de las mismas secciones de amplificación, así como las salidas de los DACs, pueden usarse como referencias de otros DACs o incluso de otras secciones de amplificación. Esta funcionalidad implica la necesidad de transportar señales provenientes del divisor resistivo de referencia a lo largo de una gran distancia (más de dos milímetros) junto a las señales de salida activa de las secciones de amplificación, resultando vital garantizar un buen aislamiento entre ellas.

Si bien para apantallar el ruido en general es bueno utilizar una capa de metal a modo de jaula de Faraday, si lo que se pretende es aislar dos pistas de metal que corren paralelas una gran distancia esta técnica resulta perjudicial ya que la reflexión de los campos electromagnéticos contra el metal pantalla hace que crezca el acoplamiento entre las líneas. En su lugar es preferible intercalar una línea más del mismo metal que las lineas activas pero conectada a masa, quizás entonces apantallando el conjunto con el metal superior.

Ahora bien, el coste de área de esta solución es cuantioso: si se intercala una línea más entre cada dos líneas activas, el área necesaria se duplica. En su lugar lo que se hizo fue intercalar líneas de masa entre conjuntos de señales relacionadas entre sí. Por ejemplo, las dos salidas diferenciales de una sección de amplificación ya están relacionadas entre sí, así que no hay porqué intentar minimizar el acoplamiento entre ellas. Lo mismo puede decirse de las nueve señales provenientes del divisor resistivo del bloque de referencia, que en cualquier caso están relacionadas de forma lineal. Sin embargo, la salida diferencial de una sección de amplificación debe aislarse tanto como sea posible de las señales de referencia, pues si no una sección de amplificación podría producir interferencias sobre la referencia de otra sección de amplificación no relacionada con la primera.

La figura 4.17 muestra este concepto. La fotografía corresponde a un multiplexor analógico de dieciséis a uno que selecciona una señal de referencia para un conjunto de secciones de amplificación. Los buses verticales en el metal inferior situados a la izquierda transportan las posibles señales de referencia, mientras que en el metal superior se realizan las conexiones a las entradas del multiplexor, que ocupa la mitad derecha. Las primeras cuatro líneas de referencia son las salidas de los DACs, que al no estar relacionadas en principio unas con otras deben apantallarse individualmente mediante líneas verticales en el mismo metal inferior conectadas a masa. La masa se conecta mediante una ristra de contactos al metal intermedio que cubre a franjas horizontales todos los buses. Sin embargo, a la izquierda de estas señales puede verse un par de señales no aisladas una de otra, ya que son salidas diferenciales de una sección de amplificación.

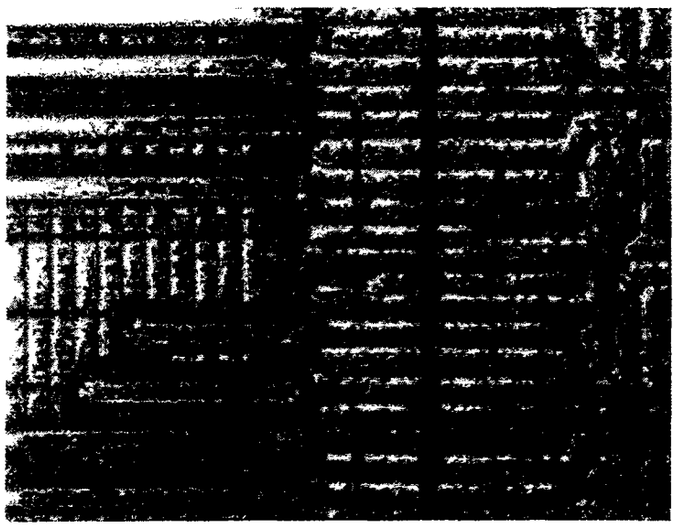

Fig 4.17: Microfotografía de un multiplexor analógico de rutado de referencias

4.4.7 El CAB en su conjunto

La figura 4.18 muestra finalmente una microfotografía del CAB en su conjunto. Para su implementación global se procedió a la unión de todos los bloques constitutivos mencionados anteriormente, de forma que las señales de interconexión recorriesen una distancia mínima y estuviesen apantalladas lo más posible.

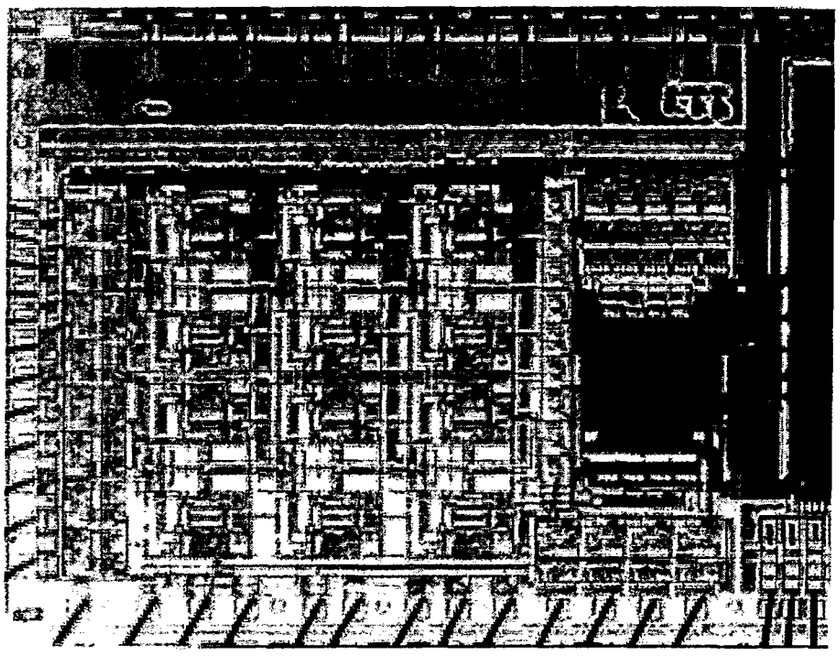

Fig 4.18: Microfotografía del CAB completo

El CAB ocupa 2.2 mm x 3.4 mm, e incluye su propio anillo de alimentación con sus *pads* particulares. La parte derecha se encuentra bastante ligada a la zona destinada a las células estándar, ya que parte del control del algoritmo de aproximaciones sucesivas del ADC no está implementada en *full custom*, si bien existen barreras de guarda entre los dos anillos de alimentación (a la derecha en la fotografía). También existen grandes barreras de guarda situadas entre la parte superior del CAB y la parte inferior de la FPGA, ya que precisamente en esta parte se encuentran los *drivers* de reloj de la FPGA que podrían llegar a funcionar a 100 MHz. Las barreras de guarda se implementan a base de contactos al substrato y pozos N+ conectados a alimentación intercalados, para así actuar como colector de corrientes de ruido provenientes del substrato.

La figura 4.19 muestra un detalle en el que puede apreciarse la esquina inferior izquierda del bloque conversor, algunas secciones de amplificación, los *buffers* de salida de los DACs, y algunos multiplexores conectados a las señales utilizadas como referencias analógicas.

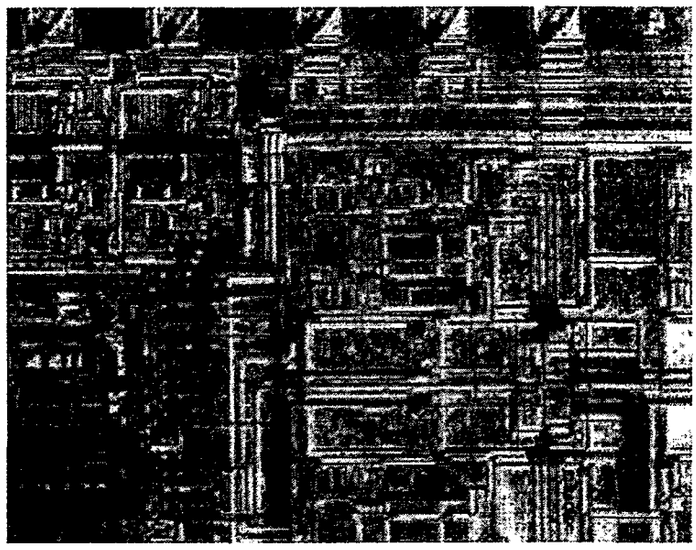

Fig 4.19: Microfotografía de un detalle del CAB

4.5 Implementación del interfaz con el microprocesador

Como ya se ha mencionado con anterioridad, el microprocesador y su interfaz han sido implementados mediante células estándar de librería a partir de síntesis de descripciones Verilog™. El mayor problema planteado por esta metodología radica en la necesidad de mezclar

las células *custom* que constituyen la mayoría del *chip* con células estándar de librería de las que sólo se dispone de una descripción lógica de comportamiento y de la caracterización de su temporización. El proceso a seguir para la correcta validación del diseño pasa por realizar la misma caracterización que se realizó en la fábrica para describir el comportamiento de las células de librería, para de esta forma conseguir una descripción compatible que permita una simulación conjunta. De esta manera, se escribieron modelos lógicos de los distintos subsistemas de la FPGA y del CAB para poder realizar una comprobación lógica del comportamiento global del *chip* y de la temporización de los interfaces, de cara a su validación final para la fabricación.

Para la correcta implementación física de ambos dominios fue necesaria una gran cantidad de trabajo manual de trazado de pistas, especialmente en la conexión entre determinadas células estándar y la parte *custom*. El riesgo introducido era grande, ya que esta metodología mixta implicaba la utilización conjunta de herramientas distintas como Silicon Ensemble™ (Cadence™) para el *placement & routing* y Virtuoso™ para el resto del *layout custom* manual, debido a lo cual resultó imprescindible realizar un LVS (*Layout vs Schematic*) final que comprobase todas las modificaciones manuales.

La figura 4.20 muestra la esquina inferior derecha del primer FIPSOC en la que se sitúa el microprocesador, los distintos bloques de memoria y todo el interfaz con la FPGA y con el CAB. En total este bloque ocupa una área de 2.5 mm x 2.4 mm, y contiene 46 *pads* entre los que se cuenta un oscilador para un cristal de cuarzo, un PLL, el interfaz completo para memoria externa de código y datos, un puerto bidireccional de propósito general, y *pads* de alimentación. El microprocesador han sido probados con éxito a frecuencias de reloj de 48 MHz.

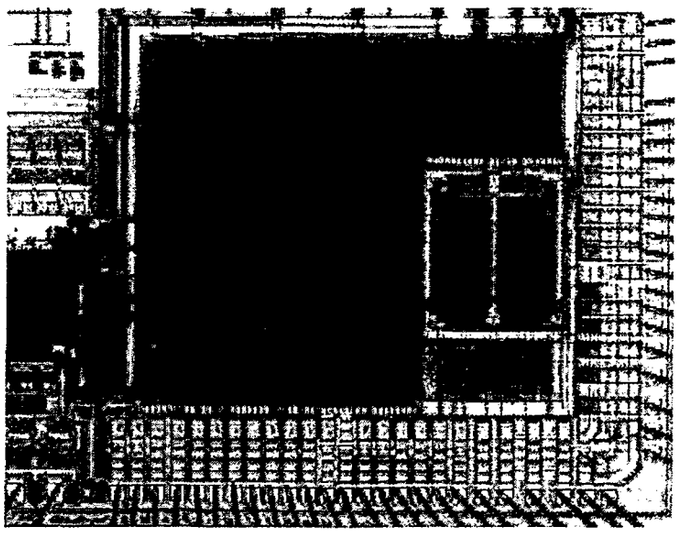

Fig 4.20: Microfotografía del microprocesador y su interfaz

4.6 Test

La estrategia propuesta para el *test* de nuestro FIPSOC se vale de la reconfiguración dinámica simultánea introducida en el capítulo dos. A base de vectores de *test* se simula un bus de memoria para hacer correr un programa en el microcontrolador que configura el *array* de DMCs de forma simultánea con la misma configuración. Para ello se activa toda la máscara de selección de DMCs a la vez, de tal forma que al escribir la configuración en el visor de memoria se escribe en realidad en el espacio correspondiente de todos los DMCs seleccionados. De esta manera se mapea una aplicación de tipo sistólico que comunica datos en direcciones vertical y horizontal, moviendo a la vez un gran número de canales de rutado y nodos internos a la lógica programable, además de cambiar los datos de la memoria de configuración. A continuación se realizan configuraciones parciales incrementales que cambian las entradas o salidas a los canales de rutado, de forma que no es necesario reconfigurar los DMCs completos sino sólo una pequeña parte de los recursos de rutado o del bloque funcional cada vez para mover una gran cantidad de nodos internos.

Dado que toda esta operación se realiza mediante accesos a memoria, el microprocesador queda probado casi en su totalidad sin más que correr este programa, aunque el resto de los periféricos del sistema necesitan unos cuantos vectores de *test* adicionales para probar su funcionamiento de forma específica.

El *test* de la parte analógica se realiza de forma específica también a base de escrituras y lecturas desde el microprocesador, el cual realiza labores de configuración dinámica de los factores de amplificación y del rutado interno de las señales, además de provocar en determinados momentos los comandos de conversión y de llevar a cabo la lectura de los datos digitales de salida.

4.7 Resumen

En este capítulo se ha presentado la implementación física del primer demostrador de una arquitectura tipo FIPSOC así como de los *chips* de prueba previos que se fabricaron para comprobar las células individualmente. Se ha discutido la metodología seguida en la implementación física de una estructura tan ecléctica como la propuesta, la cual supone un reto

ya que implica la utilización conjunta de los flujos de diseño correspondientes a *full custom* y células estándar. Se ha discutido los resultados de implementación y se ha presentado las técnicas de *layout* utilizadas, de gran importancia en el caso particular de las células analógicas, para las cuales se ha presentado resultados de caracterización real. Los resultados de área obtenidos permiten comprobar la viabilidad de las arquitecturas propuestas en el capítulo dos.

CAPÍTULO 5

APLICACIONES

5.1 Introducción

En este capítulo intentaremos apuntar algunas aplicaciones específicas de los aspectos más novedosos que presentan las arquitecturas tipo FIPSOC, como son la reconfigurabilidad dinámica de la FPGA, la interacción entre el *hardware* y el *software* a base de posiciones de memoria del microprocesador, y la disponibilidad de células analógicas complementarias.

Algunas de estas aplicaciones han sido introducidas por potenciales usuarios de este tipo de *chips*, mientras otras han sido ideadas por sus mismos co-desarrolladores, principalmente el autor, Ignacio Lacadena y José María Insenser (SIDSA), Juan Manuel Moreno y Jordi Madrenas (UPC), y Vicente Baena, Miguel Angel Aguirre y Antonio Torralba (Universidad de Sevilla). En cualquier caso es de esperar la aparición de nuevas aplicaciones en el futuro cercano debido al gran interés que este trabajo está suscitando entre la comunidad científica internacional.

Es importante notar que aunque el *software* desarrollado en el marco de este proyecto para la utilización del *chip* en su conjunto, incluyendo la FPGA, la parte analógica y el microprocesador, ha llegado a un estado de relativa madurez, el *software* necesario para el fácil manejo de la reconfiguración dinámica no ha sido todavía acometido en profundidad. Por tanto, la

implementación de aplicaciones que utilizan la reconfiguración dinámica se ha realizado de forma manual utilizando las herramientas "estáticas" para el flujo de diseño estándar. El tipo de *software* que sería necesario para explotar verdaderamente todas posibilidades del *chip* a este respecto constituye una de las principales líneas futuras de trabajo como será mencionado en el capítulo 6.

5.2 Sistemas de desarrollo integrados

La primera gran ventaja de una arquitectura FIPSOC como la que se ha estudiado frente a una posible solución mixta a base de microcontroladores programables y FPGAs existentes en el mercado radica en la integración de sus bloques y de la metodología a seguir durante el diseño de aplicaciones que precisan de los tres dominios implicados, esto es, *hardware* digital, *hardware* analógico y *software*. En efecto, a la hora de diseñar un sistema mixto en el que parte de la aplicación corre en un microprocesador y el resto se implementa a base de *hardware* programable o incluso discreto, el usuario debe seguir dos metodologías completamente distintas utilizando herramientas diferentes creadas y mantenidas por distintos proveedores. Aunque en el microprocesador el programa puede correr paso a paso y puede pararse con *breakpoints*, esta operación discretizada no puede extenderse al *hardware*, a la vez que aunque el *hardware* puede simularse lógicamente no suele ser posible extender la simulación al microprocesador ya que rara vez se dispone de su modelo lógico compatible con los modelos del resto del circuito (y aunque se tuviera la simulación de la ejecución de un programa real resultaría tediosa y precisaría muchos recursos). Sólo es posible hacer pruebas en tiempo real de todo el sistema en conjunto, lo que hace difícil su depuración.

En cambio, el sistema de desarrollo creado para un sistema de tipo FIPSOC integra ambas metodologías en una herramienta única y consistente que permite diseñar y programar aplicaciones mixtas. El sistema de desarrollo se basa en un emulador integrado que permite realizar co-emulaciones *hardware-software* concurrentes en tiempo pseudo-real.

Como se explicó en el capítulo 3, nuestra primera realización práctica de sistema FIPSOC incorpora una serie de circuitería dedicada al sistema de desarrollo del microcontrolador. Mediante ella es posible programar *breakpoints* que detendrían el programa al ejecutar una determinada instrucción o al intentar acceder a una cierta posición de memoria, a la vez que dispone de una opción de ejecución paso a paso. De forma análoga el microprocesador puede

controlar la máquina de generación de relojes de la FPGA mediante la cual puede programar paradas del reloj de referencia tras un número dado de ciclos, o bien producir estas detenciones mediante señales de propósito general provenientes de los canales de rutado de la FPGA. De esta manera, el microprocesador, controlado desde un PC o cualquier otro interfaz de usuario, puede ejecutar paso a paso de manera consistente tanto las instrucciones del programa ensamblador como los ciclos de reloj del *hardware*, presentando en la pantalla al usuario la evolución simultánea de todo el sistema refrescándose en tiempo real. Además, dado que es el PC el que controla todo el proceso, es posible implementar un gran número de funciones añadidas, tales como funciones de disparo (*trigger*) para comenzar la adquisición de datos en el analizador de estados lógicos emulado, un número ilimitado de *breakpoints* tanto *hardware* como *software*, un generador de estímulos, un osciloscopio digital emulado, etc.

La figura 5.1 muestra una captura de la pantalla de un PC donde corren las herramientas de emulación creadas para el primer demostrador. En particular puede apreciarse una ventana de código desensamblado en tiempo real, un gestor de memoria, y un visualizador de formas de onda emuladas.

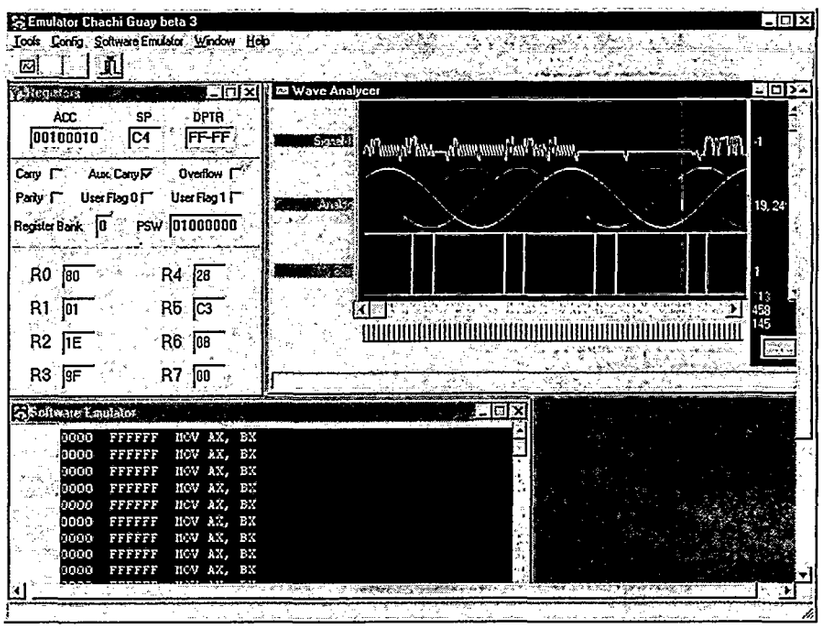

Fig 5.1: Captura de pantalla del emulador de FIPSOC

En la figura 5.2 se muestra la primera placa de desarrollo de aplicaciones con el primer FIPSOC demostrador, que incluye, además del *chip*, memoria paralelo de programa o datos, memoria

serie de configuración con protocolos SPI e I2C, un interfaz serie RS232 para comunicaciones con el PC, un bus en *daisy chain* para comunicar varios FIPSOCs en un esquema multi-esclavo, y una zona de *wire wrapping* para aplicaciones de usuario.

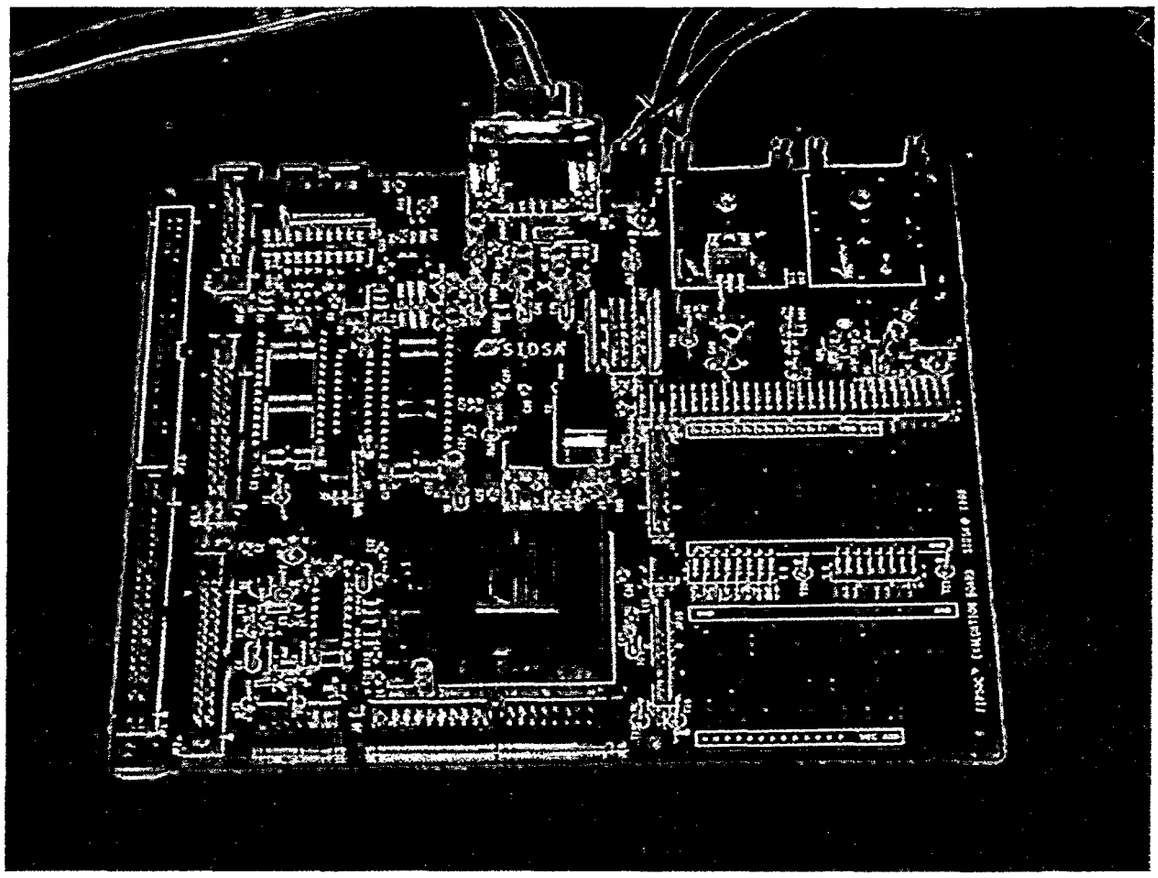

Fig 5.2: Fotografía de la placa de desarrollo de aplicaciones con el primer FIPSOC demostrador

5.3 Sistemas monochip

La aplicación más obvia de los sistemas de tipo FIPSOC consiste en la integración de aplicaciones que de otra manera se realizarían mediante componentes discretos o mediante circuitos integrados de aplicación específica o ASICs (*Application Specific Integrated Circuit*). La integración presenta un valor añadido frente a soluciones basadas en componentes discretos debido a la reducción de coste de silicio y de empaquetado (se utilizan menos piezas y son necesarias menos células de entrada-salida para comunicar los bloques funcionales de la aplicación) y la mejora de la calidad en la aplicación final (menor número de soldaduras y conexiones entre componentes); la programabilidad ofrece ventajas frente a la solución tipo

ASIC en tanto en cuanto reduce el tiempo de diseño y de prototipado del circuito final, típicamente de muchos meses a pocas semanas (el prototipado es instantáneo).

Sin embargo, el valor fundamental de una arquitectura de tipo FIPSOC en este ámbito radica en la metodología integrada de desarrollo mediante un conjunto único y consistente de herramientas sobre un substrato monolítico. Basándose en nuestras primeras experiencias de aplicación de la arquitectura a aplicaciones reales de señal mixta, se estima que el ciclo de diseño de un sistema de este tipo puede reducirse en un orden de magnitud comparado con un ASIC normal, y en más de la tercera parte comparado con una metodología basada en FPGAs y microcontroladores discretos. Además, el coste de los medios materiales necesarios para seguir esas dos metodologías resultaría varios órdenes de magnitud por encima del coste del sistema de desarrollo basado en FIPSOC.

Entre las primeras aplicaciones que han demostrado la ventaja y economía de nuestra aproximación comparada con una solución basada en ASICs o en FPGAs y microcontroladores podemos contar reconocedores de monedas, controladores de máquinas expendedoras, gestores de sistemas de alarmas contra incendios, medidores de energía eléctrica, controladores de teléfono, etc.

5.4 Sistemas con *hardware* virtual

Como fue apuntado en la introducción, los substratos reconfigurables dinámica y parcialmente son susceptibles de soportar el llamado *hardware* virtual, así llamado como analogía a la memoria virtual de los microprocesadores que ofrecen al usuario un espacio de memoria más grande del que realmente disponen. Igual que estos micros guardan en el disco el resto de los datos que deberían estar en memoria, FIPSOC puede guardar la configuración y los datos de ciertos bloques de la aplicación en el contexto no activo o incluso en la memoria externa (serie por ejemplo) de configuración, cargando sobre el *hardware* configurable la parte necesaria en cada momento con sus datos correspondientes. De esta manera es posible implementar aplicaciones que en teoría utilizan más recursos de los que en realidad dispone el *chip*. Lógicamente este tipo de aplicaciones está limitado por la compatibilidad temporal de las distintas tareas que debe llevarse a cabo en la aplicación. El tipo de aplicaciones más claramente susceptiblea a ser implementadas con este tipo de esquema es aquél en que se precisa de procesamiento secuencial en el que cada etapa tiene una duración considerable. Entonces resulta

posible utilizar el microprocesador como gestor de tareas *hardware* que iría reconfigurando el contexto no activo en cada momento con la configuración de la siguiente etapa a realizar. Entre este tipo de aplicaciones podemos contar las típicas de procesamiento de video, por ejemplo, en las que suele ser necesario realizar filtrados, convoluciones, mezclas, y otros algoritmos que en general pueden descomponerse en etapas secuenciales claramente definidas.

También es posible utilizar este tipo de técnica con aplicaciones industriales que no necesariamente ejecutan algoritmos de procesamiento numérico sino realizan tareas de control. Entre estos ejemplos podemos contar la práctica totalidad de las aplicaciones mencionadas en el párrafo anterior; así por ejemplo, el reconocedor de monedas está activo solamente durante el momento en el que el usuario introduce la moneda, funcionando el resto del tiempo como controlador de la máquina expendedora.

5.5 Sistemas adaptativos y reconfigurables

Debido a las posibilidades del *chip* para reconfigurarse parcial y dinámicamente, resulta posible que sea el mismo microprocesador, o incluso el mismo *hardware* programable, quien decida cómo y cuando debe producirse la reconfiguración. Se consigue de esta manera un sistema adaptativo que puede evolucionar por su cuenta según las necesidades de cada momento.

Entre las aplicaciones autoadaptativas para arquitecturas tipo FIPSOC propuestas hasta el momento destacan las lideradas por Moreno *et al.* [MOR98], realizadas dentro del marco del proyecto FIPSOC. Entre los casos de estudio propuestos podemos mencionar los siguientes (en algunas de las publicaciones generadas a partir de esta tesis, y en [MOR98] en especial, pueden encontrarse detalles en profundidad sobre estas aplicaciones):

- Estrategia de autorreparación de sistemas digitales, especialmente útil en entornos hostiles expuestos a radiación. Este tipo de soluciones es necesario en aplicaciones espaciales en las que los circuitos digitales están expuestos a radiaciones de alta energía capaz de producir cambios no destructivos en los datos [MA89] [PET83]. En este contexto la capacidad de reconfiguración dinámica explicada en el capítulo dos resultaría útil para refrescar continuamente los datos de configuración de tal forma que se sobreescribiría cualquier inconsistencia que se produjese debido a radiación de alta energía. Esto es posible gracias a la capacidad única del *chip* de poder escribir en la memoria de configuración mientras la aplicación está activa, produciendo de forma periódica una transferencia general de contexto

para todo el bloque. Más interesantemente aún, se puede mapear la misma aplicación en los dos contextos de un grupo de DMCs y ejecutar por duplicado cada operación *hardware* a realizar en dos ventanas de tiempo secuenciales. Finalmente se compararía ambos resultados y se generaría una señal de error en caso de existir alguna diferencia, lo que obligaría al microprocesador a reparar el contexto dañado y provocar su transferencia.

- Evolución de máquinas celulares paralelas. El caso de los autómatas celulares no uniformes es especialmente interesante ya que éstos presentan propiedades universales de computación [SIP96]. La dinámica dentro del paradigma de computación celular puede dividirse en tres etapas principales: a) inicialización de las reglas de interacción entre las células; b) ejecución de la interacción, con lo cual el estado de cada célula evoluciona en función del estado previo y del estado actual de las células vecinas, aunque las reglas de interacción no varían; y c) evolución de las reglas de interacción para la siguiente iteración. Moreno [MOR98] propone utilizar la reconfiguración dinámica multicontexto de la arquitectura para implementar los grandes multiplexores necesarios en la etapa de evolución según las soluciones *hardware* propuestas hasta el momento [SIP97]. Se trataría de multiplexores virtuales en los que cada una de las entradas se obtendría y almacenaría de forma secuencial utilizando los mismos recursos de rutado reconfigurados dinámicamente de forma distinta de una forma parecida a la propuesta en [LIN96], lo cual minimizaría los recursos totales de rutado a utilizar aunque a costa de una menor velocidad de operación debido a su naturaleza secuencial y no paralela.
- Evolución de redes neuronales. En [MOR99] se explica cómo la reconfiguración dinámica parcial puede ser usada para actualizar los pesos sinápticos en redes neuronales y, en general, implementar distintas arquitecturas de multiplicadores de forma eficiente.

5.6 Coprocesadores reconfigurables para microprocesadores

El hecho de que las señales *hardware* provenientes de la FPGA puedan leerse como posiciones de memoria desde el microprocesador permite una estrecha relación entre *hardware* y *software*. De esta manera resulta posible implementar en *hardware* bloques coprocesadores que complementen algoritmos *software* cuya ejecución es costosa en términos de tiempo. El interfaz a estos bloques es muy suave: el microprocesador escribe los datos de entrada en registros (FFs)

de determinados DMCs de la FPGA, y recoge un cierto número de ciclos (normalmente cero) más tarde el resultado de la operación desde otra posición de memoria.

Por ejemplo, un multiplicador serie de 16 *bits* puede realizar una multiplicación en *hardware* en el tiempo que el microprocesador emplea para ejecutar una instrucción rápida. Si además es necesario ejecutar un cierto número de multiplicaciones y acumulaciones, por ejemplo para una FFT (*Fast Fourier Transform*) o una DCT (*Discrete Cosine Transform*), el ahorro de tiempo es cuantioso.

Entre las operaciones susceptibles de ser aceleradas por coprocesadores *hardware* podemos contar, además de las multiplicaciones simples, las transformadas de Fourier y similares (FFT, DCT, etc.), las acumulaciones, las operaciones de matrices (cálculos de determinantes, multiplicaciones, etc.), y en general un gran número de instrucciones típicamente implementadas de forma específica en los procesadores digitales de señal o DSPs (*Digital Signal Processors*).

En nuestro caso, los coprocesadores para el microprocesador son normalmente mapeados sobre la FPGA de forma dinámica durante los momentos en que son verdaderamente necesarios para complementar al programa, quedando el resto de tiempo los recursos utilizados libres para otras partes de la aplicación global.

5.7 Librerías para co-diseño *Hardware-Software*

Utilizando la interacción entre *hardware* y *software* indicada en el párrafo anterior es posible desarrollar soluciones *hardware* encapsuladas con prototipos *software* utilizables en programas de usuario. De esta forma, un mismo prototipo de función puede tener distintas implementaciones entre las cuales el usuario puede elegir según sus necesidades y recursos en cada caso.

Tomemos el caso de un multiplicador de 16 *bits*. Una primera solución es realizar varias multiplicaciones y sumas mediante una subrutina en ensamblador que utilizaría unos parámetros de entrada convenidos y unas variables de salida. Otra solución es implementar el multiplicador en *hardware* programable e introducir los datos y extraer los resultados mediante accesos a memoria. La primera solución es más lenta que la segunda y ocupa espacio de memoria de programa, si bien la segunda ocupa *hardware* programable no necesario en la primera solución.

El usuario puede elegir por tanto entre las dos implementaciones funcionalmente equivalentes según sus necesidades de velocidad y según sus recursos disponibles.

Si se trata de un bloque más complicado, como por ejemplo una DCT, las diferencias en cuanto a tiempo de ejecución aumentan considerablemente. Incluso pueden existir soluciones mixtas en las que el multiplicador se mantiene en *hardware* mientras que las acumulaciones se realizan por programa, o soluciones en las que el multiplicador se programa en un contexto y las acumulaciones en el otro, utilizando el microprocesador para coordinar el cambio entre contextos y sus comunicaciones. Cada una de estas posibles soluciones se evaluarían según su tiempo de ejecución y según su coste en términos de recursos *hardware* (cantidad de DMCs ocupados) y *software* (espacio de memoria de programa necesaria) necesarios, para así poder elegir una u otra implementación en función de la criticalidad y los recursos disponibles en cada momento.

Entre las líneas futuras de investigación propuestas se halla la creación de una herramienta de síntesis capaz de utilizar estos costes y tiempos de ejecución para elegir de forma automática la implementación óptima de cada bloque para cada aplicación. La librería de síntesis sería en este caso mixta con soluciones *software* de bajo coste pero alto retardo, soluciones con *hardware* reconfigurable de coste medio y retardo medio, y soluciones en *hardware* permanente de coste alto y retardo bajo.

5.8 Resumen

En este capítulo se menciona algunas de las aplicaciones propuestas hasta el momento para arquitecturas tipo FIPSOC y en particular para la primera realización práctica descrita en los capítulos anteriores. Cada una de estas aplicaciones aprovechan alguno o varios de los aspectos más novedosos que el sistema presenta, desde la integración total de arquitecturas de señal mixta hasta la interacción *hardware-software* o la reconfiguración dinámica multicontexto. Se citan aplicaciones de control industrial habituales pero también aplicaciones novedosas dentro del ámbito de la investigación, como son los sistemas autorreparables, las redes neuronales o los sistemas celulares. El uso que se le da a la reconfiguración dinámica varía desde el ahorro de *hardware* mediante técnicas virtuales hasta la automodificación controlada del sistema, pasando por la utilización del *hardware* programable para aumentar el poder de procesamiento del microcontrolador. Finalmente se introduce el sistema de desarrollo integrado creado para el primer FIPSOC demostrador, el cual supone un gran avance ya que permite la co-emulación

hardware-software en tiempo pseudo-real de aplicaciones mixtas, una capacidad nunca antes ofrecida en sistemas monochip integrados ni en arquitecturas de uso general FPGA-microprocesador.

CAPÍTULO 6

CONCLUSIONES Y TRABAJO FUTURO

6.1 Introducción

En este capítulo intentaremos poner de relieve los aspectos que, a nuestro entender, constituyen las principales aportaciones de esta tesis al estado del arte en cuanto a arquitecturas programables y dinámicamente reconfigurables se refiere. Esta apreciación se apoya en el buen número de publicaciones en diversos foros internacionales a que esta tesis ha dado lugar, así como en el marcado interés que ha suscitado en la comunidad internacional tanto científica como industrial.

Así mismo se marcan las principales líneas de actuación futuras para el ulterior desarrollo de los principales aspectos novedosos introducidos, de las cuales la gran mayoría ya están siendo acometidas por varios grupos de trabajo a nivel internacional utilizando placas de desarrollo con el primer FIPSOC demostrador.

6.2 Principales aportaciones de esta tesis

Entre las principales aportaciones de esta tesis podemos contar las siguientes:

- Introducción de una metodología para el desarrollo de circuitos programables que incluyen una FPGA, un microprocesador y un *front-end* analógico. Se estudian los distintos aspectos de su diseño, integración y posibles aplicaciones, así como de las ventajas metodológicas obtenidas.

- Introducción de una metodología para el estudio, diseño e implementación de FPGAs basadas en microprocesador. Si bien existían precedentes de sistemas programables que aunque no estuvieran integrados en un *chip* incluían un microprocesador como una de sus partes constituyentes, nunca antes fue propuesta una FPGA enteramente basada en microprocesador en el sentido de que toda su configuración se realizase desde él y de que los estados de las señales digitales fueran accesibles desde el *software*.

- Introducción, diseño físico y estudio de viabilidad de un nuevo esquema de memoria de configuración de sistemas programables que permite la reutilización de memoria para propósito general, la reconfiguración dinámica multicontexto, y la configuración parcial. De esta forma es posible escribir una nueva configuración en una célula programable mientras que la célula está siendo utilizada para otro propósito. Por primera vez es además posible programar con la misma configuración un *array* de células seleccionadas por una máscara, lo cual resulta muy interesante para aplicaciones de procesamiento de señal de tipo sistólico y similares.

- Estudio e implementación práctica de memoria distribuida en sistemas de señal mixta, teniendo en cuenta todos los problemas de estabilidad del efecto memoria derivados de la distribución de la memoria a lo largo de grandes áreas de silicio sobre las que se implementa otros bloques funcionales.

- Diseño e implementación del primer *chip* completo demostrador de una arquitectura de tipo FIPSOC, incluyendo la primera realización práctica de una FPGA basada en microprocesador y todos los interfaces entre los tres subsistemas. Se demuestra así la viabilidad de la arquitectura y de los esquemas de reconfiguración dinámica multicontexto propuestos. Se trata de un sistema complejo de 65mm^2 (9.6x6.5 mm^2) en 0.5mm y tres metales, con más de 1.8 millones de transistores de los cuales 1.5 millones están implementados en *full custom* manual, alcanzando densidades de integración superiores a 40000 transistores/mm^2.

- Diseño e implementación del primer sistema completo de emulación integrado en un *chip*. El sistema incluye co-emulación *hardware-software* y es capaz de soportar un laboratorio virtual completo cuyo interfaz es un simple PC. El laboratorio puede incluir un analizador de estados, un osciloscopio digital, un co-emulador que soporta ejecución pseudo real paso a paso de *hardware* y *software* concurrente, etc. Además, todo el sistema se controla y se programa desde un PC con un simple puerto serie.

6.3 Principales publicaciones generadas a partir de esta tesis

Entre las principales publicaciones en congresos internacionales y revistas especializadas generadas a partir de esta tesis podemos contar las siguientes:

- V. Baena-Lecuyer, M.A. Aguirre, A. Torralba, L.G. Franquelo and J. Faura, *"Decoder-driven switching matrices in multicontext FPGAs: area reduction and their effects on routability"*, 1999 IEEE International Symposium on Circuit and Systems (ISCAS'99).
- J.M. Moreno, J. Cabestany, E. Cantó, J. Faura, J.M. Insenser, *"The Role of Dynamic Reconfiguration for Implementing Artificial Neural Networks Models in Programmable Hardware"*, 5th International Work-Conference on Artificial and Natural Neural Networks (IWANN'99), June 1999.
- J.M. Moreno, J. Cabestany, J. Madrenas, J. Faura, J.M. Insenser, *"Approaching Evolvable Hardware to Reality: The Role of Dynamic Reconfiguration and Virtual Meso-Structures"*, 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (MICRONEURO'99), April 1999
- J. Faura, I. Lacadena, A. Torralba and J.M. Insenser, *"Programmable Analog Hardware: A Case Study"*, IEEE International Conference on Electronic Circuits and Systems ICECS'98, Lisboa, September 7-10 1998.
- V. Baena-Lecuyer, A. Torralba, M.A. Aguirre, J. Faura and L.G. Franquelo, *"RAISE: A Detailed Routing Algorithm for SRAM based Field-Programmable Gate Arrays Using Multiplexed Switches"*, 1998 IEEE International Symposium on Circuit and Systems (ISCAS'98), session WPB4-3.
- J.M. Moreno, J. Madrenas, J. Faura, E. Cantó, J. Cabestany, M.M. Insenser, *"Facing Evolutionary and Self-Repairing Hardware by means of the dynamic reconfiguration capabilities of the FIPSOC devices"*, ICES'98.
- J. Madrenas, J.M. Moreno, J. Cabestany, J. Faura, J.M. Insenser, *"Radiation-Tolerant On-Line Monitored MAC Unit for Neural Models Using Reconfigurable-Logic FIPSOC Devices"*, IEEE International On-Line Testing Workshop (IOLTW'98), pp.114-118, July 1998.

- J.M. Moreno, J. Cabestany, E. Cantó, J. Faura, P. van Duong, M.A. Aguirre and J.M. Insenser, *"FIPSOC. A Novel Mixed FPGA for System Prototyping"*, in Mixed Design of Integrated Circuits and Systems (edited by Andrzej Napieralski et al), pp. 169-174, Kluwer Academic Publishers.
- Julio Faura, Miguel A. Aguirre, Juan M. Moreno, Phuoc van Duong, Josep Maria Insenser, *"FIPSOC: A Field Programmable System On a Chip"*, XII Design of Circuits and Integrated Systems Conference (DCIS'97), pp 597-602, November 1997.
- Julio Faura, Josep Maria Insenser, *"Tradeoffs for the Design of Programmable Interconnections in FPGAs"*, XII Design of Circuits and Integrated Systems Conference (DCIS'97), pp 415-420, November 1997.
- V. Baena-Lecuyer, M.A. Aguirre, A. Torralba, L.G. Franquelo and J. Faura, *"RAISE: A Detailed Routing Algorithm for Field-Programmable Gate Arrays"*, XII Design of Circuits and Integrated Systems Conference (DCIS'97), pp 421-424, November 1997.
- Julio Faura, Juan Manuel Moreno, Miguel Angel Aguirre, Phuoc van Duong, and Josep Maria Insenser, *"Multicontext Dynamic Reconfiguration and Real-Time Probing on a Novel Mixed Signal Programmable Device with On-Chip Microprocessor"*, 7th International Workshop on Field Programmable Logic and Applications (FPL'97), pp 1-10, Lecture Notes in Computer Science, Springer.
- Julio Faura, Chris Horton, Phuoc van Duong, Jordi Madrenas, Miguel Angel Aguirre, and Josep Maria Insenser, *"A Novel Mixed Signal Programmable Device with On-Chip Microprocessor"*, Proceedings of the IEEE 1997 Custom Integrated Circuits Conference CICC'97, pp. 103 (6.4.1) to 106 (6.4.4).
- J. Faura, J.M. Moreno, J. Madrenas and J.M. Insenser, *"VHDL modeling of fast dynamic reconfiguration on novel multicontext RAM-based FPDs"*, Proceedings of the VHDL User's Forum in Europe, SIG-VHDL Spring'97 Working Conference, April 22-24 1997, Toledo, Spain, pp. 171-177.
- Julio Faura, Chris Horton, Bernd Krah, Joan Cabestany, Miguel Angel Aguirre, and Josep Maria Insenser, *"A New Field Programmable System-on-a-chip for Mixed Signal Integration"*, Proceedings of the IEEE 1997 European Design and Test Conference ED&TC'97, pp. 610

- J. Faura, P. Paddan, B. Krah, P. Van Duong, J.M. Moreno, A. Torralba, J. Launa, J.M. Insenser, *"FIPSOC: A New Concept to Mixed Signal Integration"*, Proceedings of the Silicon Design Show 1996, pp. 47 to 52.
- J.M. Moreno, J. Cabestany, J. Faura, C. Horton, P. Van Duong, M.A. Aguirre, J.M. Insenser, *"FIPSOC: A Novel Mixed Programmable Device for System Prototyping"*, 4th International Workshop on Mixed Design of Integrated Circuits and System MIXDES'97, June 12-14 1997, Poznan, Poland.

Además debemos mencionar más de una docena de artículos aparecidos en publicaciones divulgativas y también en revistas especializadas en diseño electrónico tales como el EE Times, EDN, Mundo electrónico, El País, etc.

6.4 Líneas de trabajo futuro

Debido al gran interés que este trabajo ha suscitado en la comunidad internacional existen en la actualidad un gran número de líneas de investigación activas como continuación a este estudio. Estas líneas se dividen en dos grupos según estén más bien orientadas a *software* o a *hardware*. Las primeras intentan explotar de forma más sencilla para el usuario los recursos de reconfiguración dinámica, gestión de configuraciones e interacción entre *hardware* y *software* de que disponen las arquitecturas de tipo FIPSOC, mientras que las segundas se orientan a extender las posibilidades del *chip* y a optimizar la arquitectura *hardware* existente.

6.4.1 Esquemas de rutado jerárquico

La arquitectura de rutado diseñada para el primer FIPSOC demostrador presenta ciertos problemas de cara a la extensión del número de puertas equivalentes soportadas por versiones más grandes dentro de la misma familia. Si bien la arquitectura de rutado utiliza segmentos de distintas longitudes como se describió en el capítulo 2, no se trata de una arquitectura inherentemente jerárquica como las propuestas en [WAN94] y [BEA96].

Para el tamaño del primer demostrador (8x12 DMCs) los recursos de rutado parecen en principio suficientes a tenor de los relativamente buenos resultados de congestión (o de ausencia de ella) obtenidos hasta el momento. Sin embargo, miembros mayores de la familia, y en particular mayores de 16x16, necesitarían una ampliación del número de canales para garantizar una rutabilidad competitiva.

Si bien es posible extender el número de canales de forma pseudo-jerárquica agrupando los DMCs en *clusters* de por ejemplo 8x8 e implementado grupos de canales globales en la periferia de los *clusters*, parece en principio más potente y sencillo de usar el disponer de una arquitectura de rutado totalmente jerárquica en que la conexión entre células remotas se haría mediante una búsqueda binaria y el número de conmutadores de interconexión crecería de forma logarítmica con el tamaño de la FPGA. Una investigación más exhaustiva sería necesaria para determinar la granularidad óptima y la cantidad y el ritmo de crecimiento exponencial de los recursos de rutado en cada nivel jerárquico, a la vez que para estimar la complejidad de los algoritmos de *placement & routing* necesarios.

6.4.2 Arquitecturas analógicas programables avanzadas

Como se ha mencionado a lo largo de todo este trabajo, las células analógicas incluidas en la primera realización práctica de un sistema FIPSOC no son de propósito general y su utilidad está restringida a aplicaciones de control industrial. En el futuro sería deseable crear miembros específicos de la familia FIPSOC para aplicaciones concretas con la circuitería analógica necesaria para cada tipo de aplicación: convertidores sigma-delta de alta precisión para aplicaciones de audio, convertidores flash de 8 *bits* para aplicaciones de video, filtros programables para aplicaciones de telefonía y para *anti-aliasing* en general, o moduladores para aplicaciones de comunicaciones.

Siguiendo otra línea de actuación distinta también sería interesante mejorar las prestaciones y la flexibilidad de la parte analógica de tal forma que ésta pueda considerarse realmente como una FPAA (*Field Programmable Analog Array*) de menor granularidad con un esquema de rutado verdaderamente general. Este esquema presenta un mayor interés técnico, especialmente en los aspectos relacionados con la interacción mixta entre el *hardware* analógico y digital.

6.4.3 Co-procesadores reconfigurables fuertemente acoplados

La tercera línea futura de investigación propuesta se centra en aumentar las posibilidades de interacción entre el microprocesador y FPGA. Según el esquema propuesto, la interacción *hardware-software* se ciñe a que las salidas digitales de los bloques programables puede ser consultada desde el microprocesador, y a que éste puede admitir interrupciones provenientes de los canales de rutado.

La interacción entre *hardware* y *software* podría ampliarse si la parte programable del *chip* se pudiera usar para extender la ruta de datos del microprocesador, o si el último estuviera parcialmente implementado mediante células hasta cierto punto programables. De esta forma la ruta de datos del microprocesador se podría adaptar dinámicamente según las necesidades computacionales de cada momento, bien por el usuario a base de extender el conjunto de instrucciones del ensamblador o bien de forma autónoma llevando algún tipo de monitorización estadística de tipo *cache* análoga a las arquitecturas *cache* que regulan el acceso a memoria en los microprocesadores avanzados.

Mediante este tipo de programación sería posible por ejemplo realizar un microprocesador con un conjunto de instrucciones compatible con el de un microprocesador comercial, para de esta forma reaprovechar *software* ya escrito o utilizar herramientas *software* (compiladores, ensambladores, emuladores, etc.) previamente disponibles.

Algunas de estas ideas se están empezando a investigar dentro del proyecto BRASS liderado por André DeHon en la Universidad de Berkeley.

6.4.4 Herramientas y algoritmos de reconfiguración dinámica

El gran problema que habitualmente se encuentra para la aplicación de técnicas novedosas de computación *hardware* en problemas generales resolubles mediante técnicas clásicas es el interfaz *software* con el usuario, que debe ser suficientemente sencillo y tangible para que éste no tenga que trabajar a bajo nivel y perderse en el detalle sino que más bien pueda utilizar los recursos novedosos de forma transparente o al menos simplificada.

Con este objetivo en mente se está trabajando para diseñar un interfaz *software* capaz de utilizar las posibilidades de reconfiguración dinámica multicontexto disponibles en el *chip* de una

manera transparente al usuario, de forma que éste pueda mapear aplicaciones generales sobre el substrato reconfigurable sin preocuparse de si éstas se implementan sobre *hardware* fijo o reconfigurable. Estas herramientas deben realizar un estudio de la compatibilidad temporal de los procesos y utilizar los resultados sobre las fases de *placement* y *routing*. Algunos algoritmos de planificación temporal (*scheduling*) han sido ya propuestos [LYS96], aunque por el momento sólo han sido aplicados a simulación y no a herramientas de implementación lógica o física.

6.4.5 Herramientas de co-diseño Software-Hardware *reconfigurable*

Las propiedades reconfigurables de las arquitecturas tipo FIPSOC como las descritas y la cercana interacción entre el *hardware* y el *software* que soportan abren nuevas posibilidades al problema de la síntesis lógica automática de circuitos digitales. El objetivo de esta línea de investigación sería una herramienta capaz de simular sobre una librería mixta compuesta por elementos que a partir de un mismo prototipo tendrían tres implementaciones: una sobre *hardware* fijo convencional, otra sobre *hardware* dinámicamente multiplexado en el tiempo, y otra sobre *software* para el microcontrolador. El sintetizador analizaría entonces la compatibilidad temporal de cada proceso a implementar y determinaría su criticalidad, para así elegir la implementación más apropiada según los requerimientos de cada aplicación.

La viabilidad incluso comercial de este tipo de sintetizador dependería del nivel de trasparencia frente al usuario, el cual debería ocuparse únicamente de especificar cuál sería el enfoque en cada momento (reducir área, descargar al microprocesador, maximizar la velocidad, etc.)

6.4.6 Co-simulación y co-emulación hardware-software

La capacidad de co-emulación *hardware-software* descrita en capítulos anteriores podría extenderse a un nivel más general si su funcionamiento adoptase el mismo estándar de comunicación usado en la implementación de un simulador. De esta forma sería posible mezclar elementos del sistema que serían en parte emulados en *hardware* paso a paso con elementos simulados por ordenador, todo corriendo en tiempo pseudo-real sincronizado con la evolución del *software* del microcontrolador.

6.5 Resumen

En este capítulo se ha presentado las principales aportaciones de estas tesis y las líneas futuras de continuación de este trabajo. La gran aceptación que el concepto propuesto de sistema programable basado en microprocesador en general, y la primera implementación práctica de un sistema tipo FIPSOC en particular han recibido en la comunidad científica e industrial internacional hace albergar grandes esperanzas en cuanto a su futuro como línea de investigación básica y en cuanto a su aplicación industrial. Algunas de las líneas futuras de actuación ya están siendo comenzadas, mientras que las restantes están siendo planificadas o evaluadas para su comienzo inminente.

Finalmente, es importante destacar que las principales aportaciones de esta tesis han sido propuestas de forma aplicada a la resolución de problemas reales en aplicaciones del ámbito científico e industrial de nuestros días. Además de las correspondientes consideraciones teóricas y conceptuales se ha acometido el desarrollo de esta tesis desde un punto de vista práctico ilustrando los aspectos metodológicos mediante ejemplos concretos extraídos de la primera implementación práctica de dispositivo FIPSOC, cuyos resultados reales en silicio permiten validar el trabajo propuesto.

APENDICE

INTRODUCCION A LAS ARQUITECTURAS PROGRAMABLES: ELEMENTOS CONSTITUTIVOS Y PARAMETROS DEFINITORIOS

A.1 Introducción

En este apéndice intentamos desarrollar una pequeña introducción general a las arquitecturas programables tanto digitales como analógicas que nos será útil para definir los términos con los que trabajar a la hora de diseñar y caracterizar cualquier arquitectura programable de propósito general.

En primer lugar proponemos una estructura genérica de FPGA para explicar cada uno de sus elementos constitutivos y las relaciones entre ellos. En este apartado se revisan someramente las arquitecturas clásicas de las FPGAs que se pueden encontrar hoy día en el mercado y en la literatura científica, y se introducen los criterios hallados entre otros por Brown y Rose [BRO92a] para el problema general del diseño de FPGAs. Así mismo se explica qué metodos pueden usarse para la evaluación de arquitecturas básicas de FPGAs y que métricas proceden. Además se recopila una breve evolución histórica de las arquitecturas digitales programables.

Seguidamente se introduce las arquitecturas analógicas programables desde una perspectiva histórica, aun teniendo en cuenta los pocos avances que se han hecho en general en este campo.

Finalmente se hace una introducción al problema general de la configuración de arquitecturas programables, se define una terminología consistente para el estudio de las distintas técnicas de reconfiguración (y en especial de reconfiguración dinámica), y se hace una breve lista de los precedentes y estado del arte de las arquitecturas programables que incorporan aspectos novedosos relacionados con reconfiguración dinámica.

A.2 Arquitecturas digitales programables

En un sentido amplio entendemos por arquitectura digital programable un circuito lógico cuya funcionalidad puede ser programada por el usuario después de haber sido fabricado. Además de las memorias ROM programables eléctricamente o PROMs, en todas sus variedades (EPROM, EEPROM, Flash, etc.), y de los microprocesadores, cuya función lógica se configura mediante un programa de usuario almacenado en una memoria, las arquitecturas programables que son de interés para este trabajo se pueden alinear en dos grupos según S. Brown [BRO92a]:

- PLDs (*Programmable Logic Device*). A este grupo pertenecen los circuitos programables compuestos fundamentalmente por un plano AND y otro OR, típicamente terminados en registros opcionales. Este tipo de circuitos se pensó para implementar máquinas de estados y funciones combinacionales de propósito general, siempre descomponibles en dos planos lógicos AND y OR. A este grupo pertenecen los PALs (*Programmable Array Logic*), cuyo plano OR (el segundo) suele estar fijo, los PLAs (*Programmable Logic Array*), en los que la conectividad en el plano OR también es configurable, y los CPLDs (*Complex PLDs*), en los cuales la libertad de programación es mayor y además suele haber canales de rutado para interconectar un conjunto de este tipo de estructuras de dos planos. En la actualidad son los CPLDs, especialmente los de la casa Altera [ALT98], los de mayor difusión.
- FPGAs (*Field Programmable Gate Array*). Una FPGA es un *array* o conjunto de células programables distribuidas típicamente en una matriz bidimensional rodeada de células de entrada salida igualmente configurables e interconectadas por canales de rutado de propósito general, como se esquematiza en la figura A.1. Las funciones lógicas se implementan de distintas formas, siendo habitual la utilización de LUTs (*Look-up Table*) o tablas de *look-up* construidas a base de pequeñas memorias RAM. Ejemplos de FPGAs comerciales son las distintas familias de Xilinx [XIL98], las familias OrCA de Lucent [LUC98], y otras.

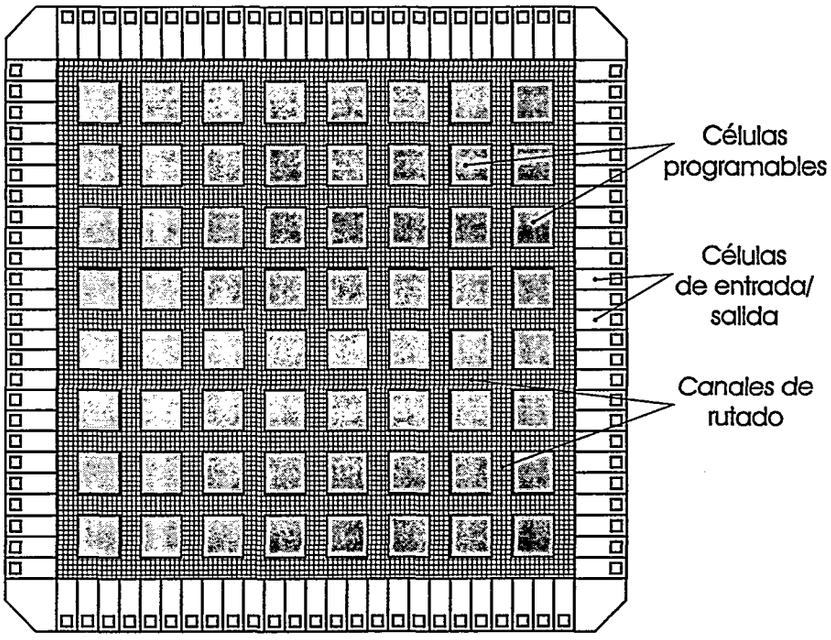

Fig. A.1: Esquema general de bloques de una FPGA

En lo sucesivo nos centraremos en las FPGAs bidimensionales basadas en LUTs como alternativa elegida de arquitectura programable, por ser la más adecuada para nuestro propósito de integrarla con un microprocesador.

A.2.1 Elementos constitutivos de las FPGAs

Como se ha mencionado, una FPGA contiene células lógicas programables, células de entrada-salida configurables, y recursos de rutado que interconectan de forma programable estas células, como se muestra esquemáticamente en la figura A.1. La figura A.2, que reproduce en mayor detalle la estructura genérica de una FPGA bidimensional, servirá de referencia al analizar cada uno de los elementos constitutivos de la misma, con el objetivo de introducir una terminología adecuada durante todo este estudio así como para comparar las alternativas propuestas hasta el momento en proyectos académicos o como productos comerciales.

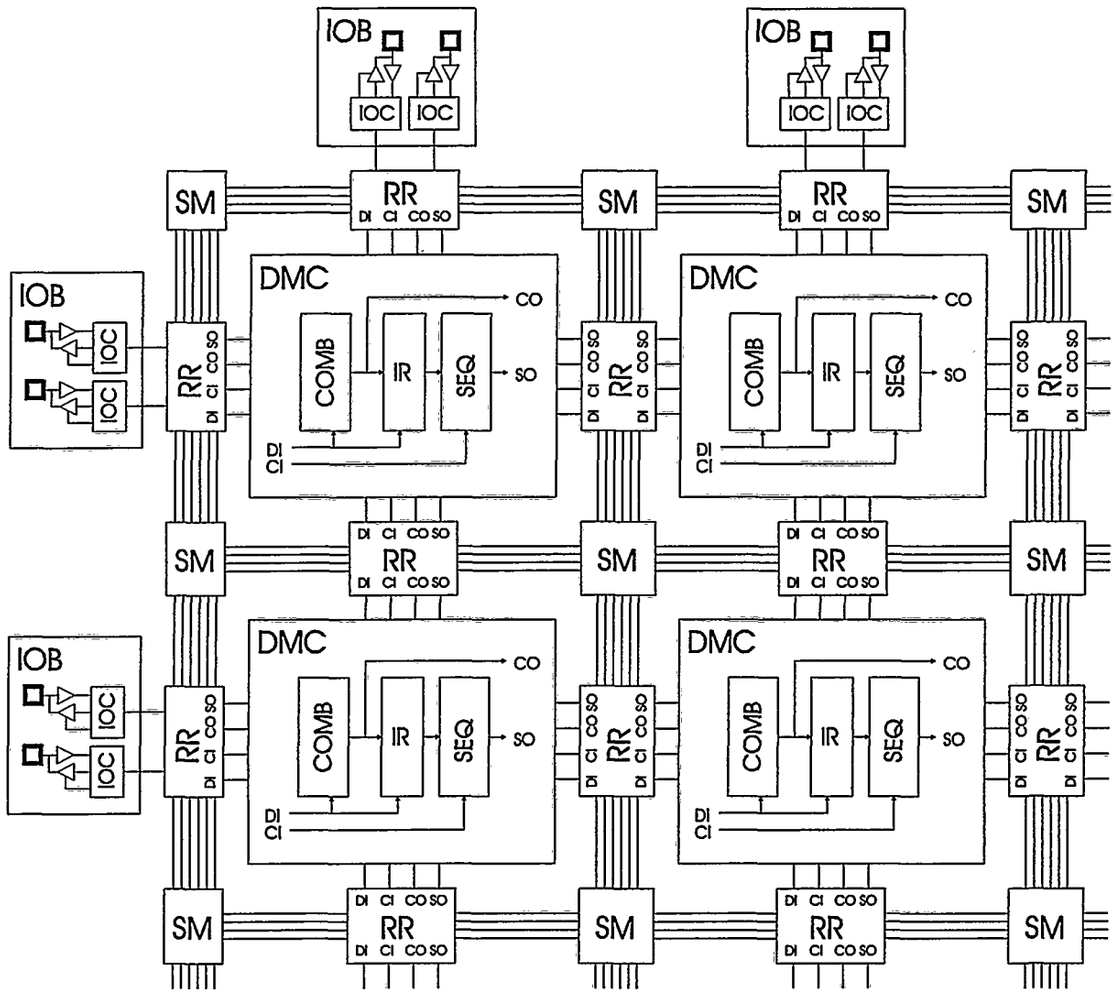

Fig. A.2: Esquema general detallado de una FPGA bidimensional

A.2.1.1 Bloques programables combinacionales y secuenciales

La célula programable está marcada con las siglas DMC (*Digital Macro Cell*), y de forma genérica está compuesta por una parte combinacional programable, una parte secuencial formada por una serie de registros configurables, y un bloque de rutado interno IR (*Internal router*) que interconecta localmente ambas partes con una cierta flexibilidad. Los DMCs pueden tener entradas de datos DI (*Data inputs*) y de control CI (*Control inputs*), y salidas combinacionales CO (*Combinational outputs*) y secuenciales (*Sequential outputs*). De esta forma, si además de poder usar las salidas combinacionales como entradas de la parte secuencial también se pueden utilizar como salidas globales del DMC igual que las salidas secuenciales, y si se pueden encaminar las señales de entrada directamente a los registros, se podría utilizar las partes

combinacional y secuencial de forma más o menos independiente. La práctica totalidad de las FPGAs supervivientes en el mercado [XIL98] [LUC98] responden de forma más o menos precisa a este esquema en el que las partes combinacional y secuencial están claramente separadas, a diferencia de algunas FPGAs de grano fino inicialmente propuestas (y aún existentes en ciertos nichos de mercado) que construyen los elementos secuenciales a partir de elementos combinacionales básicos [ACT98]

La parte combinacional programable de los DMCs puede ser tan sencilla como una puerta *nand* [MUR91], y tan compleja como un conjunto de LUTs de cuatro o cinco entradas de las cuales varias son comunes para LUTs distintas [SIN97] [BRI94] [XIL98] como en las familias de FPGAs de Lucent (OrCA) y Xilinx. Otras arquitecturas que gozaron de gran aceptación en el pasado [GAM89] [AHR90] utilizaban una pequeña estructura lógica compuesta por multiplexores a partir de la cual podía realizarse cualquier función combinacional compleja, o incluso implementar estructuras secuenciales, con mayor eficiencia que otras arquitecturas de granularidad similar [GOP93]. Un gran número de arquitecturas combinacionales básicas alternativas han sido propuestas y están siendo utilizadas en la actualidad por distintas familias comerciales de FPGAs [ATM98] [ACT98] [BIR91] [KEA89].

Además de los bloques combinacionales programables basados en LUTs genéricas como las utilizadas en las familias de Xilinx y Lucent, y los formados por multiplexores o estructuras simples especialmente pensados para la síntesis de funciones lógicas generales como los usados en las FPGAs de Actel o Atmel, un tercer tipo de FPGAs implementa las funciones lógicas programables mediante estructuras tipo PAL que incluyen planos AND y OR programables finalizados con registros configurables, como en los dispositivos ofrecidos por Altera [ALT98]. Estos últimos dispositivos son a veces denominados CPLDs (*Complex PLD*) ó PLD complejos, aunque su inclusión bajo la denominación general de FPGA es comúnmente aceptada.

La parte secuencial de la célula programable está constituida por uno o varios *flip-flops* configurables. El tipo de configurabilidad incluida suele permitir elegir entre *flip-flop* o *latch*, *reset* o *set*, *reset* (o *set*) síncrono o asíncrono, polaridad del reloj (flanco de reloj activo para los *flip-flops* o nivel activo para los *latches*), salida directa o negada, etc.

Cuando en la misma célula lógica se agrupan varias LUTs o en general cuando pueden programarse varias funciones lógicas más o menos independientes y se dispone de varias salidas, es corriente que haya varios *flip-flops* que puedan registrar todas las salidas combinacionales. En tales casos es habitual [XIL98] [ALT98] [LUC98] que los terminales de control, especialmente las líneas de reloj y de *reset*, sean comunes para todos los registros. Este tipo de estructuras

resulta satisfactorio para la implementación de sistemas síncronos y especialmente de arquitecturas de tipo *datapath* [HAR94].

Finalmente, es cada vez más común encontrar células programables de más de un bit que incluyen varias LUTs y varios registros. En este tipo de estructuras suelen existir modos de configuración especiales para la implementación de funciones más complejas habitualmente usadas, tales como sumadores/restadores (parte combinacional), contadores y registros de desplazamiento (parte secuencial), etc. También suele ser posible conectar en cascada varias células programables configuradas en estos modos para conseguir funciones con tamaños de palabra mayores. Betz [BET97] demostró además que la agrupación de varias LUTs y registros en *clusters* o células programables de mayor granularidad resulta en ahorros de recursos de rutado ya que se pueden compartir entradas sin pérdida significativa de rutabilidad.

A.2.1.2 Células de entrada-salida

Los DMCs están distribuidos en forma de matriz bidimensional rodeada de bloques programables de entrada-salida o IOBs (*Input-Output Block*), que suelen contener uno o más terminales gobernados por células programables o IOCs (*Input-Output Cell*). Estas células, responsables de la interacción de la FPGA con el mundo exterior, pueden ser normalmente programadas como entradas, como salidas, o como terminales bidireccionales [LUC98] [XIL98]. Muchas familias de FPGAs incluyen *flip-flops* para registrar opcionalmente las salidas [XIL98]. El *slew rate* de los *buffers* de salida suele ser también configurable [LUC98], o incluso los niveles lógicos de entrada y de salida para poder interconectarse a circuitos comerciales CMOS y TTL [ALT98].

A.2.1.3 Canales y recursos de rutado y matrices de conmutación

Tanto los DMCs como los IOBs se interconectan entre sí mediante un conjunto, también regular, de canales de rutado. Aparte de algunas familias primitivas de FPGAs basadas en filas análogas a las encontradas en circuitos de *standard cells* en tecnologías de dos metales [GAM89] [AHR90] [GOP93] [ROY93], en la actualidad es normal encontrar FPGAs en las que existen canales de rutado horizontales y verticales que rodean cada una de las células programables dispuestas a modo de matriz bidimensional. Cada célula programable se conecta con una cierta flexibilidad a los canales de rutado que la rodean mediante unos recursos de rutado o RRs (*Routing resources*). Los RRs de cada uno de los cuatro lados (norte, sur, este y oeste) que rodean a la célula

programable son distintos entre sí pero en principio es normal que sean iguales para dos DMCs arbitrarios dentro de la matriz bidimensional. Cada bloque RR es compartido por dos DMCs adyacentes.

Los recursos de rutado pueden conectar los terminales de las células programables a los canales de rutado de forma independiente o multiplexada. Las conexiones independientes producen menores retardos de interconexión y ocupan menos área pues son típicamente implementados mediante un conmutador CMOS. Las conexiones multiplexadas producen mayores retardos pues se implementan típicamente con multiplexores bidireccionales construidos a base de transistores de paso NMOS o conmutadores CMOS en varios niveles, que necesitan una mayor área. Sin embargo, las estructuras multiplexadas resultan casi siempre más económicas ya que necesitan menos bits de configuración, exactamente $log_2(N)$ donde N es el número de conexiones multiplexadas a realizar. Para multiplexores anchos, necesarios para rutar un terminal a muchos canales de rutado, esta diferencia resulta dramática.

Los canales de rutado se conectan a través de matrices de conmutación o SMs (*Switching matrices*). La misión de las matrices de conmutación es interconectar canales verticales con canales horizontales, y conectar tramos de los canales para dar continuidad entre izquierda y derecha y entre arriba y abajo. Las conexiones de continuidad de los canales son prácticamente siempre implementadas mediante conmutadores CMOS gobernados por bits de configuración independientes. Estos conmutadores CMOS introducen una resistencia parásita que origina retardos acumulativos muy significativos cuando la distancia a rutar es grande. Por esto, es frecuente encontrar líneas largas o canales que se extienden más allá de un sólo DMC. Las matrices de conmutación en estos casos cortocircuitarían físicamente los tramos de canales adyacentes de forma fija. En la actualidad existen FPGAs con estructuras de rutado compuestas por distribuciones de pistas de distintas longitudes, como la utilizada en las familias OrCA de Lucent [LUC98] que contiene pistas que se extienden uno, dos y cuatro bloques programables, además de las líneas largas que recorren toda la altura o anchura de la FPGA.

Una forma de paliar el problema de los retardos acumulativos en conexiones largas que se extienden a lo largo de varios DMCs consiste en introducir repetidores de línea que irían regenerando la señal cada cierto tiempo [DOB95]. Esta solución resulta especialmente indicada para FPGAs de baja granularidad en las que el número de conmutadores a activar puede ser muy alto para una conexión medianamente larga.

A.2.2 Parámetros definitorios de estructuras lógicas programables

A continuación se introducen una serie de conceptos que definen y diferencian las distintas arquitecturas programables. Si consiguiésemos valorar de forma cuantitativa cada una de estas características podríamos definir un hiperespacio en el que las distintas dimensiones estarían directamente relacionadas con estos conceptos. Cada arquitectura programable estaría así representada por un punto de este espacio con unas coordenadas determinadas. Moviéndonos en este espacio de solciones podríamos buscar puntos óptimos para distintas finalidades, y así evaluar la conveniencia de una arquitectura programable u otra en función del tipo de aplicaciones para la que será usada. En nuestro caso, el criterio sería la capacidad de interacción con un microprocesador y la accesibilidad de las señales y la configuración desde el *software*.

La idea de un espacio de arquitecturas restringido sobre el que moverse para buscar distintos puntos adecuados a uno u otro problema fue introducida por André DeHon para el caso de las arquitecturas reconfigurables para computación de propósito general [DeH96a]. Nuestro enfoque es distinto ya que nos centramos en aplicaciones típicas industriales de control más que en algoritmos de computación de propósito general.

A.2.2.1 Granularidad de la LUT

Se entiende por granularidad el tamaño del elemento programable que servirá de base para la formación de la matriz. La granularidad depende del tamaño de las LUTs y del número de ellas que hay en cada célula programable.

La granularidad de las LUTs, medida como el número de entradas de que dispone, influye decisivamente en la eficiencia del consumo de área, ya que las LUTs grandes ocupan más área pero también reducen el número de ellas que son necesarias para implementar un circuito determinado. Ha sido demostrado que el área total consumida, calculada como el producto del número de LUTs necesarias y el área de cada LUT, tiene un mínimo cuando las LUTs tienen entre tres y cuatro entradas [ROS90a] [HIL93].

La granularidad de las LUTs también influye en las prestaciones de la arquitectura, específicamente en el retardo del camino crítico y por tanto en la máxima frecuencia de funcionamiento de los circuitos a implementar, ya que las LUTs grandes resultan más lentas pero pueden implementar funciones más grandes que de otra forma tendrían que ser descompuestas en

varias más pequeñas con el consiguiente retardo de interconexión entre LUTs. Se ha demostrado que de cara al retardo del camino crítico, calculado como el producto del número de LUTs conectadas en cadena y el retardo de cada LUT, las LUTs de cuatro ó cinco entradas constituyen una buena elección [SIN91] [SIN92] [KOU91].

Como consecuencia de ambos estudios, la mayoría de las FPGAs basadas en LUTs disponibles actualmente incorporan LUTs de cuatro entradas [XIL98] [LUC98].

A.2.2.2 Granularidad del bloque programable

La granularidad del bloque programable está determinada por su número de entradas y salidas, y también por el número de LUTs que incluye. En la actualidad existen FPGAs con células programables de dos [XIL98], cuatro [BRI94] y ocho [SIN97] LUTs de cuatro entradas, que corresponderían a granularidades de mayor a menor. En estos casos, es habitual que varias de las LUTs incluidas en un mismo DMC compartan entradas, con el consiguiente ahorro en área de los recursos de rutado correspondientes. Este ahorro se produce a costa de una reducción en la flexibilidad de uso debido que las funciones lógicas a implementar con las LUTs con entradas comunes deben compartir el mismo número de variables. Se ha demostrado que el agrupamiento de LUTs en *clusters* o células programables de mayor granularidad es eficiente, resultando óptimo para el caso en el que hay cuatro LUTs por *cluster*, bastando con tener 2N+2 entradas comunes (donde N es el número de entradas de cada LUT) frente a las 4N que harían falta si no se compartieran [BET97].

Otro aspecto interesante de las células programables de grano grueso que incluyen varias LUTs es la posibilidad de cambiar su granularidad efectiva, pues mediante un sencillo multiplexor aplicado a las salidas de dos LUTs con todas sus entradas cortocircuitadas se puede disponer de una LUT con una entrada más. Se puede demostrar que las arquitecturas programables que permiten este tipo de combinaciones consiguen mayor eficiencia y prestaciones [HIL91]. De esta manera, una arquitectura que disponga de cuatro LUTs de cuatro entradas podría configurarse para tener dos LUTs de cinco o una LUT de seis entradas [LUC98].

El número de *flip-flops* incluidos en el bloque programable suele coincidir con el número de LUTs, de tal forma que sea posible registrar todas las salidas combinacionales en la configuración normal de granularidad más fina. De esta forma, una célula programable de cuatro u ocho LUTs con *flip-flops* subsiguientes se dice que está orientada a *nibble* o a *byte* respectivamente [BRI94] [SIN97]. Este tipo de arquitecturas programables de mayor

granularidad suele dar mejores resultados en aplicaciones de tipo *datapath* [HAR94], aunque su eficiencia suele verse reducida en máquinas de estado o en aplicaciones que no usen buses sino líneas monodimensionales.

De cara a nuestro propósito de implementar un bloque programable de fácil acceso desde el microprocesador, será en principio deseable una arquitectura de una granularidad media o alta, de tal forma que el número de elementos a acceder (datos internos de los registros, salidas combinacionales, datos de las LUTs, etc.) coincida o sea un múltiplo o submúltiplo de la longitud del bus del microprocesador.

A.2.2.3 Tipo de bloque programable y de memoria de configuración

Como se mencionó en el punto A.2.1.1, existen tres filosofías generales en cuanto a la estructura de la célula programable: basadas en LUTs (típicamente de cuatro entradas) como las Xilinx y Lucent, basadas en estructuras PAL anchas (de ocho o más entradas) como las de Altera, y basadas en multiplexores u otras estructuras específicamente pensadas para un algoritmo de síntesis lógica como las de Actel y Atmel. A la vista de los dispositivos disponibles en el mercado actual, resulta que las estructuras basadas en LUTs y PALs son más frecuentemente utilizadas en FPGAs de mayor granularidad, mientras que las basadas en multiplexores (Actel) son de granularidad baja. Los estudios de prestaciones y de eficiencia de área llevados a cabo hasta el momento [BRO92a] parecen confirmar la superioridad de las arquitecturas basadas en PALs y especialmente de las basadas en LUTs sobre las de menor granularidad y en especial sobre las de multiplexores, aunque la combinación del proceso de síntesis lógica e implementación suele dar mejores aprovechamientos de área en el caso de estructuras de baja granularidad, especialmente las basadas en puertas *nand*.

En cuanto al tipo de memoria de configuración utilizado, existen FPGAs configuradas mediante células de memoria RAM estática, RAM dinámica, PROM programada por fusibles o antifusibles, EPROM, e incluso flash o EEPROM [TRI94] [BRO92a]. En el pasado fueron muy populares las basadas en antifusibles [GAM89] debido al bajo coste en área del elemento de programación, si bien las basadas en RAM han terminado imponiéndose en los últimos tiempos debido a su mayor facilidad de programación y de reconfiguración dinámica. Estás últimas parecen ser las más indicadas para ser controladas mediante microprocesadores [LUC97][XIL97].

A.2.2.4 Rutabilidad externa

La rutabilidad externa (normalmente se entiende que es externa por defecto) intenta medir la probabilidad de rutar con éxito el conjunto de conexiones de los circuitos sobre las células programables ya emplazadas, y está estrechamente ligada de la cantidad y la disposición de recursos de rutado disponibles. La rutabilidad depende principalmente de cuatro factores:

- La cantidad de pistas de rutado N_p en cada canal, que puede ser distinta para los canales verticales y horizontales.
- La flexibilidad de los recursos de rutado (bloques RR de la figura A.2) F_{RR}, medida como el tanto por uno de pistas de rutado a los que se puede conectar cada conexión de la célula programable.
- La flexibilidad de las matrices de interconexión (bloques SM de la figura A.2) F_{SM} , medida como el número de pistas confluyentes a la matriz a las que se puede conectar una pista dada.
- El número T de puntos de conexión al exterior de que dispone cada entrada o salida del DMC. Si por ejemplo cada entrada puede conectarse desde el oeste o desde el sur (bloques RR de la figura A.2 situados a la izquierda y abajo de un DMC) del DMC, el parámetro T valdría 2.

Los estudios de Brown y Rose [ROS90b] [ROS91] [BRO92b] determinaron que, si bien las curvas de rutabilidad crecen monótonamente con F_{RR} y F_{SM} , el resultado prácticamente se estabiliza cuando F_{RR} llega a 0,7 (es decir, cada terminal puede conectarse a siete de cada diez pistas de rutado) con T igual a 2 (cada terminal se puede conectar a dos de los cuatro lados del DMC) y F_{SM} se limita a 3 (típicamente una conexión por cada lado restante). Por tanto, resulta más rentable según este estudio incrementar la rutabilidad de las conexiones de los terminales (F_{RR}) manteniendo relativamente limitada la rutabilidad de las matrices de conmutación (F_{SM}). El número de pistas por canal N_p debe ser superior al número teórico de pistas necesarias, medido a partir de estadísticas de rutado global sobre circuitos de prueba o *benchmarks* típicos, normalmente dos o tres pistas de más por canal.

La rutabilidad es uno de los factores que más decisivamente influyen en la eficiencia y prestaciones de una FPGA, y resulta de vital importancia para el buen funcionamiento de las herramientas automáticas de rutado de FPGAs.

A.2.2.5 Rutabilidad interna

Para una granularidad determinada, la célula programable dispondrá de un cierto número de LUTs y otro de *flip-flops* (normalmente coincidentes) que deberán poder conectarse con una determinada flexibilidad. Esta libertad a la hora de interconectar internamente los recursos programables disponibles es deseable ya que aumenta la flexibilidad de la arquitectura y simplifica en gran medida el proceso de *technology mapping* o mapeado de los circuitos diseñados mediante los recursos disponibles en el dispositivo programable. Por otra parte, los esquemas de interconexión flexibles son en general más lentos que los fijos debido a los retardos introducidos por los elementos de conmutación empleados, y lógicamente ocupan más área que una mera conexión directa. Incluso la conexión directa fija entre la salida de una LUT y la entrada de la siguiente dentro del mismo *cluster* resulta beneficioso para las prestaciones y no demasiado caro en área según [CHU91]. La rutabilidad interna resulta más importante en el caso de disponer de *flip-flops* que pueden configurarse en modos complejos de funcionamiento con varias entradas, tales como tipos J-K, multiplexor, etc. En estos casos el proceso de *technology mapping* puede complicarse considerablemente.

A.2.2.6 Accesibilidad

Una última característica cuya introducción parece interesante de cara a elegir una estructura especialmente indicada para disponer de interfaz con un microprocesador es la accesibilidad a las señales y configuración de la célula programable desde el bus de éste. Una estructura accesible nos permitiría hacer cosas tales como leer salidas combinacionales y secuenciales o sobreescribir los datos de los *flip-flops* como posiciones de memoria, mapear los datos de configuración de las LUTs o incluso de los recursos de rutado sobre el espacio de memoria del microprocesador, etc. Ninguno de los dispositivos programables dominantes en el mercado actualmente permite un gran grado de accesibilidad, a pesar de que es frecuente su utilización conjunta con un microcontrolador.

A.2.2.7 Evaluación de arquitecturas programables

Las conclusiones presentadas hasta el momento a partir de estudios realizados por diversos grupos de investigación sobre granularidades de LUTs, rutabilidades, compartición de entradas o tamaños de *clusters* han sido obtenidas utilizando un mismo método: se escoge un conjunto

variado de circuitos de prueba o *benchmarks*, típicamente algunos de los mantenidos por el Centro de Microelectrónica de Carolina del Norte para síntesis lógica [MCN91], y se implementan sobre una FPGA con un bloque programable al que se le va cambiando uno de los parámetros cada vez; finalmente se promedian los resultados obtenidos para cada valor del parámetro y se expresan en un gráfico.

Como puede verse, la metodología presenta limitaciones pues depende de la naturaleza de los circuitos de prueba escogidos, depende críticamente de la calidad del proceso de implementación (*technology mapping*, *placement* y *routing*) para cada valor del parámetro, y se basa en estimaciones de los retardos y áreas consumidas para cada caso ya que estos dependen de la implementación concreta de la estructura para cada combinación de valores de los parámetros. Sin embargo, es la única metodología posible hasta el momento y por tanto es utilizada y aceptada por la comunidad científica internacional.

Considerando estas limitaciones en la valoración cuantitativa de los parámetros definitorios introducidos, la idea de la situación de cada arquitectura programable en unas coordenadas determinadas del hiperespacio de soluciones definido por estas variables debería ser considerada solamente a título orientativo y cualitativo.

Finalmente es importante resaltar que el éxito final de una arquitectura programable depende dramáticamente en la calidad y facilidad de uso de su *software* asociado, lo cual explica que arquitecturas menos deseables desde un punto de vista puramente arquitectural hayan triunfado en el mercado sobre arquitecturas teóricamente superiores. Esto hace necesario tomar las decisiones sobre los parámetros arquitecturales en el marco de las herramientas *software* que las van a programar, pues es últimamente la combinación de ambos factores lo que dicta la economía del dispositivo en términos de puertas por dólar y las prestaciones del diseño (velocidad máxima de funcionamiento).

A.3 Arquitecturas analógicas programables

En contraste con el rápido desarrollo que las FPGAs han experimentado en los últimos tiempos, los circuitos analógicos programables se encuentran en una fase comparativamente menos avanzada. Esto podría deberse a que los subsistemas analógicos suelen estar sujetos a

especificaciones muy concretas que normalmente requieren células especialmente creadas o modificadas para una aplicación determinada, por lo cual los circuitos analógicos programables resultan de interés limitado ya que su utilización en aplicaciones de propósito general no siempre es posible.

Por otra parte, en los sistemas de señal mixta típicos usados en multitud de aplicaciones de control industrial es muy frecuente encontrar un bloque analógico cuya misión es acondicionar y convertir señales analógicas. Este tipo de bloque, llamado *front-end* analógico, se utiliza de forma habitual para adquisición de datos, y su estructura suele contener elementos de amplificación y filtrado *anti-aliasing* además de un convertidor analógico-digital o ADC. También es habitual encontrar algún conversor digital-analógico o DAC, para generar señales analógicas de salida.

Este tipo de estructuras de conversión son significativamente más reaprovechables para el abanico de aplicaciones típicas de control industrial, donde las señales suelen tener unos niveles y unos anchos de banda dentro de unos rangos no demasiado extensos. De esta manera, parece resultar más sencillo y ser más aplicable una estructura programable de acondicionamiento y conversión de señal que un *array* programable analógico de propósito totalmente general.

A.3.1 Estructura general de una célula analógica programable

La estructura básica de la célula que se repetiría bidimensionalmente en un verdadero *array* analógico dependería de la tecnología utilizada y de la granularidad deseada. Arquitecturas ya reportadas en este campo [AND97] [FRA96] utilizaban amplificadores diferenciales con lazos de realimentación configurables como célula básica, de tal forma que combinándolos mediante canales de rutado analógico de uso general podían realizarse sumadores analógicos, multiplicadores (moduladores), osciladores, filtros, etc.

Para una granularidad mayor con el propósito más específico de servir como *front-end* analógico de aplicaciones industriales de acondicionamiento de señal y adquisición de datos, el bloque analógico programable podría disponer de uno o varios canales de conversión formados por la secuencia amplificador – filtro *anti-aliasing* – conversor A/D y conversor D/A – filtro *anti-aliasing* – amplificador de salida.

A.3.2 Evolución histórica y estado del arte de las arquitecturas analógicas programables

Desde la misma aparición de los circuitos lógicos programables, también se comenzó a investigar hacia la consecución de circuitos analógicos configurables. A estos circuitos se los suele agrupar bajo la denominación FPAA (*Field Programmable Analog Array*), como contrapartida a las FPGAs. Los primeros circuitos analógicos programables, que intentaban emular a las FPGAs como sistemas de prototipado de aplicaciones analógicas [SIV88], abrieron nuevas áreas de investigación. Entre los precedentes en este campo, en general menos explorado, podemos destacar los siguientes:

- En 1991 Edward Lee presentó una FPAA implementada en tecnología CMOS [LEE91] especialmente pensada para implementar redes neuronales analógicas con pesos sinápticos programables. Esta arquitectura utilizaba estructuras CMOS analógicas polarizadas en regiones subcríticas para reducir el consumo. Más adelante, Lee propuso la utilización de trasconductores programables para la realización de funciones analógicas programables [LEE92]. Se proponía un bloque anlógico programable o CAB (*Configurable Analog Block*) que contenía un operacional con salida diferencial, y se utilizaban trasconductores programables para interconectar y realimentar CABs modulando impedancias. La trasconductancia de estos transistores de interconexión se programaba fijando tensiones de polarización que se almacenaban en memorias analógicas multivalor que usaban amplificadores de tensión o voltaje como *buffers*, o bien se obtenía a partir de funciones de las señales del circuito, lo cual servía para implementar osciladores controlados por voltaje. Más adelante se implementó un rediseño de este circuito con una estructura de trasconductores mejorada [LEE95], lo que conseguía mejores aprovechamientos de área.

- Entre 1995 y 1996 Se introdujeron las primeras FPAAs basadas en capacidades conmutadas, que utilizaban estructuras diferenciales especialmente pensadas para su compatibilidad con arquitecturas programables digitales complementarias. Así nacieron las FPMAs (*Field Programmable Mixed-Signal Array*) o *arrays* programables de señal mixta, que combinaban FPGAs y FPAAs en el mismo chip. Los casos más sobresalientes son la arquitectura propuesta por la Universidad de Toronto [CHO95] y la propuesta por Pilkington Micro-electronics Limited (PMeL) [ZHA96] [BRA96]. Motorola adquirió la tecnología programable analógica de Pilkington al igual que hiciera con sus arquitecturas de FPGA, y presentó en 1997 la primera FPAA de propósito general disponible a nivel comercial dentro de la familia

MPAAx020 [AND97]. Debido a la compatibilidad de los procesos utilizados por Motorola, es perfectamente posible el desarrollo de FPMAs que implementen las dos arquitecturas progrmables (la analógica y la digital) en un solo chip. Los bloques programables analógicos de estas familias contienen un amplificador operacional de altas prestaciones optimizado para la operación con capacidades conmutadas. La granularidad es relativamente baja, pudiéndose implementar secciones de amplificación y filtrado, osciladores, etc.

- En 1996 el Instituto Fraunhofer introdujo e incluso intentó explotar comercialmente un FPAD (*Field Programmable Analog Device*) llamado ASB (*Analog Silicon Breadboard*) [FRA96]. Se trataba de una estructura basada en células analógicas de relativamente alta granularidad pues contenían multiplicadores, sumadores, amplificadores y filtros programables, además de una serie de bloques de apoyo tales como un oscilador controlado por voltage, *choppers* para amplificadores, etc. La estructura se basaba internamente en operacionales diferenciales, y contenía canales de rutado analógico de propósito general, matrices de rutado analógico, etc., utilizando una estructura curiosamente similar a la de las FPGAs de media-alta granularidad basadas en LUTs.
- En 1996 IMP introdujo EPAC™ (*Electrically Programmable Analog Circuit*), otro dispositivo programable analógico disponible comercialmente [KLE96]. La filosofía de esta arquitectura es muy distinta a la utilizada por Pilkington, pues utiliza bloques de funcionalidad no configurable cuyas características sí son programables. Por ejemplo, dispone de secciones de amplificación cuya ganancia y *offset* son programables por el usuario, que además puede conectarlas de forma flexible. Este tipo de arquitecturas parece estar especialmente pensado para implementar *front-ends* de acondicionamiento y monitorización de señales analógicas, para lo que son necesarias las funciones de amplificación, comparación, y filtrado paso bajo de propósito general

A.4 Configuración de arquitecturas programables

La información que configura cada característica programable de la FPGA o de la FPAA puede almacenarse de distintas maneras. En los orígenes de los circuitos programables, esta información se grababa en el dispositivo mediante antifusibles al estilo de los primeros PLDs. Más adelante Xilinx introdujo sus LCAs en las que esta información se almacenaba en células de RAM estática, que tenían la ventaja de permitir reutilizar la FPGA para una aplicación distinta

sin más que reinicializar esta memoria y cargar una nueva configuración. Altera introdujo sus dispositivos programables basados en EPROM, que no permitían una reconfiguración tan sencilla como en las de Xilinx aunque resultaban útiles en entornos hostiles en los que la configuración no debía perderse aunque la tensión de alimentación fluctuase.

Una de las principales razones por las que las FPGAs basadas en RAM estática se han terminado imponiendo a las que utilizaban EPROM y antifusibles es la posibilidad de reconfigurar dinámicamente el dispositivo total o parcialmente. Esto resulta de gran utilidad en aplicaciones en las que el circuito programable debe realizar varias funciones distintas en distintos momentos del tiempo, y muy especialmente en el caso de implementar algoritmos secuenciales de varias etapas no solapadas en el tiempo y con una división muy clara entre ellas. En este tipo de aplicaciones, el circuito programable puede reconfigurarse al final de cada etapa para implementar los recursos *hardware* necesarios para la siguiente, bastando con un circuito programable para ejecutar un algoritmo que de otra forma necesitaría una serie de subsistemas fijos que sólo estarían activos un porcentaje del tiempo cada uno, ahorrándose por tanto área a costa de un cierto retraso debido al tiempo de reconfiguración entre cada etapa.

El área consumida por la circuitería de configuración de las FPGAs es altamente significativa, por lo que es deseable reducir en la medida de lo posible el número de bits de configuración necesarios para cada célula programable y los recursos de rutado y matrices de conmutación asociados a ella. Con argumentos matemáticos y estadísticos puede demostrarse que las cadenas de configuración de las FPGAs actualmente disponibles en el mercado son significativamente redundantes, lo cual provoca un empeoramiento del tiempo de reconfiguración y un desperdicio de área considerables [DeH96b].

A.4.1 Esquemas clásicos de configuración de células programables

El esquema más conocido y universalmente utilizado [LUC98] [XIL98] para la configuración de células programables es la circulación de un *bitstream* o cadena de *bits* serie de configuración sobre una ristra de registros de desplazamiento que enlaza todos los *flip-flops* que almacenan la información de configuración de toda la FPGA. La figura A.3 muestra cómo esta técnica se utilizaría para la configuración de una LUT de tres entradas implementada mediante un multiplexor de ocho a uno, cuya función lógica se almacenaría en un registro de desplazamiento de ocho *bits* por donde se circularía el *bitstream* serie de configuración.

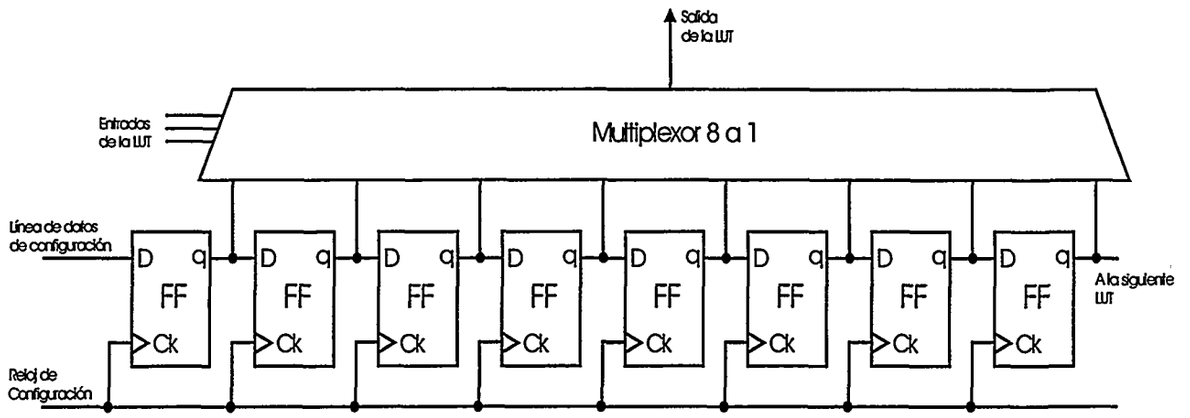

Fig. A.3: Implementación práctica de una LUT configurada mediante *bitstream* serie.

Esta técnica presenta varios problemas:

- Durante la configuración la célula debe permanecer totalmente inhabilitada ya que aparecerán sucesivamente configuraciones inconsistentes y erróneas de duración igual a un ciclo del reloj de configuración, lo cual podría producir daños físicos irreversibles al dispositivo si este reloj es suficientemente lento.
- Para hacer cualquier modificación, por mínima que ésta sea, hay que reconfigurar la célula entera recirculando la configuración completa, siendo necesario reconfigurar el dispositivo completo en la mayoría de los casos.
- La información de configuración no puede ser leída y comprobada puesto que leer los datos transmitidos implica recircular datos nuevos que sobreescribirían los comprobados. Además, cualquier error en el alineamiento de un sólo *bit* corrompería la configuración de toda la célula y habitualmente de todo el dispositivo, lo cual hace necesario un control exhaustivo del *stream* de configuración antes de transmitirlo.

En contraposición a este sistema de configuración basado en desplazamiento de *bits*, la familia 6200 de Xilinx incorpora un interfaz completo para microprocesador [CHU95] en el que los *bits* de configuración de la FPGA son transferidos y leídos como posiciones de memoria. El hecho de que la memoria de configuración está mapeada en un bus controlable por un microprocesador externo permitiría, al menos teóricamente, la incorporación de un microprocesador en el mismo chip. El interfaz de memoria de configuración es a su vez programable permitiendo trasferencias de hasta 32 bits simultáneos, lo que permite tiempos de reconfiguración del chip completo tan bajos como 100 μs. Además permite seleccionar varias células a la vez y escribir en ellas la

misma información, lo que acelera la reconfiguración de aplicaciones regulares como *arrays* sistólicos o estructuras segmentadas, espcialmente del tipo utilizado en aplicaciones de video.

A.4.2 Reconfiguración dinámica y parcial

Siguiendo la terminología introducida por Lysaght [LYS93], esquematizada en la figura A.4, una FPGA es dinámicamente reconfigurable si se pueden configurar subconjuntos de células programables sin detener la operación del resto. Según esta terminología, una estructura dinámicamente reconfigurable debe permitir la reconfiguración parcial (según Lysaght una FPGA sería parcialmente reconfigurable si para configurar un subconjunto dado de células programables no fuera necesario reconfigurar todo el *array*, aunque hubiera que parar toda la operación normal del *chip*).

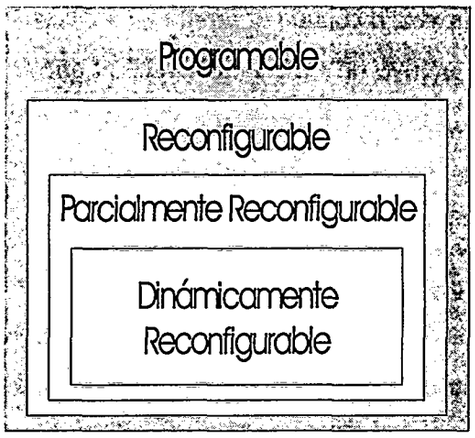

Fig. A.4: Clasificación de las FPGAs según su configurabilidad.

El hecho de que las FPGAs basadas en RAM sean reprogramables, aunque no sean dinámicamente reconfigurables según la definición anterior, ha sido uno de los factores clave por los que este tipo de FPGAs se han terminado imponiendo a las basadas en otras tecnologías. Se han reportado multitud de aplicaciones que utilizan FPGAs basadas en RAM que tienen distintas funciones en distintos instantes de tiempo [FAW93] [FOU93] [BRE95]. Todas las FPGAs basadas en RAM son por tanto reprogramables y pueden ser utilizadas para este tipo de aplicaciones, si bien las dos únicas arquitecturas dinámicamente reconfigurables de acuerdo con la nomenclatura introducida disponibles en el mercado son la familia AT6000 de Atmel [ATM98] y la XC6200 de Xilinx [XIL98]. Aunque menos numerosas, también se han reportado

aplicaciones en las que la reconfiguración dinámica es de gran utilidad, especialmente en circuitos regulares para tratamiento digital de señal [BUR96].

Finalmente, es interesante remarcar la gran importancia de la reconfiguración dinámica en entornos de computación general de alta capacidad [WIR96] [CAS93] y en el caso del fuerte acoplamiento entre arquitecturas dinámicamente programables y microprocesadores [BOL94] [DeH94] como forma de crear coprocesadores reconfigurables o rutas de datos con conjuntos de instrucciones modificables en función de las necesidades de cada momento.

A.4.3 Hardware *virtual y procedimientos* hardware

El *hardware* virtual [LYS93] nació en analogía a la memoria virtual o *swap* que utilizan las estaciones de trabajo al ejecutar grandes aplicaciones *software.* Cuando una estación de trabajo ejecuta varios programas a la vez, o cuando el programa a ejecutar necesita una gran cantidad de memoria, se dispone de un archivo de *swap* en disco en el que se almacenan todos los datos en memoria que no se están usando en cada momento. De esta forma, el espacio lógico de memoria visto por el microprocesador es más grande que la cantidad de memoria física reubicable que en realidad existe.

De igual forma, es habitual el caso de los circuitos integrados formados por varios subsistemas o bloques que no son usados al mismo tiempo, normalmente porque siguen un esquema de procesamiento secuencial en el que cada parte del circuito se activa de forma correlativa. En estos casos sería posible utilizar una FPGA que se reconfiguraría dinámicamente al inicio de cada tarea secuencial, de tal forma que sólo una tarea o parte del circuito estaría activa en cada momento. Bastaría entonces con utilizar una FPGA capaz de soportar la más grande de las tareas implicadas, sin ser necesario soportar toda la aplicación ya que el resto de las tareas estarían mapeadas en *hardware* virtual.

Utilizando esta técnica sería posible descomponer cualquier algoritmo o aplicación en etapas no solapadas en el tiempo que se mapearían y "ejecutarían" sobre la FPGA dinámicamente reconfigurable de forma correlativa, resultando esta ejecución más lenta debido al tiempo de reconfiguración y al paralelismo que se perdería entre algunas tareas, de forma parecida a intercambiar tiempo y memoria en ingeniería *software.* Por esto resulta importante reducir en la medida de lo posible el tiempo de reconfiguración, que para algunas arquitecturas dinámicamente reconfigurables especializadas en este tipo de aplicaciones se reduce a centenas e incluso a

decenas de microsegundos [ATM98]. La reducción de área debido a esta reutilización del *hardware* programable para distintas tareas puede llegar a ser espectacular en algunos casos [LYS93].

De manera análoga, se puede tratar una FPGA dinámicamente reconfigurable como un substrato programable equivalente a la memoria RAM sobre la que se ejcutarían programas o procedimientos cargados dinámicamente, como se esquematiza en la figura A.5 .

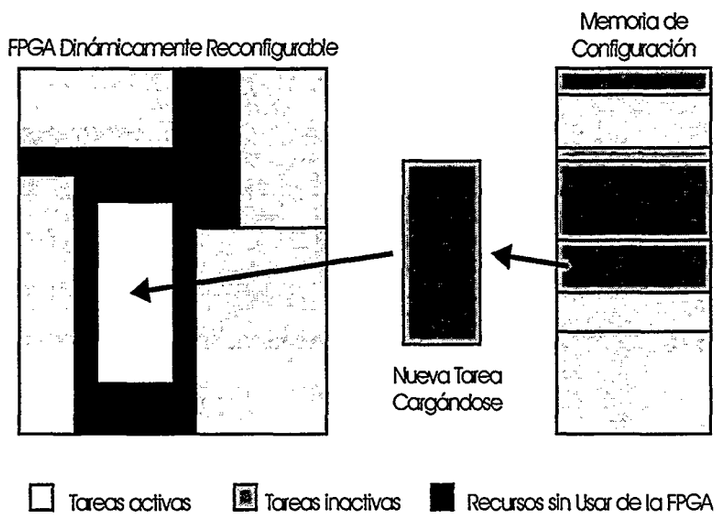

Fig. A.5: Procedimientos *hardware.*

En un esquema como el propuesto, habría una memoria de configuración que almacenaría una librería de tareas o procedimientos *hardware* que se copiarían en áreas vacías de la FPGA dinámicamente reconfigurable conforme fuera siendo necesario para la evolución del algoritmo o de la aplicación en curso. Como puede verse es necesario poder reconfigurar parcialmente la FPGA sin tener que detener el resto de la FPGA. Lysaght [LYS93] llama a esta ténica *hardware caching* en analogía a las memorias *caché* utilizadas por los ordenadores convencionales en las que el propio sistema controla y modifica dinámicamente sus propios recursos de memoria.

Más allá de los procedimientos *hardware* [HAS90], los objetos *hardware* [CAS95] tendrían asociadas cabeceras con parámetros de comunicación en un lenguaje de programación (típicamente C) de tal forma que podrían instanciarse sobre un sustrato reconfigurable general, típicamente memoria (para ser ejecutada por un microprocesador) o un FCCM (*Fast Custom Computing Machine*) [CAS93]. De esta manera podrían mezclarse e intercambiarse selectivamente *hardware* y *software* en función de la criticalidad de cada tarea del sistema.

A.4.4 Evolución histórica de la reconfiguración de arquitecturas digitales programables

Ciertas familias de circuitos programables se han especializado en aplicaciones de reconfiguración dinámica, entre las cuales podemos contar las siguientes:

- Uno de los primeros precedentes que utilizaban este tipo de técnicas es la arquitectura MAPL de National [HAW91]. Se trataba de una arquitectura PLD más o menos convencional basada en EECMOS, pero con la particularidad de que el circuito PLD de dos planos (AND y OR) estaba internamente repetido ocho veces, formando una arquitectura *paginada*. Sin más que cambiar la página activa en cada momento, el circuito se reconfiguraba implementando la función correspondiente a la página seleccionada. En realidad MAPL no estaba dirigida a aplicaciones de reconfiguración dinámica, sino que más bien intentaba utilizar los bits de selección de la página activa como entradas combinacionales adicionales, consiguiendo una arquitectura mucho más flexible que sus contemporáneas.
- En 1990 Tom Kean de Algotronix introdujo la familia CAL1024 [CAL90] cuya particularidad más notable era que exponía la memoria RAM de configuración de las células lógicas en un bus controlable por un microprocesador externo [KEA89] [GRA89]. Esto permitía la reconfiguración dinámica controlada por un microprocesador, que a base de escrituras en la RAM de configuración podía cambiar dinámicamente la funcionalidad del circuito, tanto de forma parcial como global. Algotronix fue adquirida por Xilinx en 1993 y las arquitecturas que habían desarrollado desembocaron en la familia XC6200 de Xilinx [XIL98], que básicamente tiene las mismas características que los primeros dispositivos de Algotronix, tanto en cuanto a la arquitectura del bloque lógico, de relativa baja granularidad, como de capacidad de interacción con un microprocesador externo.
- En 1993 National introdujo la familia CLAy (*Configurable Logic Array*) [CLA93], una de las primeras en permitir la reconfiguración parcial de sus módulos programables. La principal ventaja que se obtiene es la de poder reconfigurar en tiempo real partes de una aplicación sin tener que parar todo el chip mientras cambia su funcionalidad. De esta forma se pueden mapear de forma dinámica tareas *hardware* sobre el espacio libre en cada momento [HAS90].
- La familia AT6000 de Atmel [ATM98] fue una de las primeras en soportar de forma verdaderamente utilizable la reconfiguración parcial. Los bloques lógicos, de relativamente baja granularidad, se agrupan en bloques que pueden ser independientemente reconfigurados

mientras que el resto del circuito sigue funcionando normalmente. En general puede decirse que esta familia de FPGAs está especialmente pensada para aplicaciones de procesamiento digital de la señal, y particularmente para procesamiento de señales de vídeo. En este tipo de aplicaciones es necesario disponer de recursos *hardware* de altas prestaciones que realizan de forma secuencial distintas operaciones de procesamiento digital.

- Una de las arquitecturas que más directamente ataca el problema de la reconfiguración dinámica es la DPGA (*Dynamic Programmable Gate Array*) creada por André DeHon en el MIT [TAU95]. Esta arquitectura utiliza memoria dinámica para implementar distintos contextos o programaciones de los bloques lógicos. De esta manera, la reconfiguración del circuito o parte de él se reduce al cambio de contexto o selección de la página de la memoria dinámica que provee los datos de configuración, pudiéndose utilizar las entradas de control de los selectores de página como entradas del circuito lógico en una manera similar a la arquitectura MAPL [HAW91]. La posibilidad de escribir en los contextos no utilizados sin perturbar la operación del circuito permite realizar la reconfiguración sin tener que detener el funcionamiento del mismo. La utilidad de este tipo de circuitos en conjunción con microprocesadores también fue apuntada [DeH94]. De Hon encontró una reducción a un tercio del área utilizada por aplicaciones implementadas sobre su DPGA.

- En 1996 un equipo de la Universidad de California en Santa Barbara propuso la utilización de recursos de rutado multiplexados en el tiempo [LIN96], de forma que los datos de los distintos contextos se almacenarían en *flip-flops* que se interconectarían utilizando los mismos recursos de rutado en distintos momentos de tiempo. De esta forma se consigue un mejor aprovechamiento del área dedicada a interconexiones en los bloques programables, si bien los retardos de interconexión se incrementarían.

LISTA DE ACRÓNIMOS

ADC	*Analog to Digital Converter* (Conversor analógico-digital)
ASIC	*Application Specific Integrated Circuit* (Circuito integrado de aplicación específica)
CAB	*Configurable Analog Block* (Bloque analógico configurable)
CMRR	*Common Mode Rejection Ratio* (Coeficiente de rechazo al modo común)
DAC	*Digital to Analog Converter* (Conversor digital-analógico)
DCT	*Discrete Cosine Transform* (Transformada del coseno discreta)
DMC	*Digital Macro Cell* (Macrocélula digital)
DRC	*Design Rules Check* (Chequeo de reglas de diseño)
DSP	*Digital Signal Processor* (Procesador digital de señal)
CPLD	*Complex Programmable Logic Device* (Dispositvo lógico programable complejo)
EEPROM	*Electrically Erasable Programmable ROM*
EPROM	*Electrically Programmable ROM*
FCCM	*Fast Custom Computing Machine* (Máquina rápida para computación de propósito general)
FFT	*Fast Fourier Transform* (Transformada rápida de Fourier)
FIPSOC	*Field Programmable System on a Chip* (Sistema monochip programable por el usuario)
FPAA	*Field Programmable Analog Array* (*Array* analógico programable por el usuario)

FPAD *Field Programmable Analog Device* (Dispositivo analógico programable por el usuario)

FPGA *Field Programmable Gate Array* (*Array* de puertas programables por el usuario)

FPMA *Field Programmable Mixed-signal Array* (*Array* de señal mixta programable por el usuario)

IIC *Internal Interconnection Cell* (Célula de interconexión interna)

IOB *Input-Output Block* (Bloque de entrada-salida)

IOC *Input-Output Cell* (Célula de entrada-salida)

LVS *Layout Vs. Schematic* (*Layout* contra esquema)

PAL *Programmable Array Logic* (Array lógico programable)

PLA *Programmable Logic Array* (Array lógico programable)

PLD *Programmable Logic Device* (Dispositivo lógico programable)

PROM *Programmable ROM*

RAM *Random Access Memory* (Memoria de lectura y escritura)

ROM *Read Only Memory* (Memoria de sólo lectura)

SAR *Successive Approximation Register* (Registro de aproximaciones sucesivas)

SDF *Standard Delay Format* (Formato estandarizado de retardos)

SFR *Special Function Register* (Registro de función especial)

TLF *Timing Library Format* (Formato de temporización de librerías)

BIBLIOGRAFÍA

[ACT98] ACTEL Corp., *FPGA Databook*, 1998 (http://www.actel.com)

[ALT98] ALTERA Corp., *CPLD Databook*, 1998 (http://www.altera.com)

[AME96] R. Amerson et al, *"Plasma: An FPGA for Million Gate Systems"*, ACM/SIGDA 1996 International Symposium on Field Programmable Gate Arrays (FPGA'96), pp 10-16

[AND97] D. Anderson et al, *"A Field Programmable Analog Array and its Application"*, IEEE 1997 Custom Integrated Circuits Conference, May 1997

[ATH93] Peter Athanas and Harvey F. Silverman, *"Processor Reconfiguration Through Instruction-Set Metamorphosis"*, IEEE Computer, 26 (3), pp 11-18, March 1993.

[ATM98] ATMEL Corp., *FPGA Databook*, 1998 (http://www.atmel.com)

[AHR90] M. Ahrens, A. El Gamal, D. Galbraith, J. Greene et al, *"An FPGA Architecture Optimized for High Densities and Reduced Routing Delay"*, IEEE 1990 Custom Integrated Circuits Conference, July 1990.

[BAL91] Peter Baltus et al, *"A High Speed BiCMOS Table Look-up Gate"*, IEEE 1991 Custom Integrated Circuits Conference, May 1991, pp 6.3.1

[BEA96] Stephen Beavis, *"A Hierarchical Routing Structure Complementary with a Fine-Grained FPGA Architecture"*, The 4th Canadian Workshop on Field Programmable Devices (FPD'96), May 1996

[BET97] Vaughn Betz and Jonathan Rose, *"Cluster-Based Logic Blocks for FPGAs: Area Efficiency vs. Input Sharing and Size"*, IEEE 1997 Custom Integrated Circuits Conference, May 1997, pp 25.5.1

[BIR91] J. Birkner et al, *"A Very High-Speed Field Programmable Gate Array Using Metal-to-Metal Anti-fuse Programmable Elements"*, New Hardware Product Introduction at CICC '91, May 1991

[BOE95] Eduardo Boemo, *"Contribución al Diseño de Arrays VLSI con Paralelismo de Grano Fino"*, Tesis doctoral, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid, Noviembre 1995

[BOE96] E. Boemo, S. López-Buedo, and J. Meneses, *"The Wave Pipeline Effect on LUT-based FPGA Architectures"*, ACM/SIGDA 1996 International Symposium on Field Programmable Gate Arrays (FPGA'96)

[BOL94] Michael Bolotski, André DeHon and Thomas F. Knight, Jr., *"Unifying FPGAs and SMID Arrays"*, ACM/SIGDA 1994 International Symposium on Field Programmable Gate Arrays (FPGA'94)

[BRA96] Adrian Bratt and Ian Macbeth, *"Design and Implementation of a Field Programmable Analogue Array"*, ACM/SIGDA 1996 International Symposium on Field Programmable Gate Arrays (FPGA'96), pp 88-93

[BRE95] Gordon Brebner and John Gray, *"Use of Reconfigurability in Variable-Length Code Detection at Video Rates"*, 1995 Field-Programmable Logic and Applications (FPL'95)

[BRI94] Barry K. Britton et al, *"A Second Generation ORCA Architecture Utilizing 0.5μm Process Enhances the Speed and Usable Gate Capacity of FPGAs"*, IEEE International ASIC Conference and Exhibit, Sept 1994, pp. 474-478

[BRO92a] S. Brown, R. Francis, J. Rose and Z. Vranesic, *"Field Programmable Gate Arrays"*, The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, 1992.

[BRO92b] S. Brown, *"Routing Algorithms and Architectures for Field-Programmable Gate Arrays"*, Doctoral Dissertation, University of Toronto, 1992

[BUR96] Dave Bursky, *"Dynamically Reconfigurable FPGA Family Tackles Digital-Signal-Processing Applications"*, Electronica Design, April 15, 1996, pp 170, 172

[CAL90] CAL1024 Data Sheet, Algotronix Ltd., Edinburgh UK 1990.

[CAR86] W. Carter et al, *"A user programmable reconfigurable gate array"*, IEEE 1986 Custom Integrated Circuits Conference, May 1986, pp 233-235

[CAS93] Steve Casselman, *"Virtual Computing and the Virtual Computer™"*, IEEE 1993 Workshop on FPGAs for Custom Computing Machines, April 1993

[CAS95] Steve Casselman, Michael Thornburg and John Schewel, *"Creation of Hardware Objects in a Reconfigurable Computer"*, 1995 Field-Programmable Logic and Applications (FPL'95)

[CLA93] National Semiconductor, *Configurable Logic Array (CLAy) Data Sheet*, Dec 1993

[CHA96] Vi Cuong Chan and David M. Lewis *"Area-Speed Tradeoffs for Hierarchical Field-Programmable Gate Arrays"*, ACM/SIGDA 1996 International Symposium on Field Programmable Gate Arrays (FPGA'96), pp 51-57

[CHO95] P. Chwo, P. Chow & P.G.Gulak, *"A Field-Programmable Mixed-Analog-digital Array"*, ACM/SIGDA 1995 International Symposium on Field Programmable Gate Arrays (FPGA'95)

[CHU91] Kevin Chung, Satwant Singh, Jonathan Rose and Paul Chow, *"Using Hierarchical Logic Blocks to improve the Speed of FPGAs"*, FPGAs (proceedings of FPL'91), W R Moore & W Luk (eds.), © 1991 Abingdon EE&CS Books.

[CHU95] Stephen Churcher, Tom Kean, and Bill Wilkie, *"The XC6200 FastMapTM Processor Interface"*, 1995 Field-Programmable Logic and Applications (FPL'95)

[DeH94] André DeHon, *"DPGA-Coupled Microprocessors: Commodity ICs for the Early 21st Century"*, IEEE 1994 Workshop on FPGAs for Custom Computing Machines, pp 31-39, 1994

[DeH96a] André DeHon, *"Reconfigurable Architectures for General-Purpose Computing"*, Ph. D. Thesis, Technical Report #1586, MIT Artifical Intelligence Laboratory, September 1996.

[DeH96b] André DeHon, *"Entropy, Counting and Programmable Interconnect"*, ACM/SIGDA 1996 International Symposium on Field Programmable Gate Arrays (FPGA'96), pp. 73-79

[DOB95] Ivo Dobbeleare, Mark Horowith, and Abbas El Gamal, *"Regenerative Feedback Repeaters for Programmable Interconnections"*, IEEE Journal of Solid State Circuits, Vol 30, No 11, Nov 1995, pp. 1246-1253.

[EBE91] Carl Ebeling, Gaetano Borrielo et al, *"TRIPTYCH: a New FPGA Architecture"*, FPGAs (proceedings of FPL'91), W R Moore & W Luk (eds.), © 1991 Abingdon EE&CS Books. Este estudio fue republicado posteriormente en IEEE Transactions on Very Large Scale of Integration (VLSI) Systems, vol 3, No 4, Dec 1995

[FAW93] Bradly K. Fawcett, *"Applications of Reconfigurable Logic"*, More FPGAs (proceedings of FPL'93), W R Moore & W Luk (eds.), © 1994 Abingdon EE&CS Books

[FOU93] Patrick Foulk, *"Reconfigurable Computing with SRAM Programmable Gate Arrays"*, More FPGAs (proceedings of FPL'93), W R Moore & W Luk (eds.), © 1994 Abingdon EE&CS Books

[FRA96] Fraunhofer-Institut für Mikroelektronische Schaltungen und Systeme (Dr. W. Budde), *"Analog Silicon Breadboard – ABS 100"*, data sheet, 1996

[FUR90] F. Furtek, G. Stone and I.W. Jones, *"Labyrinth: A Homogeneous Computational Medium"*, IEEE 1990 Custom Integrated Circuits Conference, July 1990

[FUR92] Frederick Furtek, *"An FPGA Architecture for Massively Parallel Computing"*, 1992 Workshop on Field-Programmable Logic and Applications (FPL'92)

[GAM89] Abbas El Gamal, Jonathan Greene, Justin Reyneri, Eric Rogoyski, Khaled A. El-Ayat, and Amr Mohsen, *"An Architecture for Electrically Configurable Gate Arrays"*, IEEE Journal of Solid-State Circuits, Vol24, No 2, pp 394-398, April 1989

[GHA94] Ranjit Gharpurey, *"Modeling and Analysis of Substrate Coupling in ICs"*, PhD thesis, University of California at Berkely, 1994

[GOP93] Runip Gopisetty, Sanko H. Lan, *"Exploration of Multiplexer-based FPGA Logic Modules"*, More FPGAs (proceedings of FPL'93), W R Moore & W Luk (eds.), © 1994 Abingdon EE&CS Books

[GRA89] John Gray and Tom Kean, *"Configurable Hardware: A New Paradigm for Computation"*, Advanced Research in VLSI (Charles Seitz, ed.), 10th Decenial Caltech Conference on VLSI, pp. 279 – 195, March 1989

[HAM88] E. Hamdy et al, *"Dielectric based Antifuse for Logic and Memory ICs"*, IEDM Tech, Digest, San Francisco CA, 1998, pp. 768-789

[HAR94] Reiner Hartenstein, Rainer Kress and Helmut Reinig, *"An FPGA Architecture for Word-Oriented Datapaths"*, 1994 Canadian Workshop on Field-Programmable Devices, pp 2.4.1

[HAS90] Neil Hastie and Richard Cliff, *"The implementation of hardware subroutines on field programmable gate arrays"*, IEEE 1990 Custom Integrated Circuits Conference, July 1990

[HAU92] Scott Hauck, Gaetano Borriello, Steven Burns and Carl Ebeling, *"MONTAGE: An FPGA for Synchronous and Asynchronous Circuits"*, 1992 Workshop on Field-Programmable Logic and Applications (FPL'92)

[HAW91] David Hawley, *"Advanced PLD Architectures"*, FPGAs (proceedings of FPL'91), W R Moore & W Luk (eds.), © 1991 Abingdon EE&CS Books.

[HIL91] Dwight D. Hill and Nam-Sung Woo, *"The Benefits of Flexibility in Look-up Table FPGAs"*, FPGAs (proceedings of FPL'91), W R Moore & W Luk (eds.), © 1991 Abingdon EE&CS Books. Este estudio fue republicado posteriormente en IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol 12, No 2, Feb 1993

[HIL92] Dwight D. Hill et al, *"A New Architecture for High-Performance FPGAs"*, 1992 Workshop on Field-Programmable Logic and Applications (FPL'92)

[HOG96] Ron Hogervorst and Johan H. Huijsing, *"Design of low-voltage, low-power operational amplifier cells"*, Kluwer Academic Publishers, 1996

[HSI87] H. Hsieh et al, *"A Second Generation User Programmable Gate Array"*, IEEE 1987 Custom Integrated Circuits Conference, May 1987, pp.515-521

[HSI88] H. Hsieh et al, *"A 9000-Gate User-Programmable Gate Array"*, IEEE 1988 Custom Integrated Circuits Conference, May 1988, pp.15.3.1

[JHI93] Malkit S. Jhitta, *"Introduction of a New FPGA Architecture"*, More FPGAs (Proceedings of FPL'93), W.R.Moore & W Luk (eds.), pp. 13-23, © 1994 Abingdon EE&CS

[KAV96] Alireza Kaviani and Stephen Brown, *"Hybrid FPGA Architecture"*, ACM/SIGDA 1996 International Symposium on Field Programmable Gate Arrays (FPGA'96), pp. 3-9

[KAW90] Keiichi Kawan et al, *"An Efficient Logic Block Interconnect Architecture for User-Reprogrammable Gate Array"*, IEEE 1990 Custom Integrated Circuits Conference, July 1990, pp 31.3.1

[KEA89] Tom Kean, *"Configurable Logic: A Dynamically Programmable Cellular Architecture and its VLSI Implementation"*, Ph. D. Thesis, University of Edinburgh, January 1989.

[KHE94] Muhammad Khellah, Stephen Brown and Zvonko Vranesic, *"Minimizing Interconnection Delays in Array-Based FPGAs"*, 1994 IEEE Custom Integrated Circuits Conference, May 1994, pp. 181-184.

[KLE96] Hans W. Klein, *"The EPAC Architecture: An Expert Cell Approach to Field Programmable Analog Devices"*, ACM/SIGDA 1996 International Symposium on Field Programmable Gate Arrays (FPGA'96), pp 94-98

[KOU91] Jack L. Kouloheris and Abbas el Gamal, *"FPGA performance versus Cell Granularity"*, IEEE 1991 Custom Integrated Circuits Conference, pp. 6.2.1-6.2.4

[LAK94] Keneth Laker and Willy Sansen, *"Design of analog integrated circuits and systems"*, McGraw Hill, 1994

[LEE91] Edward K.F. Lee and P. Glenn Gulak, *"A CMOS Field-Programmable Analog Array"*, 1991 IEEE International Solid-State Circuits Conference (ISSCC'91), pp 186-187

[LEE92] Edward K.F. Lee and P. Glenn Gulak, *"Field Programmable Analogue Array Based on MOSFET Transconductors"*, Electronics Letters, vol 28, No 1, 2nd January 1992

[LEE95] Edward K.F. Lee and P. Glenn Gulak, *"A Transconductor-Based Field-Programmable Analog Array"*, 1995 IEEE International Solid-State Circuits Conference (ISSCC'95), pp 198-199

[LEE97] Miriam Leeser et al, *"Rothko: A Three Dimensional FPGA Architecture, Its Fabrication, and Design Tools"*, 1997 Field Programmable Logic and Applications (FPL'97), pp 21-30

[LIN96] Chih-chang Lin, Douglas Chang, Yu-Liang Wu, and Malgorzata Marek-Sadowska, *"Time-Multiplexed Routing Resources for FPGA design"*, IEEE 1996 Custom Integrated Circuits Conference CICC'96, pp. 8.4.1.

[LUC98] Lucent Technologies, *OrCA Databook*, 1998 (http://www.lucent.com/micro/fpga/)

[LYS93] Patrick Lysaght and John Dunlop, *"Dynamic Reconfiguration of FPGAs"*, More FPGAs (Proceedings of FPL'93), W.R.Moore & W Luk (eds.), pp. 82-94, © 1994 Abingdon EE&CS

[LYS96] P. Lysaght and J.A. Stockwood, *"A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays"*, IEEE Transactions on VLSI Systems, Sept. 1996.

[MA89] T. Ma, P. Dressendorfer, *"Ionizing Effects in MOS Devices and Circuits"*, Wiley Eds., New York 1989.

[MAL94] Franco Maloberti, *"Layout of Analog and Mixed Analog-Digital Circuits"*, in *Design of Analog-Digital VLSI Circuits for Telecommuncations and Signal Processing*, José E. Franca & Yannis Tsividis (Editors), © 1994 Prentice Hall

[MCN91] S. Yang, *"Logic Synthesis and Optimization Benchmarks User Guide - Version 3.0"*, Microelectronics Center of North Carolina, 1991

[MIY95] Toshiaki Miyasaki et al, *"Telecommunication-oriented FPGA and Dedicated CAD System"*, 1995 Field-Programmable Logic and Applications (FPL'95)

[MOR98] J.M. Moreno, J. Madrenas, J. Faura, E. Cantó, J. Cabestany, M.M. Insenser, *"Facing Evolutionary and Self-Repairing Hardware by means of the dynamic reconfiguration capabilities of the FIPSOC devices"*, International Conference on Evolvable Systems (ICES) 1998.

[MOR99] J.M. Moreno, J. Cabestany, E. Cantó, J. Faura, J.M. Insenser, *"The Role of Dynamic Reconfiguration for Implementing Artificial Neural Networks Models in Programmable Hardware"*, 5th International Work-Conference on Artificial and Natural Neural Networks (IWANN"9), Junio 1999.

[MOT92] Motorola Inc., *"MC68HC11 Datasheet"*, 1992.

[MOT93] C.D. Motchenbacher and J.A. Connelly, *"Low Noise Electronic System Design"*, Wiley-Interscience, 1993.

[MUL77] Richard S. Muller and Theodore I. Kamins, *"Device Electronics for Integrated Circuits"*, © 1997 by John Wiley & Sons, Inc.

[MUR91] H. Muroga et al, *"A Large Scale FPGA with 10K Core Cells with CMOS 0.8um 3-Layered Metal Process"*, IEEE 1991 Custom Integrated Circuits Conference, May 1991, pp 6.4.1

[PET83] E.L. Petersen, *"Single event update in space: basic concepts"*, Tutorial short course in IEEE Nuclear & Space Radiation Effects Conference (NSREC) 1997

[RAZ94] R. Razdan and M.D.Smith, *"A High-Performance Microarchitecture with Hardware-Programmable Functional Units"*, Micro 27, November 1994, pp.172-180

[RED94] S. Reddy et al, *"A High Density Embedded Array Programmable Logic Architecture"*, IEEE 1994 Custom Integrated Circuits Conference, May 1994, pp 9.2.1

[ROS90a] Jonathan Rose, Robert J. Francis, David Lewis, and Paul Chow, *"Architecture of Field-Programmable gate Arrays: The Effect of Logic Block Functionality on Area*

Efficiency", IEEE Journal of Solid-State Circuits, vol 25, No 5, pp 1217-1225, October 1990

[ROS90b] Jonathan Rose and Stephen Brown, *"The Effect of Switch Box Flexibility on Routability of Field Programmable Gate Arrays"*, 1990 IEEE Custom Integrated Circuits Conference, pp. 27.5.1-27.5.4

[ROS91] Jonathan Rose and Stephen Brown, *"Flexibility of Interconnection Structures for Field-Programmable Gate Arrays"*, IEEE Journal of Solid-State Circuits, vol 25, No 5, pp 1217-1225, October 1990

[ROY93] Kaushik Roy et al, *"Channel Architecture Optimization for Performance and Routability of Row-Based FPGAs"*, 1993 IEEE International Conference on Computer Design: VLSI in Computers & Processors, October 1993.

[SIN91] Satwant Singh et al, *"Optimization of Field-Programmable Gate Array Logic Block Architecture for Speed"*, IEEE 1991 Custom Integrated Circuits Conference, pp. 6.1.1-6.1.6

[SIN92] Satwant Singh, Jonathan Rose, Paul Chow and David Lewis, *"The Effect of Logic Block Architecture on FPGA Performance"*, IEEE Journal of Solid State Circuits, Vol 27, No3, March 1992, pp. 281-287

[SIN97] Satwant Singh et al, *"A New Synthesis Efficient, High Density and High Speed ORCA FPGA"*, IEEE 1997 Custom Integrated Circuits Conference, May 1997, pp. 25.3.1

[SIP96] M. Sipper, *"Designing evolware by cellular programming"*, Proc. of the 1st International Conference on Evolvable Systems: From Biology to Hardware (ICES96)

[SIP97] M. Sipper, *"Evolution of Parallel Cellular Machines, The Cellular Programming Approach"*, Springer-Verlag 1997

[SIV88] M.A. Sivilotti, *"A Dynamically Configurable Architecture for Prototyping Analog Circuits"*, in Advanced Research in VLSI: Proceedings of the Decennial Caltech Conference on VLSI, 1988. MA: MIT Press, 1988

[STA95] Anthony Stansfield and Ian Page, *"The Design of a New FPGA Architecture"*, 1995 Field Programmable Logic and Applications (FPL'95).

[TAU95] E. Tau, D. Chen, I. Eslick, J. Brown and A. DeHon, *"A First Generation DPGA Implementation"*, FPD'95 -- Third Canadian Workshop of Field-Programmable Devices, 1995 Montreal, Canada.

[TRE95] Nick Tredennick, *"Technology and Business: Forces Driving Microprocessor Evolution"*, Proceedings of the IEEE, vol 83, No12, pp 1641-1652, December 1995

[TRI94] Steven Trimberger, *"Field Programmable Gate Array Technology"*, © Kluwer Academic Publishers, Boston, 1994

[TRO86] Ronald R. Troutman, *"Latchup in CMOS Technology, The Problem and Its Cure"*, © Kluwer Academic Publishers, Boston, 1986

[WAN93] P.T. Wang, Y.T. Lai, and K.N. Chen, *"A Hierarchical Interconnection Structure for Field-Programmable Gate Arrays"*, IEEE 1993 TENCON, pp 557-560

[WAN94] Ping-Tsung Wang, Kun-Nen Chen, Yen-Tai Lai, *"A High Performance FPGA with Hierarchical Interconnection Structure"*, 1994

[WAZ93] M. Wazlowski et al, *"PRISM-II Compiler and Architecture"*, IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'93), April 1994

[WES93] Neil H.E. Weste and Kamran Eshraghian, *"Principles of CMOS VLSI Design: A Systems Perspective"*, © 1993 by AT&T, Addison-Wesley.

[WHA66] Sven Wahlstrom, *"Electronically Controlled Microelectronic Cellular Logic Array"*, US patent #3 473 160, 1966

[WHA67] Sven Wahlstrom, *"Programmable logic arrays – cheaper by the millions"*, Electronics, pp. 90-95, Dec. 1967

[WHI93] Telle Whitney and Jeff Schlageter, *"A New High Performance Field Programmable Gate Array Familiy"*, 1993 IEEE International Conference on Computer Design: VLSI in Computers & Processors, October 1993

[WIR96] Michael J. Wirthlin and Brad L. Hutchings, *"Sequencing Run-Time Reconfigured Hardware with Software"*, ACM/SIGDA 1996 International Symposium on Field Programmable Gate Arrays (FPGA'96), pp 122-128

[WIT95] Ralph D. Wittig, *"OneChip: An FPGA Processor With Reconfigurable Logic"*, M.A.Sc Thesis, University of Toronto, 1995

[XIL98] Xilinx Corp., *FPGA Databook*, 1998, (http://www.xilinx.com)

[ZHA96] Chenggjin Zhang, Adrian Bratt and Ian Macbeth, *"A New Field Programmable Mixed-Signal Array And Its Application"*, The 4th Canadian Workshop on Field Programmable Devices (FPD'96), May 1996

Reunido el tribunal que suscribe en el día
de la fecha, acordó calificar la presente Tesis
doctoral con SOBRESALIENTE Cum Laude POR UNANIMIDAD
Madrid, 1 - Junio - 2001

ANTONIO GARCÍA GUERRA

A. TORRADO

JOAN CARLES MARTORELL

Fco Pelayo