## Universidad Rey Juan Carlos

**Departamento de Arquitectura y Tecnología de Computadores, Ciencias de la Computación e Inteligencia Artificial**


# Sistemas de multiprocesamiento simétrico sobre FPGA

## Tesis Doctoral


**Autor: Pablo Huerta Pellitero**

**Director: José Ignacio Martínez Torre**

**Escuela Técnica Superior de Ingeniería Informática**


**Mayo de 2009**

# Agradecimientos

En primer lugar quiero dar las gracias a todos los compañeros del Grupo de Diseño Hardware/Software: Javier, Carlos, César, y especialmente a Nacho por todo el trabajo y apoyo que me ha dado.

Quisiera además agradecer a todos los compañeros del Departamento, por todos los buenos ratos pasados a lo largo de estos años, y por los buenos ratos que sin duda pasaremos en el futuro.

Un agradecimiento especial para mis padres, que me han animado a seguir adelante durante todos estos años.

A MI FAMILIA

A MI NOVIA

# Índice:

**1 INTRODUCCIÓN ........................................ 1**

*1.1 Contexto de la tesis ........................................ 1*

*1.2 Motivación y objetivos de la tesis ........................................ 4*

**2 COMPUTACIÓN PARALELA Y SISTEMAS MULTIPROCESADOR ........ 7**

*2.1 Computación paralela ........................................ 7*

2.1.1 Organización de la memoria y modelos de comunicación en sistemas MIMD 8

*2.2 Multiprocesamiento simétrico o SMP ........................................ 10*

*2.3 Algunos sistemas actuales ........................................ 11*

2.3.1 Ordenadores personales ........................................ 12

2.3.2 Videoconsolas ........................................ 12

2.3.3 Sistemas Empotrados ........................................ 14

**3 FPGAs Y SOFT-CORE PROCESSORS ........................................ 15**

*3.1 FPGAs ........................................ 15*

*3.2 Diseño basado en plataforma ........................................ 18*

*3.3 Soft-Core Processors ........................................ 21*

3.3.1 PicoBlaze ........................................ 22

3.3.2 MicroBlaze ........................................ 23

3.3.3 Nios II ........................................ 25

3.3.4 LatticeMico32 ........................................ 27

3.3.5 OpenRISC 1200 ........................................ 29

3.3.6 LEON 2 y LEON 3 ........................................ 31

**4 SISTEMAS MULTIPROCESADOR EN FPGA ........................................ 33**

*4.1 Arquitecturas MPSoC en FPGA* ... *33*

*4.2 Retos en el diseño de sistemas MPSoC en FPGA* ... *36*

4.2.1 Comunicaciones ... 36

4.2.2 Memoria ... 36

4.2.3 Software ... 37

*4.3 Algunos sistemas existentes* ... *37*

**5 OBJETIVOS ... 45**

**6 SISTEMA MULTIPROCESADOR CON COMUNICACIONES PUNTO A PUNTO 47**

*6.1 El bus FSL* ... *47*

*6.2 Test de comunicaciones* ... *48*

*6.3 Topologías para una red de MicroBlazes utilizando el bus FSL* ... *51*

6.3.1 Red completamente mallada ... 51

6.3.2 Red en anillo ... 52

6.3.3 Red en estrella ... 53

*6.4 Sistema completo* ... *53*

6.4.1 Multiplicación de matrices ... 54

6.4.2 Aplicación criptográfica ... 56

*6.5 Implementación en FPGA* ... *57*

*6.6 Conclusiones* ... *58*

**7 SISTEMA SMP, CON SISTEMAS OPERATIVOS INDEPENDIENTES EN CADA PROCESADOR ... 61**

*7.1 Retos Software* ... *62*

7.1.1 Transparencia ... 62

7.1.2 Política de planificación ... 62

7.1.3 Comunicación entre procesadores ... 62

7.1.4 Sincronización entre procesadores ... 63

*7.2 Arquitectura Hardware ... 63*

7.2.1 Hardware mutex ... 64

*7.3 Arquitectura software ... 65*

*7.4 Escribiendo aplicaciones ... 67*

*7.5 Resultados experimentales ... 68*

7.5.1 Resultados de síntesis ... 69

7.5.2 Tests hardware y software ... 70

*7.6 Conclusiones y problemas encontrados en el sistema ... 71*

**8 SISTEMA OPERATIVO PARA SISTEMAS SMP EN FPGA ... 73**

*8.1 Requisitos de la arquitectura hardware ... 73*

8.1.1 Hardware Abstraction Layer (HAL) ... 74

*8.2 Estructura del sistema operativo con soporte para SMP ... 76*

8.2.1 Estructuras de datos que utiliza el S.O ... 76

8.2.2 Rutinas principales ... 79

8.2.2.1 Inicialización ... 79

8.2.2.2 Interrupciones ... 82

8.2.2.3 Planificación ... 84

8.2.2.4 Llamadas al sistema ... 88

8.2.3 API del sistema operativo ... 91

8.2.3.1 Manejo de hilos ... 91

8.2.3.2 Semáforos ... 92

8.2.3.3 Mutex ... 92

*8.3 Resultados experimentales ... 93*

8.3.1 Plataforma hardware ... 93

8.3.2 Tests software ... 95

*8.4 Conclusiones ... 97*

**9 RENDIMIENTO DE DISTINTOS SISTEMAS SMP SOBRE FPGA ... 99**

*9.1 Aplicaciones software para evaluación de rendimiento ... 99*

9.1.1 Aplicación 1 ... 99

9.1.2 Aplicación 2 ... 102

*9.2 Sistemas SMP evaluados sobre FPGA ... 104*

9.2.1 Consideraciones previas ... 104

9.2.2 Métricas utilizadas para la evaluación de rendimiento ... 107

9.2.3 Sistema 1 ... 107

9.2.3.1 Resultados de síntesis ... 109

9.2.3.2 Resultados de los tests software ... 110

9.2.3.3 Conclusiones ... 113

9.2.4 Sistema 2 ... 113

9.2.4.1 Resultados de síntesis ... 114

9.2.4.2 Resultados de los tests software ... 115

9.2.4.3 Conclusiones ... 122

9.2.5 Sistema 3 ... 122

9.2.5.1 Arquitectura ... 122

9.2.5.2 Resultados de síntesis ... 124

9.2.5.3 Resultados de los tests software ... 126

9.2.5.4 Conclusiones ... 130

9.2.6 Sistema 4 ... 131

9.2.6.1 Arquitectura ... 131

9.2.6.2 Resultados de síntesis ... 133

9.2.6.3 Resultados de los tests software ...................................................... 134

9.2.6.4 Conclusiones ...................................................................................... 139

*9.3 Conclusiones ........................................................................................... 140*

**10 CONCLUSIONES Y TRABAJO FUTURO ........................................... 141**

*10.1 Principales aportaciones de la tesis ........................................................... 141*

*10.2 Trabajo actual y futuro ............................................................................ 142*

*10.3 Publicaciones ........................................................................................ 142*

10.3.1 Publicaciones relacionadas con la tesis ................................................. 142

10.3.2 Otras publicaciones ............................................................................ 143

**11 BIBLIOGRAFÍA ........................................................................................ 145**

# Índice de Figuras:


Figura 1: costes frente a número de unidades. 3

Figura 2: estructura básica de un sistema con memoria centralizada (11). 9

Figura 3: estructura básica de un sistema con memoria distribuida (11). 9

Figura 4: clasificación de Flynn-Johnson (15). 11

Figura 5: procesadores para equipos de sobremesa, Intel Core 2 Quad (izq) y AMD PhenomX4(dcha). 12

Figura 6: arquitectura interna del procesador Xenon (16). 13

Figura 7: esquema del procesador CELL (17). 13

Figura 8: estructura de interconexión (arriba) y detalle de un CLB (abajo) en la familia 2000 de Xilinx. 15

Figura 9: brecha de diseño (34). 19

Figura 10: flujo de diseño bidireccional. 20

Figura 11: estructura interna del microcontrolador PicoBlaze (38). 22

Figura 12: arquitectura interna del procesador MicroBlaze con sus unidades opcionales. 24

Figura 13: interfaz de la herramienta Xilinx Platform Studio. 24

Figura 14: arquitectura del procesador Nios II (40). 26

Figura 15: interfaz de la herramienta System on Programmable Chip Builder. 27

Figura 16: arquitectura del procesador LatticeMico32 (41). 28

Figura 17: interfaz de la herramienta Mico System Builder. 29

Figura 18: arquitectura OpenRISC (42). 30

Figura 19: arquitectura del procesador LEON 2 (44). 31

Figura 20: arquitectura multiprocesador con procesadores independientes. 33

Figura 21: sistema multiprocesador con comunicación entre los procesadores a través de un bus específico (arriba), y memoria compartida (abajo). ........ 34

Figura 22: arquitectura multiprocesador maestro-esclavo. ........ 35

Figura 23: arquitectura multiprocesador segmentada. ........ 35

Figura 24: arquitectura SoCrates (72). ........ 38

Figura 25: arquitectura del sistema de procesamiento masivo de datos (74). ........ 39

Figura 26: arquitectura de un sistema multiprocesador para codificación de vídeo (75). ........ 39

Figura 27: sistema heterogéneo con *hard-core* y *soft-core processors* (77). ........ 40

Figura 28: sistema multiprocesador heterogéneo segmentado para procesamiento de paquetes IP (78). ........ 41

Figura 29: arquitectura SMP con módulo de control de coherencia de cachés (79). ........ 42

Figura 30: arquitectura de CerberO (68). ........ 43

Figura 31: estructura del bus FSL. ........ 48

Figura 32: sistema utilizado para los tests de comunicaciones. ........ 49

Figura 33: velocidad del bus FSL en transferencia directa de datos. ........ 49

Figura 34: velocidad de transferencia para matrices de distintos tamaños. ........ 51

Figura 35: red completamente mallada de procesadores MicroBlaze. ........ 52

Figura 36: red en anillo de procesadores MicroBlaze. ........ 52

Figura 37: red en estrella (izq) y unión de varias redes en estrella (dcha). ........ 53

Figura 38: speedup y eficiencia para la multiplicación de matrices de enteros. ........ 55

Figura 39: speedup y eficiencia para la multiplicación de matrices de flotantes. ........ 55

Figura 40: resultados de speedup y eficiencia para la aplicación criptográfica. ........ 57

Figura 41: arquitectura del sistema implementado. ........ 64

Figura 42: sistema con 2 procesadores ejecutando varias tareas. ........ 67

Figura 43: mapa de memoria del sistema. ..................... 68

Figura 44: placa ML403 de Xilinx. ..................... 69

Figura 45: funcionamiento incorrecto del terminal debido al acceso simultáneo de los 2 procesadores. ..................... 70

Figura 46: funcionamiento correcto del terminal utilizando un *hardware-mutex* ..................... 71

Figura 47: sistema SMP utilizando procesadores soft-core MicroBlaze. ..................... 75

Figura 48: estructura de datos para almacenar el estado de un proceso. ..................... 77

Figura 49: estructura con la tabla de procesos del sistema. ..................... 78

Figura 50: estructura con los procesos asociados a cada procesador. ..................... 79

Figura 51: diagrama de flujo de la inicialización del S.O. ..................... 81

Figura 52: funcionamiento de la rutina de atención a la interrupción. ..................... 84

Figura 53: transiciones entre los distintos estados de un proceso. ..................... 86

Figura 54: diagrama de flujo del planificador. ..................... 87

Figura 55: acceso a las secciones críticas del planificador mediante el *hardware-mutex.* ..................... 89

Figura 56: funcionamiento de las llamadas al sistema. ..................... 90

Figura 57: plataforma para las pruebas del S.O. ..................... 94

Figura 58: placa RC300 de Celoxica. ..................... 95

Figura 59: salida de la aplicación de prueba. ..................... 96

Figura 60: cuanto menor sea el *time-slice*, más frecuentemente se dará la situación en la que varios procesadores están bloqueados. ..................... 106

Figura 61: cuanto mayor sea el *time-slice* más tiempo medio pasa desde que se crea una tarea hasta que se empieza a ejecutar, aún no habiendo otras tareas en cola. ..................... 106

Figura 62: arquitectura del sistema 1. ..................... 109

Figura 63: speedup y eficiencia para la configuración base. ..................... 111

Figura 64: comparación del speedup de la configuración base y la configuración 1. ................112

Figura 65: arquitectura del sistema 2. ........................................................................114

Figura 66: resultados de los tests para el sistema base. ...........................................................117

Figura 67: comparativa del speed-up para las distintas configuraciones. .................................121

Figura 68: controlador de memoria MultiChannel External Memory Controller, de Xilinx (85). ........................................................................................................................................123

Figura 69: arquitectura del sistema 3. ........................................................................124

Figura 70: placa de desarrollo ML401. ........................................................................125

Figura 71: resultados de speedup y eficiencia en el sistema 3 con la configuración base. ........127

Figura 72: comparativa del speedup para las distintas configuraciones, con 2 y 4 procesadores. ........................................................................................................................................130

Figura 73: controlador de memoria MultiChannel DDR de Xilinx (86). .....................................131

Figura 74: arquitectura del sistema 4. ........................................................................133

Figura 75: resultados de speedup y eficiencia para la configuración base. ..............................136

Figura 76: comparativa de speedup para las distintas configuraciones. ...................................139

# Índice de Tablas:

Tabla 1: transmisión de 5 palabras mediante llamadas consecutivas a la función de transmisión. ....... 50

Tabla 2: ensamblador del código de la Tabla 1. ....... 50

Tabla 3: resultados de síntesis para el sistema completo. ....... 58

Tabla 4: resultados de síntesis del sistema. ....... 69

Tabla 5: HAL para el sistema de la Figura 47. ....... 76

Tabla 6: aplicación de prueba. ....... 96

Tabla 7: parámetros de cada hilo de la aplicación 1. ....... 100

Tabla 8: código del hilo que calcula una parte del resultado en la aplicación 1. ....... 101

Tabla 9: programa principal de la aplicación 1. ....... 101

Tabla 10: parámetros de cada hilo en la aplicación 2. ....... 102

Tabla 11: código de los hilos de la aplicación 2. ....... 103

Tabla 12: programa principal de la aplicación 2. ....... 103

Tabla 13: resultados de síntesis del sistema 1. ....... 110

Tabla 14: relación de *slices*, *LUTS* y puertas equivalentes para 1 y 2 procesadores. ....... 110

Tabla 15: resultados para la configuración base. ....... 111

Tabla 16: resultados de la configuración 1. ....... 112

Tabla 17: resultados de síntesis del sistema 2 para 8 procesadores. ....... 115

Tabla 18: comparativa de *slices, LUTS* y puertas equivalentes para distintos números de procesadores en el sistema. ....... 115

Tabla 19: resultado de los tests para el sistema base. ....... 116

Tabla 20: resultados de los tests en las configuraciones 1 y 2. ....... 119

Tabla 21: resultados de los tests en las configuraciones 3 y 4. ................................................120

Tabla 22: resultados de síntesis para el sistema 3 con 4 procesadores. ..................................125

Tabla 23: comparativa de ocupación de *slices, LUTS* y puertas equivalentes para distintos números de procesadores. ..........................................................................................................125

Tabla 24: resultados del sistema 3 con la configuración base. ................................................126

Tabla 25: resultados de las distintas configuraciones. .............................................................129

Tabla 26: resultados de síntesis para el sistema con 4 procesadores. .....................................134

Tabla 27: comparativa de la ocupación de *slices¸LUTS* y puertas equivalentes para diferentes números de procesadores. ..........................................................................................................134

Tabla 28: resultados de los tests para la configuración base. ..................................................135

Tabla 29: resultados de los tests para las distintas configuraciones. .......................................138

# 1 INTRODUCCIÓN

## 1.1 Contexto de la tesis

Una de las tendencias de los últimos años, dentro del desarrollo de todo tipo de sistemas computacionales es el acercamiento a arquitecturas que usan varios procesadores, en lugar de uno solo, para obtener un mejor rendimiento. Este tipo de sistemas multiprocesador no se utiliza únicamente en el ámbito de supercomputadores y otro tipo de sistemas de altas prestaciones, si no que cada vez más se usan en todo tipo de máquinas, desde PCs de sobremesa y consolas de videojuegos, hasta reproductores multimedia y otros sistemas empotrados como él IPOD de Apple, que en sus modelos más recientes utiliza un circuito integrado PP5021C-TDF con dos procesadores ARM en su interior funcionando a 80 Mhz.

Por otra parte el rápido crecimiento de capacidad tanto de los dispositivos lógicos programables, como de los circuitos de aplicación específica (ASICs), hace posible el desarrollo de sistemas cada vez más complejos. Actualmente es normal ver sistemas complejos compuestos por multitud de elementos en un único circuito integrado. Este tipo de sistemas se conocen como System On a Chip (SoC). De esta forma, sistemas que antes se realizaban utilizando varios chips ahora se realizan en un solo circuito integrado. Además, en los últimos años los dispositivos lógicos programables se han convertido en una alternativa real a los ASICs, debido tanto al aumento de capacidad y funcionalidades, como a la reducción de su precio.

Los primeros dispositivos lógicos programables existen desde la década de los 70, cuando hicieron su aparición los primeros PAL (*Programmable Logic Array),* pero no es hasta la aparición de las FPGAs (*Field Programmbale Gate Array)* en la década de los 80 cuando estos han empezado a cobrar verdadera importancia en el mercado de los semiconductores.

Hasta esa fecha los dispositivos disponibles en sus diversas variantes, PAL, PLD, CPLD, no ofrecían la capacidad suficiente para implementar aplicaciones complejas en su interior, limitándose por tanto su uso a funciones muy básicas como pueden ser por ejemplo decodificación de memoria o enmascaramiento de interrupciones. Este tipo de

aplicaciones primigenias recibían el nombre de *glue logic* debido a su función de adaptar (pegar) un sistema a otro (ej. un microprocesador a una memoria).

Desde las primeras FPGAs aparecidas en los 80 hasta las que se comercializan en la actualidad, han aparecido multitud de mejoras e innovaciones que permiten implementar sistemas cada vez más grandes y complejos dentro de ellas. Algunas de estas mejoras van desde bloques de memoria o multiplicadores embebidos dentro de la propia FPGA, hasta microprocesadores completos como los PowerPC que incluyen algunas FPGAs de las familias Virtex2 Pro, Virtex 4 y Virtex 5 de Xilinx.

A pesar de que por su propia naturaleza, las FPGA están muy lejos de las prestaciones ofrecidas por los ASICs, éstas ofrecen otra serie de importantes ventajas que hacen que su uso se haya popularizado en las pequeñas y medianas empresas que carecen de los recursos necesarios para realizar implementaciones en ASIC. Estas ventajas se pueden resumir en los siguientes puntos:

- El coste de realizar un diseño en una FPGA es mucho menor que en un ASIC.
- Las herramientas de diseño para FPGA son más baratas que las correspondientes para ASIC.
- El tiempo de desarrollo hasta la llegado al mercado de un diseño basado en FPGA es mucho menor que el equivalente basado en ASIC.
- En una FPGA es posible corregir un error en el diseño, incluso cuando ya ha sido lanzado el producto final.

Por el contrario la implementación de un diseño en un ASIC es ventajosa cuando el volumen de circuitos a fabricar es elevado o éstos requieren trabajar a altas velocidades. En la Figura 1 se muestra una aproximación del coste total frente al número de unidades fabricadas.

Como puede verse, uno de los grandes atractivos del diseño con FPGA es que para pequeños volúmenes su coste es muy inferior al de los ASIC tradicionales o los más actuales ASIC estructurados (1) pero en cuanto el número de unidades crece, esta ventaja desaparece. De nuevo, apelando únicamente al coste de fabricación sin tener en cuenta otras características, las FPGA han supuesto un decisivo impulso para las pequeñas y medianas empresas con bajos volúmenes de producción.

Aun siendo la capacidad de las FPGA muy inferior a la de los ASIC o incluso a la de los ASIC estructurados, ésta es a día de hoy lo suficientemente grande como para poder albergar en su interior diseños de gran magnitud. Hasta tal punto es así que la tendencia actual es integrar dentro de una FPGA sistemas digitales completos que incluyen un microprocesador de propósito general y todo el hardware de propósito específico que requiere la aplicación. Estos sistemas implementados sobre dispositivos programables se han dado en denominar SoPC *(System on Programmable Chip)* derivado del término SoC *(System on Chip)*, definiendo un SoC como "un circuito integrado formado por diversos módulos VLSI con distinta funcionalidad que interconectados entre sí ofrecen una funcionalidad específica para una aplicación" (2).

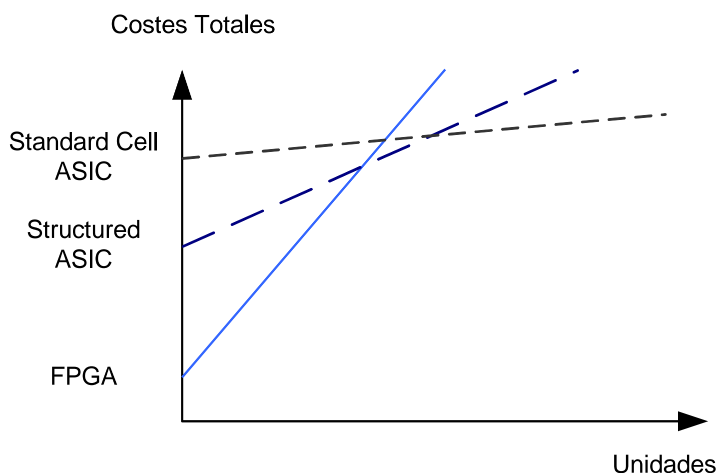


Figura 1: costes frente a número de unidades.

Esta tendencia a integrar módulos VLSI dentro de un circuito ha hecho necesaria la redefinición de las metodologías de diseño, haciendo necesaria la aparición de nuevos mecanismos de interacción hardware-software, así como una nueva serie de lenguajes de diseño (3)(4) que facilitan la generación de modelos de alto nivel de estos sistemas de forma rápida y eficiente, facilitando también la reusabilidad de los diferentes bloques que conforman el sistema. Estos bloques se conocen como bloques IP (*Intellectual Property*) (5). Dentro de los bloques IP una nueva tendencia es la aparición de bloques IP de código abiertos basados en la misma filosofía que los programas software de código abierto. Es sencillo encontrar por la red comunidades de usuarios que ofrecen bloques IP de alta calidad listos para ser integrados dentro de un SoC. Entre estas comunidades cabe destacar OpenCores que como bloque IP estrella ofrece el microprocesador OpenRISC, además de

multitud de diseños fácilmente integrables en cualquier SoC gracias a su interfaz de bus Wishbone (6).

Un paso más en la dirección de reducir al máximo los tiempos de desarrollo de los sistemas electrónicos es el llamado diseño basado en plataforma entendiendo como tal, "un circuito integrado flexible donde la aplicación se realiza programando uno o más componentes del chip" (7). Dentro del mundo de las FPGA los fabricantes ponen ya a disposición de los clientes plataformas de diseño donde sólo deben escribir el software y el hardware específico para su aplicación dejando de lado el esfuerzo de integrar microprocesadores, controladores de memoria, y jerarquías de buses, pudiendo todo esto ser configurado utilizando una herramienta gráfica.

Una de las innovaciones que se ha puesto de moda en los últimos años dentro del mundo del diseño de SOCs, son los *soft-core processors*: microprocesadores completamente descritos en un lenguaje de descripción de hardware como VHDL o Verilog, y orientados a ser integrados dentro de un diseño con FPGA, aunque también pueden ser sintetizados dentro de un diseño para ASIC, como sucede con el *soft-core* OpenRISC que ha sido usado satisfactoriamente tanto en proyectos en FPGA (8) como en proyectos ASIC.

## 1.2 Motivación y objetivos de la tesis

Los sistemas electrónicos modernos demandan cada día mayor capacidad de cómputo. Por ejemplo, las aplicaciones de red requieren dispositivos capaces de ofrecer un buen rendimiento a velocidades de transferencia del orden de gigabits por segundo. También los sistemas multimedia están aumentando sus necesidades de potencia de cálculo debido principalmente tanto al incremento de sus funciones (codificación/decodificación de audio y video, conectividad a redes, etc.) como a la aparición de nuevos algoritmos de compresión/descompresión de audio y video que ofrecen mayor calidad a costa de una mayor complejidad y costo computacional.

Estos dos grupos de sistemas son un buen ejemplo del tipo de aplicaciones embebidas más implementadas habitualmente. Muchos de estos sistemas embebidos suelen estar desarrollados en forma de SoC que materializa las funciones necesarias mediante una solución integrada hardware/software que utiliza un procesador de propósito general y una serie de coprocesadores, buses, memorias y periféricos.

Con el aumento de las necesidades computacionales de los sistemas embebidos actuales, los procesadores de propósito general se ven desbordados y se hace necesario buscar soluciones alternativas. Una de las posibles soluciones pasa por utilizar varios procesadores para poder hacer frente a toda la carga de trabajo de los sistemas modernos.

Las FPGAs actuales tienen los suficientes recursos para poder implementar en ellas sistemas multiprocesador complejos utilizando *soft-core processors*, así como distintas arquitecturas de memoria y comunicaciones para este tipo de sistemas, por lo que son una plataforma ideal para evaluar este tipo de sistemas.

El objetivo de esta tesis es evaluar las distintas posibilidades que ofrecen las FPGAs a la hora de diseñar e implementar sistemas multiprocesador basados en *soft-core processors*.

A lo largo de la tesis se presentarán varias arquitecturas multiprocesador implementadas en FPGA, analizando las ventajas e inconvenientes de cada una de ellas, utilizando para ello distintos programas de prueba para evaluar su rendimiento.

El objetivo final del trabajo es proponer tanto una arquitectura multiprocesador tipo SMP (*symmetric multiprocessing*) como un sistema operativo y entorno de desarrollo de aplicaciones para ella, que reúnan una serie de características deseables en todo sistema como son: escalabilidad, portabilidad a otras plataformas, transparencia para el programador de aplicaciones software, etc.

# 2 COMPUTACIÓN PARALELA Y SISTEMAS MULTIPROCESADOR

A lo largo de este capítulo se presentarán las principales características y modelos de clasificación de los distintos sistemas multiprocesador existentes a lo largo de la historia de la computación, así como algunos sistemas multiprocesador actuales.

## *2.1 Computación paralela*

La computación paralela es una forma de computación en la que se realizan varias operaciones de forma simultánea (9), basándose en el principio de que los problemas grandes pueden ser divididos en problemas más pequeños que pueden ser resueltos de forma concurrente en varias unidades de procesamiento, permitiendo que el trabajo se complete en un tiempo menor. La idea de utilizar varias unidades de procesamiento para incrementar el rendimiento de las aplicaciones no es nueva, existe desde los inicios de la era de las computadoras electrónicas. Hace ya 40 años, Flynn (10) propuso un modelo para clasificar todos los computadores, que se sigue utilizando actualmente. El modelo tiene en cuenta el paralelismo tanto en las instrucciones como en los datos, y según esa clasificación, todo computador pertenece a alguna de las siguientes categorías:

- Flujo simple de instrucciones y flujo simple de datos (s*ingle instruction stream, single data stream o SISD*): Esta categoría es la de los sistemas uniprocesador.

- Flujo simple de instrucciones y flujo múltiple de datos (*single instruction stream, multiple data stream o SIMD*): La misma instrucción es ejecutada por varios procesadores, sobre distintos flujos de datos.

- Flujo múltiple de instrucciones y flujo simple de datos (*multiple instruction stream, single data stream o MISD*): Hasta la fecha no existe ningún multiprocesador comercial de este tipo, aunque algunos procesadores de propósito específico que trabajan con flujos de datos, se aproximan a esta clasificación.

- Flujo múltiple de instrucciones y flujo múltiple de datos (*multiple instruction stream, multiple data stream o MIMD*): cada procesador ejecuta sus propias instrucciones sobre sus propios datos.

Los primeros multiprocesadores eran de tipo SIMD, pero ese tipo de arquitectura cayó en desuso en la década de los 90, a excepción de los procesadores vectoriales. Los multiprocesadores de tipo MIMD han calado con fuerza como la principal arquitectura para multiprocesadores de propósito general. Una de las principales razones del auge de los multiprocesadores de tipo MIMD es su flexibilidad. Con un soporte adecuado de hardware y software pueden funcionar como un multiprocesador monoprogramado, enfocado a aumentar el rendimiento de una aplicación concreta, o también pueden funcionar como multiprocesadores multiprogramados ejecutando diferentes tareas, o una combinación de ambas funciones.

### 2.1.1 Organización de la memoria y modelos de comunicación en sistemas MIMD

Dentro de los multiprocesadores de tipo MIMD, se pueden distinguir dos tipos bien diferenciados, atendiendo a la organización de la memoria y al modo en que se interconectan los procesadores: sistemas con memoria centralizada, y sistemas con memoria distribuida (11):

- Sistemas con memoria centralizada: para sistemas multiprocesador donde el número de procesadores no es muy elevado, es posible utilizar una memoria centralizada que comparten todos los procesadores a través de un bus compartido (Figura 2). Haciendo uso de unas cachés eficientes, la memoria y el bus compartidos pueden satisfacer las necesidades de un número no muy grande de procesadores. El hecho de tener una memoria y un bus compartidos al que acceden de forma simétrica todos los procesadores ha derivado a que a estos sistemas se les conozca como sistemas de multiprocesamiento simétrico o SMP (*symmetric multiprocessing).* En estos sistemas, la comunicación entre procesadores se realiza de forma implícita a través de la memoria.

- Sistemas con memoria distribuida: para poder soportar un mayor número de procesadores, se hace necesario distribuir la memoria entre ellos en lugar de utilizar una memoria centralizada. De otra forma, la memoria centralizada no podría soportar el ancho de banda necesario para satisfacer las demandas de todos los procesadores.

Para la comunicación entre procesadores se hace necesario algún tipo de red de interconexión, tal como se muestra en la Figura 3.

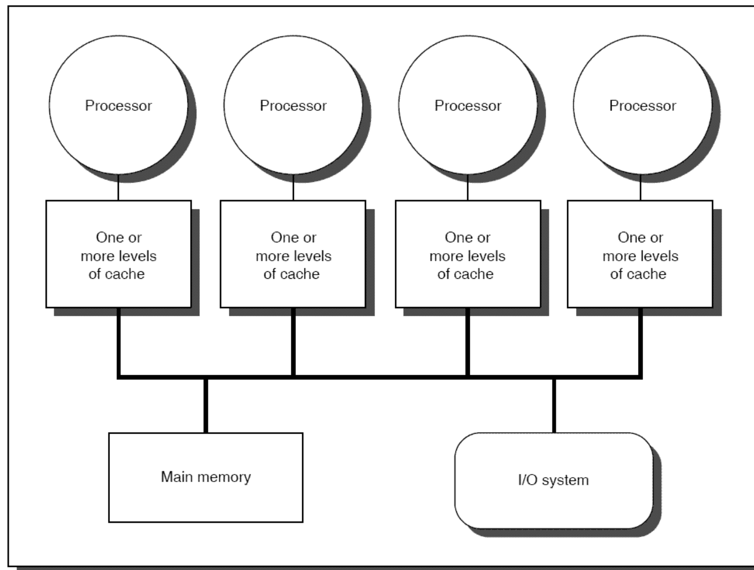


Figura 2: estructura básica de un sistema con memoria centralizada (11).

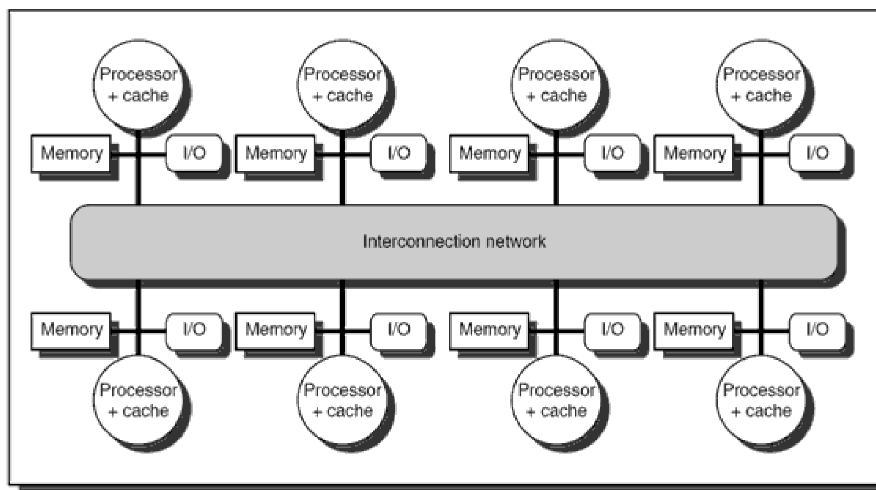


Figura 3: estructura básica de un sistema con memoria distribuida (11).

En sistemas multiprocesador existen dos alternativas bien diferenciadas a la hora de intercambiar información entre los distintos procesadores que forman un sistema (11)(12)(13), independientemente de que se use una arquitectura de memoria centralizada o distribuida.

Uno de los modelos de comunicación existentes es el de variables compartidas, donde todos los procesadores tienen acceso a toda la memoria del sistema a través de un único espacio de direcciones y los intercambios de información se realizan mediante operaciones de lectura y escritura en memoria, independientemente de que la arquitectura de memoria subyacente sea de memoria centralizada o de memoria distribuida.

El otro modelo de comunicación en sistemas multiprocesador es el de paso de mensajes, donde cada procesador tiene su propio espacio de direcciones y los distintos procesadores se comunican a través de mensajes para intercambiar información.

Teniendo en cuenta tanto la organización de la memoria como el modelo de comunicación, E.E. Johnson (14) propuso la siguiente clasificación de los sistemas MIMD:

- GMSV (*Global Memory / Shared Variables*): memoria centralizada y variables compartidas.
- GMMP (*Global Memory / Message Passing*): memoria centralizada y paso de mensajes.
- DMSV (*Distributed Memory / Shared Variables*): memoria distribuida y variables compartidas.
- DMMP (*Distributed Memory / Message Passing*): memoria distribuida y paso de mensajes.

Esta clasificación junto con la clasificación de Flynn permite clasificar cualquier computador teniendo en cuenta tanto el tipo de flujo de instrucciones y de datos, como la organización de la memoria y las comunicaciones tal como se muestra en la Figura 4.

## 2.2 Multiprocesamiento simétrico o SMP

Un tipo concreto de sistemas multiprocesador, y sobre el que se centrará la parte principal de este trabajo doctoral, son los sistemas de multiprocesamiento simétrico o SMP (*Symmetric MultiProcessing*), que son sistemas multiprocesador donde 2 o más procesadores idénticos están conectados a una misma memoria principal compartida y a un mismo interfaz de entrada/salida (11). El término "simétrico" se usa porque todos los procesadores utilizan el mismo mecanismo para acceder a la memoria y a los periféricos y compiten en igualdad de condiciones para obtener dicho acceso, por lo que este tipo de sistemas también se conoce con el nombre de UMA (*Unified Memory Acces*) en

contraposición a otro tipo de sistemas multiprocesador con una arquitectura de memoria no uniforme que se conoce como NUMA (*NonUniform Memory Acces*) (13).

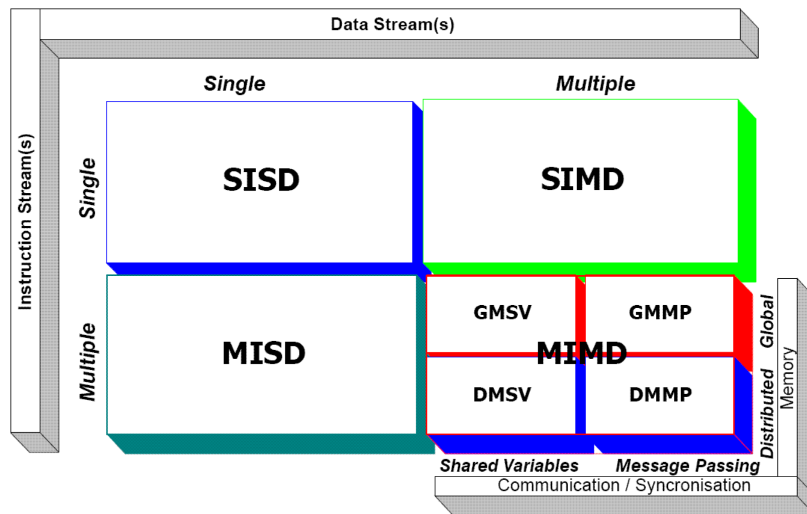


Figura 4: clasificación de Flynn-Johnson (15).

La comunicación entre los distintos procesadores en los sistemas SMP se hace de forma implícita a través de la memoria compartida, por lo que dentro de la clasificación de Flynn-Johnson los sistemas SMP entran dentro de la categoría MIMD-GMSV.

Los sistemas SMP van desde sistemas con varios procesadores de propósito general o específico conectados a una misma placa base con soporte para varios procesadores, hasta los más recientes procesadores *multi-core* donde dentro de una misma pieza de silicio se encuentran varios procesadores.

## 2.3 Algunos sistemas actuales

Actualmente existen diversos y muy variados sistemas multiprocesador en ámbitos que ya no se restringen a las grandes supercomputadoras de centros de investigación y grandes empresas, sino que ya han llegado a los ordenadores personales, a las videoconsolas, teléfonos móviles, sintonizadores de TDT y un largo etcétera.

A continuación se presentan algunos de los múltiples sistemas multiprocesador existentes, tanto sistemas simétricos como asimétricos, que se usan actualmente en el campo de la informática de consumo.

### 2.3.1 Ordenadores personales

En los últimos años se ha popularizado el uso de multiprocesadores o *multicores* en el ámbito de los ordenadores personales de gama media y alta. En 2005 Intel lanzó al mercado su procesador Pentium D consistente en dos procesadores Pentium 4 metidos en un único encapsulado, y en 2006 lanzó su gama de procesadores Core Duo y posteriormente en el mismo año el Core 2 Duo que integran dos procesadores en la misma pastilla de silicio. En la actualidad ya se encuentra en el mercado el procesador Core 2 Quad, que consiste en dos Core 2 Duo integrados en un único encapsulado, teniendo así en total cuatro núcleos.

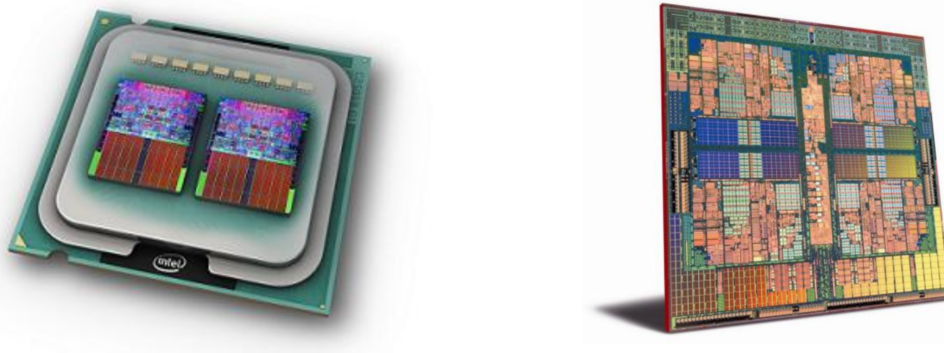

Figura 5: procesadores para equipos de sobremesa, Intel Core 2 Quad (izq) y AMD PhenomX4(dcha).

El otro gran fabricante de procesadores para ordenadores personales, AMD, también tiene sus propios modelos de multiprocesador para este tipo de ordenadores como son el TurionX2 lanzado en 2006, que integra dos procesadores Turion en un único encapsulado, los Athlon64 X2 lanzados en 2007 con dos procesadores Athlon64 en una única pastilla de silicio o los actuales procesadores Phenom X3 y X4 con tres y cuatro procesadores respectivamente en una misma pastilla de silicio.

### 2.3.2 Videoconsolas

En Noviembre de 2005 salió al mercado la primera videoconsola de la generación actual, la Xbox 360, siendo la primera que utiliza un multiprocesador como unidad central de proceso. El procesador que utiliza la Xbox 360 responde al nombre de Xenon (16) y contiene tres procesadores de la familia Power PC integrados en un único chip, con una caché de nivel 1 para cada procesador y una caché de nivel 2 compartida entre los tres procesadores (Figura 6).

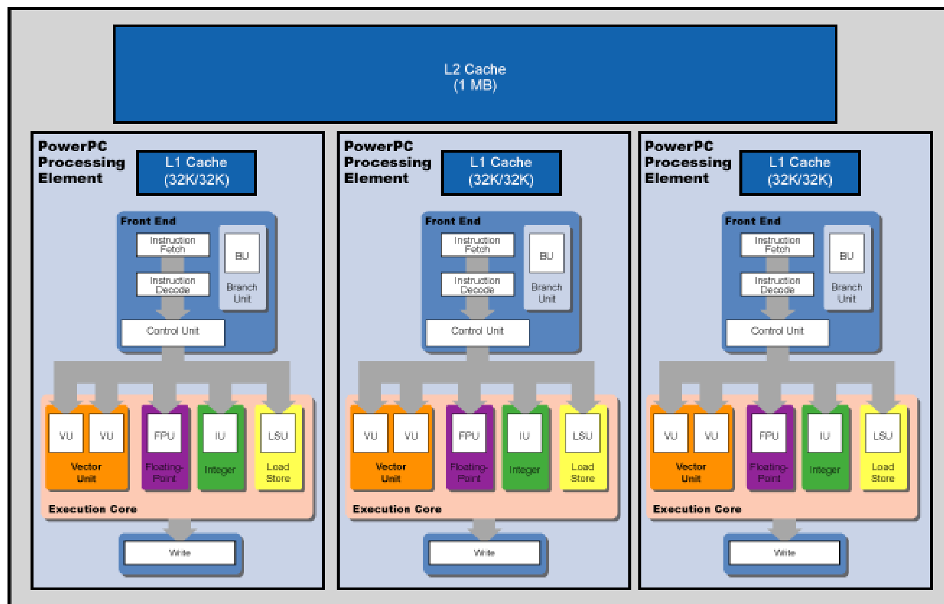


Figura 6: arquitectura interna del procesador Xenon (16).

Otra de las consolas de última generación, la Play Station 3, también utiliza un multiprocesador como unidad central de proceso: el procesador CELL de IBM (17). Al contrario que el Xenon que es un multiprocesador simétrico, el procesdor CELL es un multiprocesador asimétrico formado por un procesador con arquitectura Power (18) y 8 coprocesadores llamados SPEs (*Synergistic Processing Elements*) (19). Cada SPE es un procesador RISC de 128 bits con una estructura SIMD orientado principalmente a realizar cálculos complejos en coma flotante. En la Figura 7 se muestra la distribución interna de los distintos componentes del procesador.

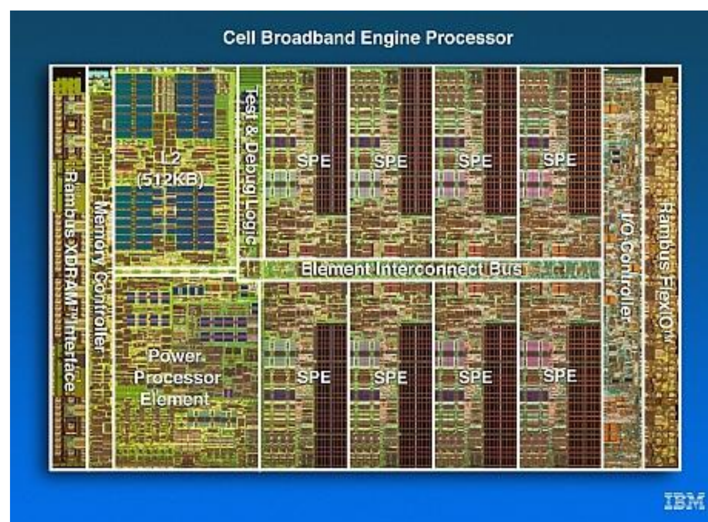


Figura 7: esquema del procesador CELL (17).

### 2.3.3 Sistemas Empotrados

En el campo de los sistemas empotrados también se utilizan multiprocesadores, generalmente asimétricos, con procesadores especializados para tareas de audio/video o *networking* combinados con procesadores de propósito general para realizar tareas de control. En la actualidad y gracias a los continuos avances de la tecnología microelectrónica este tipo de multiprocesadores suelen estar integrados en un único chip, permitiendo reducir tanto la complejidad de las PCBs como el consumo eléctrico del producto final. A continuación se comentan algunos de los multiprocesadores existentes en este ámbito:

- TI TMS320DM6467 (20): es un multiprocesador de Texas Instruments que incluye en un único chip un procesador VLIW C64x, un procesador RISC ARM 9 y un DSP DaVinci. Se utiliza principalmente para aparatos de decodificación de audio/video como receptores de TDT, satélite, reproductores de DVD, BlueRay, etc

- Intel IXP2850 (21): es un procesador diseñado para aplicaciones de red que requieran procesamiento de paquetes, procesamiento de contenidos y seguridad, como *routers* y *firewalls*. Está formado por una matriz de 16 *microengines* para manejar paquetes, un procesador Intel Xscale para operaciones de control y dos procesadores criptográficos para acelerar los algoritmos de seguridad.

- Viper PNX-8500 (22): contiene en su interior dos procesadores: un Trimedia TM32 VLIW y un MIPS PR3940 de 32 bits, además de una serie de aceleradores para distintas aplicaciones de vídeo. También una serie de interfaces de entrada/salida: UART, PCI, IEEE 1394. Está diseñado para ser utilizado en aplicaciones de vídeo y red: televisión digital, pasarela de red, *set-top-boxes*, etc.

Estos son sólo algunos ejemplos de los sistemas multiprocesador que se utilizan en el ámbito de los sistemas empotrados, ya que es probablemente el área donde más cantidad y variedad de multiprocesadores existen.

# 3 FPGAs Y SOFT-CORE PROCESSORS

## 3.1 FPGAs

Desde que en 1985 Xilinx lanzase al mercado su primera FPGA, la XC2064 (23) con una capacidad de 1000 puertas lógicas equivalentes, la tecnología microelectrónica ha evolucionado siguiendo de una forma bastante exacta la Ley de Moore (24). A día de hoy, una FPGA de 1000 puertas puede parecer irrisoria, pero esta familia de FPGAs estableció las bases de la arquitectura de las FPGAs de la compañía Xilinx, que son en la actualidad las que dominan el mercado y sobre las que se realizarán la parte experimental de esta tesis.

Las FPGAs son circuitos integrados digitales que contienen bloques de lógica configurable (CLBs) e interconexiones también configurables entre dichos bloques (25), que permiten al desarrollador programarlas para realizar tareas muy diversas. En la Figura 8 se muestra la estructura interna de las primeras FPGAs de la familia 2000 de Xilinx.

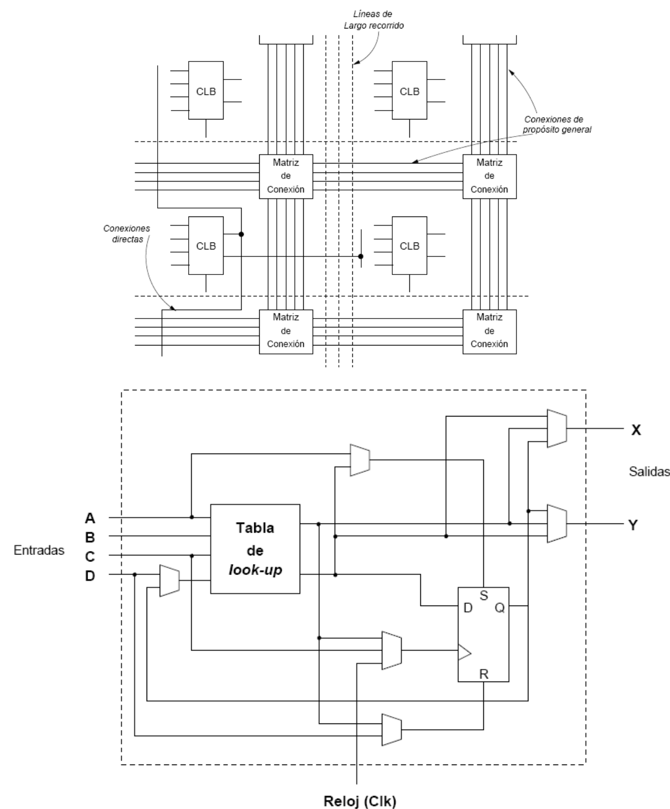


Figura 8: estructura de interconexión (arriba) y detalle de un CLB (abajo) en la familia 2000 de Xilinx.

Como se puede ver en la Figura 8, la FPGA está formada por una estructura simétrica en la que cada elemento CLB está rodeado por líneas dedicadas que conforman la red de interconexión. A su vez cada CLB está compuesto por un elemento LUT (*LookUp Table*) que puede implementar hasta dos funciones lógicas de 4 variables de entrada, y un biestable tipo D. Esta estructura básica de matriz simétrica ha sido mantenida por Xilinx en sus modernas familias de FPGA, aumentando con cada nueva familia la capacidad de las CLBs e interconexiones. Otros fabricantes han optado por otro tipo de estructuras internas para sus FPGAs, cada una con sus propias ventajas e inconvenientes. No es objeto de este trabajo evaluar las estructuras internas de las FPGAs, baste sólo con enumerar diferentes tipos como puede ser basado en filas (26), jerárquica, *sea-of-gates* (27), o estructuras de una única dimensión (28) (29).

Atendiendo al tipo de tecnología que utilizan las FPGAs para almacenar sus datos de configuración, estas se pueden clasificar en tres tipos principales:

- **Basadas en memoria RAM estática (SRAM)**: las FPGAs guardan su configuración en una memoria interna de tipo SRAM. Cada vez que se enciende el sistema es necesario reprogramar la FPGA con su configuración, por lo tanto es necesario almacenar esta configuración en un dispositivo ROM externo o descargarla desde un PC conectado a la FPGA. Este tipo de FPGA es la más flexible y permite utilizar técnicas de reconfiguración dinámica.

- **Basadas en memoria ROM:** las FPGAs almacenan su configuración en una memoria interna de tipo ROM, normalmente FLASH o EEPROM, siendo la tecnología FLASH la dominante en estos momentos. La ventaja es que al apagar la FPGA la configuración no se pierde, por lo que no son necesarios componentes externos para almacenarla. Por contra es menos flexible ya que la configuración es más lenta y no permiten utilizar técnicas de reconfiguración dinámica.

- **Basadas en fusibles:** La configuración se almacena "quemando" fusibles durante el proceso de programación. Actualmente la tecnología más utilizada en estas FPGAs es la basada en anti-fusibles (30). La configuración queda grabada en la FPGA de forma que no es posible reprogramarla, lo que elimina cualquier tipo de flexibilidad.

En los primeros años de la década de los 90 la evolución tanto en tamaño como en sofisticación de las FPGAs las llevo a ser utilizadas en aplicaciones más complejas, principalmente en el área de las telecomunicaciones. Ya a finales de los 90 su uso comenzó

a popularizarse en diversas aplicaciones industriales, de automoción, electrónica de consumo, y un largo etcétera.

Uno de los campos donde son utilizadas frecuentemente es en el prototipado de diseños en ASIC o como plataforma hardware donde verificar la implementación física de nuevos algoritmos, aunque su bajo coste de desarrollo en relación a los ASIC y su corto tiempo de desarrollo hasta la puesta en el mercado hacen que cada vez más se utilicen en productos finales.

Las FPGAs del presente siglo poco tienen que ver con las primeras aparecidas en los 80, y ofrecen densidades de millones de puertas lógicas además de microprocesadores empotrados, interfaces de entrada/salida de alta velocidad, bancos de memoria, multiplicadores hardware, DSPs (*Digital Signal Processor*), etc. El resultado final es que las FPGAs de hoy en día permiten implementar una gran variedad de sistemas electrónicos: dispositivos de comunicaciones, complejos algoritmos de procesamiento de señal, microprocesadores, etc.

Además de todos estos componentes integrados ya dentro de la propia FPGA, otro avance que ha propiciado que se popularice el uso de este tipo de dispositivos es la gran variedad de bloques IP, también llamados IP *cores* (*Intellectual Property*) disponibles. Los IP *cores* son bloques de lógica descritos en algún lenguaje de descripción de hardware o en algún formato propietario y que pueden integrarse de forma sencilla en diseños de SoC (*System on Chip*) para FPGA o ASIC, permitiendo acortar mucho el tiempo de diseño. Estos bloques IP son proporcionados por los propios fabricantes de FPGAs, por otras empresas dedicadas a la electrónica, e incluso por algunas comunidades de hardware libre, y van desde interfaces de comunicaciones como UARTs, buses I2C, interfaces Ethernet, hasta controladores de memoria, buses de sistema y procesadores.

El aumento de recursos de las FPGAs junto con la amplia disponibilidad de bloques IP ha hecho posible la integración de sistemas completos en la propia FPGA y la aparición de una nueva terminología para clasificarlos. Algunos de los términos que se usan para designar este tipo de sistemas son:

- **System on Chip (SoC):** Circuito integrado formado por diversos módulos VLSI con distinta funcionalidad que interconectados entre sí ofrecen una funcionalidad específica para una aplicación (31).

- **System-on-Programable-Chip (SoPC):** Se aplica este término específicamente cuando el dispositivo utilizado para realizar el SoC es reconfigurable.

- **Configurable-System-on-Chip (CSoC)** (32)**:** Mediante este término se definen los sistemas SoC en los que se hace uso de la capacidad de reconfiguración de los mismos para aplicaciones de computación reconfigurable. Pueden incluirse bajo la denominación CSoC tanto los sistemas que admiten diferentes configuraciones estáticas según ciertos condicionantes, como los que utilizan la reconfiguración parcial dinámica para modificar en tiempo de ejecución una sección hardware.

- **Multiprocessor-Configurable-System-on-Chip (MCSoC):** Se aplica esta definición a los sistemas CSoC que incluyen varias unidades procesador funcionando de forma simultánea.

## 3.2 Diseño basado en plataforma

El diseño basado en plataforma es una de las respuestas actuales a las necesidades del mundo del diseño electrónico de hoy. Gordon Moore (24) predijo en 1965 que el número de transistores por pulgada dentro de un circuito integrado se duplicaría cada año y que la tendencia continuaría durante las siguientes dos décadas. Algo más tarde reajustó su cálculo diciendo que la densidad se doblaría cada 18 meses. Este dato, que se conoce comúnmente como Ley de Moore, se ha venido cumpliendo de forma sorprendentemente exacta durante las últimas décadas. Lamentablemente, los ingenieros de desarrollo no han podido incrementar al mismo ritmo su capacidad de utilización de estos recursos. La diferencia entre el número de recursos disponibles y el número de recursos disponibles realmente utilizados se denomina brecha de diseño (*design gap*) (33) y es uno de los retos fundamentales a los que se enfrenta la industria de los semiconductores.

El *Internacional Technology Roadmap for Semiconductors* en su edición de 2005 remarca que el principal problema para poder continuar la hoja de ruta siguen siendo los costes de diseño. El desarrollo de nuevas metodologías que permitan abordar el diseño de circuitos de gran envergadura es uno de los puntos en los que más se incide en dicho documento. Las soluciones propuestas pasan por utilizar flujos de diseño que suban el nivel de abstracción de tal forma que el diseñador "olvide" los aspectos de más bajo nivel, centrándose en la arquitectura del sistema. A partir de esa especificación de alto nivel harían su aparición distintas metodologías de verificación a nivel de sistema, herramientas de particionado, síntesis de alto nivel y co-síntesis hardware-software, permitiendo a los

diseñadores reducir de forma drástica el tiempo de diseño. Cabe mencionar que para conseguir este objetivo es importante aumentar la reusabilidad de los bloques que conforman los sistemas electrónicos (34), tanto software como hardware. En relación directa con este punto merece la pena destacar el auge de los bloques IP de código abierto de los cuales se hablará posteriormente.

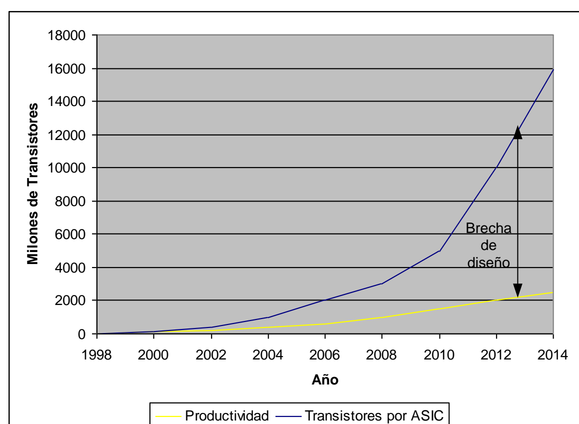


Figura 9: brecha de diseño (35).

Obviamente, no sólo es importante contar con metodologías y herramientas de diseño sino también con plataformas hardware sobre las que aplicar dichas metodologías. Una plataforma dentro del contexto de los sistemas electrónicos se puede definir como un sistema flexible donde la aplicación se crea mediante la programación de uno o más componentes físicos del sistema (7). Dentro de cada una de las tecnologías que se pueden encontrar en el mundo de los sistemas electrónicos, los componentes programables de los que dispone el diseñador son obviamente diferentes. Si nos centramos en las plataformas basadas en FPGAs, desde hace tiempo es habitual disponer de bloques de lógica configurable, memorias RAM internas, elementos de control del reloj, elementos DSP, e incluso en algunas familias (como Virtex-II Pro o Virtex-4 FX) de microprocesadores embebidos en la FPGA (como PowerPC) que permiten diseñar sistemas completos en un periodo de tiempo razonablemente corto. No obstante, el gran crecimiento de la capacidad de las FPGAs auspiciado por la Ley de Moore ha hecho aparecer nuevas metodologías de diseño para plataformas "centradas en el procesador" *(processor-centric)* (36). Estas plataformas están compuestas por uno o varios procesadores, un sistema operativo de tiempo real, una jerarquía de memoria, y bloques IP estándar. Mediante herramientas de diseño gráficas o textuales todos estos elementos, incluido el sistema operativo, se configuran de una forma lo más transparente posible para el diseñador, teniendo éste que

ocuparse principalmente de desarrollar el software de aplicación que correrá sobre los microprocesadores y de diseñar los bloques IP específicos que necesite su aplicación. De esta manera se evita tener que rediseñar el sistema desde el principio y se reducen drásticamente los tiempos de desarrollo, siendo posible tener un sistema plenamente operativo en cuestión de minutos.

Comúnmente las metodologías de diseño son de tipo descendente *(top-down)* donde se parte de una especificación lo más alta posible del sistema y se va refinando hasta llegar a la implementación final, o ascendente *(bottom-up)* donde se va formando un sistema completo partiendo de componentes de bajo nivel, pudiendo de esta forma crear diseños muy optimizados. Sin embargo, en el diseño basado en plataforma el flujo de diseño es tanto ascendente como descendente, denominándose bidireccional o *meet-in-the middle* ya que existen dos puntos de inicio: la plataforma y la aplicación. Cuando a la plataforma se le añaden nuevos periféricos o bloques IP estamos yendo hacia arriba en el flujo de diseño, sin embargo cuando se añade nuevo software de aplicación nos movemos de forma descendente en el flujo.

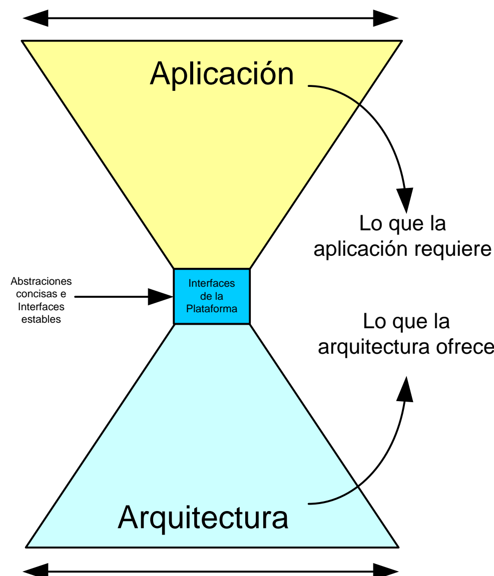


Figura 10: flujo de diseño bidireccional.

En la actualidad comienzan a existir entornos de desarrollo cada vez más estables, que permiten realizar diseño basado en plataforma sobre dispositivos FPGA de forma gráfica e intuitiva y que se enfrentan al desarrollo de soluciones desde distintos grados de abstracción.

Como ejemplo de grado de abstracción medio-bajo, cabe destacar el entorno EDK de Xilinx que permite diseñar sistemas basados tanto en el *hard-core processor* PowerPC incluido en algunas familias de FPGAs de Xilinx como en el procesador diseñado sobre la lógica de la FPGA (*Soft-Core Processor – SCP*) denominado Microblaze. En este entorno se puede tanto añadir al procesador los bloques IP que se desee para generar la arquitectura hardware como desarrollar el software que correrá sobre dicho procesador gracias a las herramientas de desarrollo hardware y software que incluye. Altera ofrece una alternativa similar para su SCP Nios-II.

Como ejemplo de grado de abstracción medio-alto cabe destacar el entorno para el diseño basado en plataforma ofrecido por Celoxica. En esta metodología se describen algoritmos en un lenguaje muy próximo a C, denominado Handel-C, de forma independiente a la plataforma hardware sobre la que posteriormente serán implementados gracias a lo que Celoxica denomina PAL (*Platform Abstraction Layer*). Esto permite utilizar los periféricos de la plataforma, como bancos de memoria, o interfaces de comunicación utilizando un API en C genérico para todas las placas de prototipado soportadas por el entorno.

## 3.3 Soft-Core Processors

Un *soft-core processor* o abreviado SCP, también llamado *soft microprocessor* o *soft processor*, es un microprocesador que puede ser implementado utilizando síntesis lógica en diferentes dispositivos lógicos programables como CPLDs o FPGAs, o incluso puede ser incluido en un diseño para ASIC (37). Suelen ser distribuidos en forma de código fuente en algún lenguaje de descripción de hardware, principalmente VHDL o Verilog, aunque algunos SCPs comerciales se distribuyen en formatos propietarios.

Los principales fabricantes de dispositivos lógicos programables tienen su propio SCP comercial especialmente diseñado y optimizado para funcionar en sus propias FPGAs. Así Xilinx tiene el PicoBlaze (38) y el MicroBlaze (39), Altera proporciona el Nios II (40), y Lattice distribuye su LatticeMico32 (41).

También existen una gran variedad de SCPs distribuidos en forma de código abierto, como el OpenRISC 1200 (42) mantenido por la comunidad Opencores (43), o los SCPs LEON2 y LEON3 (44) (45) que proporciona la compañía Gaisler Research (46).

A continuación se presentan más detalladamente las características de estos SCPs, y un resumen comparativo de sus características.

### 3.3.1 PicoBlaze

PicoBlaze es un microcontrolador RISC de 8 bits desarrollado por Xilinx y optimizado para sus FPGAs. Esta optimizado para ocupar muy poco área, tan sólo 96 *slices*, y en cuanto a velocidad puede llegar a ejecutar entre 44 y 100 millones de instrucciones por segundo, dependiendo del *speedgrade* y la familia de FPGA sobre la que se implemente.

Las principales características del PicoBlaze son:

- 16 Registros de propósito general de 1 byte.
- 1024 bytes de instrucciones en memoria *on-chip* que se cargan al programarse la FPGA.
- 256 puertos de entrada y 256 de salida para conectar lógica o periféricos externos.

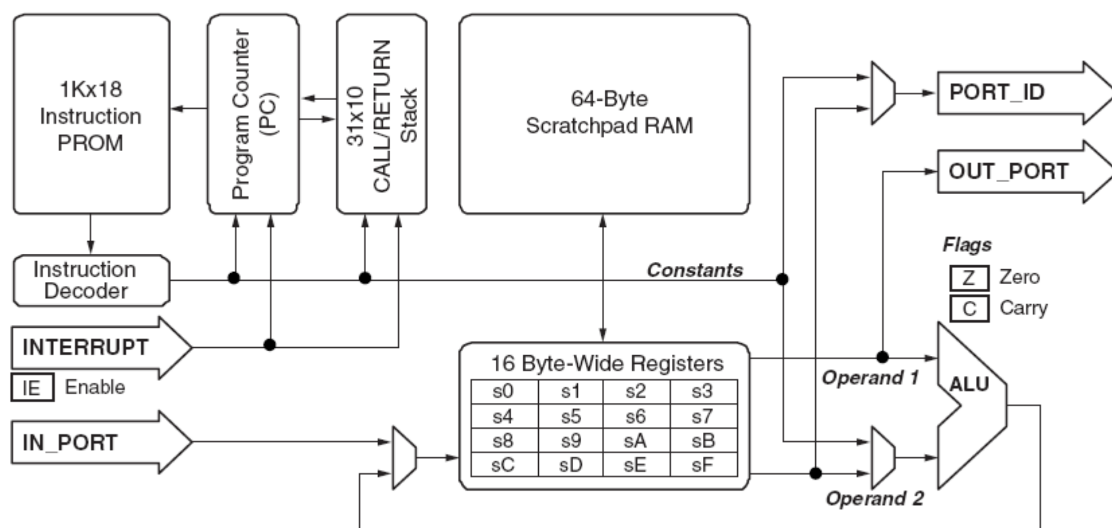


Figura 11: estructura interna del microcontrolador PicoBlaze (38).

Aunque está más orientado para ser usado en pequeñas tareas de control más que en ejecutar grandes aplicaciones, se ha usado ya en varios sistemas multiprocesador como se mostrará en el capítulo 4.

### 3.3.2 MicroBlaze

MicroBlaze es un microprocesador RISC de 32 bits desarrollado por Xilinx para sus FPGAs de las familias Spartan y Virtex. Sigue una arquitectura Harvard con buses de memoria de datos e instrucciones separados.

Una de sus principales características es que es muy configurable, pudiendo incluir o excluir una serie de elementos del microprocesador según las necesidades de la aplicación objetivo permitiendo una gran variedad de configuraciones más o menos rápidas y que ocupan más o menos área en la FPGA.

Las características más destacables de este SCP son:

- 32 registros de propósito general de 32 bits.
- Instrucciones de 32 bits, con 3 operandos y 2 modos de direccionamiento.
- Bus de direcciones de 32 bits.
- Pipeline configurable de 3 o 5 etapas.
- FPU, *barrel-shifter* y multiplicador y/o divisor de enteros opcionales.
- Módulo de depuración opcional.
- Caché de instrucciones y caché de datos opcionales.
- 3 interfaces de bus disponibles para conectar distintos tipos de periféricos:
    - LMB (47)(*Local Memory Bus*): Bus síncrono de alta velocidad utilizado principalmente para conectar los bloques de memoria interna de la FPGA.
    - OPB (48)(*On-Chip Memory Bus*) : Bus síncrono utilizado para conectar periféricos con tiempos de acceso variables. Tiene soporte para hasta 8 maestros.
    - FSL (49)(*Fast Simplex Link*): Canales punto a punto dedicados, para *streaming* de datos. Dispone de 8 canales, cada uno con un puerto de entrada y otro de salida.

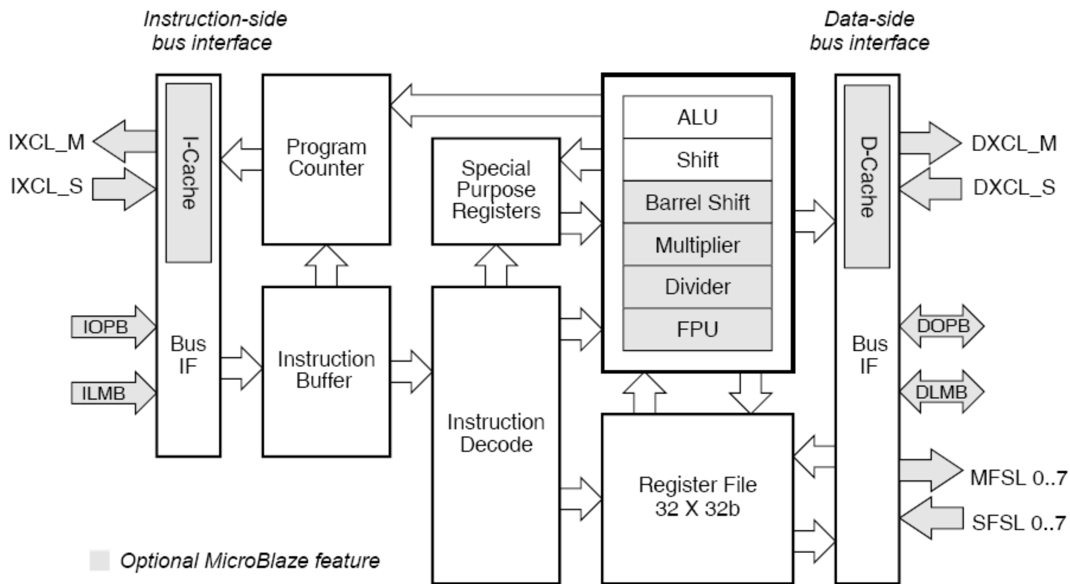


Figura 12: arquitectura interna del procesador MicroBlaze con sus unidades opcionales.

Actualmente MicroBlaze es uno de los SCP más utilizado, y parte del éxito se debe a las herramientas que proporciona Xilinx para crear sistemas basados en este microprocesador. La herramienta Xilinx Platform Studio (50) permite de forma gráfica e intuitiva interconectar tanto el procesador como los distintos periféricos y buses que forman el sistema (Figura 13).

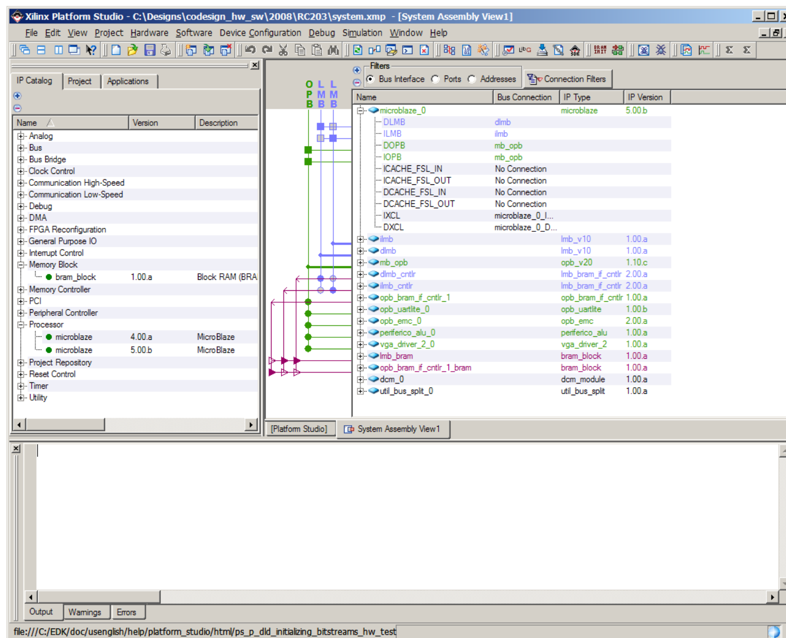


Figura 13: interfaz de la herramienta Xilinx Platform Studio.

También ofrece una librería de IPs ya diseñadas y listas para integrarse en cualquier diseño como:

- Controladores de memoria: SRAM, SDRAM, DDR y DDR2.
- Dispositivos de comunicación serie: bus I2C, UART16550
- Controladores Ethernet y Ethernet Lite.

La herramienta además de ayudar y simplificar la parte de diseño hardware de un sistema basado en el procesador MicroBlaze también ofrece facilidades para el desarrollo de software para el sistema, proporcionando tanto drivers para los distintos periféricos como todo un conjunto de herramientas de desarrollo software (compilador, depurador, *bootloader*, …), un micro kernel con interfaz de hilos POSIX y diversas librerías, todo ello integrado en un entorno basado en Eclipse (51) que facilita mucho la tarea del programador de aplicaciones.

### 3.3.3 Nios II

Nios II es el procesador que ofrece la compañía Altera para ser usado en sus FPGAs de las familias Stratix y Ciclone. Es un procesador RISC de 32 bits con arquitectura Harvard.

Las principales características de este SCP son:

- 32 registros de propósito general de 32 bits.
- Operaciones de punto flotante de precisión simple.
- Bus de direcciones de 32 bits.
- Posibilidad de añadir lógica a la ALU para crear nuevas instrucciones de propósito específico.
- Caché de instrucciones y caché de datos opcionales.

Altera ofrece tres versiones diferentes del Nios II, con diferentes características en cuanto a rendimiento y consumo de área. Las principales características de cada versión del procesador son:

- Nios II *fast*: diseñado para ser lo más rápido posible, cuenta con un *pipeline* de 6 etapas. Incluye operación de multiplicación por hardware en un solo ciclo, previsión dinámica de saltos y cachés separadas de instrucciones y datos.

- Nios II *economy:* diseñado para ocupar la menor área posible, ocupa tan solo 700 *Logic Elements*. Sin caches.

- Nios II *standard:* orientado a aplicaciones de bajo coste, con un rendimiento medio en las aplicaciones. Incluye caché de instrucciones, *pipeline* de 5 etapas y predicción estática de saltos. También ofrece multiplicador, divisor y desplazador hardware opcionales.

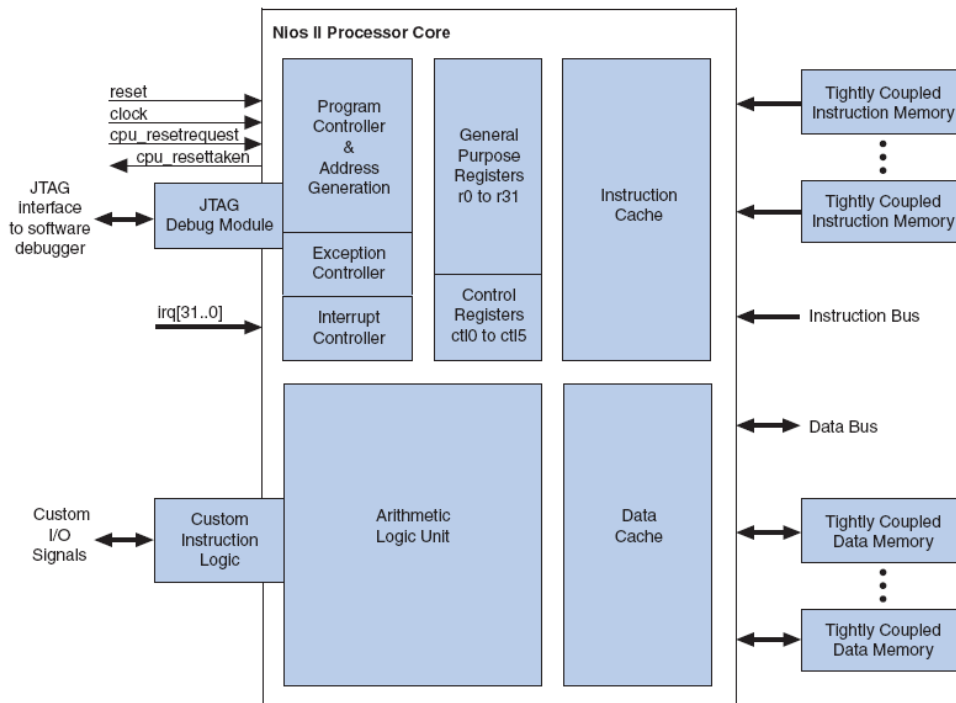


Figura 14: arquitectura del procesador Nios II (40).

Al igual que Xilinx, Altera también ofrece una herramienta gráfica que permite crear de forma rápida y sencilla sistemas basados en su microprocesador: *System on Programmable Chip Builder* (52)(Figura 15). También ofrece una librería de componentes como UARTs, dispositivos Ethernet, controladores de memoria, etc, que se pueden integrar a cualquier diseño basado en Nios II.

Además de facilitar la tarea de diseñar la parte hardware del sistema, también ofrece un entorno de desarrollo de software basado en Eclipse que facilita la tarea del

desarrollador de software proporcionándole en un único IDE todas las herramientas necesarias como editor, compilador, depurador, *bootloader*, etc, además de dar soporte para desarrollar aplicaciones que hagan uso del sistema operativo MicroC/OS-II (53).

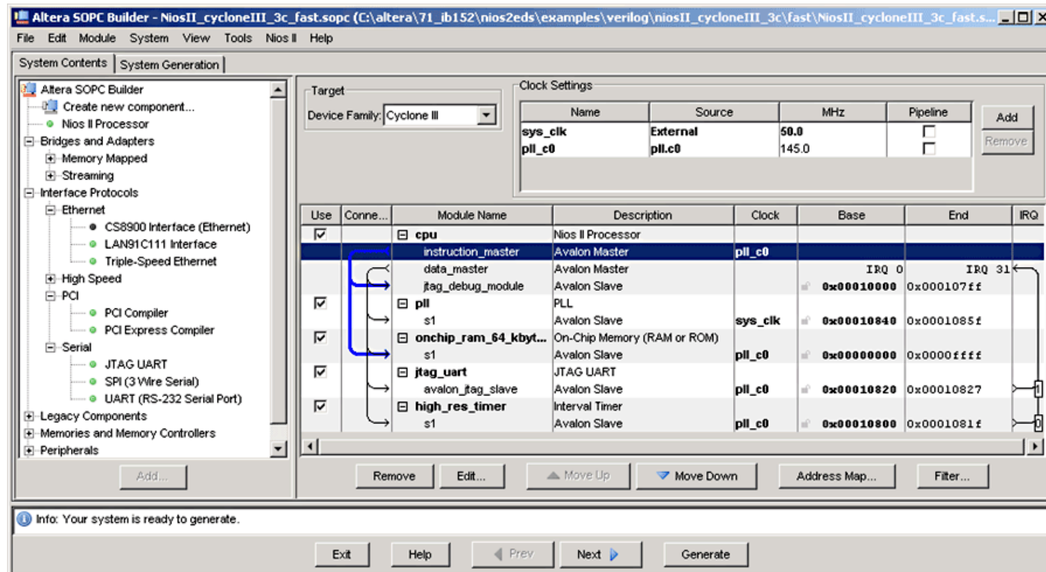


Figura 15: interfaz de la herramienta System on Programmable Chip Builder.

### 3.3.4 LatticeMico32

La compañía Lattice ofrece el procesador LatticeMico32 para usarlo en sus FPGAs. Una de las principales características de este procesador es que aunque proviene de una compañía privada se ofrece bajo una licencia libre. Sigue una arquitectura RISC, y utiliza dos interfaces de bus WISHBONE (6) separados para instrucciones y datos.

El procesador se ofrece en 3 posibles configuraciones con distintas relaciones de área y rendimiento:

- Configuración básica: Sin multiplicador hardware ni cachés. Desplazador multiciclo. Ocupa 1571 LUTs de una FPGA LatticeEC/ECP y puede alcanzar una frecuencia de reloj de 81 MHz.

- Configuración estándar: Con multiplicador hardware, desplazador segmentado y 8 KB de caché de instrucciones. Ocupa 2040 LUTS y alcanza una frecuencia de reloj de 89 MHz.

- Completa: Igual que la configuración estándar pero con 8 KB de caché de datos. Ocupa 2230 LUTs y alcanza los 92 MHz.

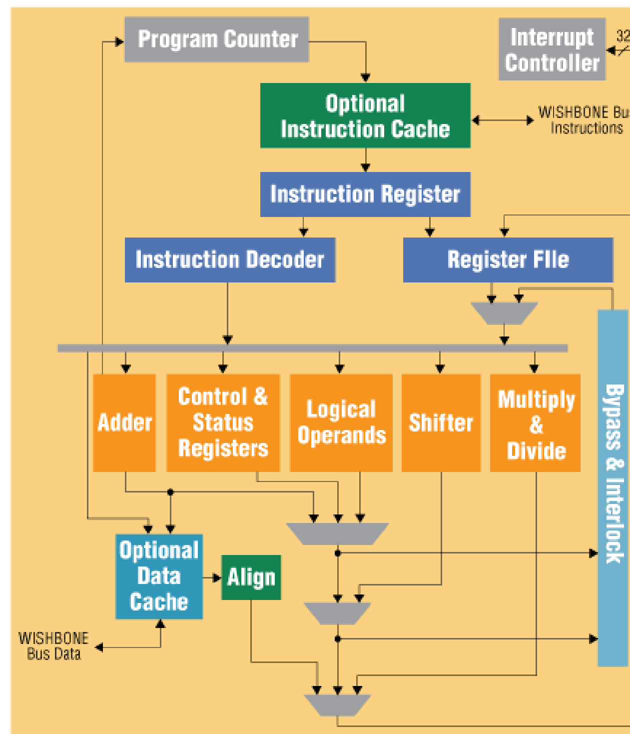


Figura 16: arquitectura del procesador LatticeMico32 (41).

Al igual que sus competidoras Xilinx y Altera, Lattice ofrece también un entorno de desarrollo para generar sistemas basados en su microprocesador llamada *Mico System Builder*. Desde dicha herramienta permite de forma sencilla interconectar el procesador con los distintos periféricos y buses que integra la librería de IPs proporcionada por Lattice.

También se proporciona un entorno de desarrollo software basado en Eclipse que permite un manejo sencillo por parte del desarrollador de software de todas las herramientas necesarias para manejar un proyecto software: editor, compilador, depurador, bootloader, etc. Además, al igual que la herramienta de Altera, da soporte directo para utilizar el sistema operativo MicroC/OS-II.

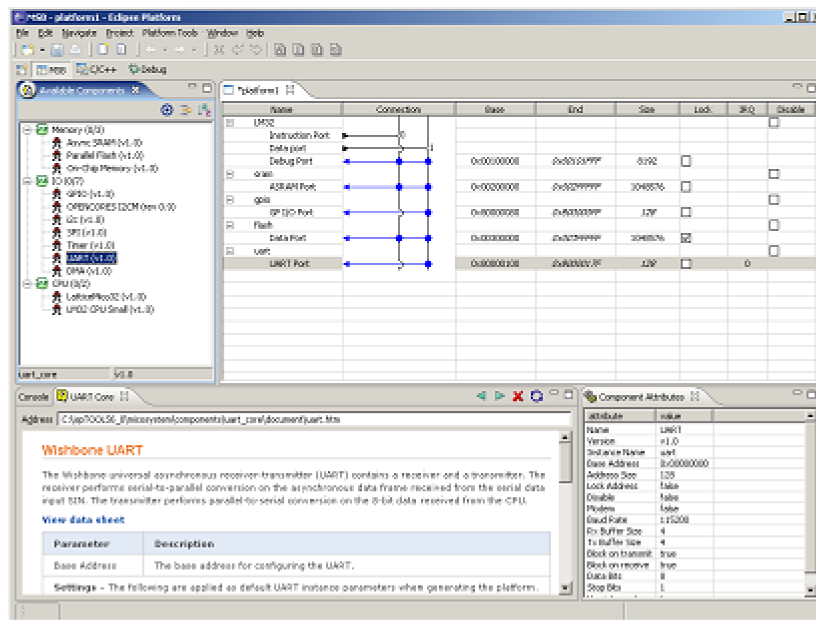


Figura 17: interfaz de la herramienta Mico System Builder.

### 3.3.5 OpenRISC 1200

Opencores (43) es una comunidad de hardware de código abierto que ofrece una gran variedad de IPs entre los que se encuentra su producto estrella: el OpenRISC 1200, un microprocesador RISC de 32 bits que se distribuye como código abierto.

Algunas características de este SCP son:

- Arquitectura Harvard.
- *Pipeline* de 5 etapas.
- Unidad de manejo de memoria (MMU)
- Cachés de instrucciones y de datos de 8 KB, de mapeo directo.
- Unidades opcionales como: unidad de depuración, *tick-timer*, controlador de interrupciones y unidad de manejo de potencia (PMU).
- Interfaz WISHBONE para los buses de instrucciones y de datos.

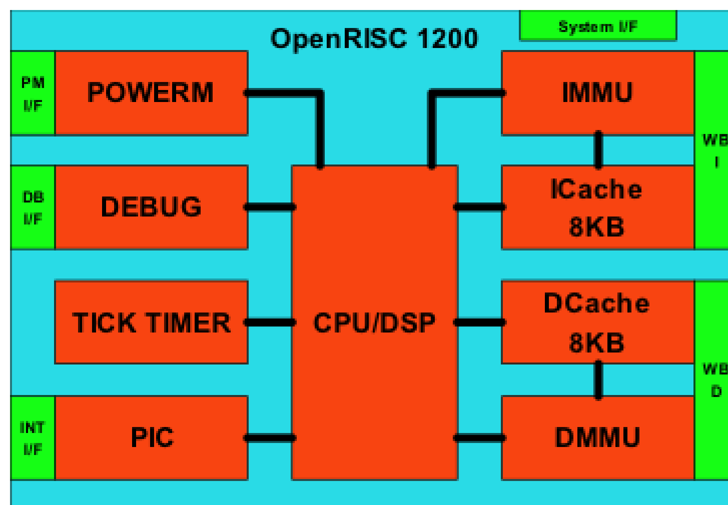


Figura 18: arquitectura OpenRISC (42).

Las herramientas existentes para desarrollar sistemas basados en el OpenRISC 1200 son muy limitadas y no existe una herramienta similar a las existentes para los procesadores ya comentados que permita interconectar el procesador y los distintos buses y periféricos de una forma sencilla e intuitiva. Existe una herramienta gráfica basada en TCL/TK que permite configurar las diferentes características del procesador de forma sencilla (54), en lugar de tener que editar a mano el fichero de configuración del microprocesador.

En cuanto a herramientas para el desarrollo de software está también bastante limitado. Se proporcionan las herramientas de compilación y depuración de GNU, y existen también proyectos que mantienen *ports* de algunos sistemas operativos libres como eCos (55)o uCLinux (56). Pero no está todo integrado en un entorno de desarrollo ni ofrece las características que ofrecen las herramientas de Xilinx o Altera como generación automática de drivers, generación de scripts de enlazado, generación de BSPs para distintos sistemas operativos, etc.

El rendimiento que ofrece el OpenRISC 1200 es superior al de sus competidores comerciales, a costa de un mayor consumo de área. En (57) se ofrecen diversos datos de comparación de distintos procesadores en distintas configuraciones y se desprende que el área ocupada por un sistema basado en OpenRISC 1200 es entre dos y tres veces superior al de un sistema equivalente basado en MicroBlaze.

### 3.3.6 LEON 2 y LEON 3

LEON 2 es un procesador RISC de 32 bits basado en la arquitectura SPARC-V8 (58). Está desarrollado por Gaisler Research y la Agencia Espacial Europea (ESA). Está completamente descrito en VHDL sintetizable y se distribuye bajo una licencia GPL (*General Public License*). Algunas de sus características más importantes son:

- Arquitectura Harvard.
- Caches de instrucciones y datos configurables.
- *Pipeline* de 5 etapas.
- Interfaz de bus AMBA (59).
- Caché de datos con protocolo *snoop*, que permite coherencia de cachés en sistemas multiprocesador.
- Unidad de manejo de memoria (MMU).

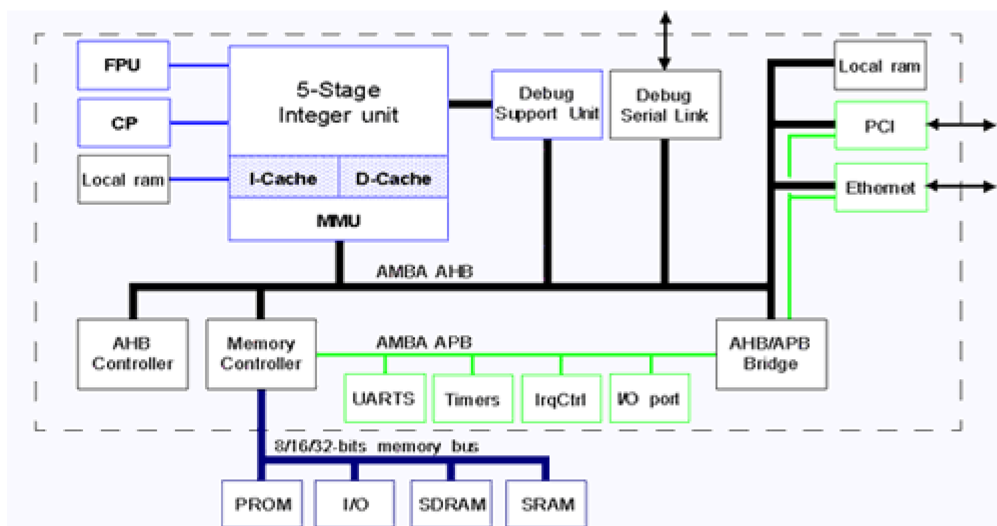


Figura 19: arquitectura del procesador LEON 2 (44).

Existe una herramienta gráfica que permite configurar las partes opcionales del procesador, pero no existe ninguna herramienta que permita realizar de forma sencilla las conexiones entre buses y periféricos, añadir nuevos periféricos, etc que ofrecen las

herramientas comerciales de otros procesadores. En la parte de desarrollo software Gaisler Research proporciona el compilador basado en gcc LECCS (60), y RTEMS (61) un *kernel* de tiempo real para ser usado con su procesador. También existen *ports* de otros *kernels* como eCos o Linux, mantenidos por terceras partes.

En cuanto a rendimiento el LEON 2 es muy superior a todos los procesadores presentados hasta ahora, aunque también es el que ocupa más área: de los resultados que se presentan en (57) se puede ver que en media ocupa entre 3 y 4 veces más que un sistema equivalente basado en MicroBlaze y entre 1 y 1,5 veces más que un sistema equivalente basado en OpenRISC 1200.

El LEON 2 dejó de estar soportado y dio paso al LEON 3: un procesador RISC de 32 bits, también compatible con la arquitectura SPARC V8 y que incluye muchas mejoras con respecto a su predecesor:

- *Pipeline* de 7 etapas.
- Unidad de punto flotante completamente segmentada.
- Cachés de instrucciones y de datos mejoradas, hasta 4 veces más grandes.
- Hasta un 50 % más de frecuencia de reloj.
- Soporte para multiprocesamiento simétrico.

Todas estas mejoras suponen también un considerable aumento del área necesaria, en torno a un 30 % más.

# 4 SISTEMAS MULTIPROCESADOR EN FPGA

Como se ha visto en el capítulo anterior las FPGAs ofrecen la capacidad de integrar varios procesadores *soft-core* dentro de un mismo diseño, así como la presencia en algunas ocasiones de procesadores *hard-core*. Esto permite implementar arquitecturas muy diversas de sistemas multiprocesador, tanto para aumentar el rendimiento de una aplicación concreta como para experimentar con nuevas arquitecturas.

En este capítulo se presentan por una parte las posibilidades que ofrecen las FPGAs a la hora de implementar sistemas multiprocesador, así como los retos que supone integrar este tipo de sistemas dentro de un diseño basado en FPGA. También se presentarán algunos de los sistemas multiprocesador sobre FPGA que existen actualmente.

## *4.1 Arquitecturas MPSoC en FPGA*

A la hora de diseñar un sistema multiprocesador en FPGA hay infinidad de posibles arquitecturas (62), cada una con sus ventajas e inconvenientes y más o menos idóneas según el tipo de aplicación para la que se va a utilizar.

La arquitectura más sencilla en que se puede pensar es una en la que se incluyen varios procesadores pero no se comunican entre ellos, y cada uno se dedica a una tarea independiente de las que se ejecutan en los otros procesadores. Este tipo de arquitectura sólo es útil para aplicaciones que tengan paralelismo a nivel de tarea, y las tareas sean independientes unas de otras. En la Figura 20 se muestra una posible arquitectura de este tipo.

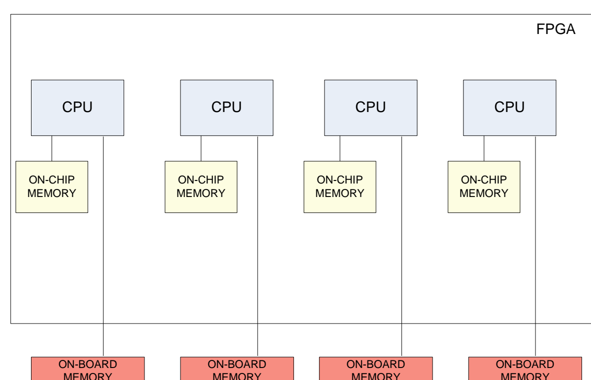


Figura 20: arquitectura multiprocesador con procesadores independientes.

En este tipo de arquitectura es posible que los procesadores no sean iguales unos a otros, y se combine incluso el uso de procesadores *soft-core* con procesadores *hard-core*.

En el caso de que las tareas que se quieren ejecutar no sean independientes, se hace necesario dotar al sistema con algún mecanismo de comunicación que permita que las aplicaciones que se ejecutan en los distintos procesadores puedan intercambiar información. El mecanismo de comunicación puede implementarse de múltiples maneras: buses de propósito general, buses de propósito específico, NoC (*network on chip*), memoria compartida, etc (Figura 21).

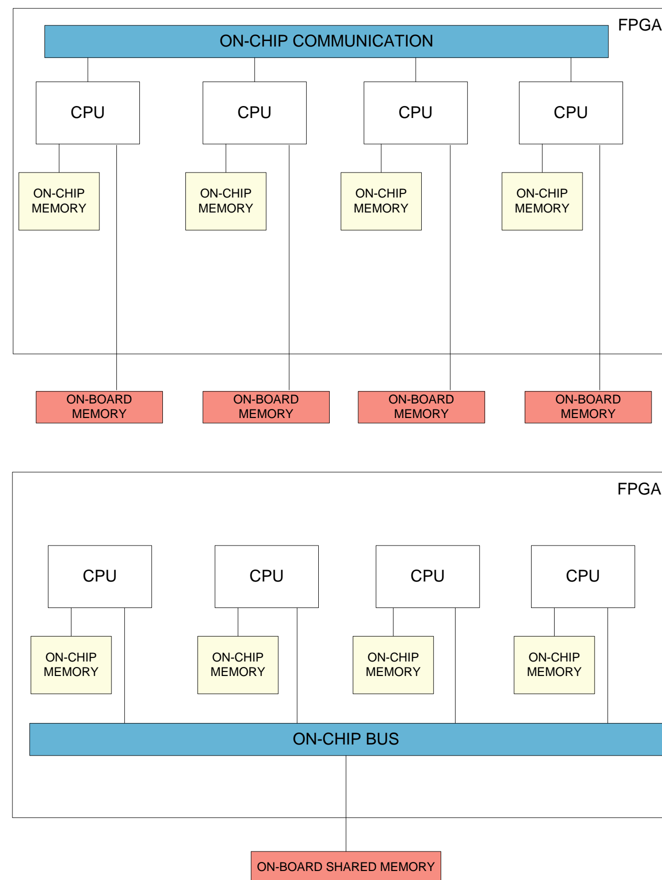


Figura 21: sistema multiprocesador con comunicación entre los procesadores a través de un bus específico (arriba), y memoria compartida (abajo).

Otra posible arquitectura multiprocesador consiste en una arquitectura maestro esclavo, en la que un maestro reparte el trabajo entre distintos procesadores esclavos. Esta arquitectura es idónea para aplicaciones de procesamiento masivo de datos, donde todos los procesadores esclavos ejecutan el mismo código y es el procesador maestro el

encargado de recibir los datos a procesar de una o varias fuentes y repartirlos entre los distintos procesadores esclavos. La Figura 22 muestra una arquitectura de este tipo.

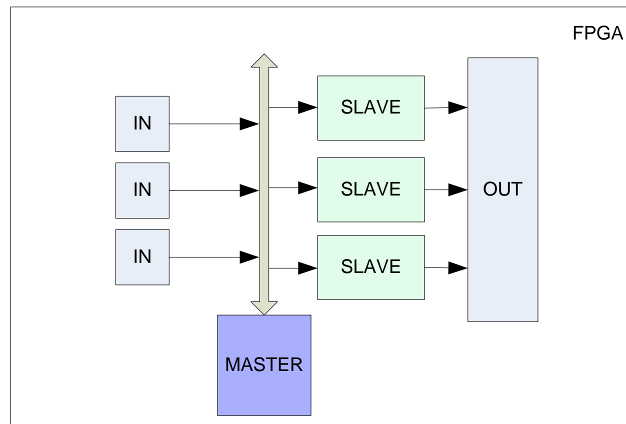


Figura 22: arquitectura multiprocesador maestro-esclavo.

También se puede utilizar esta arquitectura en aplicaciones con paralelismo a nivel de tarea, siendo el maestro el encargado de repartir las distintas tareas entre los esclavos.

Esta arquitectura puede sacar provecho también de la presencia de un procesador *hard-core* en la FPGA, utilizándolo como procesador maestro y encargado de la comunicación con el exterior y el reparto de tareas entre los esclavos.

En la Figura 23 se presenta otro tipo de arquitectura multiprocesador que se puede implementar en una FPGA. Consiste en una arquitectura segmentada que puede utilizarse para aplicaciones en las que no existe paralelismo intrínseco, pero que se pueden dividir en tareas más pequeñas que se pueden segmentar. En una arquitectura de este tipo, cada procesador ejecuta una parte de la tarea y pasa los resultados al siguiente procesador.

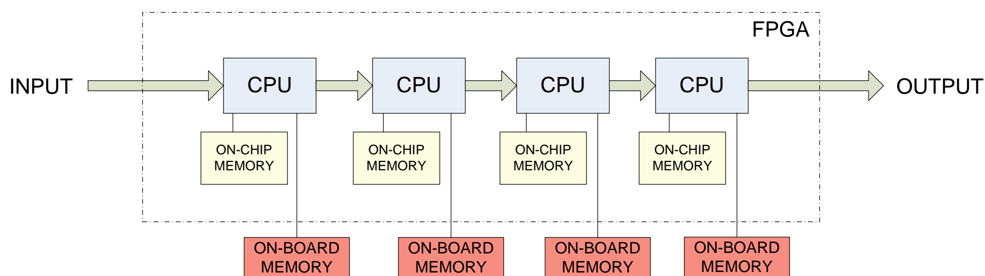


Figura 23: arquitectura multiprocesador segmentada.

A partir de esta tres arquitecturas básicas: en red, maestro-esclavo y segmentada, se pueden desarrollar múltiples sistemas multiprocesador combinándolas o modificándolas para adaptarlas a las necesidades específicas de cada aplicación.

## 4.2 Retos en el diseño de sistemas MPSoC en FPGA

### 4.2.1 Comunicaciones

Las posibilidades para conectar entre sí varios procesadores en una FPGA son múltiples(63) tanto en topología: red (64), maestro/esclavo o en cadena; como en implementación: NoC (65), buses compartidos (66)(67) o memoria compartida (68). No existe un estándar establecido, lo que hace que cada sistema emplee la solución más apropiada para la aplicación concreta que se desea implementar.

Como consecuencia de esta falta de estándar, la integración de MPSoCs es difícil y las oportunidades de reusar módulos hardware o de migrar software de un MPSoC a otro son complicadas.

### 4.2.2 Memoria

Las configuraciones de memoria que utilizan los distintos MPSoCs en FPGA existentes son muy diversas y para cada tipo de configuración existen distintas técnicas para mejorar el rendimiento del sistema completo.

Una organización típica de la memoria en este tipo de sistemas es aquella en la que cada procesador dentro del sistema tiene acceso a una zona de memoria privada interna a la FPGA, y también acceso a una memoria externa a la FPGA y compartida entre todos los procesadores. Esta compartición puede ser sólo a nivel físico: se comparte el módulo de memoria, pero cada procesador tiene asignado un espacio de direcciones en el módulo, o puede ser una compartición lógica en la que los procesadores utilizan la memoria externa por igual accediendo a variables compartidas, intercambiando mensajes, etc.

Uno de los principales problemas de los sistemas que utilizan una memoria compartida de datos y los procesadores hacen uso de una caché de datos, es el de mantener la coherencia de las cachés de datos (69).

### 4.2.3 Software

Muchos de los sistemas multiprocesador en FPGA existentes en la actualidad están diseñados para una aplicación muy específica, lo cual dificulta la tarea de desarrollar nuevas aplicaciones software para el sistema o adaptar las aplicaciones existentes para otro sistema multiprocesador diferente (70). Las dificultades se acentúan cuando el sistema cuenta con diferentes tipos de procesadores (71), ya que entran en juego nuevos problemas como: el arranque del sistema, la depuración de errores que afectan a 2 procesadores de distinta familia, o el balanceo de carga en un sistema heterogéneo.

En sistemas SMP se echa en falta un sistema operativo que permita ejecutar aplicaciones multi-hilo en este tipo de sistemas de forma transparente para el programador, y que facilite la portabilidad de aplicaciones entre distintos sistemas SMP. Sobre este aspecto se centra el trabajo principal de esta tesis, que se desarrolla a lo largo de los capítulos 7,8 y 9.

## *4.3 Algunos sistemas existentes*

Existen diversas implementaciones en FPGA de sistemas multiprocesador, principalmente orientadas a aplicaciones específicas y poco adaptables a otro tipo de aplicación. A continuación se muestran algunos de los sistemas existentes en la actualidad y como afrontan los distintos aspectos de comunicaciones, memoria y software presentados en el punto 4.2.

SoCrates(72),(73) es una plataforma que permite diseñar sistemas multiprocesadores heterogéneos combinando varias CPUs y/o DSPs. La CPU que utiliza es un clon del ARM7, y el DSP es una variación del CMUDSP de la universidad de Carnegie-Mellon. Utiliza una arquitectura de memoria compartida distribuida, con tiempo de acceso a memoria no uniforme (NUMA). Utiliza un bus compartido para interconectar los distintos CPUs y DSPs que se incluyan en el sistema (Figura 24).

Un sistema basado en SoCrates permite ejecutar aplicaciones software multi-hilo, pero queda en manos del programador de la aplicación el asignar a cada CPU los hilos que debe ejecutar. Además, los hilos que se asignan a un procesador no pueden ser migrados en tiempo de ejecución a otro procesador, lo cual puede llevar a situaciones de ineficiencia donde un procesador puede tener varios hilos pendientes de ejecutar mientras otros están parados al haber finalizado ya sus tareas. Al utilizar una arquitectura de memoria NUMA,

el enlazado de toda la aplicación en un único ejecutable que pueda ser descargado en el sistema se vuelve una tarea un tanto complicada y que debe realizarse en varias fases.

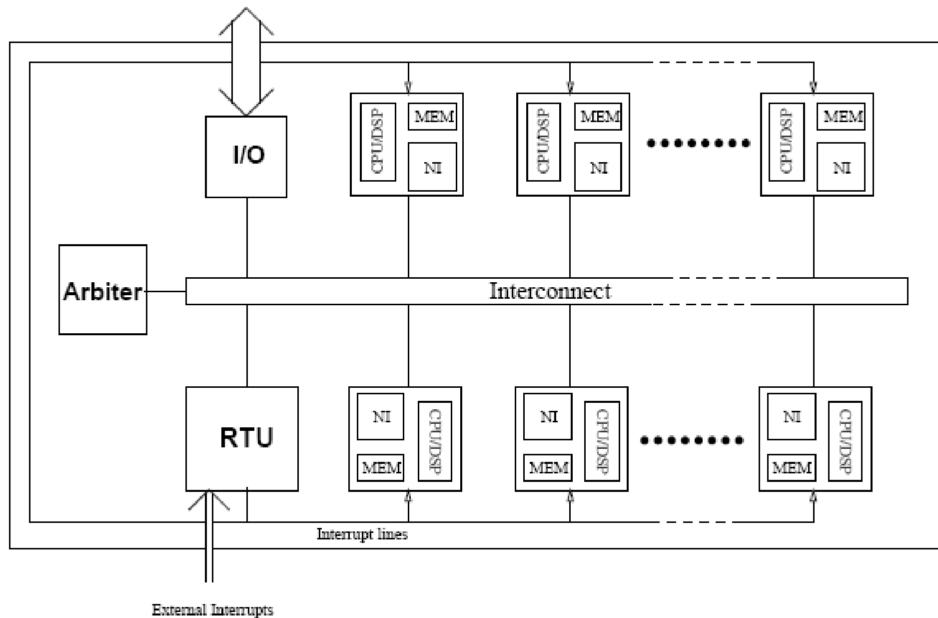


Figura 24: arquitectura SoCrates (72).

En (74) se presenta un sistema multiprocesador basado en MicroBlaze y orientado a procesamiento masivo de datos. El sistema utiliza un bus compartido para dar acceso a todos los procesadores a la memoria de datos, mientras que cada procesador almacena el código del programa que ejecutará en una memoria local no compartida. Utilizando un mecanismo del tipo maestro-esclavo, un procesador maestro se encarga de indicar a cada esclavo en que zona de la memoria compartida están los datos que debe procesar.

Para escribir aplicaciones para este sistema, se deben escribir dos aplicaciones distintas. Por una parte la aplicación que ejecutarán los esclavos, que es la misma para todos, y que consiste en un bucle que espera mensajes del procesador maestro con las indicaciones de los datos que se deben procesar y una vez recibidos son procesados. Por otra parte está la aplicación que ejecuta el maestro, que se encarga de mandar los mensajes necesarios a los esclavos, además de procesar también datos cuando no tiene mensajes que enviar.

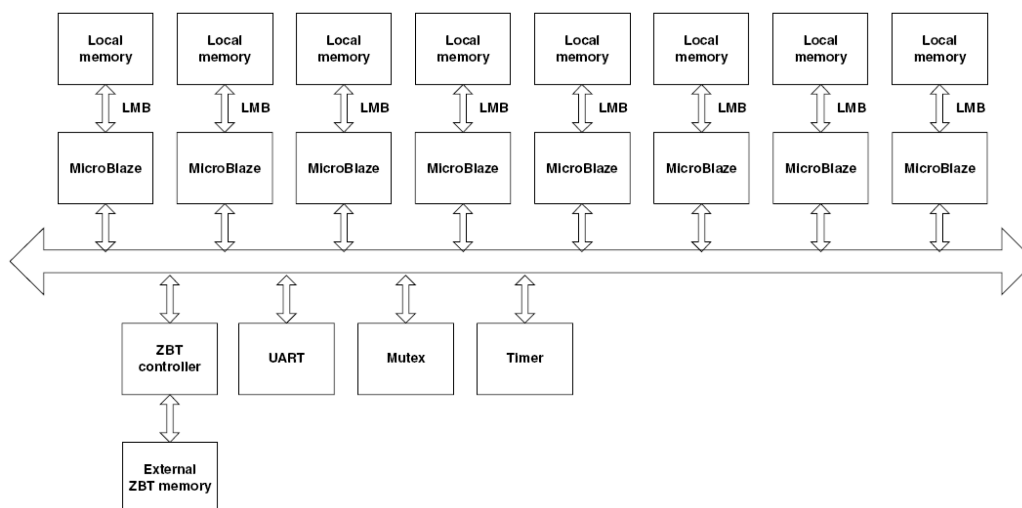


Figura 25: arquitectura del sistema de procesamiento masivo de datos (74).

En (75) y (76) se presenta otro sistema orientado a procesamiento masivo de datos. Utiliza también una arquitectura maestro/esclavo y los procesadores se conectan todos a un bus compartido. El sistema utiliza 1 procesador Nios I como maestro, y 3 procesadores Nios II como esclavos (Figura 26). A través del bus compartido se puede acceder a la memoria compartida de datos, y a través de un bus local cada procesador tiene acceso a una memoria no compartida para almacenar el programa. El sistema está pensado para una aplicación muy concreta de codificación de vídeo en formato MPEG, y no tiene mucha utilidad para aplicaciones de propósito más general al no proporcionarse una forma sencilla de escribir las aplicaciones.

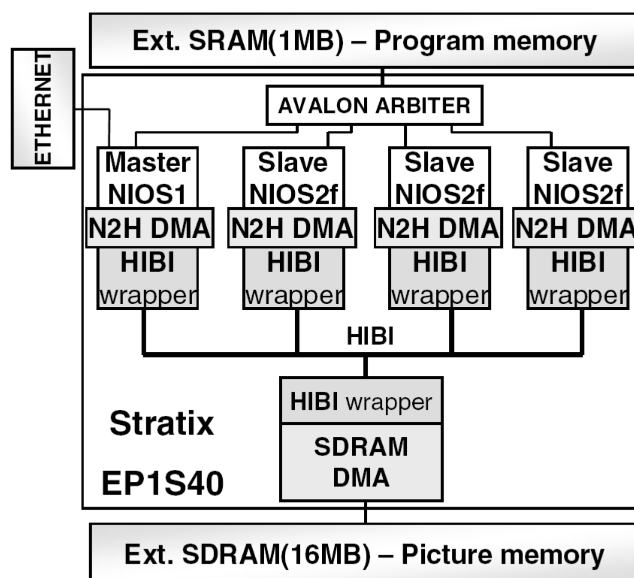


Figura 26: arquitectura de un sistema multiprocesador para codificación de vídeo (75).

En (77) se presenta un sistema multiprocesador en FPGA que combina el uso de procesadores *soft-core* y *hard-core* en una arquitectura de tipo maestro esclavo. Utiliza por una parte 4 procesadores *soft-core* Nios II como esclavos, y un procesador ARM como maestro. Todos los procesadores tienen acceso a través de un bus compartido a una memoria que utilizan sólo para datos, y cada procesador tiene una memoria no compartida para instrucciones y datos conectada a través de un bus local (Figura 27).

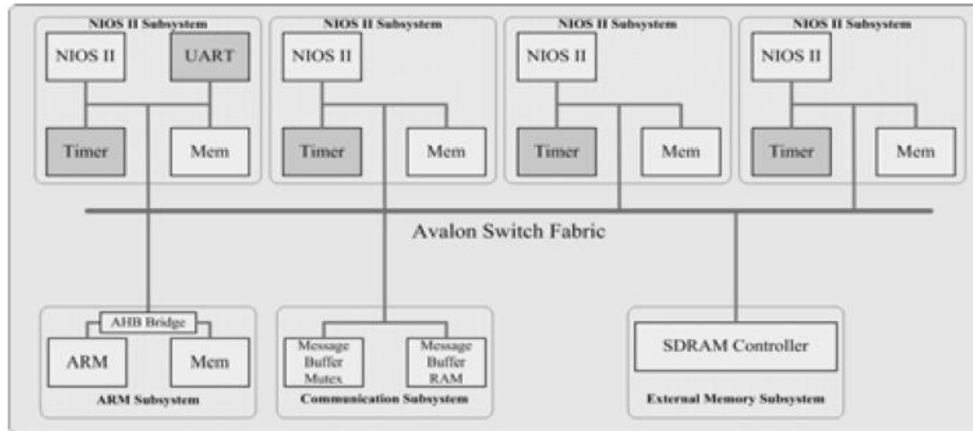


Figura 27: sistema heterogéneo con *hard-core* y *soft-core processors* (77).

Los autores utilizan una aplicación de multiplicación de matrices para probar el funcionamiento y rendimiento del sistema. La aplicación utiliza un mecanismo de paso de mensajes a través de un buffer *hardware* para comunicar el procesador maestro con los esclavos. La programación de aplicaciones para este sistema se realiza por separado para el maestro y los esclavos, y es el programador de la aplicación el que se tiene que encargar de manejar el mecanismo de paso de mensajes de forma manual lo que hace que las aplicaciones no sean portables a otro sistema, o que haya que reescribir la aplicación del procesador maestro si se añaden o eliminan procesadores esclavos al sistema.

En (78) se presenta un sistema multiprocesdor heterogéneo que utiliza una arquitectura segmentada en la que cada procesador realiza una parte del trabajo y pasa los datos al siguiente procesador. Utiliza por una parte dos procesadores PowerPC para controlar el acceso a los dos interfaces de red Ethernet que tiene el sistema y pasarle los datos a una serie de procesadores MicroBlaze que se encargan de procesar los paquetes (Figura 28).

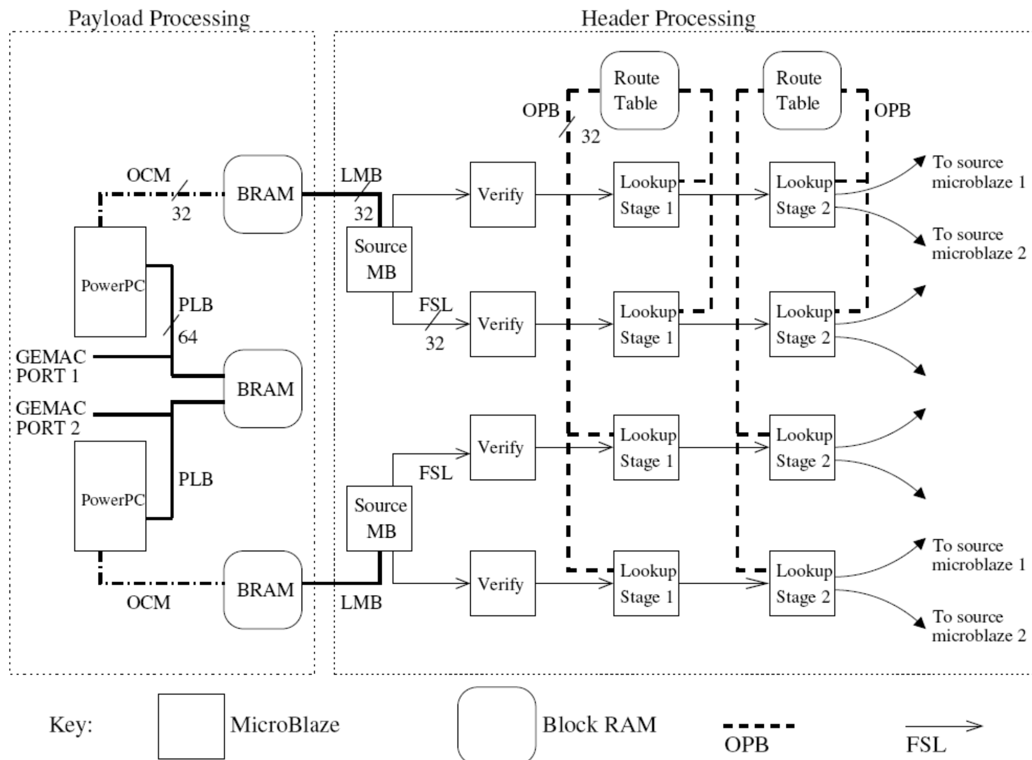


Figura 28: sistema multiprocesador heterogéneo segmentado para procesamiento de paquetes IP (78).

El sistema está diseñado para una aplicación muy específica como es el procesmiento de paquetes IP (*Internet Protocol*), por lo que utilizarlo para otro tipo de aplicación puede resultar muy complicado además de ineficiente.

Como se comentó en el apartado 4.2.2, uno de los retos en el diseño de sistemas multiprocesador en FPGA es garantizar la coherencia de la memoria cuando se utilizan procesadores con caché de datos. En (79) y (80) se presenta un módulo hardware que permite mantener la coherencia en sistemas SMP basados en el procesador *soft-core* Nios II (Figura 29). La memoria del sistema aunque está compartida completamente de forma física, no está compartida completamente de forma lógica si no que hay regiones de la memoria privadas para que cada procesador mantenga en esa región su pila y su marco de pila. Los autores dejan en manos del programador de la aplicación el gestionar de forma correcta esos espacios de memoria privada, no dando ningún tipo de librería o S.O para poder escribir aplicaciones para el sistema SMP.

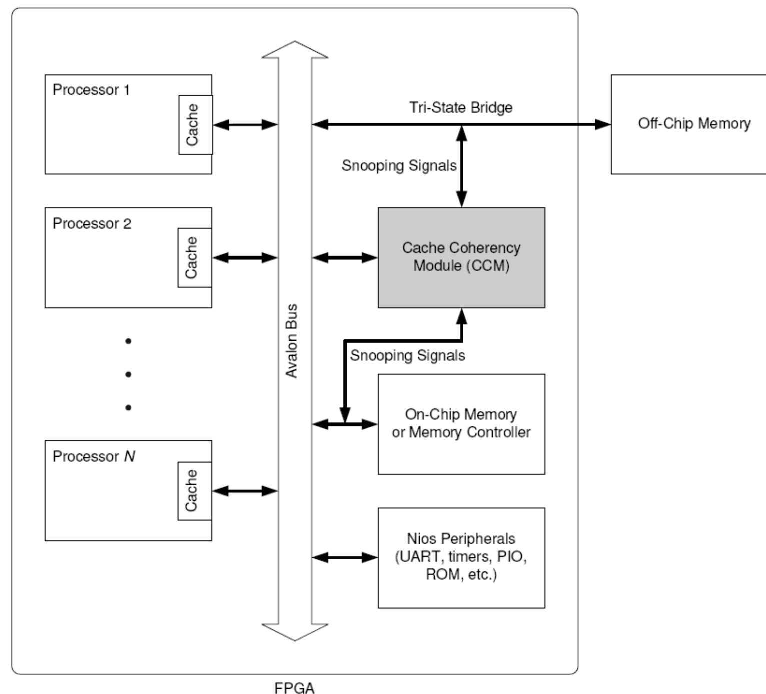


Figura 29: arquitectura SMP con módulo de control de coherencia de cachés (79).

CerberO (68) es otro sistema multiprocesador basado en el *soft-core* MicroBlaze que sigue una arquitectura SMP (Figura 30). El sistema utiliza por una parte una memoria compartida a la que tienen acceso todos los procesadores y en la que se almacena la aplicación software que se ejecute en el sistema. Por otra parte también cuenta con un sistema de sincronización y comunicación que permite intercambiar mensajes entre los distintos procesadores de forma más rápida que si se hiciese a través de la memoria compartida. Los procesadores también cuentan con una memoria local cada uno para almacenaje de datos locales al procesador, además de utilizarla también como pila en las llamadas a funciones.

El modelo de programación del sistema permite ejecutar aplicaciones multi-hilo a través de un pequeño *kernel* llamado CNK (*CerberO Nano Kernel*). Este *kernel* permite planificar hilos de ejecución en los distintos procesadores de forma dinámica, pero siguiendo una política de "ejecución hasta completar": cuando un procesador empieza a ejecutar un hilo no aceptará otro hilo del planificador hasta que termine la ejecución del hilo actual. No existe un mecanismo que permita que un hilo que empiece a ejecutarse en un procesador se planifique posteriormente en otro procesador, ni que un procesador pueda ejecutar varios hilos simultáneamente mediante una política de planificación local al procesador.

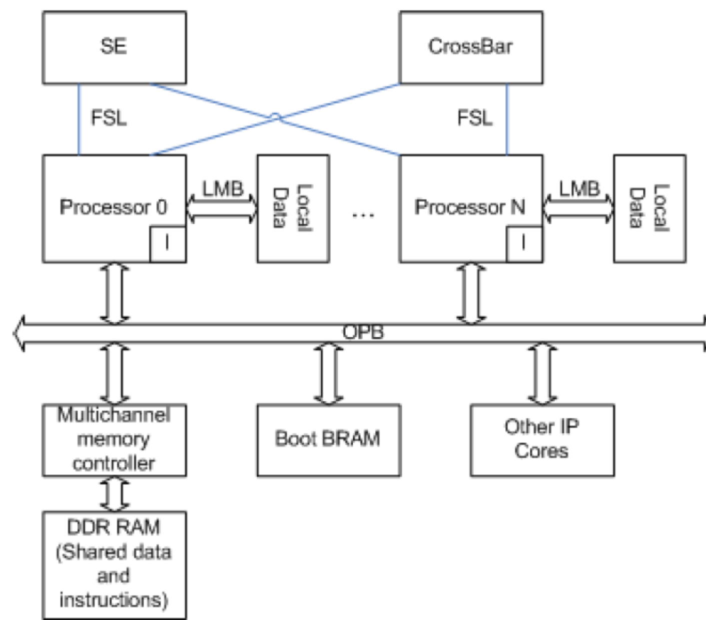


Figura 30: arquitectura de CerberO (68).

# 5 OBJETIVOS

En el estado del arte presentado se ha realizado un recorrido por los distintos tipos de sistema multiprocesador existentes, y cómo alguno de ellos han sido implementados en FPGA, tanto sistemas de propósito específico como sistemas de propósito general.

Una de las carencias que presentan los sistemas que se han estudiado es la ausencia de herramientas que permitan escribir aplicaciones software de forma sencilla, así como que estas aplicaciones sean fácilmente adaptables a otros sistemas.

Durante la realización de este trabajo doctoral se han diseñado he implementado diversos sistemas multiprocesador, con el objetivo de identificar cuáles son los principales obstáculos a la hora de implementar este tipo de sistemas y como sortearlos.

El objetivo final de este trabajo es proponer una arquitectura hardware que permita implementar de forma sencilla y eficiente sistemas multiprocesador del tipo SMP sobre FPGAs utilizando procesadores *soft-core*, además de un sistema operativo que permita escribir aplicaciones multi-hilo para estos sistemas.

Cabe destacar que todos los sistemas que se han desarrollado y todo el trabajo realizado, están hechos sobre FPGAs de Xilinx y orientado al procesador *soft-core* MicroBlaze. El motivo principal para haber escogido este fabricante es que ya se tenía experiencia en el manejo de sus herramientas de diseño, además de que ya se disponía en el Grupo de Diseño de Hardware/Software de varías placas de prototipado que integran FPGAs de prácticamente todas las familias que ofrece Xilinx (Spartan2, Virtex 2, Virtex 2P y Virtex 4).

Para mostrar el camino recorrido hasta llegar a cumplir el objetivo final, se presentan en los siguientes capítulos los distintos sistemas que se han ido implementando a lo largo del desarrollo de esta tesis. En el capítulo 6 se presenta un sistema multiprocesador con comunicaciones punto a punto, en el que se evalúa la posibilidad de utilizar los buses FSL como elemento de comunicaciones entre procesadores. El capítulo 7 presenta una primera aproximación a un sistema SMP sobre FPGA además de una librería software para desarrollar aplicaciones. En dicho capítulo se identifican algunos de los requisitos necesarios para desarrollar este tipo de sistemas. En el capítulo 8 se presenta un sistema

operativo con funcionalidad SMP junto con los requisitos de hardware mínimos que debe tener un sistema SMP sobre FPGA. Finalmente, en el capítulo 9 se presentan 4 sistemas SMP diferentes que hacen uso de distintas arquitecturas de memoria y sobre los cuales se realizan una serie de pruebas de rendimiento utilizando aplicaciones que hacen uso del sistema operativo presentado en el capítulo 8.

# 6 SISTEMA MULTIPROCESADOR CON COMUNICACIONES PUNTO A PUNTO

En este capítulo se presenta el primer sistema que se ha desarrollado para evaluar las posibilidades del *soft-core processor* MicroBlaze y el bus FSL a la hora de implementar sistemas multiprocesador en FPGA.

El sistema está formado por hasta 8 procesadores interconectados a través de un bus de alta velocidad a través del cual se intercambian datos para ser procesados. A continuación se presentarán las características principales del bus FSL, un sistema con sólo 2 procesadores para medir la velocidad de las comunicaciones, y finalmente un sistema con 8 procesadores y algunas aplicaciones que se han utilizado para evaluar el rendimiento del sistema.

## 6.1 El bus FSL

El microprocesador MicroBlaze dispone de 8 interfaces FSL de entrada y 8 de salida. Los canales FSL son canales unidireccionales punto a punto dedicados para *streaming* de datos. Tienen un ancho de palabra de 32 bits, y dos modos de transferencia: de datos o de información de control.

El rendimiento del bus FSL puede llegar teóricamente a los 300 MB/ seg (49), dependiendo del dispositivo FPGA sobre el que se implemente. Es un bus ideal para comunicación procesador-procesador, o para comunicaciones en *streaming* de entrada salida.

Las principales características del interfaz FSL son:

- Comunicación unidireccional punto a punto.
- Mecanismo de comunicación no compartido ni arbitrado.
- Soporte para comunicaciones tanto de datos como de información de control.
- Comunicación basada en FIFOs.
- Tamaño de datos configurable.

- Frecuencia de hasta 600 MHz en modo *standalone*.

El bus FSL está manejado por un maestro y maneja a un esclavo. En la Figura 31 se muestra el bus FSL y las señales disponibles.

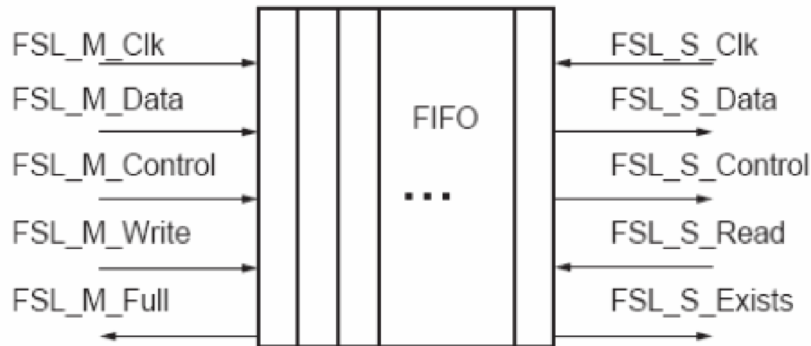


Figura 31: estructura del bus FSL.

La herramienta EDK (*Embedded Development Kit*) de Xilinx proporciona una serie de macros para leer y escribir de o hacia un interfaz FSL. Se proporcionan dos métodos de acceso al bus para lectura/escritura, bloqueante o no bloqueante, así como diferentes instrucciones para datos o palabras de control.

## 6.2 Test de comunicaciones

Para evaluar las posibilidades del bus FSL a la hora de transmitir datos entre procesadores, se ha implementado un sistema con dos procesadores para poder realizar una serie de tests de velocidad.

El sistema cuenta con dos procesadores MicroBlaze (MB_0 y MB_1) interconectados a través de 2 interfaces FSL, una para cada sentido de la comunicación. Se ha utilizado memoria de bloque para los datos e instrucciones de cada procesador, conectada al bus LMB. Además, el procesador MB_0 tiene acceso a un puerto RS232 a través del bus OPB para depuración y entrada/salida de texto, y un temporizador para poder medir el tiempo de las transferencias de datos. En la Figura 32 se muestra un esquema del sistema desarrollado.

Para probar la velocidad del enlace FSL se desarrolló un programa que transfiere diferentes cantidades de datos entre los dos procesadores, y mide el tiempo en ciclos de

reloj que tardan en realizarse dichas transferencias. Los datos transferidos van de una sola palabra de 32 bits, hasta un matriz de 32x32 enteros (4 Kbytes).

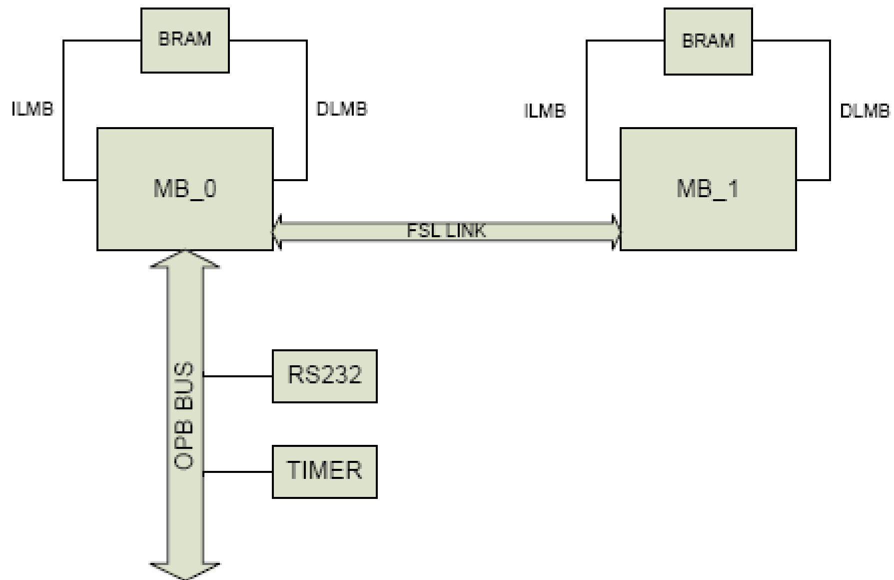


Figura 32: sistema utilizado para los tests de comunicaciones.

En la Figura 33 se muestra las distintas velocidades que se alcanzaron para transferencia directa de distintos tamaños de datos.

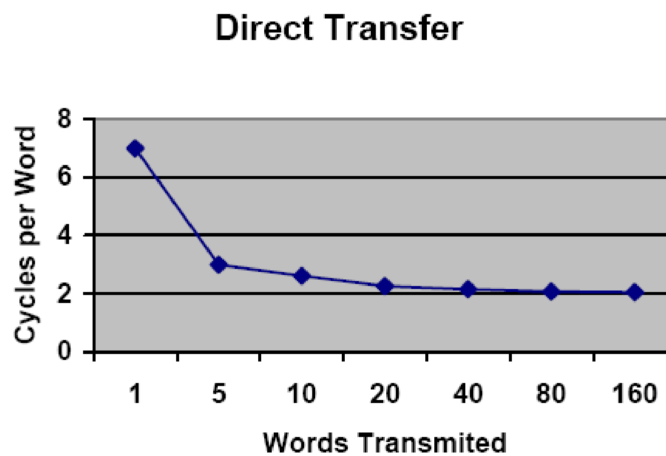


Figura 33: velocidad del bus FSL en transferencia directa de datos.

Transferencia directa de datos se refiere a que se invocan directamente las funciones de transferencia a través del bus, una vez por cada palabra que se desea transferir, sin utilizar ningún tipo de bucle de control y escribiendo directamente el dato que se quiere

transferir en el bus (no leyéndolo de memoria). Es decir, para medir cuánto tarda el bus en transferir 5 palabras, se invoca 5 veces a la función de escritura en el bus directamente con la palabra que se quiere transmitir.

```
FSL_WRITE(0x1234);
FSL_WRITE(0x1122);
FSL_WRITE(0x2233);
FSL_WRITE(0x1212);
FSL_WRITE(0x3131);
```

Tabla 1: transmisión de 5 palabras mediante llamadas consecutivas a la función de transmisión.

Este test sólo sirve para medir la máxima velocidad teórica que se podría alcanzar, ya que nunca se utilizará de este modo para realizar transferencias reales de datos, ya que como mínimo habría que considerar también la sobrecarga que supone leer los datos de memoria antes de pasarlos al bus FSL. Como se puede ver en el gráfico, la latencia mínima que se puede alcanzar al enviar datos de esta forma es de 2 ciclos por palabra enviada, lo cual concuerda con cómo es realmente el código ensamblador que ejecuta el micro para este tipo de transferencias. (Nótese que al estar el código en memoria de bloque, cada instrucción normal tarda un ciclo en ejecutarse). El código ensamblador que se corresponde con las 5 transferencias del ejemplo anterior se muestra en la Tabla 2.

```
ori    r5, r0, 0x1234;
put    r5, rfsl0;
ori    r5, r0, 0x1122;
put    r5, rfsl0;
ori    r5, r0, 0x2233;
put    r5, rfsl0;
ori    r5, r0, 0x1212;
put    r5, rfsl0;
ori    r5, r0, 0x3131;
put    r5, rfsl0;
```

Tabla 2: ensamblador del código de la Tabla 1.

Para transferencias de datos más realistas, donde no se transfieren valores inmediatos ni se utiliza una instrucción para cada transferencia, siempre será necesario realizar accesos a memoria para leer el dato que se quiere enviar, utilizar algún mecanismo para recorrer los datos que se quieren enviar, etc, lo cual introduce una sobrecarga en el tiempo de transferencia de los datos. En la Figura 34 se muestran tiempos más realistas de transferencias de datos entre los dos procesadores. Los datos que se están transfiriendo son matrices de enteros (4 bytes por cada elemento), y se hacen necesarias instrucciones de control para recorrer un bucle así como accesos a memoria para leer los datos antes de escribirlos al bus FSL.

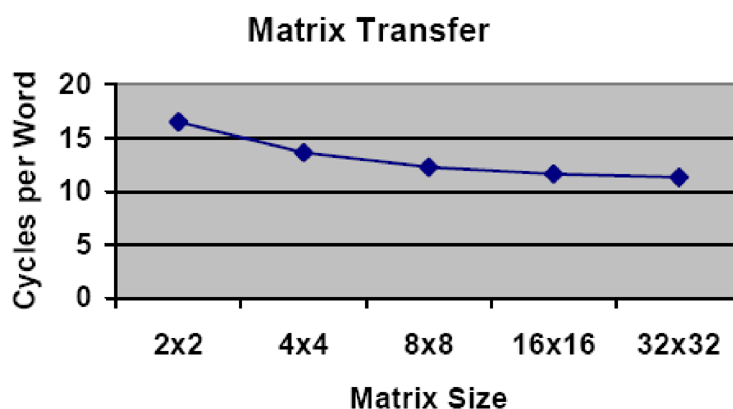


Figura 34: velocidad de transferencia para matrices de distintos tamaños.

Como se puede ver en el gráfico, la latencia más baja que se consigue para una transferencia de este tipo es 5,5 veces más lenta que en transferencia directa. Aún así, 11 ciclos por cada palabra transmitida puede ser lo suficientemente rápido como para que no introduzca una sobrecarga excesiva en según qué tipo de aplicación.

## 6.3 Topologías para una red de MicroBlazes utilizando el bus FSL

Una vez probadas las posibilidades que ofrece el bus FSL a la hora de transmitir datos entre dos procesadores, se presentan las distintas topologías que se pueden utilizar para unir varios procesadores utilizando este bus.

Utilizando conexiones punto a punto se pueden implementar distintas topologías para unir varios procesadores. A continuación se muestran las ventajas y desventajas de las tres más representativas.

### 6.3.1 Red completamente mallada

Una red completamente mallada es aquella en la que cada nodo está conectado con todos los nodos del sistema (Figura 35). Es una buena topología para reducir el tiempo de transferencia entre dos nodos ya que la información pasa de uno a otro directamente, sin pasar por nodos intermedios. El principal problema de esta topología es que el número de enlaces crece mucho si el número de nodos es grande, siguiendo la fórmula $número\ enlaces = \frac{N*(N-1)}{2}$, siendo N el número de nodos. Cómo el procesador MicroBlaze sólo tiene 8 interfaces FSL, el máximo número de procesadores que podría

tener una red completamente mallada sería de 9 procesadores y en total habría 36 buses FSL en el sistema, lo cual puede ser un gasto de recursos de la FPGA bastante elevado.

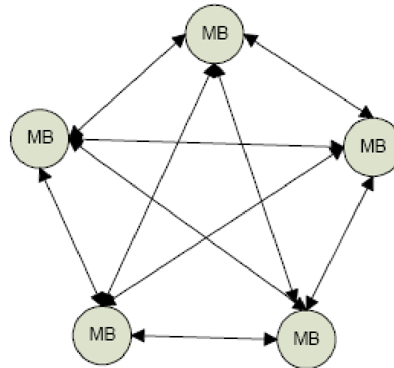


Figura 35: red completamente mallada de procesadores MicroBlaze.

### 6.3.2 Red en anillo

Una red en anillo es aquella en la que cada nodo está conectado al nodo anterior y al posterior dentro del anillo (Figura 36). Los datos pasan de nodo en nodo hasta que alcanzan el nodo destino, lo cual introduce mucha sobrecarga en las comunicaciones. Como ventaja de esta topología se puede considerar que utilizando procesadores MicroBlaze y el bus FSL no habría límite en el número de procesadores en el anillo, ya que cada procesador sólo utiliza 2 de sus 8 interfaces FSL. Cuantos más procesadores haya en el sistema, más grande será la sobrecarga que se introduce en las comunicaciones al tener que pasar los datos por más nodos intermedios antes de llegar al destino.

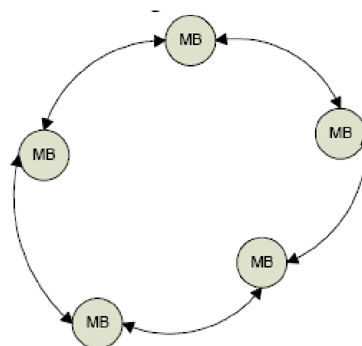


Figura 36: red en anillo de procesadores MicroBlaze.

### 6.3.3 Red en estrella

Una red en estrella es aquella en la que cada nodo está conectado a un nodo central. La principal debilidad de este tipo de redes es que si falla el nodo central, falla el sistema completo. Otra desventaja es que al pasar todas las comunicaciones por un nodo central, si la aplicación tiene muchas comunicaciones el nodo central puede llegar a saturarse convirtiéndose en un cuello de botella para el sistema. Utilizando esta topología se pueden construir sistemas con 9 MicroBlazes, uno actuando de nodo central y los 8 restantes actuando como esclavos. También se podrían implementar sistemas más grandes enlazando varios subsistemas juntos (Figura 37). Está topología es la que se ha escogido para implementar un sistema multiprocesador con *soft-cores* MicroBlaze, y se presenta a continuación.

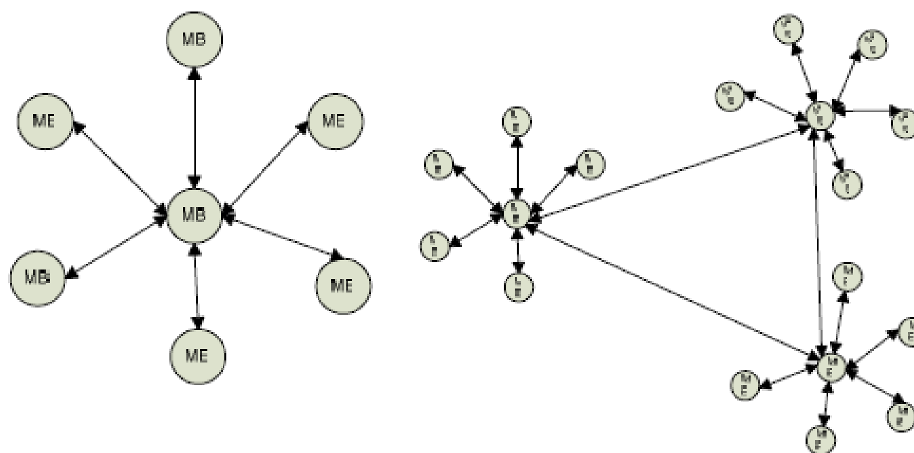


Figura 37: red en estrella (izq) y unión de varias redes en estrella (dcha).

## 6.4 Sistema completo

Una vez estudiadas las posibilidades del bus FSL y las distintas posibilidades de topologías de interconexión, se implementó un sistema con 8 procesadores interconectados en estrella, donde uno de los procesadores actuará como maestro, y los otros 7 como esclavos. El sistema completo es similar al presentado en el punto 6.2 para el test de comunicaciones, pero con 7 esclavos en lugar de 1.

Sobre este sistema se ejecutaron dos aplicaciones paralelizables diferentes: una aplicación de multiplicación de matrices, y una aplicación que hace uso de funciones

criptográficas. A continuación se presentan ambas aplicaciones y los resultados de rendimiento del sistema al ejecutarlas de forma paralela.

### 6.4.1 Multiplicación de matrices

La operación de multiplicar dos matrices es una aplicación sencilla de paralelizar y que se escogió como primera aplicación de prueba debido a que es de sobra conocida y fácil de programar de forma paralela.

El algoritmo utilizado consiste en dividir la primera matriz en tantos grupos de filas como procesadores tenga el sistema, y enviar a cada procesador las filas de la primera matriz que tiene que procesar así como la segunda matriz completa. Cada procesador calculará los elementos da la matriz resultado que le corresponda, y enviará el resultado de vuelta al procesador maestro, que los juntará en una única matriz resultado.

El algoritmo se usó para multiplicar tanto matrices de enteros como matrices de flotantes, y poder así comparar los resultados entre dos tipos de operaciones diferentes en intensidad de cómputo con la misma cantidad de datos, es decir con el mismo tiempo de comunicaciones pero distinto tiempo de cómputo.

Se debe remarcar que en el momento en el que se realizaron estas pruebas el procesador MicroBlaze aún no disponía de unidad de punto flotante, por lo que la diferencia de tiempo entre multiplicar dos número enteros y multiplicar dos números flotantes era mucho más grande que la que se podría obtener repitiendo el experimento usando la última versión del procesador Microblaze que se encuentra actualmente disponible y que dispone de unidad de punto flotante.

En la Figura 38 y en la Figura 39 se muestran los resultados obtenidos para ambos tipos de datos. Como se puede ver, para operaciones de multiplicación de matrices de enteros los resultados son bastante pobres bajando la eficiencia hasta un valor del 20 % para 8 procesadores. Esto se debe a que la multiplicación de enteros no es una operación muy intensa computacionalmente frente al número de datos que maneja, por tanto la sobrecarga que introducen las comunicaciones entre el procesador maestro y los esclavos afecta mucho al rendimiento global del sistema.

En la Figura 39 se puede ver que los resultados son mucho mejores para las matrices de números flotantes que  para las matrices enteras. Esto es debido a que para la misma cantidad de datos transmitidos el tiempo de cómputo es mucho mayor, lo que hace

que el impacto de la sobrecarga de las comunicaciones afecte mucho menos al rendimiento global del sistema.

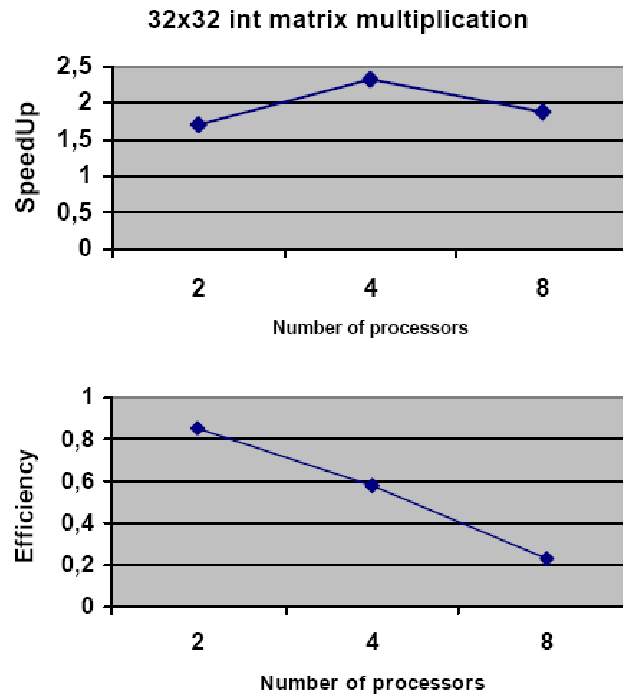


Figura 38: speedup y eficiencia para la multiplicación de matrices de enteros.

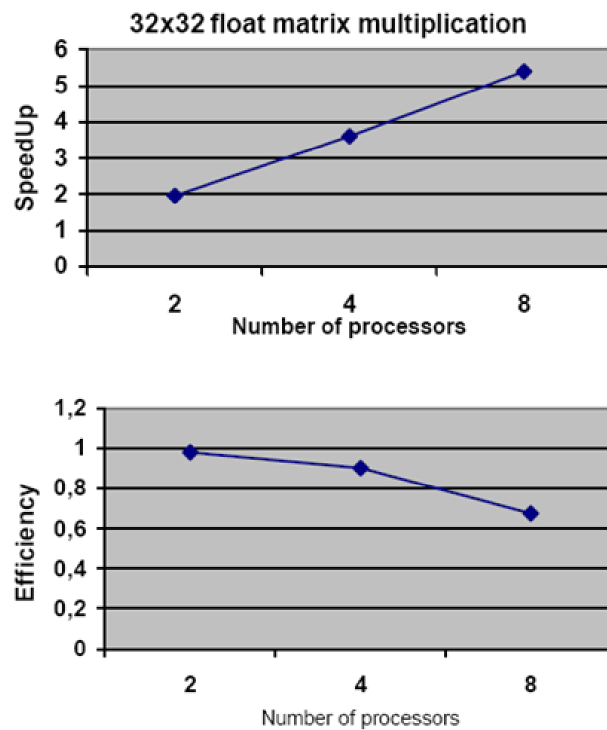


Figura 39: speedup y eficiencia para la multiplicación de matrices de flotantes.

### 6.4.2 Aplicación criptográfica

La segunda aplicación que se utilizó para probar el rendimiento del sistema es una aplicación que utiliza funciones criptográficas, y que se podría clasificar como una aplicación de procesamiento masivo de datos. Se escogió este tipo de aplicación debido a que las funciones criptográficas tienen mucha carga computacional en comparación con el número de datos con los que trabajan, lo cual permite que se pueda optimizar mucho su ejecución si se ejecutan sobre un sistema paralelo.

El test consiste en encriptar o desencriptar bloques de texto de 1 KB, utilizando el algoritmo AES(*Advanced Encryption Standard*) (81)(82). AES, también conocido como Rijndael, es un algoritmo de cifrado que se convirtió en estándar del NIST(*National Institute of Standards*) (83) en el año 2000. Este algoritmo utiliza tamaños de datos y de claves diversos, pero para esta implementación se ha escogido la versión de 128 bits de datos y 128 bits de clave. El texto a cifrar se divide en fragmentos de 128 bits y el procesador maestro va enviando a cada esclavo un fragmento a procesar así como la clave de encriptación/desencriptación que debe utilizar. Cuando los procesadores esclavos terminan sus operaciones, devuelven el resultado al procesador maestro que se encarga de ir recogiendo todos los fragmentos y unirlos.

En la Figura 40 se muestran los resultados obtenidos con este test. Como se puede observar en las gráficas, el *speedup* crece casi linealmente al aumentar el número de procesadores y la eficiencia se mantiene siempre por encima del 90 %. Esto es debido a que la cantidad de datos que se deben transmitir cuando se aumenta el número de procesadores no aumenta, al contrario que en la aplicación de multiplicación de matrices: por cada procesador que había en el sistema había que transmitir la segunda matriz una vez (4 KB para un tamaño de 32x32).

Puede parecer extraño que los resultados sean diferentes para encriptación que para desencriptación, pero esto es simplemente porque el proceso de desencriptación es más costoso computacionalmente para la misma cantidad de datos y por tanto se beneficia más del hecho de ser ejecutado en paralelo.

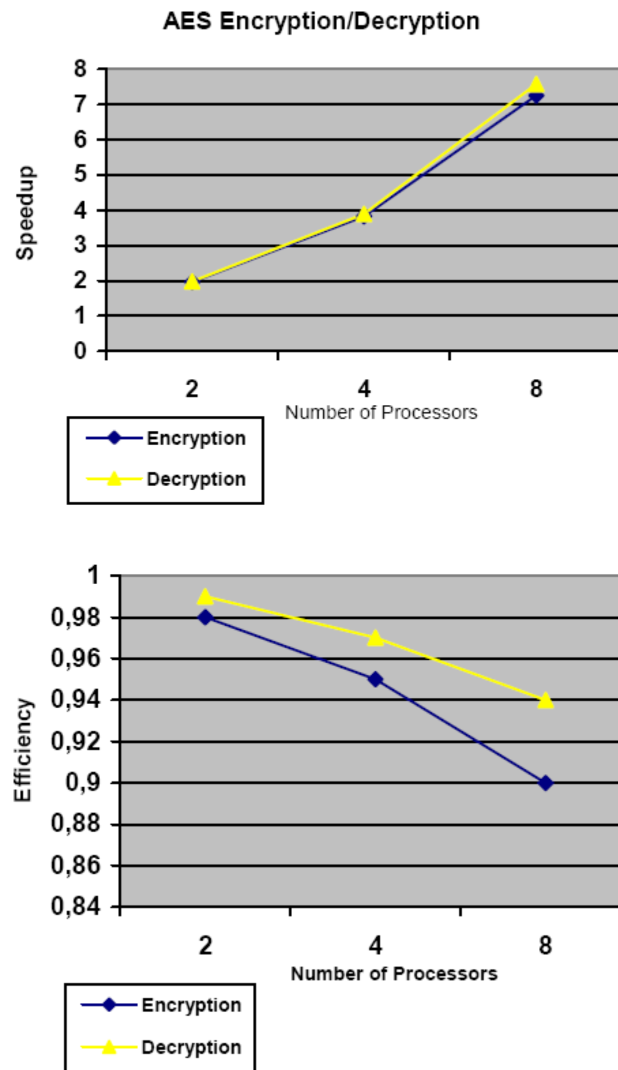


Figura 40: resultados de speedup y eficiencia para la aplicación criptográfica.

## 6.5 Implementación en FPGA

La implementación del sistema presentado en este capítulo para la ejecución de las distintas aplicaciones paralelas evaluadas se realizó sobre una placa de prototipado RC300 de Celoxica, equipada con una FPGA XC2V6000-4 de la familia Virtex II de Xilinx.

El sistema completo funciona a una frecuencia de reloj de 50 MHz, aunque los resultados de síntesis muestran que se puede utilizar un reloj más rápido, en concreto de hasta 116 MHz. En la Tabla 3 se muestran los resultados de ocupación de la FPGA para el sistema completo con 8 procesadores.

| | | | | |
|---|---|---|---|---|
| Logic Utilization: | | | | |
| Total Number Slice Registers: | 3,017 | out of | 67,584 | 4% |
| Number used as Flip Flops: | 2,985 | | | |
| Number used as Latches: | 32 | | | |
| Number of 4 input LUTs: | 5,760 | out of | 67,584 | 8% |
| Logic Distribution: | | | | |
| Number of occupied Slices: | 5,098 | out of | 33,792 | 15% |
| Total Number 4 input LUTs: | 8,940 | out of | 67,584 | 13% |
| Number used as logic: | 5,760 | | | |
| Number used as a route-thru: | 108 | | | |
| Number used for Dual Port RAMs: | 2,048 | | | |
| Number used as Shift registers: | 1,024 | | | |
| Number of bonded IOBs: | 4 | out of | 824 | 1% |
| Number of Block RAMs: | 64 | out of | 144 | 44% |
| Number of MULT18X18s: | 24 | out of | 144 | 16% |
| Number of GCLKs: | 1 | out of | 16 | 6% |

Tabla 3: resultados de síntesis para el sistema completo.

A la vista de que el grado de ocupación de *slices* de la FPGA es del 15 %, se podría pensar que se pueden construir sistemas con muchos más procesadores en FPGAs de este tamaño. El principal problema es que al utilizar la memoria de bloque de los procesadores, que no es un recurso muy abundante en las FPGAs, se consume casi la mitad de toda la memoria de bloque disponible. Cada procesador utiliza sólo 16 KB, que es una cantidad que no permite ejecutar programas muy complejos. Para programas más complejos que tengan requerimientos de memoria más grandes, se hace necesario utilizar memoria externa a la FPGA y compartirla entre los distintos procesadores lo que puede introducir un cuello de botella al tener que acceder todos los procesadores a la memoria a través de un bus compartido. En los siguientes capítulos se presentarán otros sistemas desarrollados que hacen utilización de una memoria compartida tanto para albergar las instrucciones y datos de los programas como para comunicarse los distintos procesadores a través de ella.

## 6.6 Conclusiones

En este capítulo se ha presentado un sistema multiprocesador en FPGA que utiliza varios *soft- core processors* Microblaze y buses FSL para comunicarlos.

Se han presentado algunas aplicaciones que se han utilizado para evaluar el rendimiento del sistema, obteniendo buenos resultados para aplicaciones en las que se necesita poca comunicación en comparación con el tiempo de cómputo.

En cuanto a las posibilidades de estos sistemas para ser implementados en FPGAs se ha visto que para aplicaciones que tengan requisitos de memoria bajos, se pueden utilizar hasta 8 procesadores utilizando sólo la mitad de la memoria de bloque disponible en la FPGA, si las aplicaciones necesitan más memoria se hace necesario utilizar memoria externa a la FPGA.

Este tipo de sistemas multiprocesador con comunicaciones punto a punto pueden resultar útiles para aplicaciones concretas, y se puede ajustar el diseño del sistema según los requerimientos de la aplicación concreta. Esto hace que las aplicaciones sean poco portables, y estén muy ligadas a la arquitectura hardware subyacente.

Para buscar más portabilidad entre aplicaciones se presentarán en los siguientes capítulos otro tipo de sistemas multiprocesador en FPGA: los SMP o sistemas de multiprocesamiento simétrico.

# 7 SISTEMA SMP, CON SISTEMAS OPERATIVOS INDEPENDIENTES EN CADA PROCESADOR

En el capítulo anterior se mostró un sistema multiprocesador en FPGA y algunas aplicaciones que se ejecutaron sobre él para valorar el rendimiento que podía ofrecer ese tipo de sistemas. Uno de los problemas del sistema presentado, y de arquitecturas similares, es que el software que se desarrolla para ser ejecutado en ese tipo de sistemas está muy ligado a la arquitectura que tenga el sistema, haciendo que la portabilidad de las aplicaciones a sistemas parecidos pero con algunas características diferentes sea un trabajo complicado.

En este capítulo se presentará otro sistema desarrollado dentro del trabajo de esta tesis. El trabajo se desarrolló para evaluar las posibilidades y retos, tanto hardware como software, que ofrecen las FPGAs para implementar sistemas de multiprocesamiento simétrico.

El sistema que se presenta busca como uno de sus objetivos el permitir que las aplicaciones sean portables a otros sistemas similares sin necesidad de hacer cambios en el programa. También se busca que el programador de aplicaciones vea el sistema como un todo, sin tener que preocuparse sobre cómo está implementada la arquitectura hardware subyacente, cómo están interconectados los procesadores, cuántos hay, etc.

El sistema que se presenta en este capítulo sigue una arquitectura SMP, donde todos los procesadores utilizan el mismo mecanismo para acceder a la memoria y a los periféricos. Así mismo se presenta una capa software que se ha desarrollado para correr sobre el sistema operativo *xilkernel*, dotándolo con las características necesarias para tener funcionalidad SMP. Esta capa software permite desarrollar y ejecutar de forma transparente aplicaciones en el sistema, y permite que éstas sean portables entre sistemas similares.

También se presenta en este capítulo un periférico diseñado para permitir sincronización entre procesos que se están ejecutando en distintos procesadores, y que permita acceder a los recursos compartidos de forma ordenada.

## 7.1 Retos Software

Existen muchos sistemas operativos que se pueden utilizar con los *soft core processors* más populares, como pueden ser eCos, MicroC/OS II o *xilkernel*, pero ninguno de ellos tiene soporte nativo para sistemas SMP.

En esta sección se presentan algunas de las características que un S.O debería tener para soportar sistemas SMP.

### 7.1.1 Transparencia

Una de las condiciones que debe tener un sistema operativo con soporte para sistemas de multiprocesamiento simétrico es que el número de procesadores en el sistema no sea relevante para la aplicación, de forma que la aplicación pueda ser ejecutada haya más o menos procesadores en el sistema y que además, la aplicación no necesite ser modificada cuando se lleva a otro sistema con distinto número de procesadores.

### 7.1.2 Política de planificación

En un sistema operativo con soporte para SMP se pueden considerar dos posibilidades a la hora de planificar tareas en los distintos procesadores:

- Planificación estática: cuando una tarea es asignada a un procesador no es posible migrarla luego a otro procesador.
- Planificación dinámica: una tarea que está siendo ejecutada por un procesador puede ser migrada a otro procesador.

Una planificación dinámica puede obtener mejores resultados en cuanto a rendimiento, pero es más difícil de implementar que una planificación estática.

### 7.1.3 Comunicación entre procesadores

En un sistema de memoria compartida, como son los sistemas SMP, la comunicación entre procesadores se hace de forma intrínseca al compartir las variables localizadas en la memoria compartida.

Si existe algún otro mecanismo de comunicación en el sistema, como pueden ser buses dedicados, *network-on-chip*, etc, el S.O debe proporcionar mecanismos para abstraer a los programadores de los mecanismos de comunicación que están por debajo.

### 7.1.4 Sincronización entre procesadores

Múltiples procesadores accediendo simultáneamente a recursos compartidos puede conducir a problemas críticos, que deberían evitarse. Se hace necesario un mecanismo de sincronización que permita a los procesadores acceder de forma ordenada a los distintos recursos compartidos. Las implementaciones software de los mecanismos clásicos de sincronización que proporcionan la mayoría de los sistemas operativos, como pueden ser los semáforos, *mutex*, etc, pueden no funcionar de la forma que se espera si no se añade algún soporte hardware extra.

## *7.2 Arquitectura Hardware*

El sistema desarrollado inicialmente consta de 2 procesadores MicroBlaze. El sistema se muestra en la Figura 41, y sus componentes principales son:

- 2 procesadores MicroBlaze con 32 Kbytes de memoria local para cada uno, a través del bus LMB.
- Un bus OPB compartido por ambos procesadores.
- Una UART RS232 para entrada/salida y para depuración.
- Un display LCD para mensajes de salida.
- 1 Mbyte de memoria SRAM.
- Un componente para sincronización entre procesadores: *hardware_mutex*.

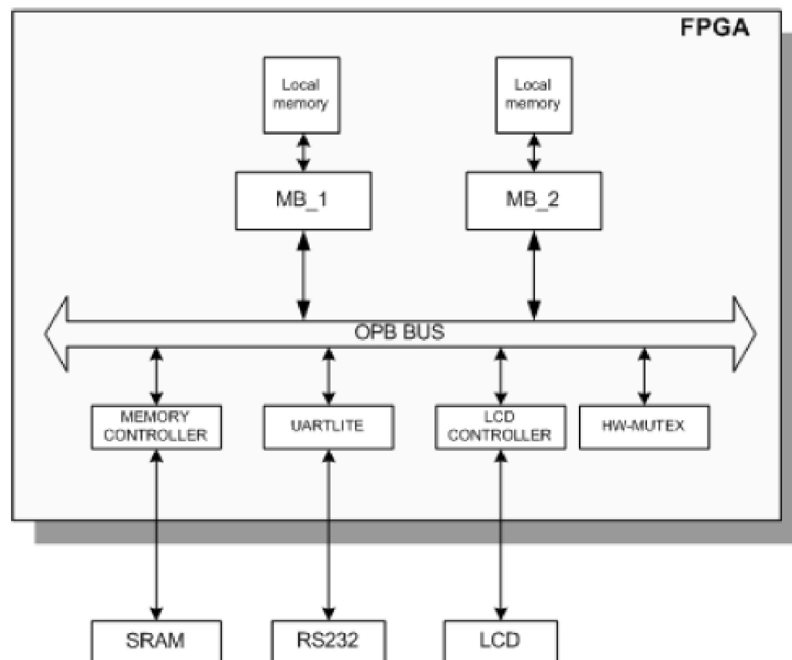


Figura 41: arquitectura del sistema implementado.

### 7.2.1 Hardware mutex

Los mecanismos de sincronización que proporciona el sistema operativo sobre el que se ha montado el sistema, *xilkernel*, no son apropiados para ser usados por procesos que corren en procesadores distintos. Estos mecanismos, tal como están diseñados, son sólo apropiados para ser usados en sistemas con un único procesador.

Cualquier operación de un mecanismo de sincronización debe ser atómica, y la forma que tiene *xilkernel* de garantizar esa atomicidad consiste en desactivar las interrupciones cuando se produce una llamada a estas funciones, garantizando así que la llamada no será interrumpida a la mitad, pudiendo entrar a ejecutar otro proceso diferente que interfiera en el correcto funcionamiento de esa llamada.

Si el sistema tiene varios procesadores este método no es válido, ya que la operación de desactivar las interrupciones solo desactiva las del procesador que lo solicita. Se hace necesario algún mecanismo hardware que permita garantizar la atomicidad de ciertas operaciones, y con ese propósito se ha desarrollado un periférico específico: el *hardware_mutex*.

El *hardware_mutex* es un periférico que se conecta al bus OPB, y que tiene un número configurable de registros internos accesibles desde los procesadores conectados al

bus. Cada registro se comporta como un *mutex* tradicional, que puede estar en dos posibles estados: bloqueado o desbloqueado. Inicialmente todos los registros están en estado desbloqueado. Cuando un procesador intenta leer un *mutex* que está desbloqueado, la operación devuelve un '1' y el *mutex* pasa al estado bloqueado. Cualquier operación de lectura sobre un *mutex* que está bloqueado devuelve un '0'. La operación de desbloquear el *mutex* se realiza escribiendo sobre él un '0', y esta operación sólo debería ser realizada por el proceso que lo bloqueó ya que en esta primera versión del periférico no se ha incluido información sobre el procesador que mantiene al *mutex* bloqueado. La lógica de arbitraje del bus OPB garantiza que sólo un procesador puede acceder al *mutex* en un momento dado, garantizándose así la atomicidad de las operaciones de lectura y escritura sobre él.

Para facilitar la utilización de este periférico, se ha creado una pequeña librería que contiene las funciones y tipos de datos necesarios para manejar el periférico. Las funciones principales de esta librería son:

- *void hw_mutex_lock(hw_m m)*: intenta bloquear el *mutex m*. Esta función no retorna hasta que se consigue bloquear el *mutex*.

- *int hw_mutex_try_lock (hw_m m)*: intenta bloquear el *mutex m*. Si se consigue bloquear devuelve un '1', en caso contrario un '0'.

- *void hw_mutex_unlock(hw_m m)*: desbloquea el *mutex m*. Esta función sólo debe ser ejecutada por un proceso que previamente haya bloqueado el *mutex*, ya que en esta primera versión el *mutex* no guarda información de que proceso lo ha bloqueado.

Tras haber realizado el diseño e implementación del *hardware_mutex*, y como prueba de la necesidad y oportunidad de su diseño, Xilinx añadió a su librería un periférico denominado *opb_mutex* con algunas funciones similares y otras ampliadas. Este componente está presente desde la versión 9.1 de la herramienta Xilinx Platform Studio, bastante posterior a la 8.2 con la que se realizó este trabajo.

## 7.3 Arquitectura software

Para poder desarrollar programas y ejecutarlos aprovechando la existencia de varios procesadores, se ha desarrollado una capa software que corre sobre el sistema operativo *xilkernel* proporcionado por Xilinx. Esta capa software incluye una serie de funciones para escribir hilos que se ejecutarán en los distintos procesadores. Además de estas funciones

también se incluye un planificador SMP, que será el encargado de repartir los distintos hilos entre los procesadores disponibles. Este *schdeuler* SMP sólo se ejecutará en uno de los procesadores, que será el procesador maestro. Todos los procesadores, incluyendo el maestro, tendrán su propia imagen de *xilkernel* en su memoria local.

El planificador SMP será el responsable de asignar tareas a los distintos procesadores dependiendo de la disponibilidad de cada uno de ellos. Para ello mantendrá una lista con las tareas que está ejecutando cada procesador, y cuando se cree una nueva tarea se la asignará al procesador que menos tareas tenga en ese momento.

Una vez que una tarea ha sido asignada a un procesador, la imagen de *xilkernel* que maneja ese procesador será la encargada de planificar localmente esa tarea, y se quedará asignada a ese procesador hasta que la tarea finalice, no pudiendo ser traspasada a otro procesador.

En el arranque del sistema todos los procesadores salvo el maestro se encuentran ejecutando un hilo de comunicaciones que espera mensajes del planificador SMP. Cuando una tarea se crea, el planificador SMP selecciona un procesador y le envía un mensaje indicando la dirección de inicio de la nueva tarea así como los parámetros que lleva asociada. El procesador que recibe este mensaje crea un hilo POSIX con esa información y su copia de *xilkernel* pasará a manejarlo y planificarlo. Cuando el hilo finaliza, se envía un mensaje al planificador SMP para que actualice la información de carga del sistema. La Figura 42 muestra un posible escenario con dos procesadores que ejecutan cada uno dos tareas además del hilo de comunicaciones.

La principal función que proporciona la API del planificador SMP es la que se utiliza para crear nuevas tareas. Su prototipo es:

```
int SMP_thread_create(smp_pthread thread,
smp_pthread_attr_t* attr, void* (*start_func)(void*), void*param)
```

Como se puede observar, la llamada es similar a la llamada tradicional de POSIX para crear procesos ligeros. La diferencia es que esta llamada va a crear un hilo de más alto nivel que se asignará a uno de los procesadores del sistema SMP, y posteriormente ese procesador será el encargado de localmente convertirlo en un hilo POSIX que será ejecutado por la copia local de *xilkernel*.

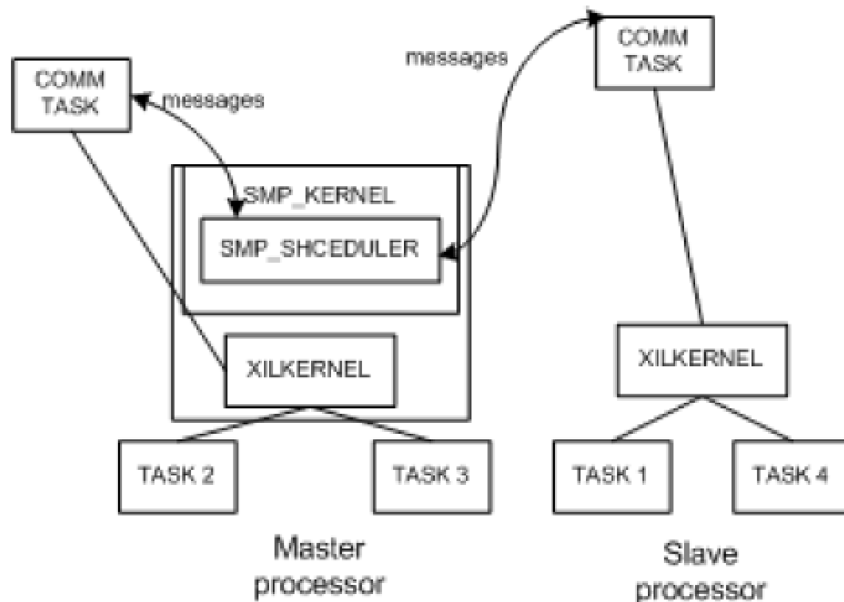


Figura 42: sistema con 2 procesadores ejecutando varias tareas.

## 7.4 Escribiendo aplicaciones

El proceso de escribir aplicaciones para un sistema como el descrito en los puntos anteriores consta de dos partes diferenciadas. Por una parte se debe compilar una copia de *xilkernel* para cada uno de los procesadores que forman parte del sistema. Esta copia estará situada en la memoria local de cada procesador, y será la responsable de planificar de forma local las tareas que se asignen a cada procesador además de comunicarse a través de un hilo de comunicaciones con el planificador SMP que correrá en el procesador maestro. La copia local de *xilkernel* que se ejecuta en el procesador maestro será la encargada de inicializar el planificador SMP.

Por otra parte se debe compilar la aplicación que se desea ejecutar en el sistema, y que usará la API del planificador SMP para crear las distintas tareas en que esté dividida la aplicación.

El mapa de memoria del sistema completo cuando ejecuta tareas de este tipo se muestra en la Figura 43.

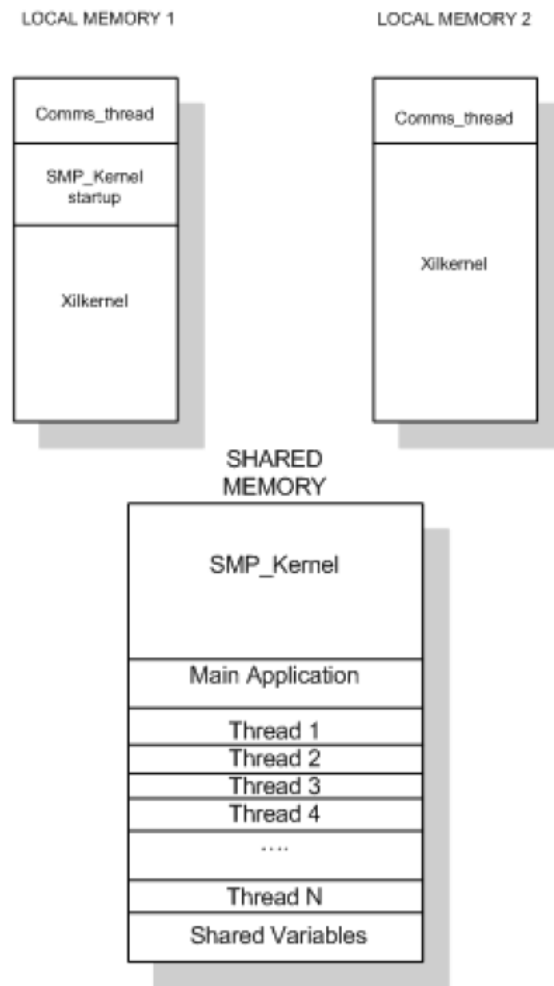


Figura 43: mapa de memoria del sistema.

## 7.5 Resultados experimentales

El sistema que se ha presentado en este capítulo se construyó y probó sobre una placa de desarrollo ML403 de Xilinx, que incluye una FPGA XC4VFX12 de la familia Virtex 4 (Figura 44).

Figura 44: placa ML403 de Xilinx.

### 7.5.1 Resultados de síntesis

En la Tabla 4 se muestran los resultados de síntesis del sistema sobre una FPGA XC4VFX12. Como se puede ver en la tabla el sistema ocupa el 39 % de los *slices* de la FPGA y el 88 % de la memoria de bloque. Con ese grado de ocupación de la memoria interna de la FPGA no es posible implementar un sistema con más de 2 procesadores MicroBlaze.

```
Design Summary:

Logic Utilization:
  Number of Slice Flip Flops:        1,577 out of  10,944   14%
  Number of 4 input LUTs:            2,684 out of  10,944   24%

Logic Distribution:
  Number of occupied Slices:        2,171 out of   5,472   39%

Total Number 4 input LUTs:           3,556 out of  10,944   32%
  Number used as logic:             2,684
  Number used as a route-thru:         21
  Number used for Dual Port RAMs:     640
  Number used as Shift registers:     211

Number of FIFO16/RAMB16s:             32 out of      36   88%

Number of DSP48s:                      6 out of      32   18%
```

Tabla 4: resultados de síntesis del sistema.

### 7.5.2 Tests hardware y software

Para evaluar el correcto funcionamiento del periférico *hardware_mutex* se escribió una aplicación que envía mensajes a un PC a través del puerto serie desde los distintos procesadores del sistema. Para ello la aplicación crea dos tareas, una en cada procesador, que envían un mensaje de texto al terminal del PC. Si no se utiliza ningún mecanismo de sincronización, las dos tareas utilizarán simultáneamente el periférico RS232, provocando que los caracteres de los mensajes de cada tarea se mezclen con los de la otra tarea e incluso se pierdan algunos caracteres, como se muestra en la Figura 45.

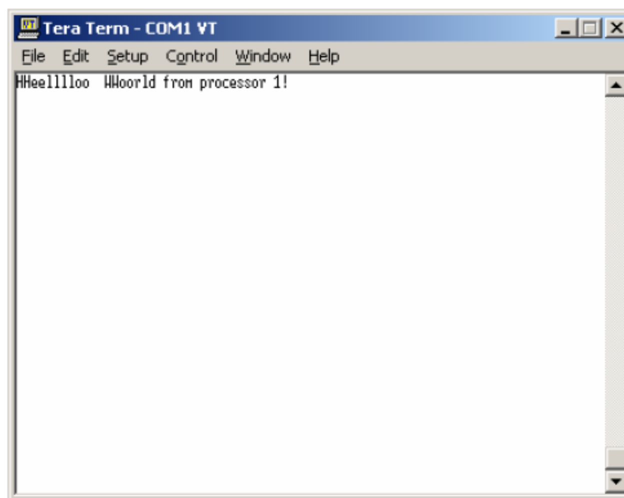


Figura 45: funcionamiento incorrecto del terminal debido al acceso simultáneo de los 2 procesadores.

Si se utiliza el *hardware_mutex* que se ha desarrollado para garantizar el acceso exclusivo al puerto RS232, se obtiene el resultado esperado. En la Figura 46 se muestran los resultados de la ejecución correcta del ejemplo anterior al utilizar el *hardware_mutex*.

Para comprobar el correcto funcionamiento del planificador SMP se desarrolló una aplicación que crea varios hilos que se van ejecutando en los dos procesadores que tiene el sistema. La aplicación envía varios mensajes a través del terminal mostrando cómo se van creando y distribuyendo los distintos hilos.

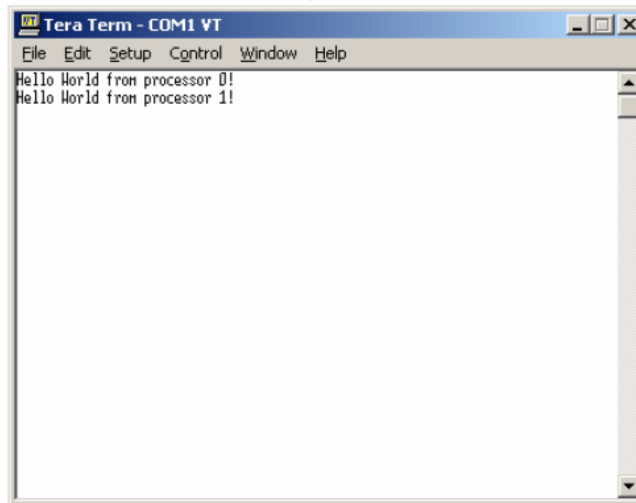


Figura 46: funcionamiento correcto del terminal utilizando un *hardware-mutex*.

## 7.6 Conclusiones y problemas encontrados en el sistema

En este capítulo se ha presentado un sistema SMP junto con una capa software que permite repartir hilos para que se ejecuten sobre la copia local del S.O *xilkernel* que almacena cada procesador.

El elevado uso de memoria de bloque que requiere cada procesador para almacenar su copia local del S.O no permite que en el sistema se puedan integrar más de dos procesadores. En los capítulos siguientes se presenta una forma diferente de enfrentar el problema que permite implementar sistemas más elaborados con un único sistema operativo para todos los procesadores en lugar de una copia local para cada uno.

También se ha presentado el diseño a medida de un periférico hardware que permite la sincronización de los dos procesadores para que accedan de forma ordenada a los recursos compartidos del sistema. La prueba de que un periférico de este tipo es necesario es que Xilinx incluye un periférico similar desde una versión bastante posterior a la que se utilizó para este desarrollo.

El uso del periférico *hardware_mutex* es responsabilidad del programador de la aplicación software, lo que puede llevar a usos incorrectos que conduzcan a un funcionamiento no esperado de la aplicación. En el capítulo siguiente se presenta un sistema operativo que va a utilizar por debajo este mismo periférico, pero el programador no tiene que manejarlo directamente si no que se utilizarán los mecanismos de sincronización software clásicos del estándar de hilos POSIX.

# 8 SISTEMA OPERATIVO PARA SISTEMAS SMP EN FPGA

En este capítulo se presenta un sistema operativo para sistemas multiprocesador MicroBlaze con una arquitectura de memoria compartida tipo SMP.

El sistema operativo que se ha desarrollado en este trabajo doctoral se basa en *xilkernel*, el *microkernel* proporcionado por Xilinx para trabajar con los procesadores MicroBlaze y PowerPC, y se ha modificado para que pueda soportar de forma transparente y eficiente sistemas SMP basados en MicroBlaze. En este capítulo se muestran los requisitos hardware que son necesarios para que el S.O implementado pueda funcionar, y se presentan los fundamentos de funcionamiento de dicho S.O.

## 8.1 Requisitos de la arquitectura hardware

Una arquitectura SMP sobre la que se quieran ejecutar aplicaciones que hagan uso del sistema operativo que se va a presentar, debe disponer de una serie de características hardware que son necesarias para poder realizar algunas de las tareas propias del S.O como son la planificación de procesos entre procesadores, la comunicación y la sincronización entre procesos, etc. Las características imprescindibles para que el S.O pueda realizar dichas tareas son:

- Una región grande de memoria compartida, mapeada en las mismas direcciones para todos los procesadores. En esta región es donde residirá tanto el sistema operativo como la aplicación de usuario.

- Una pequeña región de memoria no compartida, para cada uno de los procesadores, que será usada como pila cuando el sistema ejecute ciertas funciones en modo *kernel*. Con un tamaño de 1 KB es suficiente para esta región de memoria.

- Un mecanismo de identificación del procesador. En determinadas rutinas del S.O, este debe tomar unas decisiones u otras dependiendo de qué procesador esté ejecutando esas rutinas. La principal necesidad de esta identificación es durante el proceso de planificación y en los cambios de contexto, momentos en los cuales el *kernel* necesita saber que procesador es el que está realizando dichas operaciones.

- Un mecanismo hardware de sincronización que permita proporcionar acceso exclusivo a secciones críticas del código a los distintos procesadores.

### 8.1.1 Hardware Abstraction Layer (HAL)

El sistema operativo puede ser utilizado por diversos tipos de arquitecturas SMP, con diferentes características o diferentes implementaciones de los requisitos mencionados en el punto anterior. Para separar la parte funcional del sistema operativo de la parte dependiente del hardware, se ha creado una capa de abstracción del hardware o HAL (*Hardware Abstraction Layer*) donde se describe cómo están implementadas las partes de bajo nivel dependientes del hardware que permiten satisfacer los 4 requisitos descritos en el apartado anterior.

Las características que se definen en la capa de abstracción son las siguientes:

- NUM_CPUS: número de procesadores que tiene la arquitectura sobre la que se va a ejecutar el S.O.

- SHARED_MEMORY_BEGIN: inicio de la zona de memoria compartida por todos los procesadores.

- SHARED_MEMORY_SIZE: tamaño de la zona de memoria compartida.

- PRIVATE_MEMORY_BEGIN: inicio de la zona de memoria privada de cada procesador.

- PRIVATE_MEMORY_SIZE: tamaño de la zona de memoria privada.

- GET_PROCESSOR_ID(VAR): instrucción o conjunto de instrucciones necesarias para leer el identificador del procesador que utilice la arquitectura para la que se va a utilizar el S.O.

- HARDWARE_MUTEX_LOCK / HARDWARE_MUTEX_UNLOCK(ADDRESS): instrucción o conjunto de instrucciones necesarias para tomar / ceder el control del mecanismo de sincronización hardware que implemente la arquitectura sobre la que se va a utilizar el S.O.

El objetivo del HAL es que el sistema operativo se pueda portar con la máxima sencillez posible para que funcione con cualquier arquitectura SMP que implemente los requisitos hardware mencionados en el punto 8.1.

A continuación se muestra un ejemplo de cómo se debería escribir el HAL para una arquitectura concreta como la mostrada en la Figura 47, que consta de:

- Dos procesadores MicroBlaze.
- Una zona de memoria compartida a través del bus OPB.
- Una zona de memoria privada accesible a través del bus LMB.
- Un mecanismo de sincronización, *hardware_mutex*, como el presentado en el capítulo 7.
- Un registro accesible a través del bus FSL con el identificador del procesador.

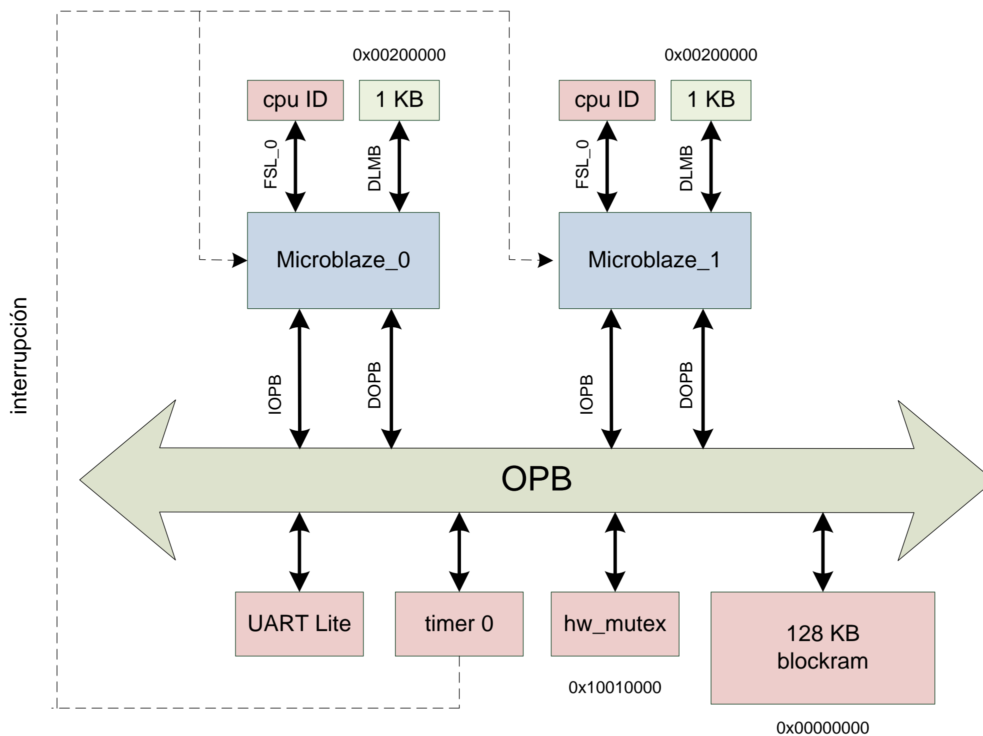


Figura 47: sistema SMP utilizando procesadores soft-core MicroBlaze.

Para este sistema se tendría un HAL como el que se muestra en la Tabla 5. Con esas definiciones el S.O ya sabe todo lo necesario sobre las características de la plataforma, y puede usar esa información y esas funciones para realizar de forma correcta tanto la planificación de procesos en los distintos procesadores, como otro tipo de operaciones que se ven afectadas por la coexistencia de varios procesadores en el sistema.

```
#define NUM_CPUS 2
#define SHARED_MEMORY_BEGIN      0x00000000
#define SHARED_MEMORY_SIZE       0x00200000  // (128 KB)
#define PRIVATE_MEMORY_BEGIN     0x10000000
#define PRIVATE_MEMORY_SIZE      0x00000400 // (1 KB)
#define GET_PROCESSOR_ID (VAR)   \
             microblaze_bread_datafsl(VAR, 0)
#define HARDWARE_MUTEX_LOCK            \
             hw_mutex_lock(0x10010000)
#define HARDWARE_MUTEX_UNLOK     \
             hw_mutex_unlock(0x10010000)
```

Tabla 5: HAL para el sistema de la Figura 47.

## 8.2 Estructura del sistema operativo con soporte para SMP

En esta sección se presentan los detalles de implementación del sistema operativo. Por una parte se mostrarán las estructuras de datos que se utilizan para almacenar la información sobre los procesos que se están ejecutando y los procesadores que hay en el sistema. Por otra parte se presentará como están implementadas y cómo funcionan las funciones principales del *kernel* que permiten realizar las tareas de inicialización, atención a interrupciones, planificación de procesos y llamadas al sistema.

### 8.2.1 Estructuras de datos que utiliza el S.O

El sistema operativo utiliza distintas estructuras de datos para almacenar diversa información sobre los procesos que están en el sistema, así como de los procesadores.

La estructura de datos que utiliza el S.O para albergar las características de los procesos que se ejecutan en el sistema se llama *process_struct* y contiene la siguiente información acerca de cada proceso (Figura 48):

- *process_context*: estructura de datos para almacenar el estado del procesador para cada proceso (el contenido de los 32 registros de propósito general, más el registro de estado del procesador *MSR*)
- *state*: estado del proceso ( *new, run, ready, wait, dead, . . .*).

- *pid:* identificador del proceso.

- *priority*: prioridad del proceso. Sólo va a ser útil en el caso de que se haya escogido un esquema de planificación basado en prioridades.

- *is_allocated:* indica si este *process_struct* está asignado a un proceso o está libre para ser utilizado por un proceso de nueva creación.

- *block_q*: si el proceso está bloqueado, esta variable indica en qué cola está bloqueado.

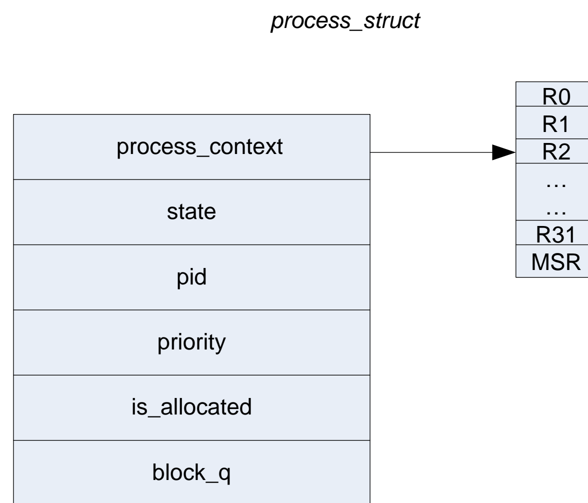


Figura 48: estructura de datos para almacenar el estado de un proceso.

El sistema operativo dispone una tabla de procesos llamada *ptable[ ]*, que contiene un array de *process_structs* (Figura 49) con tantos elementos como procesos simultáneos pueda soportar el S.O, lo cual se puede configurar a través de un parámetros del S.O llamado *MAX_PROCS*.

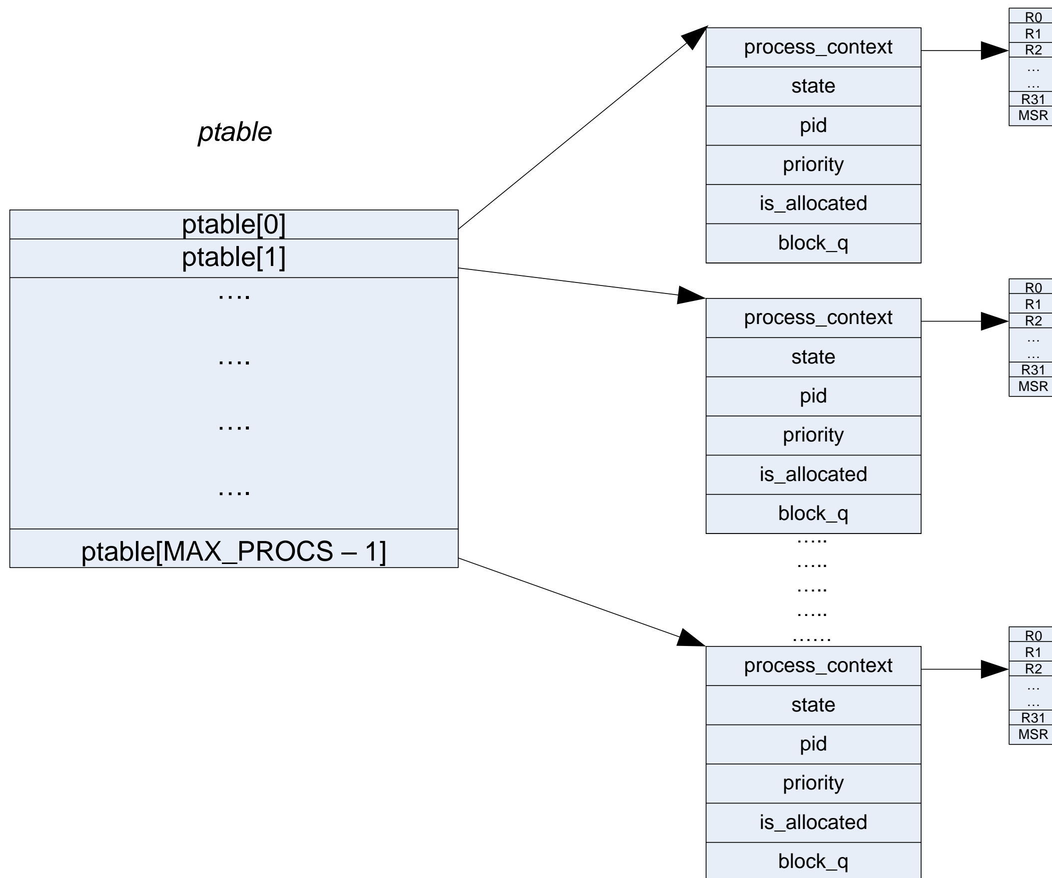


Figura 49: estructura con la tabla de procesos del sistema.

Además de estas dos estructuras de datos para almacenar el estado de los procesos, se hace necesario llevar un control de qué proceso se está ejecutando en cada procesador. Para ello, se utiliza un array de punteros a *process_struct*, llamado *current_process[ ]*. El array tendrá tantos elementos como procesadores haya en el sistema, y cada elemento apuntará al *process_struct* del proceso que esté ejecutando ese procesador (Figura 50). Así, *current_process[0]* apuntará en todo momento al proceso que esté ejecutando el procesador número 0, *current_process[1]* apuntará al proceso que esté ejecutando el procesador número 1, etc.

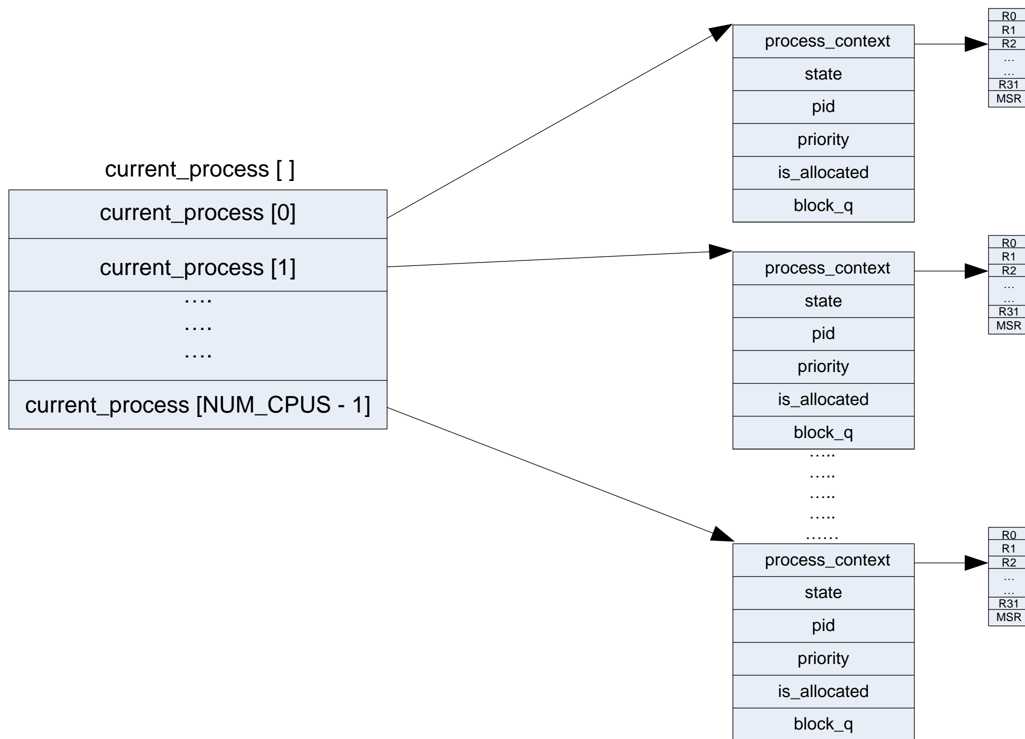


Figura 50: estructura con los procesos asociados a cada procesador.

### 8.2.2 Rutinas principales

Las estructuras de datos presentadas en el punto anterior son manejadas por el sistema operativo cuando se realizan determinadas acciones, como son la inicialización del sistema, el manejo de interrupciones, la planificación de procesos y las llamadas al sistema. A continuación se muestra como están implementadas dichas acciones en el sistema operativo.

#### 8.2.2.1 Inicialización

Durante la inicialización, el sistema operativo tiene que configurar varios elementos. Por una parte tiene que configurar los dispositivos hardware que utiliza para realizar ciertas tareas, como son el controlador de interrupciones y el temporizador del sistema. Por otra parte debe inicializar una serie de estructuras de datos y colas para dar soporte a los distintos módulos del S.O: estructuras de datos para los procesos, la cola del planificador y otras estructuras y colas para los módulos opcionales que tiene el S.O (hilos, semáforos, colas de mensajes, etc).

En concreto la secuencia ordenada de acciones que realiza el sistema al iniciarse es:

- Configurar el temporizador.
- Inicializar las estructuras de datos para los procesos.
- Inicializar la cola del planificador.
- Inicializar el módulo *pthread*, para dar soporte a procesos ligeros con interfaz POSIX.
- Crear una tarea *idle_task* que no hace nada útil y que se encarga de mantener ocupados a los procesadores si no hay procesos listos para ejecutar.
- Crear los hilos estáticos definidos por el usuario, y ponerlos en la cola del planificador.
- Activar las interrupciones.
- Ejecutar el proceso *idle_task* en cada procesador.

Realizada esta inicialización, cuando llegue la primera interrupción del temporizador comenzarán a planificarse los distintos procesos que están listos para ejecutar en los procesadores disponibles en el sistema.

El código de inicialización del sistema operativo no debe ser ejecutado íntegramente por todos los procesadores, si no que sólo uno de ellos debe realizar ciertas tareas. El procesador encargado de las principales tareas de inicialización recibe el nombre de procesador de arranque (*boot processor*), y será el que tenga como identificador de procesador el número ‘0’.

En la Figura 51 se muestra el diagrama de flujo del código de inicialización del S.O, y se puede ver que partes son realizadas solamente por el *boot-processor*.

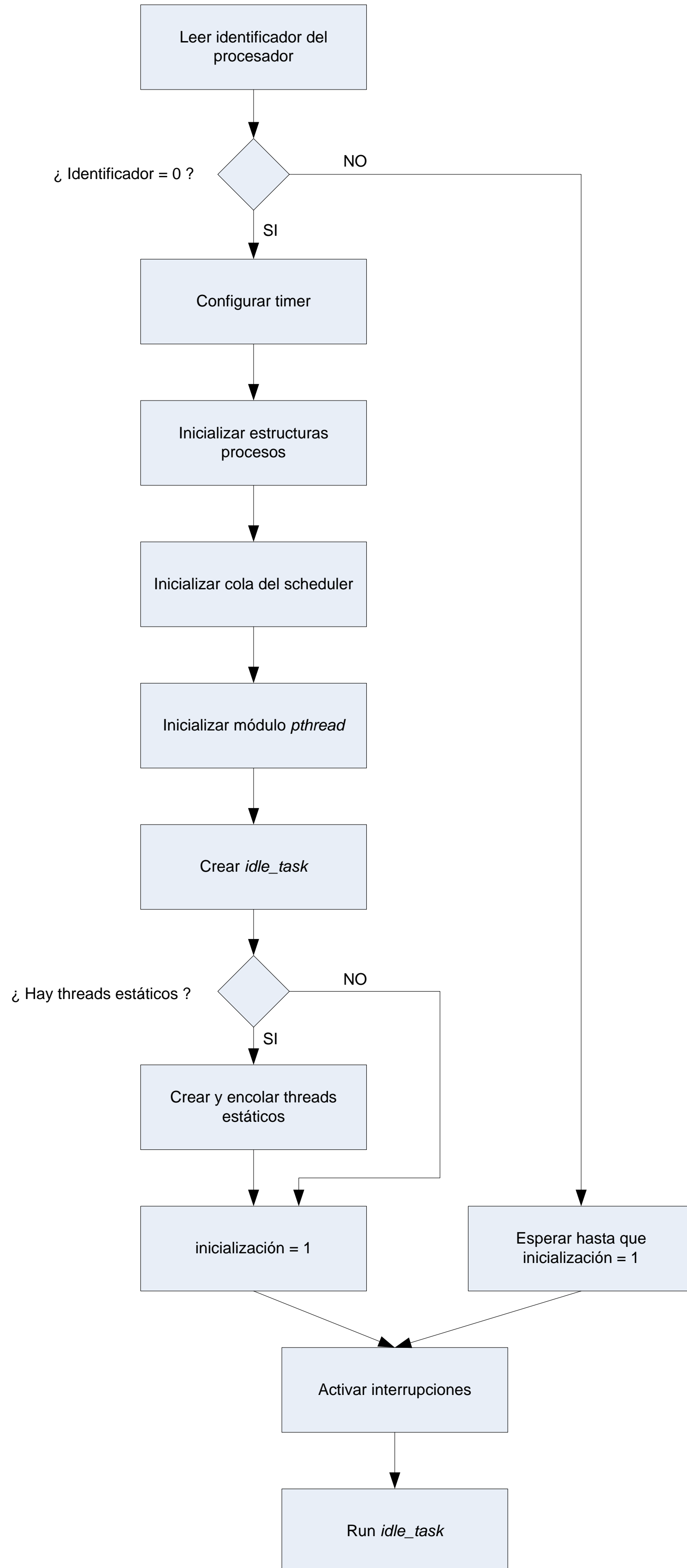


Figura 51: diagrama de flujo de la inicialización del S.O.

#### 8.2.2.2 Interrupciones

La rutina de atención a la interrupción opera de forma similar a como se hace en el tratamiento clásico de interrupciones: deshabilitar las interrupciones, salvar el estado del procesador, cambiar a la pila (*stack*) del *kernel*, atender la interrupción, restaurar el estado del procesador y rehabilitar las interrupciones.

El hecho de tener varios procesadores hace que se complique un poco el diseño, ya que todos son interrumpidos simultáneamente, pero sólo uno de ellos debe atender la interrupción. La única interrupción que debe ser atendida por todos los procesadores es la interrupción del temporizador, que es la que se utiliza para lanzar el planificador y pasar a ejecutar nuevos procesos que estuvieran en espera. En esta primera versión del S.O no se da soporte a otras fuentes de interrupción que no sea el temporizador, y se deja ese aspecto para una versión futura en la que probablemente se necesite de algún controlador de interrupciones preparado para trabajar con varios procesadores (84).

Otro problema a la hora de tratar las interrupciones surge a la hora de cambiar a la pila del *kernel*. Esta pila se utiliza cuando se ejecutan funciones del *kernel*, como llamadas al sistema o rutinas de atención a la interrupción. El hecho de tener varios procesadores hace recomendable que exista una pila por procesador para simplificar la gestión, ya que si sólo existiese una pila compartida por todos los procesadores sería necesario implementar mecanismos de control de acceso a la pila para evitar situaciones de conflicto, como que todos los procesadores a la vez puedan atender a la interrupción del temporizador, o que dos procesadores realicen simultáneamente una llamada al sistema, que podrían llevar a un comportamiento no deseado del sistema completo.

Para tener una pila por procesador, se ha optado por exigir que cada procesador tenga una región de memoria no compartida, y que todos la tengan mapeada en la misma dirección, tal como se explicó en el punto 8.1. Esta forma de resolver el problema de pilas separadas no es la única posible, pero se ha escogido como primera opción por que simplifica el diseño de las rutinas de inicialización del S.O así como de los *scripts* de enlazado de las aplicaciones de usuario. En (80) utilizan un sistema que hace uso de pilas separadas en la memoria compartida, pero dificulta la tarea de compilar y enlazar las aplicaciones. En un futuro se añadirá la opción de que en lugar de tener esta región no compartida de memoria, se puedan tener en la zona de memoria principal varias pilas separadas, una para cada procesador.

A continuación se detallan las distintas acciones que realiza la rutina de atención a las interrupciones:

- Salvar el estado del procesador en el bloque de control de proceso que corresponda (*process_struct*). Para ello es necesario saber que procesador está ejecutando la rutina de atención a la interrupción, y obtener un puntero al *process_struct* del proceso que está ejecutando dicho procesador.

- Cambiar el puntero de pila para que apunte a la pila del *kernel*.

- Al tener soporte sólo para las interrupciones del temporizador, se llama directamente a la rutina del planificador para que lance otro proceso que esté listo para ejecución. Si existiese soporte para distintas fuentes de interrupción habría que llamar a la rutina específica para la interrupción que se hubiera producido. Después de que finalice la rutina del planificador, se obtiene un puntero al *process_struct* del nuevo proceso planificado.

- Restaurar el estado del procesador a partir de la información del *process_struct*.

- Retornar de la interrupción.

Entre las acciones descritas en la rutina no aparece la habilitación/deshabilitación de las interrupciones, ya que las realiza automáticamente el procesador tanto al saltar a la rutina de atención a la interrupción como al ejecutar la instrucción de retorno de interrupción.

El diagrama de flujo del código de la rutina de atención a la interrupción se muestra en la Figura 52.

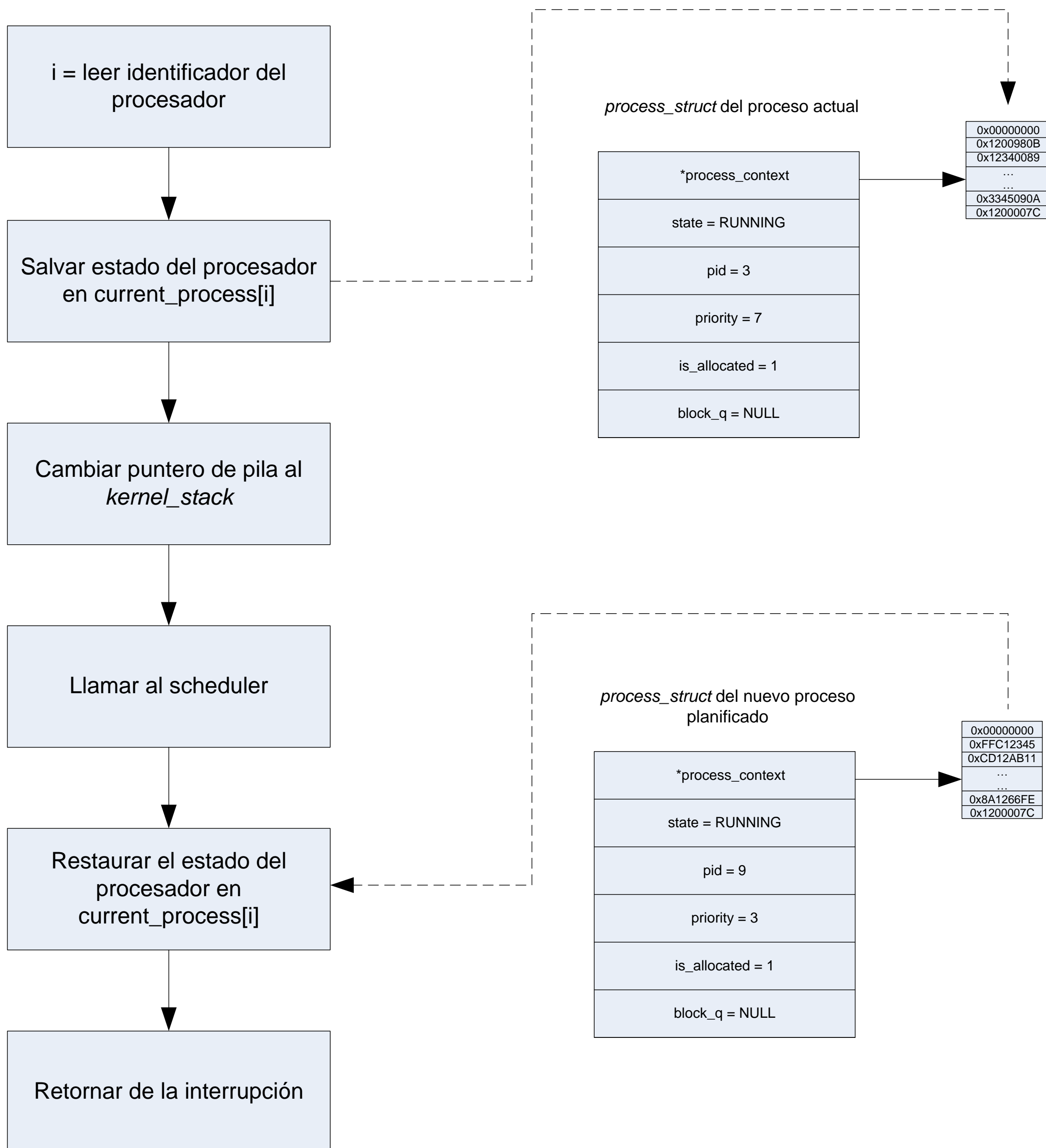


Figura 52: funcionamiento de la rutina de atención a la interrupción.

#### 8.2.2.3 Planificación

El planificador o *scheduler*, es la parte del sistema operativo que se encarga de repartir el tiempo de procesamiento entre los distintos procesos que estén listos para ser ejecutados.

El modelo de procesos que implementa el sistema operativo distingue entre seis posibles estados en los que puede estar un proceso:

- PROC_NEW: proceso nuevo, recién creado.
- PROC_READY: proceso listo para ser ejecutado.
- PROC_RUN: proceso actualmente en ejecución.
- PROC_WAIT: proceso bloqueado en algún recurso.
- PROC_DELAY: proceso que está esperando por un *timeout*.
- PROC_TIMED_WAIT: proceso bloqueado en un recurso, y con un *timeout* asociado.

Cuando un proceso termina pasa a un estado llamado PROC_DEAD. También se puede llegar a ese estado si el proceso recibe una señal de finalización (*kill*). El planificador libera posteriormente los recursos de los procesos que están en este estado.

La Figura 53 representa las distintas transiciones que pueden hacer los procesos entre los distintos estados. La función del planificador consiste en cambiar periódicamente a un proceso que está en el estado PROC_RUN al estado PROC_READY, y pasar el primer proceso de la cola de procesos listos del estado PROC_READY al estado PROC_RUN. Cuando un proceso está bloqueado, por ejemplo en un mutex o un semáforo, y posteriormente se desbloquea pasa al estado PROC_READY, quedando listo para ser ejecutado cuando le llegue el turno.

En la versión del sistema para un único procesador sólo puede estar un proceso en el estado PROC_RUN en un determinado instante de tiempo, mientras que en el sistema modificado para múltiples procesadores puede haber tantos procesos en ese estado como procesadores. En este tipo de sistema, cada procesador ejecuta la rutina de planificación cuando llega una interrupción del temporizador, que se encarga de expulsar al proceso que esté ejecutando el procesador en ese instante y pasa a ejecutar un proceso de la lista de procesos listos.

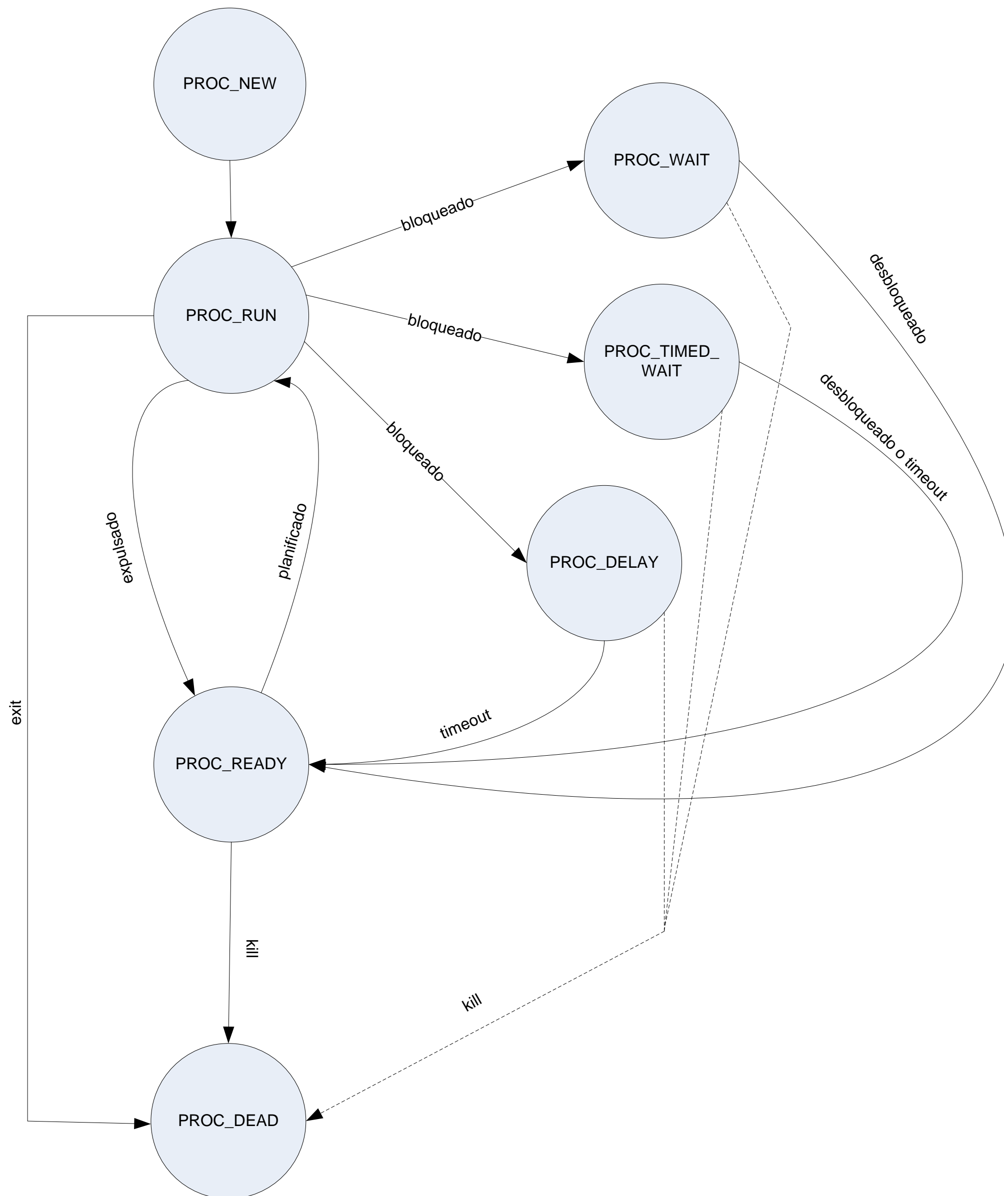


Figura 53: transiciones entre los distintos estados de un proceso.

La Figura 54 muestra los distintos pasos que realiza la rutina de planificación para realizar estas acciones. Se debe tener en cuenta que esta rutina no sólo es ejecutada cuando hay una interrupción del temporizador, sino que también se activa con algunas llamadas al sistema o cuando finaliza un proceso. Por eso, el estado del proceso que estaba asignado al procesador cuando se lanza el planificador podría ser distinto de PROC_RUN: podría ser PROC_WAIT si el proceso se encuentra bloqueado, o podría ser PROC_DEAD si el proceso acaba de finalizar su ejecución.

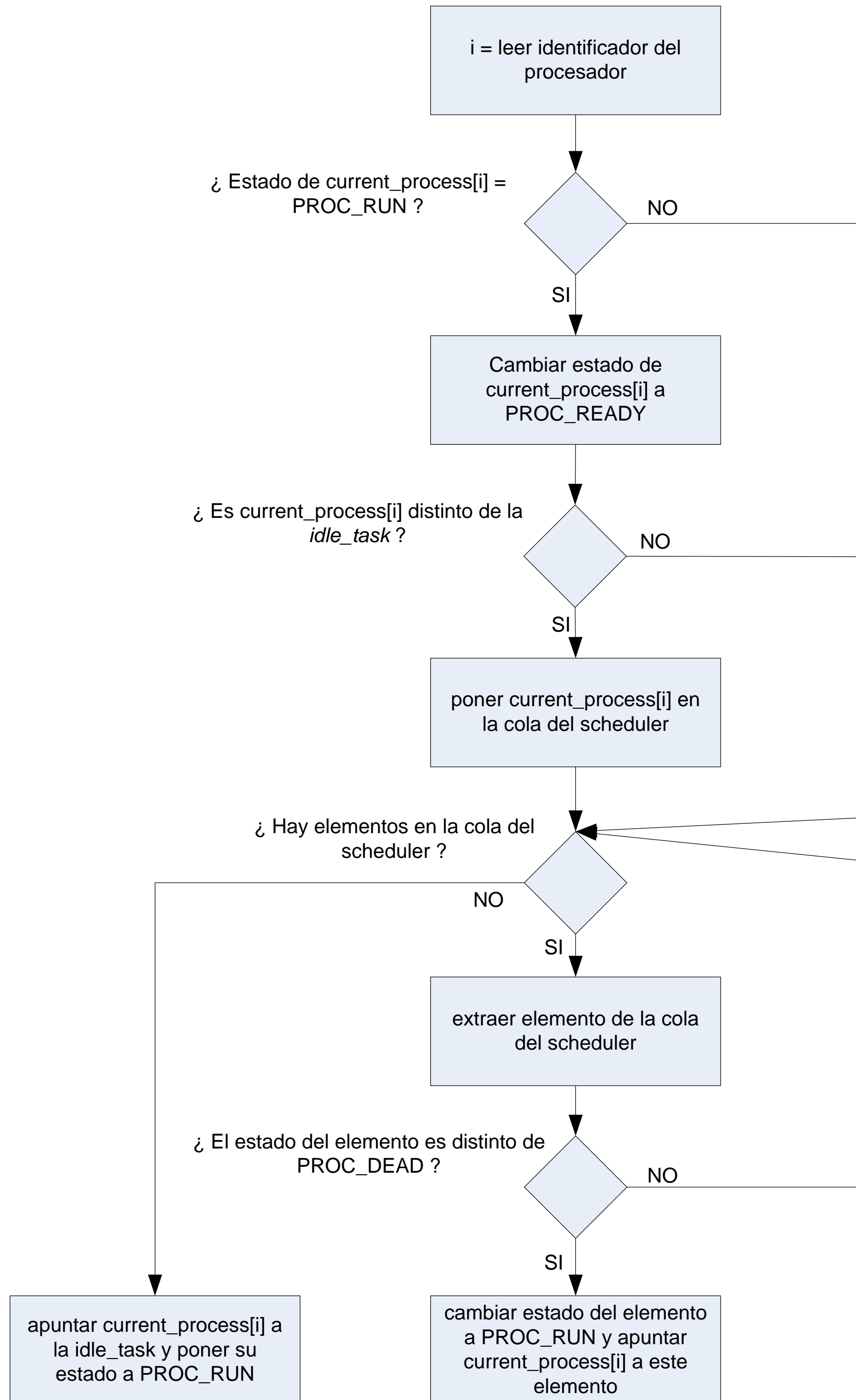


Figura 54: *diagrama de flujo del planificador.*

El sistema operativo permite escoger dos políticas de planificación: *round-robin*, y por prioridades. La rutina que se ha presentado es la que sigue la política *round-robin*. La rutina del planificador para una política de reemplazo basada en prioridades es muy similar a la presentada, sólo que en lugar de tener una única cola, hay una cola para cada nivel de

prioridad, y no se planifican procesos de una cola de prioridad inferior mientras haya procesos en estado PROC_READY en alguna cola de prioridad superior.

El hecho de que la interrupción del temporizador llegue a todos los procesadores simultáneamente hace que surja un problema: ¿Qué sucede si dos procesadores intentan extraer un proceso de la cola de procesos listos simultáneamente? Cabría la posibilidad de que los dos procesadores extrajesen el mismo elemento de la cola y los dos ejecutasen el mismo proceso, lo cual no debería pasar bajo ninguna circunstancia. También podría pasar que dos procesadores intenten introducir en la cola del planificador dos procesos simultáneamente, ocupando los dos la misma posición y perdiéndose por tanto uno de ellos. Se hace pues necesario un mecanismo de sincronización que permita solucionar este problema de sección crítica, tal como se describió en la sección 8.1. Dicho mecanismo será utilizado en este caso para que sólo un procesador esté dentro de la rutina de planificación a un mismo tiempo. Para ello se modifica la rutina de planificación presentada introduciendo las llamadas necesarias para utilizar dicho mecanismo de sincronización garantizando el acceso exclusivo a las secciones críticas de la rutina, como se muestra en la Figura 55.

#### 8.2.2.4 Llamadas al sistema

El sistema operativo proporciona una serie de llamadas al sistema para acceder a los distintos servicios del sistema operativo: crear procesos, manejar semáforos, manejar colas de mensajes, etc.

La mayor parte de las llamadas al sistema acceden a datos compartidos, como puede ser la tabla de procesos, la cola de un semáforo, etc. Además el resultado de una llamada al sistema pueda hacer necesaria una activación del *kernel*, como por ejemplo sucede con un *process_exit* (finalización de un proceso), o un *sem_post* (liberación de un semáforo). Es por tanto necesario garantizar el acceso exclusivo a los datos compartidos mediante algún mecanismo de sincronización. En los sistemas monoprocesador esto se consigue deshabilitando las interrupciones cuando se realiza una llamada al sistema, garantizando así que ningún otro proceso accede a los datos compartidos mientras se lleva a cabo la llamada. En un sistema multiprocesador esto no es suficiente, ya que al estar varios procesadores ejecutando procesos de forma simultánea no basta con desactivar las interrupciones, ya que estas solo se desactivarían en el procesador que realiza la llamada, y por tanto se hace necesario un mecanismo de sincronización hardware como el

*hardware_mutex* descrito en el capítulo 7. En la Figura 56 se muestra el funcionamiento de las llamadas al sistema.

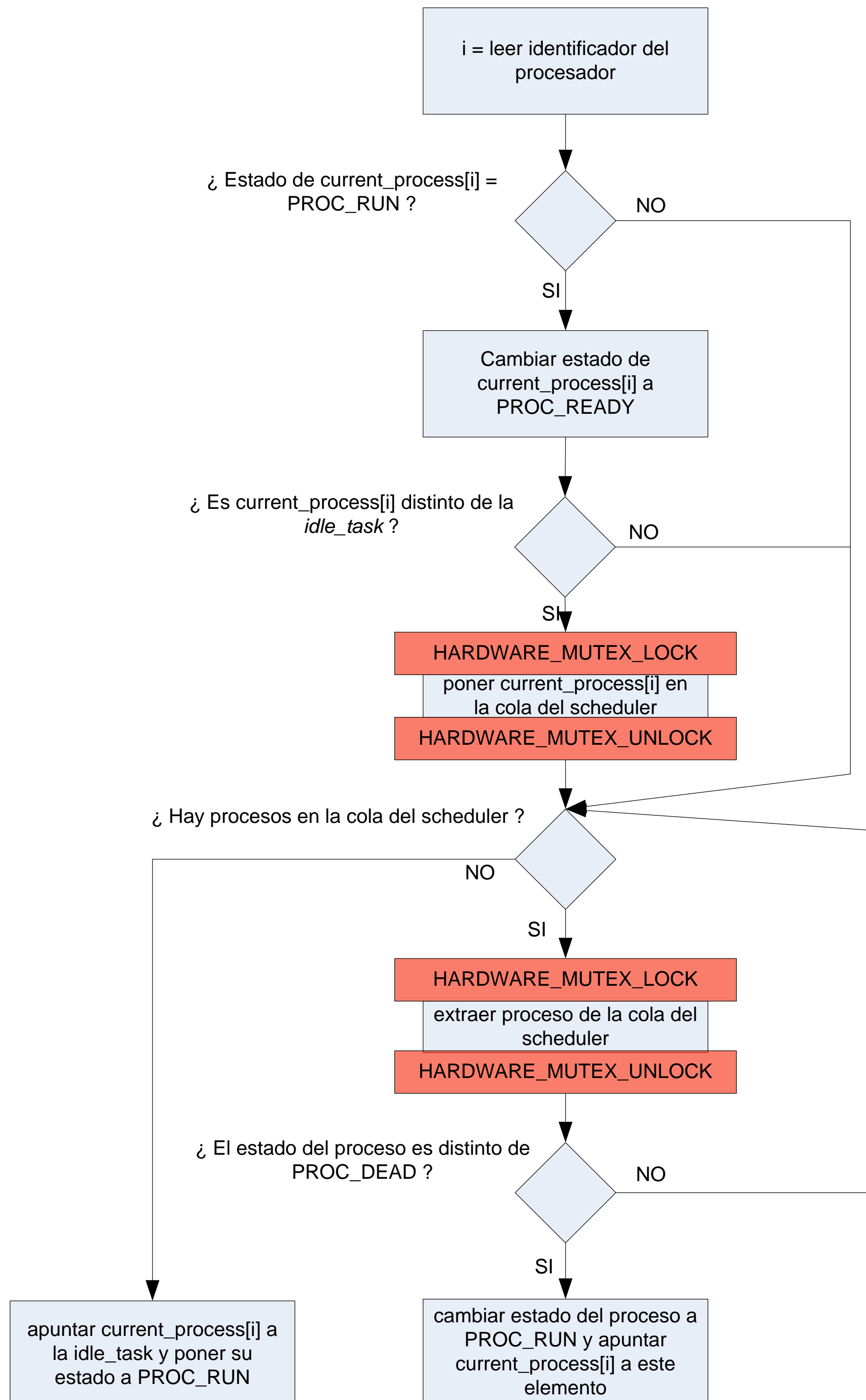


Figura 55: acceso a las secciones críticas del planificador mediante el *hardware-mutex.*

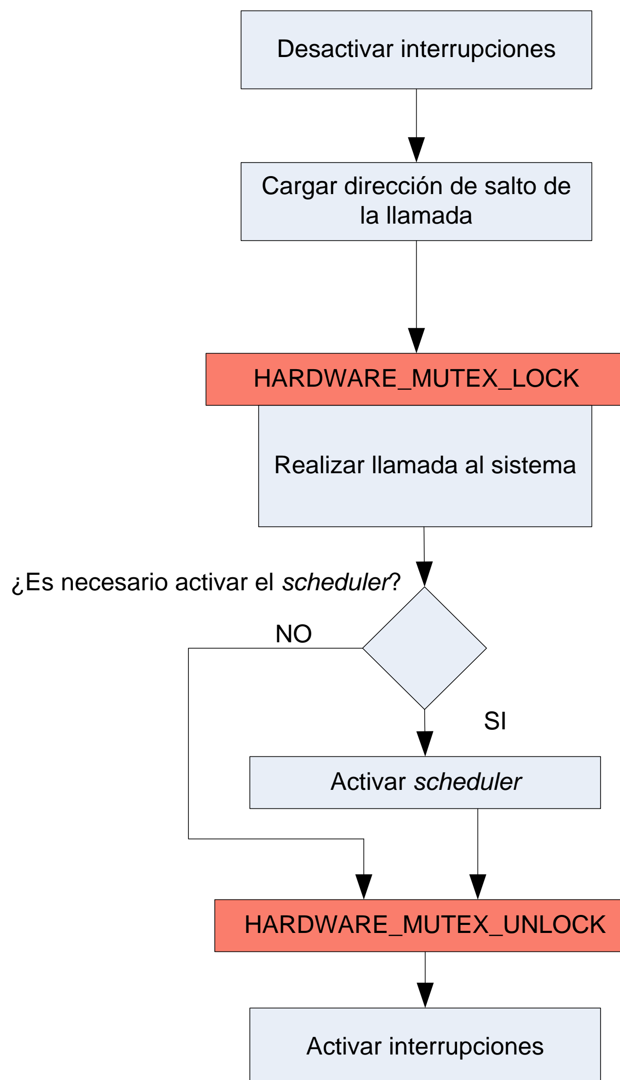


Figura 56: funcionamiento de las llamadas al sistema.

Como se puede observar en la Figura 56, se utiliza el mismo *mutex* hardware que se usaba en la rutina de *planificación*. Esto implica que mientras un procesador está realizando una llamada al sistema, otro procesador no puede ejecutar la rutina de *planificación*. Además, nunca dos procesadores podrán realizar una llamada al sistema de forma simultánea. Como ya se comentó, esto es así debido a que sería posible que se accediese a datos compartidos durante las llamadas al sistema, aunque no siempre dos llamadas al sistema simultáneas tienen porque acceder a los mismos datos: por ejemplo, una llamada a *sem_lock* (bloquear semáforo) es perfectamente compatible con una llamada simultánea de otro procesador a *process_getpid* (obtener el identificador de proceso).

Se podría implementar una mejora en el sistema de gestión de las llamadas al sistema que permitiese que ejemplos como el anterior se pudieran ejecutar simultáneamente, pero complicaría bastante el tratamiento de las llamadas al sistema además de requerir más instancias del componente *hardware_mutex*, y el aumento del rendimiento no sería significativo.

### 8.2.3 API del sistema operativo

A continuación se enumeran algunas de las principales funciones que ofrece el sistema operativo y que han sido adaptadas para funcionar en un sistema SMP:

#### 8.2.3.1 Manejo de hilos

Xilkernel soporta la interfaz básica de hilos POSIX. Los hilos están identificados por un identificador único del tipo *pthread_t*. Este identificador se utiliza para identificar al hilo en determinadas operaciones. Él módulo de hilos es opcional, y configurable. El número de hilos máximo que puede manejar el S.O, cuánto *stack* se reserva para cada uno, y otras características, son configurables. Las principales funciones para el manejo de hilos son las siguientes:

- int **pthread_create** (pthread_t *thread_id*, pthread_attr_t* *attr*, void* (*start_func)(void*), void* *param)* : crea un hilo con los atributos especificados en *attr*. El identificador del hilo se alamcena en *thread_id*. El nuevo hilo ejecutará la función *start_func* y recibirá como parámtero *param*. La llamada a *pthreaad_create* devuelve un ‘0’ si el hilo se consigue crear correctamente, o un código de error en caso contrario.
- void **pthread_exit** (void **value_ptr*) : finaliza la ejecución del hilo que la invoca y pone el valor de *value_ptr* disponible para cualquier hilo que haga una llamada a *pthrea_join* sobre el hilo que finaliza.
- int **pthread_join** (pthread_t *target_thread*, void **value_ptr) : suspende la ejecución del hilo que ejecuta la función hasta que el *target_thread* finaliza. En *value_ptr* se devuelve el valor que especificó el *target_thread* en su llamada a *pthread_exit*.
- pthread_t **pthread_self** (void) : devuelve el identificador del hilo actual.
- int **pthread_equal** (pthread_t *t1*, pthread_t *t2*) : devuelve ‘1’ si *t1* y *t2* se refieren a dos hilos que son iguales. En caso contrario devuelve ‘0’.

#### 8.2.3.2 Semáforos

Xilkernel soporta semáforos POSIX que pueden usarse para sincronización entre procesos. El módulo de semáforos es opcional y configurable, pudiendo elegir el número máximo de semáforos y la longitud de sus colas de espera. A continuación se presentan algunas de las funciones para manejo de semáforos disponibles:

- int **sem_init** (sem_t **sem*, int *pshared*, unsigned *value*) : inicializa el semáforo referido por *sem* con el valor inicial *value*. El parámetro *pshared* indica si el semáforo va a ser compartido entre procesos, pero no se usa actualmente. La llamada devuelve un '0' si el semáforo se creó correctamente, o -1 en caso contrario.
- int **sem_destroy** (sem_t **sem*) : destruye el semáforo referenciado por *sem* y libera los recursos que utiliza.
- int **sem_wait** (sem_t **sem*) : bloquea el semáforo referenciado por *sem*. Si el semáforo ya está bloquado por otro hilo, el hilo se queda bloqueado hasta que consiga bloquear el semáforo o éste sea destruido.
- int **sem_trywait** (sem_t **sem*) : bloquea el semáforo referenciado por *sem* en caso de que este actualmente desbloqueado. En caso contrario no bloquea el semáforo y devuelve '-1'.
- int **sem_timdewait** (sem_t **sem*, unsigned *ms*) : bloquea el semáforo referenciado por *sem*. Si el semáforo ya está bloquado por otro hilo, el hilo se queda bloqueado hasta que consiga bloquear el semáforo, éste sea destruido o el timeout *ms* haya vencido.
- int **sem_post** (sem_t **sem*): desbloquea el semáforo referenciado por *sem*. Devuelve '0' si se desbloquea con éxito, o '-1' en caso contrario.

#### 8.2.3.3 Mutex

Xilkernel proporciona soporte para *mutex* POSIX. Este mecanismo de sincronización entre procesos se utiliza conjuntamente con el módulo de manejo de hilos visto en la sección 8.2.3.1. El módulo es opcional, y se puede configurar el número máximo de *mutex* disponibles y la longitud de sus colas de espera. A continuación se muestran algunas de las funciones para manejo de *mutex* que soporta el sistema:

- int **pthread_mutex_init** (pthread_mutex_t **mutex_id*, const pthread_mutexattr_t **attr*) : inicializa el *mutex* referenciado por *mutex_id* con los atributos especificados por *attr*. Devuelve un '0' si se realiza la inicialización correctamente y coloca en *mutex_id* el identificador del *mutex*.
- int **pthread_mutex_destroy** (pthread_mutex_t **mutex_id*) : destruye el *mutex* referenciado por *mutex_id* y libera los recursos que utiliza.
- int **pthread_mutex_lock** (pthread_mutex_t **mutex_id*) : devuelve un '0' y deja el *mutex* referenciado por *mutex_id* en estado bloqueado. Si el *mutex* ya estaba bloqueado, el hilo se queda bloqueado hasta que el *mutex* esté disponible.
- int **pthread_mutex_trylock** (pthread_mutex_t **mutex_id*) : devuelve un '0' y deja el *mutex* referenciado por *mutex_id* en estado bloqueado. Si el *mutex* ya estaba bloqueado devuelve EBUSY y continua la ejecución.
- int **pthread_mutex_unlock** (pthread_mutex_t **mutex_id*) : desbloquea el *mutex* referenciado por *mutex_id*.

## 8.3 Resultados experimentales

Para comprobar el correcto funcionamiento del sistema operativo implementado se ha diseñado una plataforma hardware con los componentes necesarios para poder ejecutar el sistema operativo y algunas aplicaciones de prueba que permitan verificar el funcionamiento de las distintas partes del *kernel* que se han modificado para dar soporte a sistemas SMP.

### 8.3.1 Plataforma hardware

La plataforma sobre la que se han realizado las pruebas del sistema operativo (Figura 57) consta de:

- 4 procesadores MicroBlaze, cada uno con:
  - 8 KBytes de memoria de bloque conectada al bus de datos LMB.
  - Un registro conectado al bus FSL con un valor diferente para cada procesador, que se utilizará como identificador del procesador.
- Un bus OPB compartido por los 8 procesadores, al qué están conectados los siguientes periféricos:

- Una UART para entrada/salida.
- Un *hardware-mutex*.
- 128 KBytes de memoria de bloque.
- Un temporizador, que interrumpirá simultáneamente a todos los procesadores.

En la Figura 57 se muestra un esquema del sistema completo.

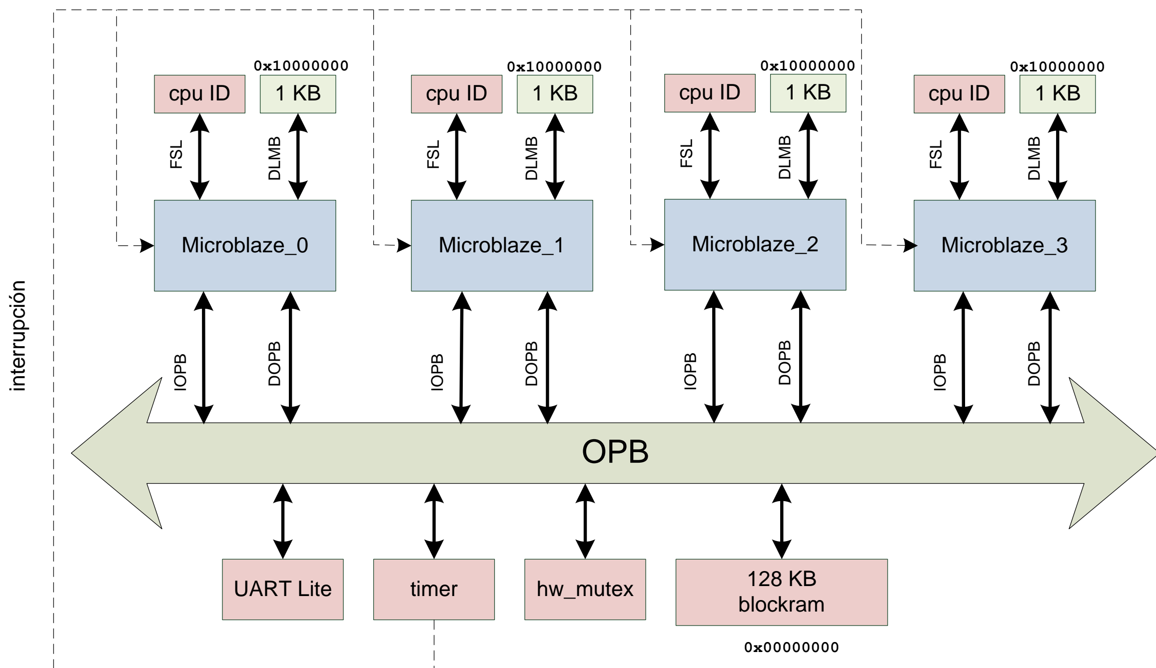


Figura 57: plataforma para las pruebas del S.O.

Este sistema se implementó sobre una placa de desarrollo RC-300 de Celoxica (Figura 58), equipada con una FPGA XC2V6000-4 de la familia Virtex 2 de Xilinx.

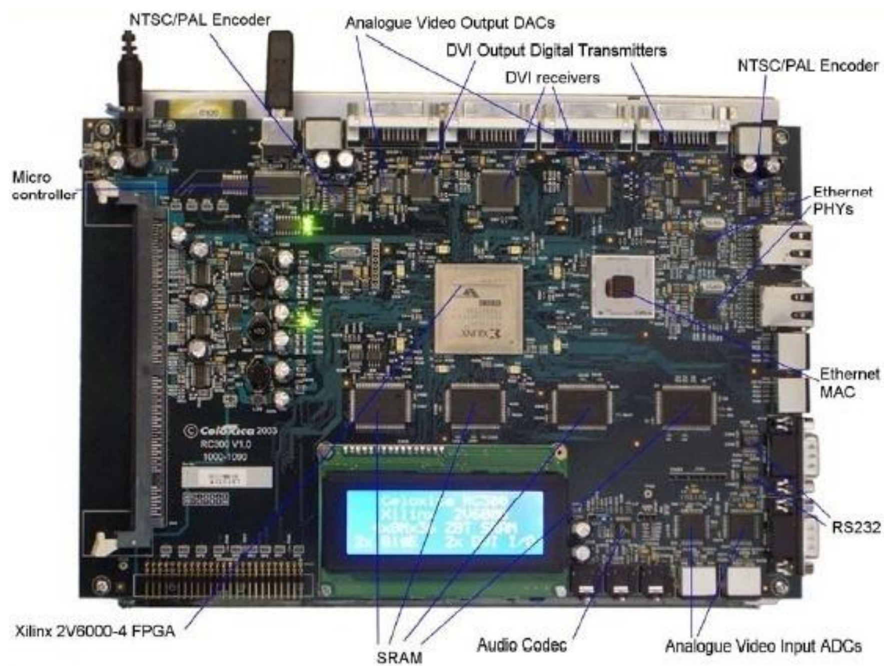


Figura 58: placa RC300 de Celoxica.

### 8.3.2 Tests software

Sobre la plataforma descrita en el punto anterior se ejecutó un test software para comprobar el correcto funcionamiento de los distintos módulos del *kernel*. Este test consiste en una aplicación que crea varios hilos que lanzan mensajes a través del puerto serie mostrando el identificador del hilo y el identificador del procesador que está ejecutando dicho hilo en ese momento. Los distintos hilos se sincronizan mediante semáforos para acceder de forma ordenada al puerto serie. En la Tabla 6 se muestra el código de la aplicación de prueba.

Este test permite mostrar como los distintos procesadores del sistema ejecutan cada uno un hilo diferente en un momento dado (Figura 59). También se muestra que los hilos no están siempre ligados al mismo procesador sino que en las distintas ejecuciones del planificador un hilo puede pasar a ser ejecutado por cualquiera de los procesadores del sistema.

```
#define NUM_THREADS 8
pthread_t p_th[NUM_THREADS];
sem_t  semaforo;

void main_thread(void)
{
        sem_init(&semaforo, NULL, 1);
        sem_wait(&semaforo);
        print("Lanzando threads\r\n");
        sem_post(&semaforo);
        for(i = 0; i < NUM_THREADS; i++)
                pthread_create(&p_th[i], NULL, (void*)thread_1, NULL);
        for(i = 0; i < NUM_THREADS; i++)
                pthread_join(p_th[i], NULL);
        sem_wait(&semaforo);
        print("Todos los threads finalizaron\r\n");
        sem_post(&semaforo);
}

void thread_1(void)
{
        int i, processor_id, thread_id;
        thread_id = pthread_self();
        sem_wait(&semaforo);
        GET_PROCESSOR_ID(processor_id);
        xil_printf("Al entrar: thread_id = %d\t
                      processor_id = %d\r\n", thread_id, processor_id);
        sem_post(&semaforo);

        for(i = 0; i < 5000000; i++);

        sem_wait(&semaforo);
        GET_PROCESSOR_ID(processor_id);
        xil_printf("Al salir:  thread_id = %d\t
                      processor_id = %d\r\n", thread_id, processor_id);
        sem_post(&semaforo);

        pthread_exit(NULL);
}
```

Tabla 6: aplicación de prueba.

```
Tera Term - COM1 VT
File  Edit  Setup  Control  Window  Help
Lanzando threads
Al entrar: thread_id = 1        processor_id = 3
Al entrar: thread_id = 3        processor_id = 1
Al entrar: thread_id = 2        processor_id = 2
Al entrar: thread_id = 4        processor_id = 0
Al entrar: thread_id = 5        processor_id = 0
Al entrar: thread_id = 7        processor_id = 1
Al entrar: thread_id = 6        processor_id = 3
Al entrar: thread_id = 8        processor_id = 0
Al salir:  thread_id = 1        processor_id = 2
Al salir:  thread_id = 3        processor_id = 3
Al salir:  thread_id = 4        processor_id = 0
Al salir:  thread_id = 2        processor_id = 3
Al salir:  thread_id = 5        processor_id = 2
Al salir:  thread_id = 7        processor_id = 3
Al salir:  thread_id = 6        processor_id = 1
Al salir:  thread_id = 8        processor_id = 0
Todos los threads finalizaron
```

Figura 59: salida de la aplicación de prueba.

## 8.4 Conclusiones

En este capítulo se ha presentado un sistema operativo que se ha desarrollado para permitir la ejecución de aplicaciones multihilo en un sistema SMP. A diferencia del sistema presentado en el capítulo 7, este S.O utiliza una única imagen del *kernel* para todos los procesadores, con el ahorro de memoria que conlleva, además de permitir que los distintos hilos se puedan planificar libremente en cualquier procesador en cada cambio de contexto, en lugar de estar anclados siempre al mismo procesador.

En el próximo capítulo, utilizando el S.O operativo que se ha desarrollado, se probarán distintas aplicaciones paralelas para evaluar el rendimiento de distintos sistemas SMP.

# 9 RENDIMIENTO DE DISTINTOS SISTEMAS SMP SOBRE FPGA

Una vez terminado el desarrollo y pruebas del sistema operativo con soporte para SMP descrito en el capítulo anterior, se implementaron distintos sistemas SMP sobre FPGA para comprobar el rendimiento que pueden ofrecer dichos sistemas, así como para estudiar más a fondo las posibilidades que ofrecen las FPGAs y los SCPs actuales para implementar este tipo de sistemas.

Se han implementado sistemas muy diversos para probar tanto distintas versiones del SCP Microblaze como distintas arquitecturas de memoria compartida.

Los tests que se desarrollaron para probar el rendimiento de los distintos sistemas son similares a los descritos en el apartado 6.4. Se utilizaron por una parte una aplicación de multiplicación de matrices, y por otra parte una aplicación criptográfica para encriptar/desencriptar cantidades grandes de información utilizando el algoritmo AES.

A continuación se presentan primero las distintas aplicaciones software que se utilizaron, para posteriormente mostrar los distintos sistemas evaluados y los resultados de rendimiento que se obtuvieron para cada uno de ellos.

## *9.1 Aplicaciones software para evaluación de rendimiento*

Uno de los objetivos de diseño de las aplicaciones que se van a presentar en este punto, aparte de que permitan evaluar el rendimiento de un sistema SMP, es que muestren que el sistema operativo diseñado y presentado en el capítulo anterior permite escribir aplicaciones paralelas de forma transparente a la arquitectura subyacente y que sean portables entre distintos sistemas SMP. Las aplicaciones que se van a presentar no necesitan ninguna modificación para correr en cualquiera de los 4 sistemas que se van a presentar en este capítulo.

### 9.1.1 Aplicación 1

La primera aplicación que se desarrolló consiste en una aplicación que multiplica 2 matrices, similar a la presentada en el punto 6.4. En este caso, dado que se dispone de un

sistema operativo que permite la creación de hilos, sincronización entre procesos y otra serie de servicios, la aplicación es más sencilla y "elegante" de escribir ya que la interfaz POSIX que ofrece el sistema operativo es de sobra conocida y permite la manipulación y sincronización de procesos ligeros de forma transparente y eficiente.

El algoritmo que sigue el programa consiste en crear una serie de hilos, uno por cada procesador en el sistema, que calculan la multiplicación de una serie de filas de la primera matriz por toda la segunda matriz, de forma que cada hilo calcula una parte de la matriz resultado.

Cada hilo recibe como parámetros las direcciones donde se encuentran las matrices a multiplicar, la dirección de la matriz resultado así como el número de filas y columnas y la fila inicial y final que tienen que procesar, como se muestra en la Tabla 7.

```
struct params_thread_matrices
{

	unsigned int address1; // Matriz 1
	unsigned int address2; // Matriz 2
	unsigned int address3; // Matriz resultado
	unsigned int num_columnas_1;
	unsigned int num_columnas_2;
	unsigned int ini;         //fila inicial
	unsigned int fin;         //fila final
};
```

Tabla 7: parámetros de cada hilo de la aplicación 1.

En la Tabla 8 y en la Tabla 9 se muestran por una parte el código del hilo que calcula una parte del resultado final, y por otra el código del programa principal que prepara los parámetros de ejecución de los hilos y crea tantos hilos como procesadores hay en el sistema.

El código utiliza la variable de entorno del sistema operativo NUM_CPUS, que está definida en la capa de abstracción de hardware del sistema operativo, consiguiendo así que el código del programa sea independiente de la plataforma sobre la que se ejecute, quedando todo lo que es dependiente de la plataforma dentro del HAL del sistema operativo.

```
void* thread_mul_row_float(void *param)
{
struct params_thread *ptr;
unsigned int i;
ptr = (struct params_thread *)param;

for(i = ptr->ini; i <= ptr->fin; i++)
{
	mul_row_float((float *) \
	(i * (4 * ptr->num_columnas_1) + \
	ptr->address1), (float *)ptr->address2, \
	(float *)(i * (4 * ptr->num_columnas_2) \
	 + ptr->address3), ptr->num_columnas_1, \
	ptr->num_columnas_2);
}
}

void mul_row_float(float *row,float *matrix, \
	float *row_result,unsigned int _num_columns_a, \
	unsigned int _num_columns_b)
{
float aux;
int j, k;

for(j = 0; j < _num_columns_b; j++)
{
	aux = 0;
	for(k = 0; k < _num_columns_a; k++)
	aux += *(row+k)* *(matrix+k*_num_columns_b+j);

	 *(row_result+j) = aux;
}
}
```

Tabla 8: código del hilo que calcula una parte del resultado en la aplicación 1.

```
pthread_t p_th_array[NUM_CPUS];
struct params_thread params_array[NUM_CPUS];

for(i = 0; i < NUM_CPUS; i++)
{
	params_array[i].address1= &m1[0][0];
	params_array[i].address2 = &m2[0][0];
	params_array[i].address3 = &m3[0][0];
	params_array[i].ini= 0 + i * M_SIZE / NUM_CPUS;
	params_array[i].fin = ((i+1)*(M_SIZE/NUM_CPUS))-1;
	params_array[i].num_columnas_1 = M_SIZE;
	params_array[i].num_columnas_2 = M_SIZE;
}

for(i = 0; i < NUM_CPUS; i++)
	pthread_create(&p_th_array[i], \
	NULL, (void *)thread_mul_row_float, \
	(void *)&params_array[i]);

for(i = 0; i < NUM_CPUS; i++)
	pthread_join(p_th_array[i], NULL);
```

Tabla 9: programa principal de la aplicación 1.

### 9.1.2 Aplicación 2

Otra de las aplicaciones utilizadas para evaluar los distintos sistemas implementados consiste en una aplicación que encripta/desencripta bloques de texto de diferentes tamaños utilizando el algoritmo criptográfico AES. Al igual que en la aplicación de multiplicación de matrices, la aplicación crea tantos hilos como procesadores tiene el sistema y cada hilo se encargará de encriptar o desencriptar una porción de los datos totales. Cada hilo recibe en una estructura de datos como la mostrada en la Tabla 10 los parámetros necesarios para procesar una parte de los datos. Estos parámetros son: la dirección del buffer donde están los datos a tratar, la dirección del buffer donde se deben almacenar los resultados, la clave a usar y el número de datos a tratar.

```
struct params_thread_aes
{
	unsigned int address1;	//Dirección datos
	unsigned int address2;	//Dirección resultado
	unsigned int key;	//Dirección clave
	unsigned int ini;	//Posición inicial
	unsigned int size;	//Número de datos
};
```

Tabla 10: parámetros de cada hilo en la aplicación 2.

La Tabla 11 y la Tabla 12 muestran respectivamente el código que ejecuta cada hilo en el caso de encriptación con el algoritmo AES, y el código del programa principal que se encarga de crear todos los hilos y pasarles los parámetros necesarios para que cada uno de ellos calcule una parte del resultado final. Al igual que en la aplicación de multiplicación de matrices, se hace uso de la variable de entorno del sistema operativo NUM_CPUS para que la aplicación cree tantos hilos como procesadores tenga el sistema para el que se compila la aplicación.

```
void* aes_encrypt_th(void *param)
{
	struct params_thread_aes *ptr;
	unsigned char *key_original;
	unsigned char *data_original;
	unsigned char *resultado_original;
	unsigned char key[16];
	unsigned char data[16];
	unsigned int i, j;

	ptr = (struct params_thread_aes *)param;

	data_original = (unsigned char *) \
	(ptr->ini + ptr->address1);

	resultado_original = (unsigned char *) \
	(ptr->ini + ptr->address2);

	for(j = 0; j < ptr->size; j++)
	{
	key_original = (unsigned char *)ptr->key;
	for(i = 0; i < 16; i++)
	{
		key[i] = *key_original;
		data[i] = *data_original;
		key_original++;
		data_original++;
	}

	encrypt_aes(&data, &key);

	for(i = 0; i < 16; i++)
	{
		*resultado_original = data[i];
		resultado_original++;
	}
	}
}
```

Tabla 11: código de los hilos de la aplicación 2.

```
pthread_t p_th_array[NUM_CPUS];
struct params_thread_aes params[NUM_CPUS];

for(i = 0; i < NUM_CPUS; i++)
{
	params[i].address1 = &buff_in;
	params[i].address2 = &buff_out;
	params[i].key = &key;
	params[i].ini = 0 + i * (BUFF_SIZE / NUM_CPUS);
	params[i].size = (BUFF_SIZE / 16) / NUM_CPUS;
}

for(i = 0; i < NUM_CPUS; i++)
	pthread_create(&p_th_array[i], \
	NULL,(void *)aes_encrypt_th, (void *)&params[i]);

for(i = 0; i < NUM_CPUS; i++)
	pthread_join(p_th_array[i], NULL);
```

Tabla 12: programa principal de la aplicación 2.

## 9.2 Sistemas SMP evaluados sobre FPGA

### 9.2.1 Consideraciones previas

Como se vio en el capítulo anterior, el sistema operativo necesita cumplir una serie de requisitos hardware para poder funcionar correctamente, que pueden ser implementados de diversas formas. En todos los sistemas que se van a presentar en este capítulo se utilizarán los siguientes elementos necesarios por el sistema operativo:

- Un bloque de memoria blockram para cada procesador, accesible a través del interfaz de datos del bus LMB de cada procesador. El tamaño de ese bloque dependerá del tamaño mínimo que soporte la FPGA sobre la que se vaya a materializar el sistema.
- Un periférico hardware_mutex como el presentado en el capítulo 6.
- Un registro para cada procesador, accesible a través del interfaz FSL, que se usará para identificar al procesador.

El tipo de sistemas que se van a llevar a una versión multiprocesador son sistemas que hacen uso de caché de instrucciones, ya que si no lo hicieran la versión multiprocesador no tendría sentido: todos los procesadores estarían accediendo simultánea y continuamente al bus compartido para traer instrucciones de la memoria, haciendo que siempre haya un procesador consiguiendo traer una instrucción de memoria y el resto de procesadores estén esperando a que el bus quede libre.

Si los procesadores hacen uso de una caché de instrucciones, siempre que los procesadores estén ejecutando instrucciones que están en la caché no se producirán esperas en el bus. Si bien por un lado la caché de instrucciones es indispensable en el tipo de sistemas que se van a llevar a una versión multiprocesador, por otro la caché de datos no es posible utilizarla aún. Los controladores de caché y controladores de memoria que proporciona Xilinx para ser usados con su procesador Microblaze aún no disponen de un mecanismo para mantener la coherencia de la memoria de datos en sistemas multiprocesador.

Para todos los sistemas que se van a presentar en este capítulo se hará primero una breve descripción de la arquitectura del sistema general, esto es: número de procesadores,

memorias, buses, periféricos, etc; y posteriormente se presentarán los resultados de rendimiento obtenidos para las distintas aplicaciones utilizando distintas configuraciones del sistema.

En concreto, los parámetros del sistema que se van a variar dentro de las distintas pruebas son, donde sea posible hacerlo:

- Tamaño de las cachés de instrucciones: con la variación de este parámetro se pretende evaluar en qué grado afecta el tamaño de las cachés al rendimiento general del sistema.

- Optimizaciones de compilación: las aplicaciones se ejecutarán compiladas con y sin optimizaciones del compilador. Con esto se pretende evaluar cómo afecta la ausencia de caché de datos al sistema. Para un mismo programa, compilado con optimizaciones y sin optimizaciones habrá muchas más instrucciones de acceso a memoria en la versión que no usa optimizaciones. Cuando varios procesadores ejecutan simultáneamente instrucciones que acceden a la memoria de datos, sólo uno de ellos podrá acceder al bus teniendo los otros que esperar a que quede libre. Así, cabe esperar que la misma aplicación con optimizaciones tendrá un mejor rendimiento al ser ejecutada en un sistema multiprocesador.

- *Time-slice* del sistema operativo: en el capítulo anterior se mostró cómo funcionaba la rutina de atención a la interrupción del sistema operativo. Esta rutina sólo podía ser ejecutada por un procesador a la vez, estando el resto de procesadores bloqueados esperando para poder ejecutarla. Este funcionamiento hará que un *time-slice* muy pequeño haga que se tenga que atender la rutina muy frecuentemente, haciendo que muy frecuentemente varios procesadores estén bloqueados a la espera de atender la rutina en lugar de realizar trabajo útil. Por otra parte, un *time-slice* grande hará que el tiempo medio que tarda una nueva tarea en ser asignada a un procesador que estaba inactivo sea más grande, haciendo que el rendimiento de tareas que consumen pocos *time-slices* se vea deteriorado. En la Figura 60 y en la Figura 61 se muestran ambos tipos de comportamientos.

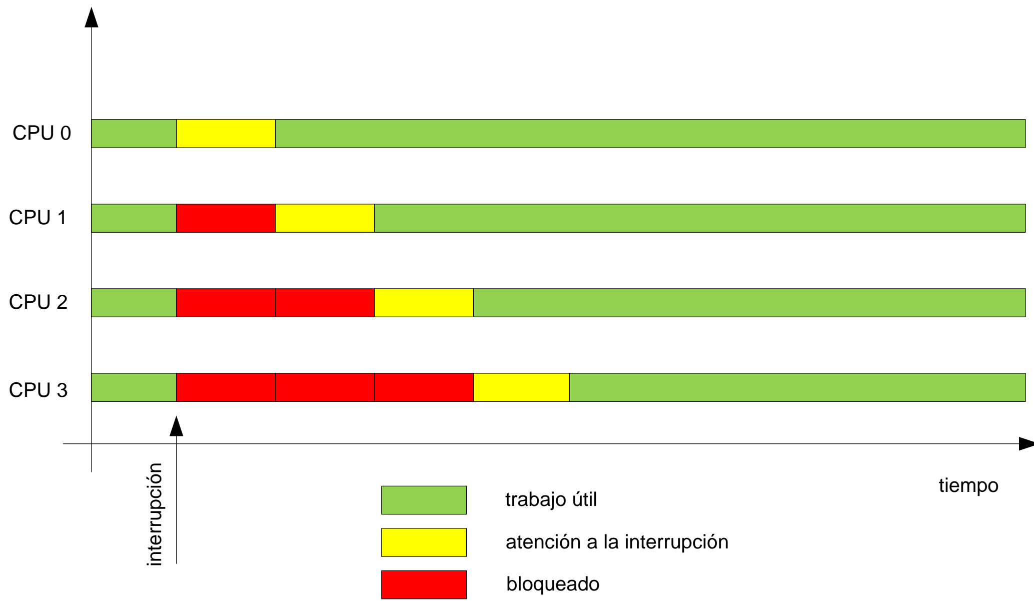


Figura 60: cuanto menor sea el *time-slice*, más frecuentemente se dará la situación en la que varios procesadores están bloqueados.

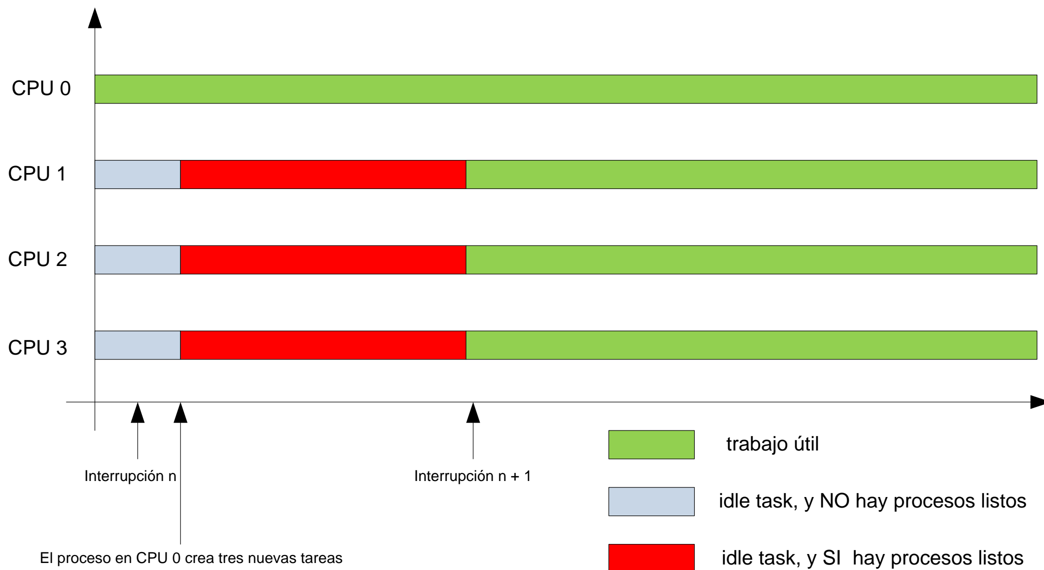


Figura 61: cuanto mayor sea el *time-slice* más tiempo medio pasa desde que se crea una tarea hasta que se empieza a ejecutar, aún no habiendo otras tareas en cola.

### 9.2.2 Métricas utilizadas para la evaluación de rendimiento

Para evaluar el rendimiento que obtienen los distintos sistemas que se van a probar, se ejecutaron las aplicaciones que se han descrito en el punto 9.1 midiendo el tiempo de ejecución sobre cada uno de los sistemas.

En cada sistema evaluado, se ejecutaron las aplicaciones primero sobre 1 procesador, y luego sobre 2, 3, etc, hasta el máximo de procesadores que soporta cada sistema.

A la hora de presentar los resultados se utilizan las siguientes métricas:

- SpeedUp: representa la aceleración del sistema a la hora de ejecutar una aplicación respecto al sistema con sólo un procesador, esto es el cociente entre el tiempo que tarda el programa en ejecutarse sobre un solo procesador y el tiempo que tarda en $p$ procesadores.

$$S_p = \frac{T_1}{T_p}$$

- Eficiencia relativa al número de procesadores: representa el grado de aprovechamiento de los procesadores del sistema a la hora de ejecutar una aplicación paralela. Se define como el cociente entre el speedup y el número de procesadores.

$$E_p = \frac{S_p}{p}$$

- Eficiencia relativa al área: representa el grado de aprovechamiento de los recursos de la FPGA al ejecutar una aplicación paralela utilizando varios procesadores. Se define como el cociente entre el speedup y la razón de las áreas del sistema con un procesador y el sistema con $p$ procesadores.

$$R_a = \frac{A_1}{A_p} \qquad E_a = \frac{S_p}{R_a}$$

### 9.2.3 Sistema 1

El primer sistema que se va a evaluar es un sistema compuesto por:

- 2 procesadores Microblaze versión 4.0a.

- 64 Kbytes de blockram para instrucciones conectados a través del bus LMB de ambos procesadores.

- 64 Kbytes de blockram para datos conectados a través del bus LMB de ambos procesadores.

- 2 bloques de 8 Kbytes de memoria privada accesibles a través del bus LMB, uno por procesador, necesarios para el sistema operativo. El motivo de utilizar 8 Kbytes por procesador es por ser el tamaño mínimo para memoria de bloque que acepta la FPGA sobre la que se va a materializar el sistema. En realidad con una zona privada de 1 Kbyte sería suficiente.

- Un bus OPB compartido por los dos procesadores, y al que están conectados:

    - Un periférico opb_uartlite para entrada/salida.

    - Un periférico opb_timer, para ser usado como fuente de interrupción de ambos procesadores.

    - Otro periférico opb_timer que se utilizará para medir el tiempo que tardan en ejecutarse los distintos programas.

    - Un hardware-mutex, necesario para poder ejecutar el sistema operativo.

- 2 registros con interfaz FSL, uno para cada procesador, que contendrán el identificador del procesador.

La principal característica de este sistema, y lo que lo diferenciará de todos los demás sistemas tratados en este capítulo, es que los dos procesadores comparten la memoria pero no comparten el bus para acceder a la memoria, ya que cada procesador accede a través de su propio bus LMB. Esto es posible debido a que la memoria de bloque que incluyen las FPGAS de Xilinx es de doble puerto, permitiendo su uso compartido por dos procesadores simultáneamente.

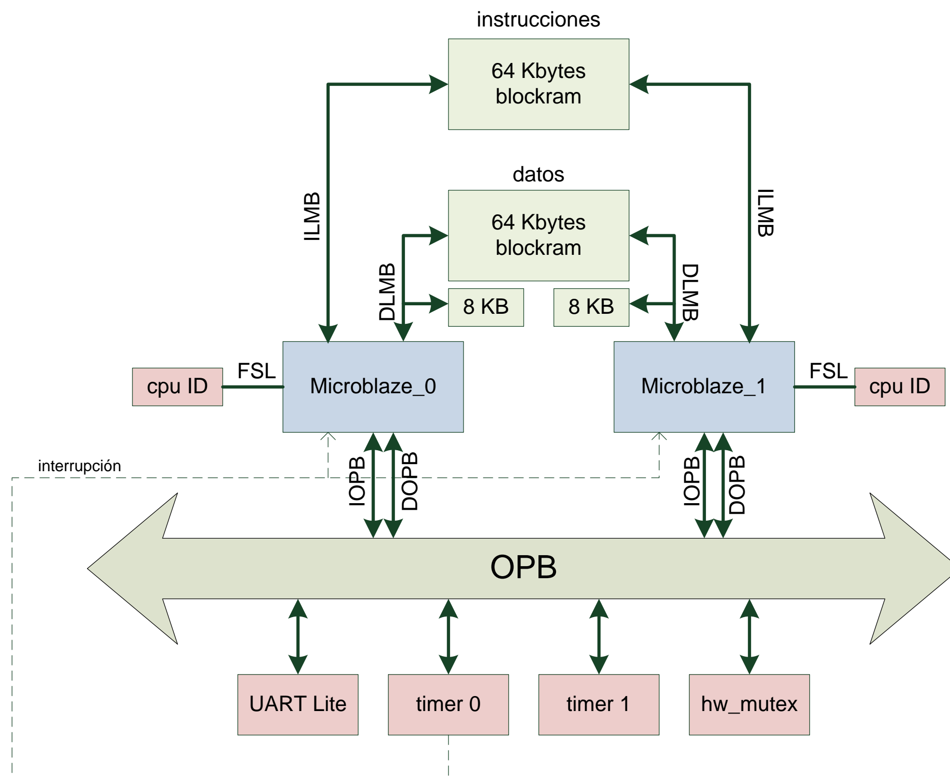


Figura 62: arquitectura del sistema 1.

En esta arquitectura no tiene sentido utilizar cachés, ya que el acceso a la memoria de bloque a través del bus LMB se hace a ciclo. Al no usarse cachés no se ejecutarán los tests software para distintas configuraciones de la caché. Tampoco se harán tests para evaluar el impacto de utilizar código compilado con o sin optimizaciones, ya que al no compartirse el bus para acceder a la memoria no tiene importancia el hecho de que haya más o menos instrucciones de acceso a memoria en el rendimiento relativo del sistema. Por tanto, los únicos tests que se ejecutarán en este sistema son el test estándar, con dos configuraciones de *time-slice* del sistema operativo diferentes.

#### 9.2.3.1 Resultados de síntesis

El sistema se sintetizó para una FPGA XC2V6000-FF1152-4 de la familia Virtex 2 de Xilinx, incluida en la placa de desarrollo RC300 de Celoxica sobre la que se ejecutaron posteriormente los distintos tests software. La Tabla 13 muestra los resultados de síntesis para el sistema con 2 procesadores, y la Tabla 14 muestra las relaciones entre *slices*, *LUTS* y puertas equivalentes para el sistema con 1 y 2 procesadores.

Como se puede observar en la Tabla 14, el número de puertas equivalentes no da una imagen muy realista del aumento de área que se produce al incluir un segundo procesador. Esto es debido a que el cálculo del número de puertas equivalentes incluye los 128 KB de memoria compartida, que representan un número mucho mayor de puertas equivalentes que un procesador Microblaze. Debido a esto se utilizará la relación entre el número de *slices* de los distintos sistemas para representar los resultados de eficiencia relativa al área de los tests software, cifra que da una imagen más acertada de la cantidad de recursos de la FPGA consumidos por el sistema.

```
Design Summary:

Logic Utilization:
  Number of Slice Flip Flops:       1,902 out of  67,584    2%
  Number of 4 input LUTs:           2,765 out of  67,584    4%
Logic Distribution:
  Number of occupied Slices:        2,344 out of  33,792    6%
Total Number 4 input LUTs:          3,561 out of  67,584    5%
Number of bonded IOBs:                 11 out of     824    1%
Number of Block RAMs:                 100 out of     144   69%
Number of MULT18X18s:                   6 out of     144    4%
Number of GCLKs:                        1 out of      16    6%
Number of DCMs:                         1 out of      12    8%

Total equivalent gate count for design:  6,701,024
```

Tabla 13: resultados de síntesis del sistema 1.

| | SLICES | | LUTS | | NAND GATES | |
|---|---|---|---|---|---|---|
| | Número | Razón | Número | Razón | Número | Razón |
| 1 CPU | 1657 | 1 | 2366 | 1 | 6639998 | 1 |
| 2 CPUs | 2344 | 1,4146 | 3561 | 1,5051 | 6701024 | 1,009 |

Tabla 14: relación de *slices*, *LUTS* y puertas equivalentes para 1 y 2 procesadores.

#### 9.2.3.2 Resultados de los tests software

Sobre el sistema se ejecutaron los tests software para dos configuraciones diferentes, la configuración base con 20 ms de *time-slice*, y la configuración 1 con 100 ms de *time-slice*. La Tabla 15 muestra los resultados obtenidos para la configuración base.

| CPUS | Área | Matrices: Ciclos | Speedup | Eficiencia P | Eficiencia A |
|---|---|---|---|---|---|
| 1 | 1 | 35241 | 1 | 1 | 1 |
| 2 | 1,4146 | 17751 | 1,9853 | 0,9927 | 1,4035 |

| CPUS | Área | Encriptación: Ciclos | Speedup | Eficiencia P | Eficiencia A |
|---|---|---|---|---|---|
| 1 | 1 | 65614 | 1 | 1 | 1 |
| 2 | 1,4146 | 33371 | 1,9662 | 0,9831 | 1,3899 |

| CPUS | Área | Desencriptación: Ciclos | Speedup | Eficiencia P | Eficiencia A |
|---|---|---|---|---|---|
| 1 | 1 | 106552 | 1 | 1 | 1 |
| 2 | 1,4146 | 53789 | 1,9809 | 0,9905 | 1,4003 |

Tabla 15: resultados para la configuración base.

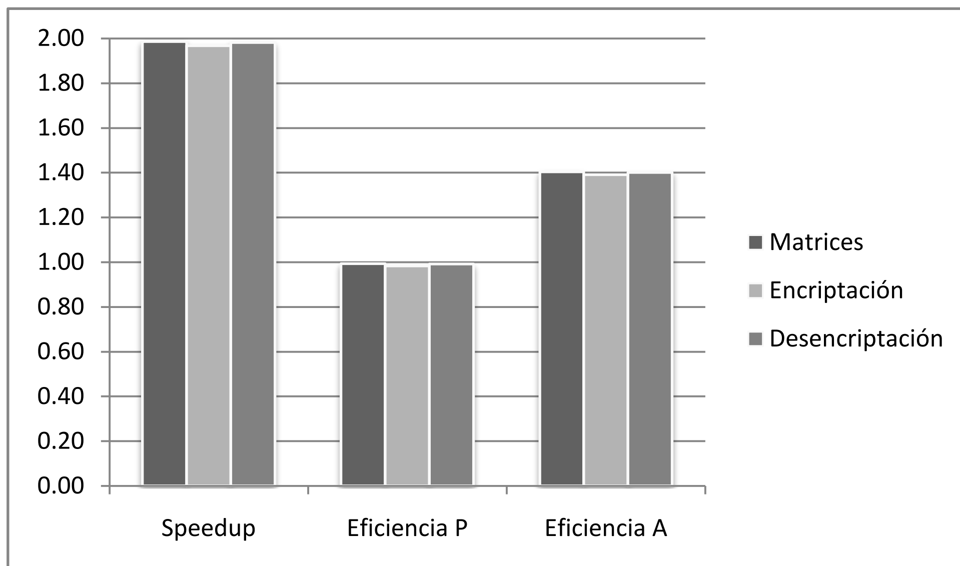


Figura 63: speedup y eficiencia para la configuración base.

Como se puede ver en los resultados el sistema obtiene un rendimiento casi ideal para los distintos tests realizados, con eficiencias muy cercanas al 100 %. Si se tiene en cuenta el rendimiento frente al área ocupada el sistema tiene un rendimiento superior al 100%, por lo tanto si se está trabajando con un sistema MicroBlaze que sólo utiliza memoria interna a la FPGA es completamente recomendable incluir en el diseño un segundo procesador si se necesita un incremento en las prestaciones a la hora de ejecutar aplicaciones multi-hilo o aplicaciones susceptibles de ser paralelizadas y se dispone de espacio libre en la FPGA.

En la Tabla 16 se muestran los resultados para la configuración 1, y la Figura 64 los compara con la configuración base. Como se puede observar, para las aplicaciones evaluadas decae el rendimiento cuando se aumenta el time-slice del sistema operativo debido al fenómeno ya explicado en la Figura 61 (sección 9.2.1).

| | | Matrices | | | |
|---|---|---|---|---|---|
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 35182 | 1 | 1 | 1 |
| 2 | 1,4146 | 18177 | 1,9355 | 0,9678 | 1,3682 |
| | | Encriptación | | | |
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 65502 | 1 | 1 | 1 |
| 2 | 1,4146 | 34904 | 1,8766 | 0,9383 | 1,3266 |
| | | Desencriptación | | | |
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 106370 | 1 | 1 | 1 |
| 2 | 1,4146 | 55532 | 1,9155 | 0,9577 | 1,3541 |

Tabla 16: resultados de la configuración 1.

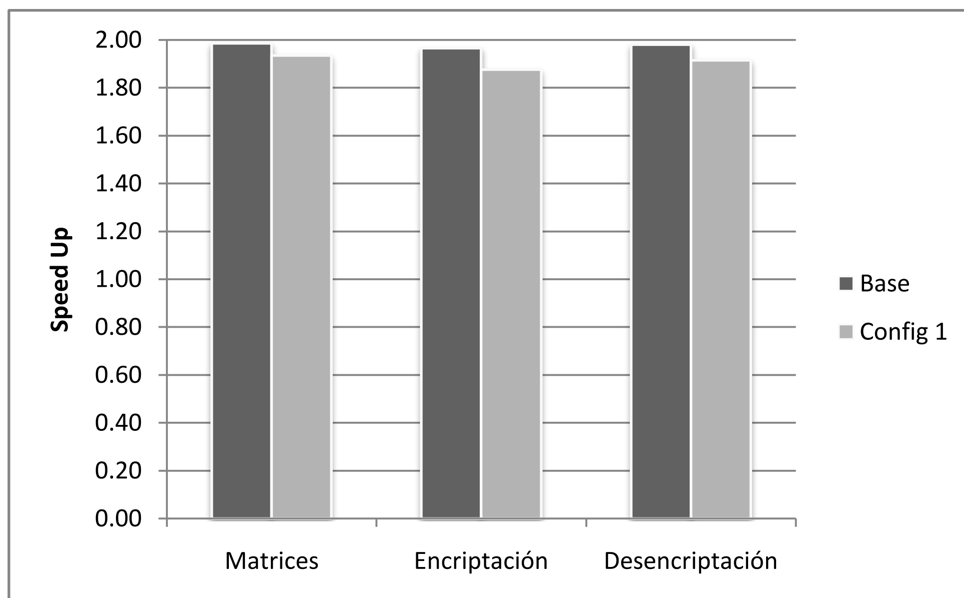


Figura 64: comparación del speedup de la configuración base y la configuración 1.

#### 9.2.3.3 Conclusiones

Se ha presentado un sistema SMP con dos procesadores que obtiene unos rendimientos excelentes debido principalmente al uso de memorias de doble puerto, que permiten el acceso simultáneo y sin bloqueos a la memoria compartida.

El principal punto débil de la arquitectura presentada es la poca cantidad de memoria que permite utilizar, ya que dependiendo de la FPGA sobre la que se vaya a implementar permitirá entre 64 y 256 KB de memoria total, limitando la variedad de aplicaciones que se pueden ejecutar sobre él.

### 9.2.4 Sistema 2

El segundo sistema que se ha evaluado es un sistema con memoria compartida a través de un bus compartido, y el objetivo de este sistema es analizar las posibilidades y limitaciones que ofrece el bus OPB y el procesador Microblaze a la hora de implementar sistemas SMP.

El sistema está formado por:

- 8 procesadores Microblaze versión 4.0a, con la interfaz de caché de instrucciones habilitada.
- 8 módulos, uno por procesador, de 8 Kbytes de memoria de bloque. Esta memoria hará las funciones de zona de memoria privada que necesita el sistema operativo para funcionar correctamente. El motivo de utilizar 8 Kbytes por procesador es el mismo que en el sistema 1, es el tamaño mínimo que soporta esa FPGA.
- Un bus OPB compartido por todos los procesadores y al que están conectados:
    - 128 Kbytes de memoria de bloque para instrucciones y datos.
    - Un periférico opb_uartlite para entrada/salida.
    - Un periférico opb_timer, para ser usado como fuente de interrupción de todos los procesadores.
    - Otro periférico opb_timer que se utilizará para medir el tiempo que tardan en ejecutarse los distintos programas.

  - Un hardware-mutex, necesario para poder ejecutar el sistema operativo.

- 8 registros con interfaz FSL, uno para cada procesador, que contendrán el identificador del procesador.

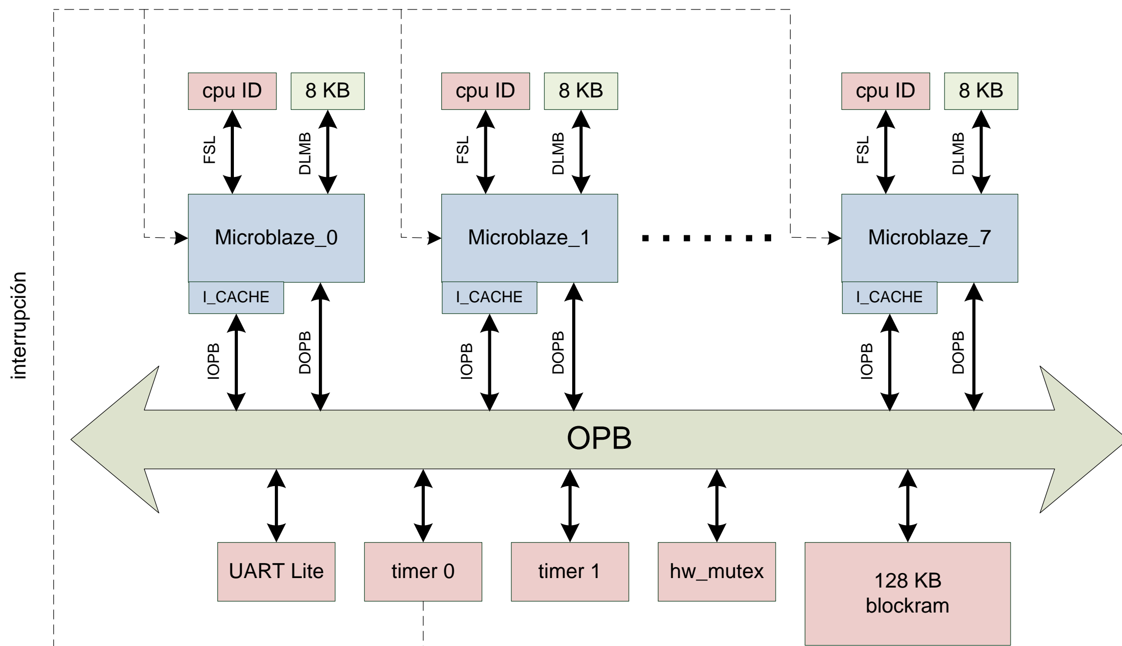


Figura 65: arquitectura del sistema 2.

#### 9.2.4.1 Resultados de síntesis

El sistema se implementó con 1, 2, 3, 4, 5, 6, 7 y 8 procesadores. La Tabla 17 muestra los resultados de síntesis para el sistema con 8 procesadores, y la Tabla 18 muestra las relaciones entre *slices*, *LUTS* y puertas equivalentes para los distintos sistemas. La FPGA para la que se realizaron las síntesis es una XC2V6000-FF1152-4 de la familia Virtex 2 de Xilinx, incluida en la placa de desarrollo RC300 de Celoxica sobre la que se ejecutaron posteriormente los distintos tests software.

Como se puede observar en la Tabla 18, el número de puertas equivalentes no da una imagen muy realista del aumento de área que se produce al aumentar el número de procesadores. Esto es debido a que el cálculo del número de puertas equivalentes incluye los 128 KB de memoria compartida, que representan un número mayor de puertas equivalentes que 8 procesadores Microblaze. Debido a esto se utilizará la relación entre el

número de *slices* de los distintos sistemas para representar los resultados de eficiencia relativa al área de los tests software, cifra que da una imagen más acertada de la cantidad de recursos de la FPGA consumidos por cada sistema.

```
Design Summary:

Logic Utilization:
  Number of Slice Flip Flops:        4,500 out of  67,584    6%
  Number of 4 input LUTs:            8,617 out of  67,584   12%
Logic Distribution:
  Number of occupied Slices:         7,134 out of  33,792   21%
Total Number 4 input LUTs:          11,806 out of  67,584   17%
Number of bonded IOBs:                 11 out of     824    1%
Number of Block RAMs:                 120 out of     144   83%
Number of MULT18X18s:                  24 out of     144   16%
Number of GCLKs:                        1 out of      16    6%
Number of DCMs:                         1 out of      12    8%

Total equivalent gate count for design:  8,384,309
```

Tabla 17: resultados de síntesis del sistema 2 para 8 procesadores.

| | SLICES | | LUTS | | NAND GATES | |
|---|---|---|---|---|---|---|
| | Número | Razón | Número | Razón | Número | Razón |
| 1 CPU | 1660 | 1 | 2329 | 1 | 4738779 | 1 |
| 2 CPUs | 2341 | 1,4102 | 3539 | 1,5195 | 5258661 | 1,1097 |
| 3 CPUs | 3093 | 1,8633 | 4834 | 2,0756 | 5779133 | 1,2195 |
| 4 CPUs | 3867 | 2,3295 | 6137 | 2,6350 | 6229528 | 1,3146 |
| 5 CPU | 4683 | 2,8211 | 7504 | 3,2220 | 6820591 | 1,4393 |
| 6 CPUs | 5473 | 3,2970 | 8935 | 3,8364 | 7341739 | 1,5493 |
| 7 CPUs | 6242 | 3,7602 | 10292 | 4,4191 | 7862595 | 1,6592 |
| 8 CPUs | 7134 | 4,2976 | 11806 | 5,0691 | 8384309 | 1,7693 |

Tabla 18: comparativa de *slices, LUTS* y puertas equivalentes para distintos números de procesadores en el sistema.

#### 9.2.4.2 Resultados de los tests software

Sobre este sistema se probaron una serie de configuraciones partiendo de una configuración base que tiene las siguientes características: 4 KB de caché de instrucciones, *time-slice* del sistema operativo de 20 ms y programa compilado con gcc –O2. La tabla y gráficas que se muestran a continuación muestran los resultados obtenidos para los distintos tests ejecutados sobre esta configuración base.

| | | Matrices | | | |
|---|---|---|---|---|---|
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 65881 | 1 | 1 | 1 |
| 2 | 1,5195 | 45141 | 1,4594 | 0,7297 | 0,9605 |
| 3 | 2,0756 | 36458 | 1,8070 | 0,6023 | 0,8706 |
| 4 | 2,6350 | 34602 | 1,9040 | 0,4760 | 0,7226 |
| 5 | 3,2220 | 34934 | 1,8859 | 0,3772 | 0,5853 |
| 6 | 3,8364 | 35956 | 1,8323 | 0,3054 | 0,4776 |
| 7 | 4,4191 | 37227 | 1,7697 | 0,2528 | 0,4005 |
| 8 | 5,0691 | 38513 | 1,7106 | 0,2138 | 0,3375 |
| | | Encriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 69854 | 1 | 1 | 1 |
| 2 | 1,5195 | 41774 | 1,6722 | 0,8361 | 1,1005 |
| 3 | 2,0756 | 35288 | 1,9795 | 0,6598 | 0,9537 |
| 4 | 2,6350 | 33676 | 2,0743 | 0,5186 | 0,7872 |
| 5 | 3,2220 | 32776 | 2,1313 | 0,4263 | 0,6615 |
| 6 | 3,8364 | 33022 | 2,1154 | 0,3526 | 0,5514 |
| 7 | 4,4191 | 33801 | 2,0666 | 0,2952 | 0,4677 |
| 8 | 5,0691 | 34102 | 2,0484 | 0,2560 | 0,4041 |
| | | Desencriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 118331 | 1 | 1 | 1 |
| 2 | 1,5195 | 67718 | 1,7474 | 0,8737 | 1,1500 |
| 3 | 2,0756 | 54716 | 2,1626 | 0,7209 | 1,0419 |
| 4 | 2,6350 | 50421 | 2,3469 | 0,5867 | 0,8906 |
| 5 | 3,2220 | 48023 | 2,4640 | 0,4928 | 0,7648 |
| 6 | 3,8364 | 47806 | 2,4752 | 0,4125 | 0,6452 |
| 7 | 4,4191 | 47917 | 2,4695 | 0,3528 | 0,5588 |
| 8 | 5,0691 | 48590 | 2,4353 | 0,3044 | 0,4804 |

Tabla 19: resultado de los tests para el sistema base.

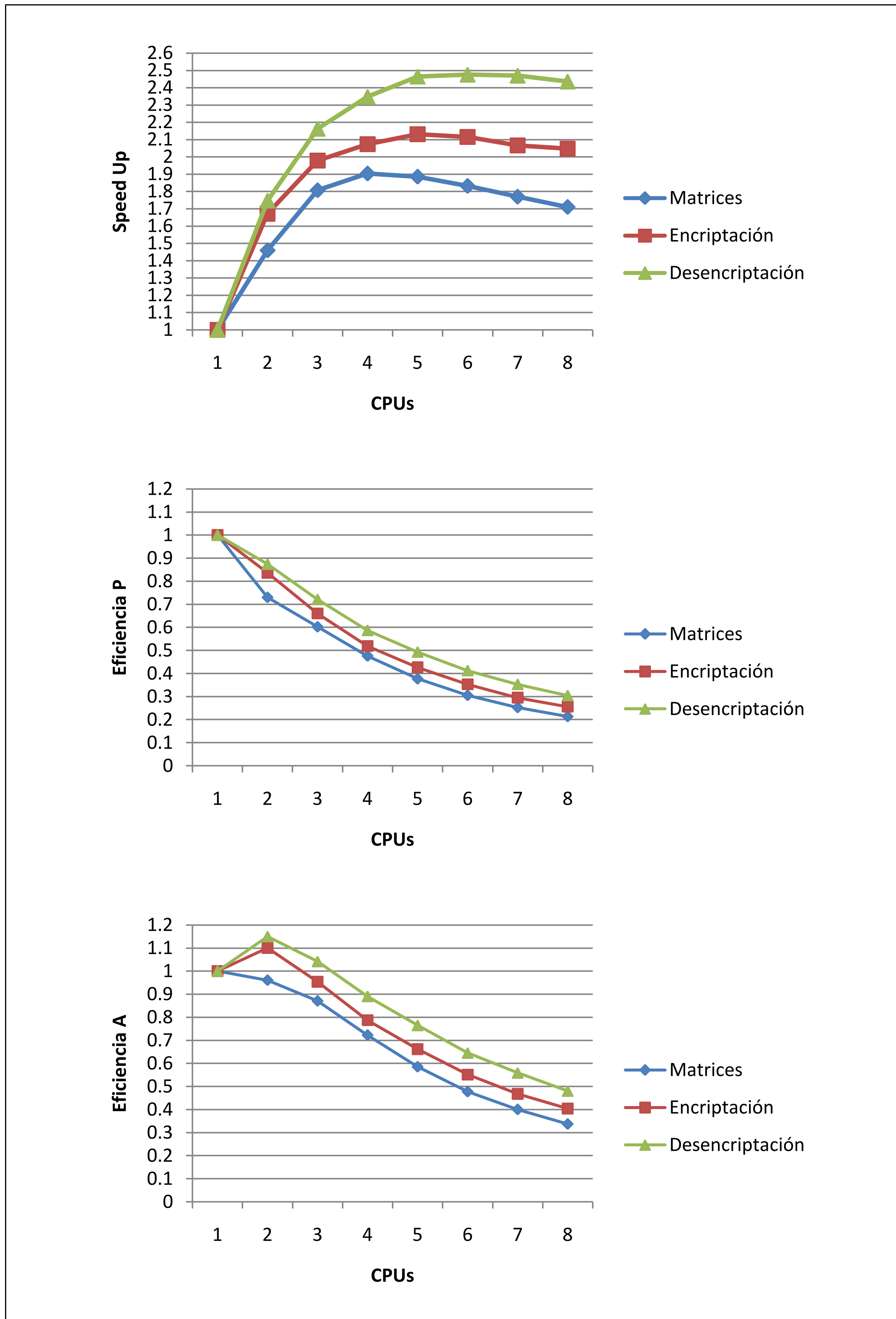


Figura 66: resultados de los tests para el sistema base.

A la vista de los resultados se puede observar como el speed-up del sistema no crece a partir de 4 o 5 procesadores, llegando incluso a bajar con 6, 7 y 8 procesadores. Estos resultados no se corresponden con lo que se esperaba: se podía esperar que el rendimiento no creciese a partir de un número de procesadores ya que los accesos al bus serían más frecuentes y habría más bloqueos, pero no se esperaba que una aplicación pudiese llegar a tardar más en ejecutarse en un sistema con 8 procesadores que en uno con 2. Después de realizar una serie de experimentos y analizar a fondo los manuales del microprocesador y del bus OPB, se llegó a la conclusión de que este fenómeno se debía a una característica del microprocesador poco conocida que no se había tenido en cuenta y que afectaba al rendimiento de sistemas multiprocesador como el que se presenta en este apartado. Dicha característica es el acceso especulativo a memoria, que consiste en que cuando el microprocesador necesita acceder a la memoria lanza la petición a través de todos los interfaces de memoria disponibles y acepta la primera respuesta que le llegue. Esto hace que aún cuando un microprocesador esté accediendo a su memoria privada, o a una instrucción que esté en la caché de instrucciones, la petición es lanzada también a través del bus OPB ocupándolo durante varios ciclos de reloj de forma innecesaria. La consecuencia que tiene esa característica del microprocesador en el sistema que se está evaluando es que cuanto mayor es el número de procesadores más tarda en realizarse un acceso al bus OPB, aunque sólo haya un procesador intentando acceder al bus.

A partir de la configuración base, se probaron otras configuraciones modificando algunos parámetros para evaluar su impacto en el rendimiento del sistema. Las distintas configuraciones que se testearon a partir del sistema base son las siguientes:

- Configuración 1: time-slice 100 ms.
- Configuración 2: optimizaciones gcc –O
- Configuración 3: caché de instrucciones 8 KB.
- Configuración 4: caché de instrucciones 1 KB.

Las siguientes tablas muestran los resultados de las distintas configuraciones para los distintos tests software.

| | | Configuración 1 | | | | Configuración 2 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Matrices | | | | Matrices | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 65417 | 1 | 1 | 1 | 66813 | 1 | 1 | 1 |
| 2 | 1,5195 | 44481 | 1,4707 | 0,7353 | 0,9679 | 46811 | 1,4273 | 0,7136 | 0,9393 |
| 3 | 2,0756 | 35650 | 1,8350 | 0,6117 | 0,8841 | 36563 | 1,8273 | 0,6091 | 0,8804 |
| 4 | 2,6350 | 32681 | 2,0017 | 0,5004 | 0,7597 | 34267 | 1,9498 | 0,4874 | 0,7399 |
| 5 | 3,2220 | 32924 | 1,9869 | 0,3974 | 0,6167 | 34823 | 1,9186 | 0,3837 | 0,5955 |
| 6 | 3,8364 | 34650 | 1,8879 | 0,3147 | 0,4921 | 36545 | 1,8283 | 0,3047 | 0,4766 |
| 7 | 4,4191 | 35648 | 1,8351 | 0,2622 | 0,4153 | 37323 | 1,7901 | 0,2557 | 0,4051 |
| 8 | 5,0691 | 38438 | 1,7019 | 0,2127 | 0,3357 | 39599 | 1,6872 | 0,2109 | 0,3328 |
| | | Encriptación | | | | Encriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 69353 | 1 | 1 | 1 | 122491 | 1 | 1 | 1 |
| 2 | 1,5195 | 44789 | 1,5484 | 0,7742 | 1,0190 | 84139 | 1,4558 | 0,7279 | 0,9581 |
| 3 | 2,0756 | 37691 | 1,8400 | 0,6133 | 0,8865 | 79034 | 1,5499 | 0,5166 | 0,7467 |
| 4 | 2,6350 | 35448 | 1,9565 | 0,4891 | 0,7425 | 79874 | 1,5336 | 0,3834 | 0,5820 |
| 5 | 3,2220 | 34022 | 2,0385 | 0,4077 | 0,6327 | 80451 | 1,5226 | 0,3045 | 0,4725 |
| 6 | 3,8364 | 33436 | 2,0742 | 0,3457 | 0,5407 | 82289 | 1,4885 | 0,2481 | 0,3880 |
| 7 | 4,4191 | 33387 | 2,0772 | 0,2967 | 0,4701 | 83948 | 1,4591 | 0,2084 | 0,3302 |
| 8 | 5,0691 | 33296 | 2,0829 | 0,2604 | 0,4109 | 87278 | 1,4035 | 0,1754 | 0,2769 |
| | | Desencriptación | | | | Desencriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 117502 | 1 | 1 | 1 | 184524 | 1 | 1 | 1 |
| 2 | 1,5195 | 70277 | 1,6720 | 0,8360 | 1,1004 | 117533 | 1,5700 | 0,7850 | 1,0332 |
| 3 | 2,0756 | 56515 | 2,0791 | 0,6930 | 1,0017 | 107528 | 1,7161 | 0,5720 | 0,8268 |
| 4 | 2,6350 | 51390 | 2,2865 | 0,5716 | 0,8677 | 106314 | 1,7357 | 0,4339 | 0,6587 |
| 5 | 3,2220 | 48119 | 2,4419 | 0,4884 | 0,7579 | 106296 | 1,7359 | 0,3472 | 0,5388 |
| 6 | 3,8364 | 47160 | 2,4916 | 0,4153 | 0,6495 | 108542 | 1,7000 | 0,2833 | 0,4431 |
| 7 | 4,4191 | 45983 | 2,5553 | 0,3650 | 0,5782 | 110449 | 1,6707 | 0,2387 | 0,3781 |
| 8 | 5,0691 | 45470 | 2,5842 | 0,3230 | 0,5098 | 113100 | 1,6315 | 0,2039 | 0,3219 |

Tabla 20: resultados de los tests en las configuraciones 1 y 2.

| | | Configuración 3 | | | | Configuración 4 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Matrices | | | | Matrices | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 49405 | 1 | 1 | 1 | 91814 | 1 | 1 | 1 |
| 2 | 1,5195 | 28738 | 1,7192 | 0,8596 | 1,1314 | 77842 | 1,1795 | 0,5897 | 0,7762 |
| 3 | 2,0756 | 20848 | 2,3698 | 0,7899 | 1,1418 | 75887 | 1,2099 | 0,4033 | 0,5829 |
| 4 | 2,6350 | 19541 | 2,5283 | 0,6321 | 0,9595 | 74867 | 1,2264 | 0,3066 | 0,4654 |
| 5 | 3,2220 | 19774 | 2,4984 | 0,4997 | 0,7754 | 75854 | 1,2104 | 0,2421 | 0,3757 |
| 6 | 3,8364 | 20250 | 2,4398 | 0,4066 | 0,6360 | 76444 | 1,2011 | 0,2002 | 0,3131 |
| 7 | 4,4191 | 20549 | 2,4042 | 0,3435 | 0,5440 | 77149 | 1,1901 | 0,1700 | 0,2693 |
| 8 | 5,0691 | 20957 | 2,3574 | 0,2947 | 0,4651 | 78036 | 1,1765 | 0,1471 | 0,2321 |
| | | Encriptación | | | | Encriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 69471 | 1 | 1 | 1 | 70934 | 1 | 1 | 1 |
| 2 | 1,5195 | 41677 | 1,6669 | 0,8334 | 1,0970 | 43035 | 1,6483 | 0,8241 | 1,0848 |
| 3 | 2,0756 | 35198 | 1,9737 | 0,6579 | 0,9509 | 36632 | 1,9364 | 0,6455 | 0,9329 |
| 4 | 2,6350 | 33781 | 2,0565 | 0,5141 | 0,7805 | 35231 | 2,0134 | 0,5033 | 0,7641 |
| 5 | 3,2220 | 32618 | 2,1298 | 0,4260 | 0,6610 | 34613 | 2,0493 | 0,4099 | 0,6360 |
| 6 | 3,8364 | 32926 | 2,1099 | 0,3517 | 0,5500 | 34819 | 2,0372 | 0,3395 | 0,5310 |
| 7 | 4,4191 | 33671 | 2,0632 | 0,2947 | 0,4669 | 35079 | 2,0221 | 0,2889 | 0,4576 |
| 8 | 5,0691 | 34013 | 2,0425 | 0,2553 | 0,4029 | 35513 | 1,9974 | 0,2497 | 0,3940 |
| | | Desencriptación | | | | Desencriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 118189 | 1 | 1 | 1 | 119983 | 1 | 1 | 1 |
| 2 | 1,5195 | 67607 | 1,7482 | 0,8741 | 1,1505 | 69722 | 1,7209 | 0,8604 | 1,1325 |
| 3 | 2,0756 | 54517 | 2,1679 | 0,7226 | 1,0445 | 57073 | 2,1023 | 0,7008 | 1,0129 |
| 4 | 2,6350 | 50061 | 2,3609 | 0,5902 | 0,8960 | 52851 | 2,2702 | 0,5676 | 0,8616 |
| 5 | 3,2220 | 48167 | 2,4537 | 0,4907 | 0,7616 | 51916 | 2,3111 | 0,4622 | 0,7173 |
| 6 | 3,8364 | 47918 | 2,4665 | 0,4111 | 0,6429 | 52182 | 2,2993 | 0,3832 | 0,5993 |
| 7 | 4,4191 | 48083 | 2,4580 | 0,3511 | 0,5562 | 52937 | 2,2665 | 0,3238 | 0,5129 |
| 8 | 5,0691 | 48638 | 2,4300 | 0,3037 | 0,4794 | 53619 | 2,2377 | 0,2797 | 0,4414 |

Tabla 21: resultados de los tests en las configuraciones 3 y 4.

Las siguientes gráficas comparan el *speed-up* de los distintos sistemas en sus versiones de 2, 4 y 8 procesadores respectivamente. En ellas se observa como el factor que más afecta a la aplicación de multiplicación de matrices es el tamaño de la caché de instrucciones, mejorando considerablemente al utilizar una caché de 8 KB (configuración 3), y empeorando considerablemente con una caché de 1 KB (configuración 4). Sin embargo las aplicaciones de encriptación y desencriptación apenas se ven afectadas por el tamaño de la caché, ya que al ser aplicaciones con un tamaño (de código) más pequeño caben prácticamente completas en una caché de 1 KB. También se puede observar como baja el rendimiento cuando no se usan las optimizaciones del compilador (configuración 2), ya que al ejecutarse más instrucciones de acceso a memoria se producen más esperas en el bus.

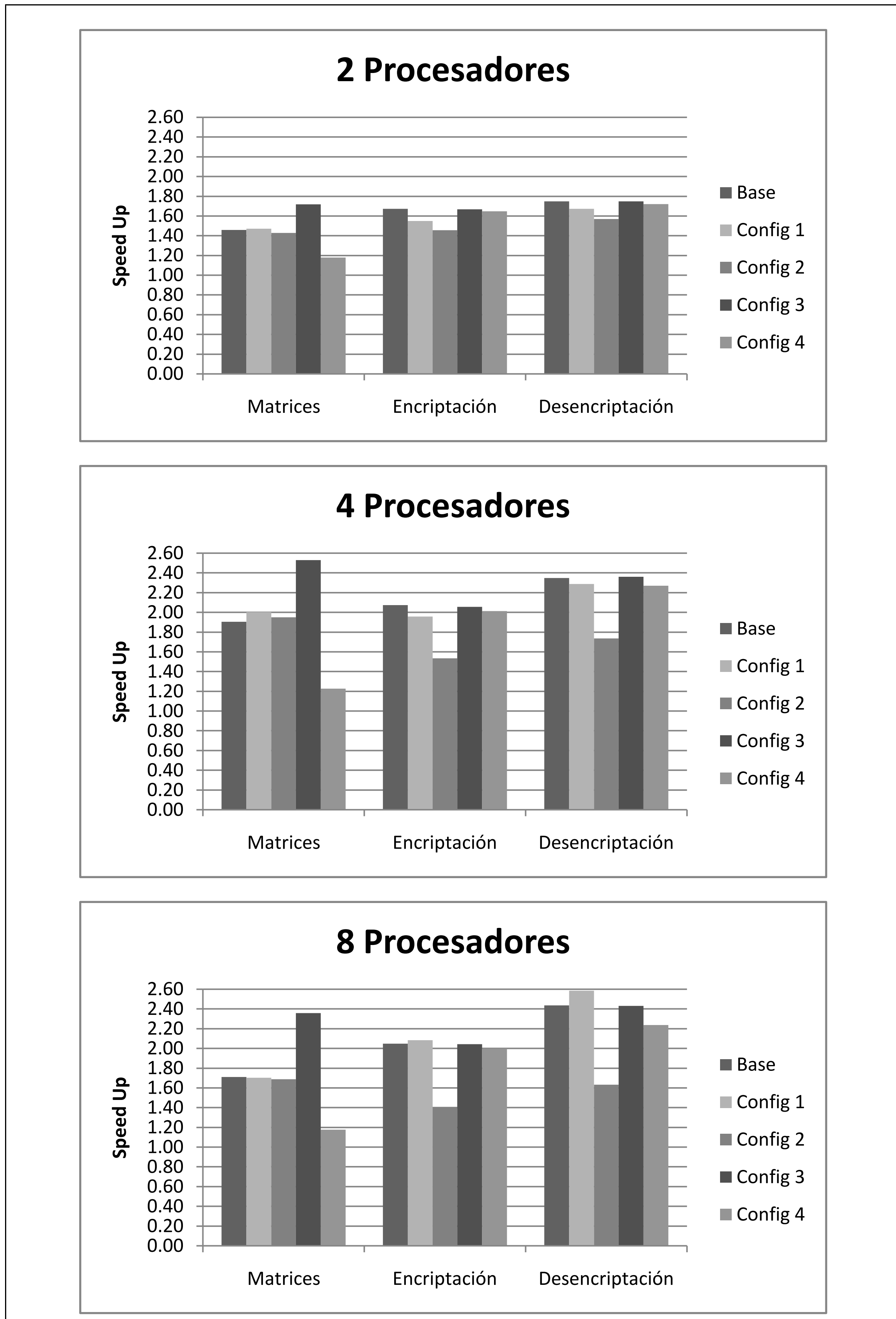


Figura 67: comparativa del speed-up para las distintas configuraciones.

#### 9.2.4.3 Conclusiones

El sistema que se ha evaluado obtiene un rendimiento relativo al área cercano al 100% para 2 procesadores, pero a medida que aumenta el número de procesadores el rendimiento cae más de lo que cabría esperar, debido a una característica de la versión 4.00a del procesador MicroBlaze que es el acceso especulativo a memoria. Por lo tanto, esta versión del procesador no es nada recomendable para implementar sistemas SMP que hagan uso de una memoria compartida a través del bus OPB.

### 9.2.5 Sistema 3

El siguiente sistema que se evaluó hace uso de una versión del procesador Microblaze más moderna, y que incluye unas interfaces para las cachés distintas a las que se utilizaban en el sistema anterior. El sistema también incluye memoria externa, y con él se pretende evaluar las posibilidades de implementar sistemas SMP utilizando los controladores de memoria que proporciona Xilinx para este tipo de memorias.

#### 9.2.5.1 Arquitectura

La nueva versión del procesador MicroBlaze utilizada en este sistema, 6.0 a, incluye una serie de mejoras entre las que se incluyen unas nuevas cachés que tienen un funcionamiento diferente al de las utilizadas en el sistema 2. A diferencia de las cachés del MicroBlaze v4.0a que podían cachear cualquier rango de direcciones de cualquier controlador de memoria conectado al bus OPB, estas nuevas cachés utilizan un interfaz diferente, llamado Xilinx Cache-Link (XCL), y sólo son compatibles con controladores de memoria que tengan un interfaz de ese tipo. En el sistema que se va a evaluar se utiliza un banco de memoria SRAM externo a la FPGA, y para acceder a ella es necesario utilizar el controlador de memoria que proporciona Xilinx: el Multi-Channel OPB External Memory Controler. Este controlador dispone de dos interfaces diferentes: un interfaz OPB estándar, y un número configurable de interfaces XCL para conectarse a las nuevas cachés del procesador MicroBlaze.

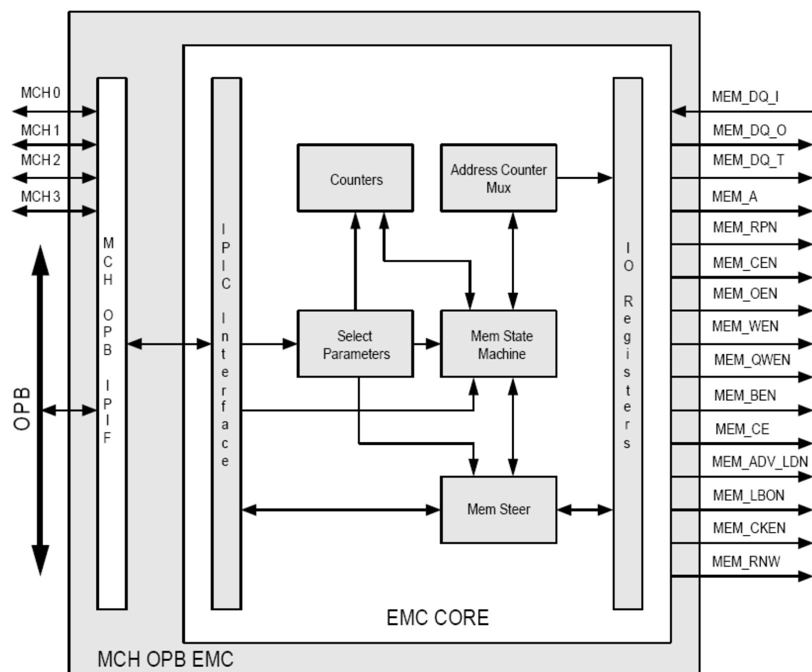


Figura 68: controlador de memoria MultiChannel External Memory Controller, de Xilinx(85).

Utilizando este tipo de memoria y controlador, se ha implementado el sistema mostrado en la Figura 69, que incluye:

- 4 procesadores Microblaze versión 6.0a, con la interfaz de instrucciones Xilinx Cache Link habilitada y 4 KB de caché de instrucciones cada uno.

- 4 módulos, uno por procesador, de 2 Kbytes de memoria de bloque. Esta memoria hará las funciones de zona de memoria privada que necesita el sistema operativo para funcionar correctamente.

- Un bus OPB compartido por todos los procesadores y al que están conectados:
    - 1 Megabyte de memoria SRAM.
    - Un periférico opb_uartlite para entrada/salida.
    - Un periférico opb_timer, para ser usado como fuente de interrupción de todos los procesadores.
    - Un segundo periférico opb_timer que se utilizará para medir el tiempo que tardan en ejecutarse los distintos programas.

- Un hardware-mutex, necesario para poder ejecutar el sistema operativo.

- 2 registros con interfaz FSL, uno para cada procesador, que contendrán el identificador del procesador.

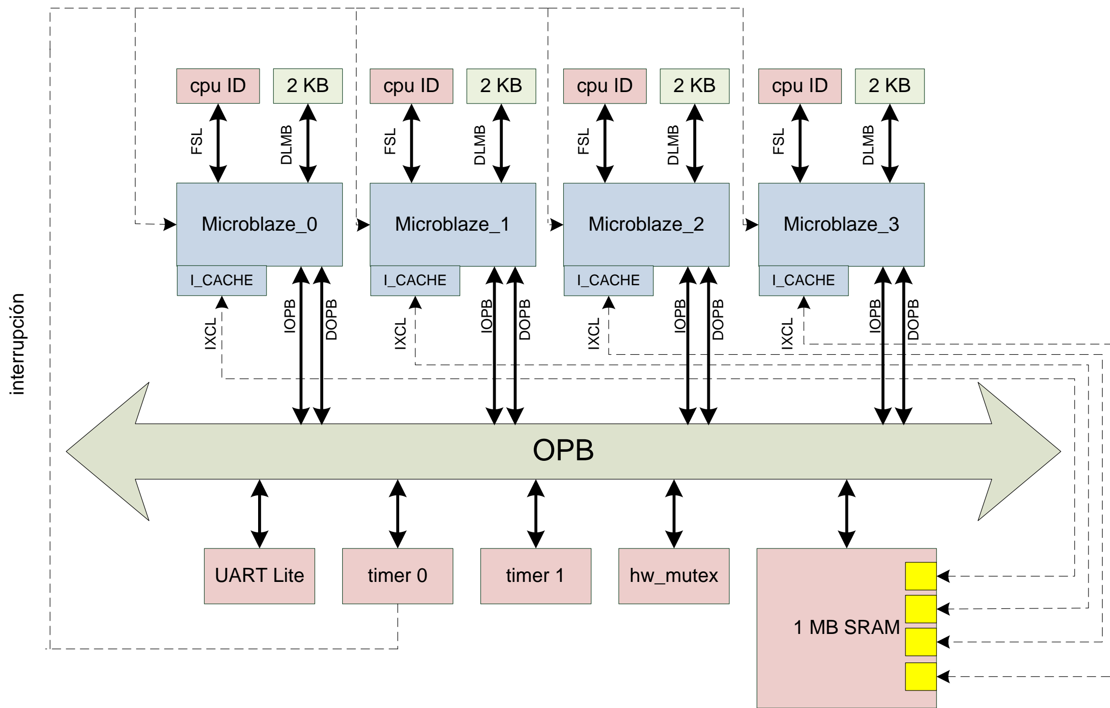


Figura 69: arquitectura del sistema 3.

#### 9.2.5.2 Resultados de síntesis

El sistema se implementó para 1, 2, 3 y 4 procesadores para poder obtener las diferencias de ocupación al usar más o menos procesadores. La FPGA para la que se realizaron las síntesis es una Virtex4 XC4VLX25-FF668-10C, incluida en la placa de desarrollo Xilinx ML401 (Figura 70), sobre la que se probaron posteriormente los distintos sistemas. La Tabla 22, muestra los resultados totales de síntesis para el sistema completo con los 4 procesadores, y la Tabla 23 muestra las relaciones entre el número de *slices, LUTS* y puertas equivalentes para 1, 2, 3 y 4 procesadores.

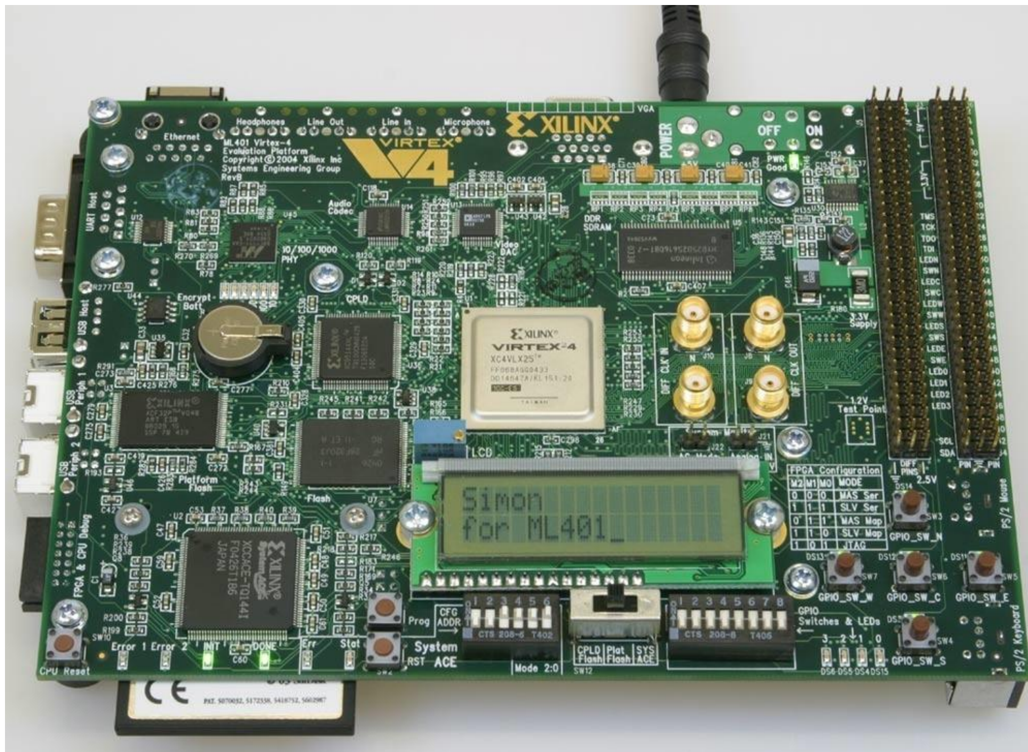


Figura 70: placa de desarrollo ML401.

```
Design Summary:
Logic Utilization:
  Number of Slice Flip Flops:        6,549 out of  21,504   30%
  Number of 4 input LUTs:            8,871 out of  21,504   41%
Logic Distribution:
  Number of occupied Slices:         7,569 out of  10,752   70%
Total Number of 4 input LUTs:       11,555 out of  21,504   53%
Number of bonded IOBs:                  69 out of     448   15%
Number of BUFG/BUFGCTRLs:                2 out of      32    6%
Number of FIFO16/RAMB16s:               32 out of      72   44%
Number of DSP48s:                       12 out of      48   25%
Number of DCM_ADVs:                      1 out of       8   12%

Total equivalent gate count for design:  2,342,351
```

Tabla 22: resultados de síntesis para el sistema 3 con 4 procesadores.

| | SLICES | | LUTS | | NAND GATES | |
|---|---|---|---|---|---|---|
| | Número | Razón | Número | Razón | Número | Razón |
| 1 CPU | 3132 | 1 | 4415 | 1 | 813618 | 1 |
| 2 CPUs | 4581 | 1,4626 | 6740 | 1,5266 | 1322872 | 1,6259 |
| 3 CPUs | 6041 | 1,9288 | 9112 | 2,0639 | 1832458 | 2,2522 |
| 4 CPUs | 7569 | 2,4167 | 11555 | 2,6172 | 2342351 | 2,8789 |

Tabla 23: comparativa de ocupación de *slices, LUTS* y puertas equivalentes para distintos números de procesadores.

#### 9.2.5.3 Resultados de los tests software

Sobre este sistema se probaron una serie de configuraciones partiendo de una configuración base con: 4 KB de caché de instrucciones, *time-slice* del sistema operativo de 20 ms y programa compilado con gcc –O2. Al igual que en los sistemas ya presentados, se utilizará la relación entre el número de *slices* a la hora de representar el área relativa que ocupa cada sistema. En la tabla siguiente se muestran los resultados obtenidos para los distintos tests en el sistema con la configuración base.

| | | Matrices | | | |
|---|---|---|---|---|---|
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 80182 | 1 | 1 | 1 |
| 2 | 1,4626 | 50234 | 1,5962 | 0,7981 | 1,0913 |
| 3 | 1,9288 | 44861 | 1,7874 | 0,5958 | 0,9267 |
| 4 | 2,4167 | 44214 | 1,8135 | 0,4534 | 0,7504 |
| | | Encriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 101047 | 1 | 1 | 1 |
| 2 | 1,4626 | 64888 | 1,5573 | 0,7786 | 1,0647 |
| 3 | 1,9288 | 57531 | 1,7564 | 0,5855 | 0,9106 |
| 4 | 2,4167 | 55721 | 1,8134 | 0,4534 | 0,7504 |
| | | Desencriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 162022 | 1 | 1 | 1 |
| 2 | 1,4626 | 99639 | 1,6261 | 0,8130 | 1,1118 |
| 3 | 1,9288 | 84597 | 1,9152 | 0,6384 | 0,9930 |
| 4 | 2,4167 | 80114 | 2,0224 | 0,5056 | 0,8369 |

Tabla 24: resultados del sistema 3 con la configuración base.

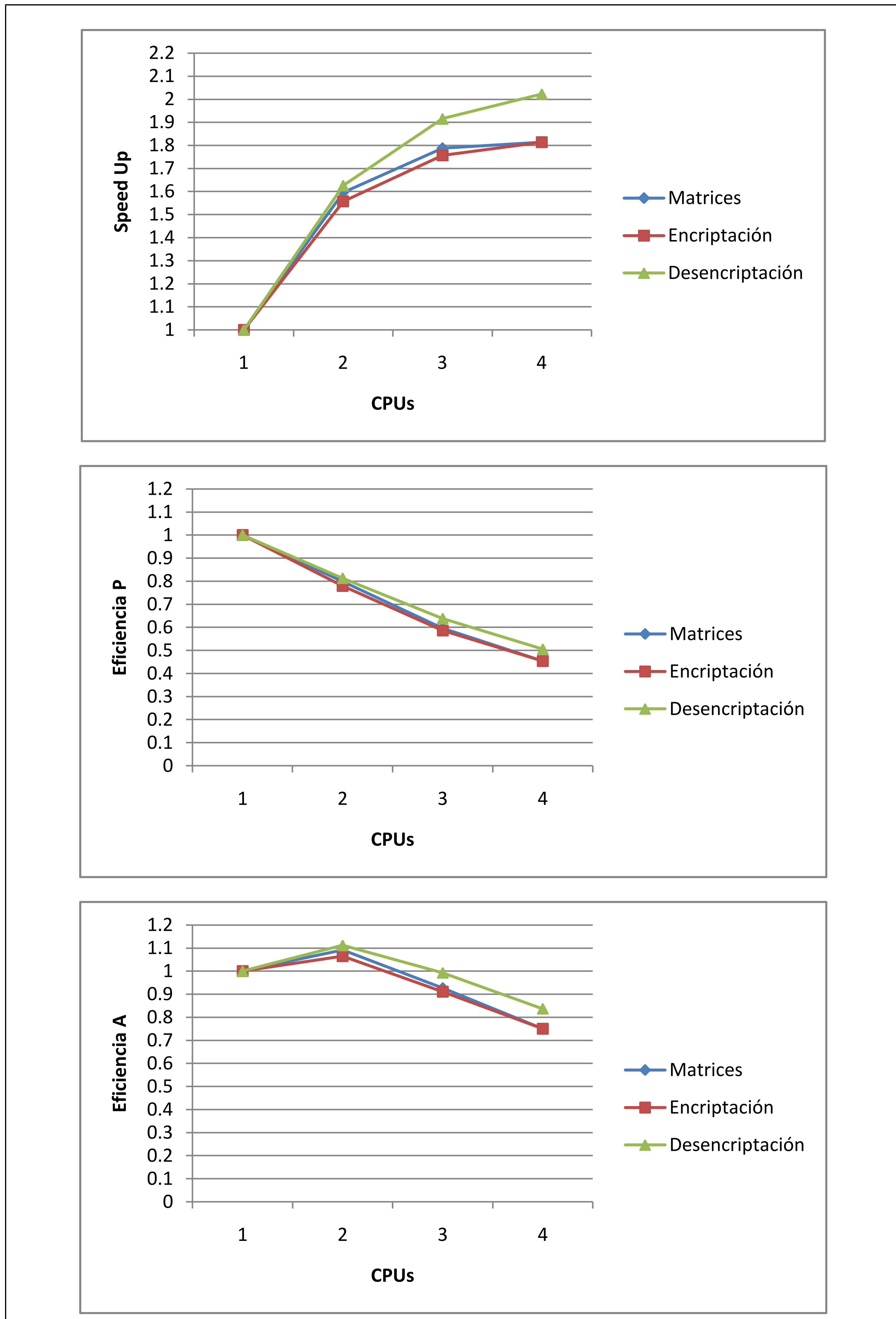


Figura 71: resultados de speedup y eficiencia en el sistema 3 con la configuración base.

Como se puede apreciar en la Figura 71, el sistema obtiene una aceleración significativa al incluir más procesadores, aunque el porcentaje de mejora es cada vez menor al aumentar el número de procesadores debido a la mayor utilización concurrente del bus del sistema. Para 2 procesadores se obtienen unos resultados muy buenos, siendo la eficiencia relativa al área ocupada superior al 100% para dos de las tres aplicaciones evaluadas. Las diferencias de aceleración entre 3 y 4 procesadores son ya bastante pequeñas ya que el bus del sistema en esas condiciones empieza a estar saturado.

Las distintas configuraciones que se testearon a partir del sistema base son las siguientes:

- Configuración 1: *time-slice* 100 ms.
- Configuración 2: optimizaciones gcc –O
- Configuración 3: caché de instrucciones 8 KB.
- Configuración 4: caché de instrucciones 1 KB.

En la Tabla 25 se presentan los resultados absolutos para las distintas configuraciones.

| | | Configuración 1 | | | | Configuración 2 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Matrices | | | | Matrices | | | |
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 79093 | 1 | 1 | 1 | 93980 | 1 | 1 | 1 |
| 2 | 1,4626 | 49899 | 1,5850 | 0,7925 | 1,0837 | 59082 | 1,5907 | 0,7953 | 1,0875 |
| 3 | 1,9288 | 44822 | 1,7646 | 0,5882 | 0,9149 | 53070 | 1,7709 | 0,5903 | 0,9181 |
| 4 | 2,4167 | 44111 | 1,7930 | 0,4483 | 0,7419 | 50051 | 1,8777 | 0,4694 | 0,7770 |
| | | Encriptación | | | | Encriptación | | | |
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 100599 | 1 | 1 | 1 | 259733 | 1 | 1 | 1 |
| 2 | 1,4626 | 65259 | 1,5415 | 0,7708 | 1,0539 | 182300 | 1,4248 | 0,7124 | 0,9741 |
| 3 | 1,9288 | 57022 | 1,7642 | 0,5881 | 0,9147 | 177430 | 1,4639 | 0,4880 | 0,7590 |
| 4 | 2,4167 | 55072 | 1,8267 | 0,4567 | 0,7559 | 176041 | 1,4754 | 0,3689 | 0,6105 |
| | | Desencriptación | | | | Desencriptación | | | |
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 161286 | 1 | 1 | 1 | 370866 | 1 | 1 | 1 |
| 2 | 1,4626 | 99242 | 1,6252 | 0,8126 | 1,1111 | 247664 | 1,4975 | 0,7487 | 1,0238 |
| 3 | 1,9288 | 83776 | 1,9252 | 0,6417 | 0,9981 | 234373 | 1,5824 | 0,5275 | 0,8204 |
| 4 | 2,4167 | 78893 | 2,0444 | 0,5111 | 0,8459 | 233585 | 1,5877 | 0,3969 | 0,6570 |

| | | Configuración 3 | | | | Configuración 4 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Matrices | | | | Matrices | | | |
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 65856 | 1 | 1 | 1 | 142023 | 1 | 1 | 1 |
| 2 | 1,4626 | 40296 | 1,6343 | 0,8172 | 1,1174 | 95290 | 1,4904 | 0,7452 | 1,0190 |
| 3 | 1,9288 | 30942 | 2,1284 | 0,7095 | 1,1035 | 93146 | 1,5247 | 0,5082 | 0,7905 |
| 4 | 2,4167 | 32232 | 2,0432 | 0,5108 | 0,8455 | 91912 | 1,5452 | 0,3863 | 0,6394 |
| | | Encriptación | | | | Encriptación | | | |
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 100987 | 1 | 1 | 1 | 120037 | 1 | 1 | 1 |
| 2 | 1,4626 | 64737 | 1,5600 | 0,7800 | 1,0665 | 100928 | 1,1893 | 0,5947 | 0,8131 |
| 3 | 1,9288 | 57397 | 1,7595 | 0,5865 | 0,9122 | 96218 | 1,2475 | 0,4158 | 0,6468 |
| 4 | 2,4167 | 55635 | 1,8152 | 0,4538 | 0,7511 | 95891 | 1,2518 | 0,3130 | 0,5180 |
| | | Desencriptación | | | | Desencriptación | | | |
| CPUS | Área | Ciclos | Speedup | Eficiencia P | Eficiencia A | Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 161934 | 1 | 1 | 1 | 152943 | 1 | 1 | 1 |
| 2 | 1,4626 | 99717 | 1,6239 | 0,8120 | 1,1103 | 130427 | 1,1726 | 0,5863 | 0,8017 |
| 3 | 1,9288 | 84515 | 1,9160 | 0,6387 | 0,9934 | 120118 | 1,2733 | 0,4244 | 0,6601 |
| 4 | 2,4167 | 79966 | 2,0250 | 0,5063 | 0,8379 | 119854 | 1,2761 | 0,3190 | 0,5280 |

Tabla 25: resultados de las distintas configuraciones.

En la Figura 72 se compara el *speedup* de los distintos sistemas en sus versiones de 2 y 4 procesadores. En ellas se observa como el factor que más afecta a la aplicación de multiplicación de matrices es el tamaño de la caché de instrucciones, mejorando considerablemente al utilizar una caché de 8 KB (configuración 3), y empeorando considerablemente con una caché de 1 KB (configuración 4). Sin embargo las aplicaciones

de encriptación y desencriptación apenas se ven afectadas por el tamaño de la caché, ya que al ser aplicaciones con un tamaño (de código) más pequeño caben prácticamente completas en una caché de 1 KB. También se puede observar como baja el rendimiento cuando no se usan las optimizaciones del compilador (configuración 2), ya que al ejecutarse más instrucciones de acceso a memoria se producen más esperas en el bus.

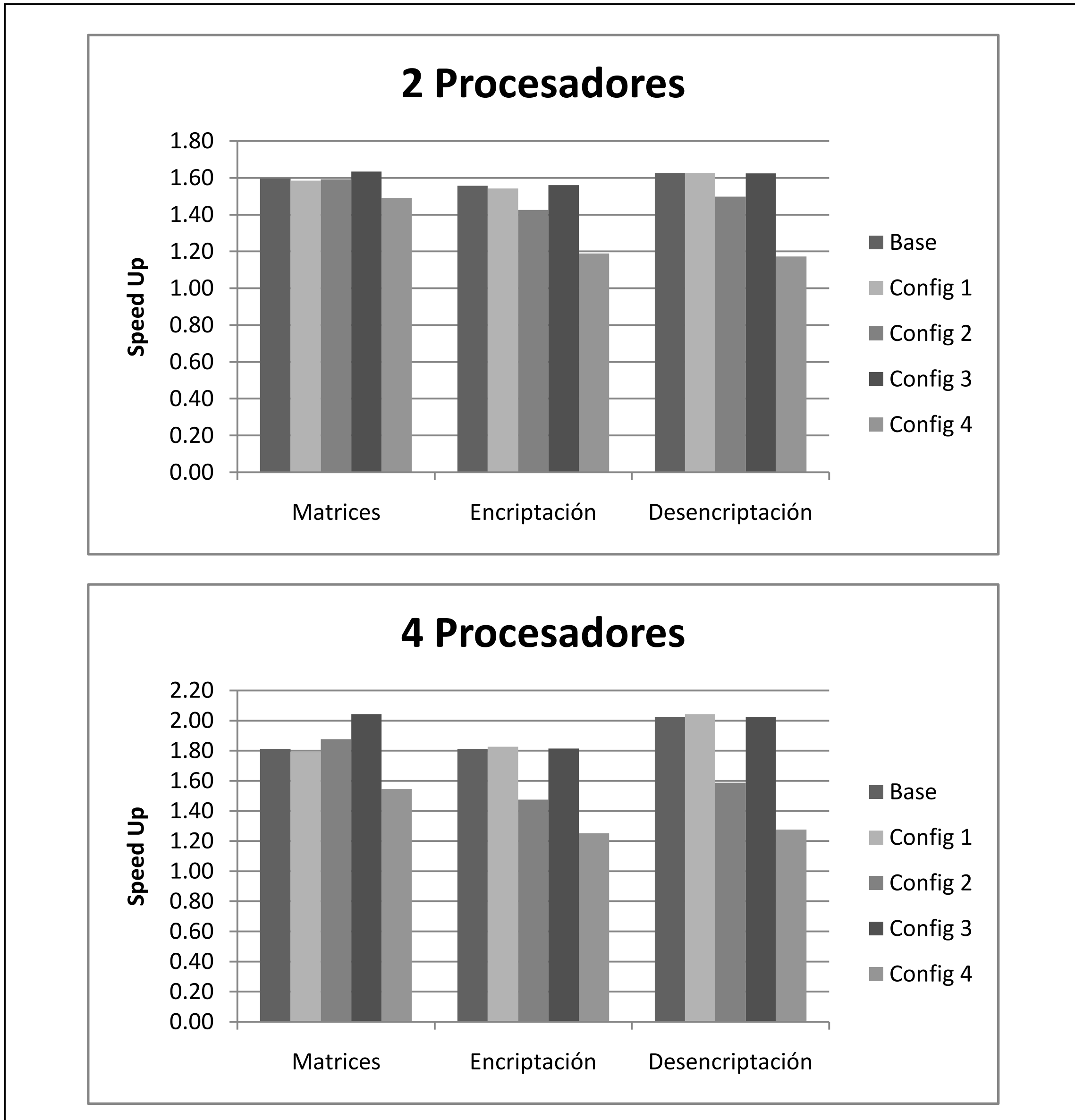


Figura 72: comparativa del speedup para las distintas configuraciones, con 2 y 4 procesadores.

#### 9.2.5.4 Conclusiones

El sistema evaluado obtiene un buen rendimiento, y para 2 procesadores presenta unos resultados de eficiencia relativa al área superiores al 100 % para las aplicaciones que se han probado.

El controlador de memoria que proporciona Xilinx para interactuar con la memoria SRAM sólo tiene 4 interfaces XCL, lo que hace que el número máximo de procesadores que pueden compartir la memoria sea 4, en el caso de utilizar sólo caché de instrucciones. Si se utilizara también caché de datos, lo cual aún no es posible, sólo podrían implementarse sistemas con 2 procesadores, ya que cada uno consumiría 2 interfaces del controlador: una para la caché de instrucciones y otra para la caché de datos.

### 9.2.6 Sistema 4

El último sistema es similar al descrito en el apartado anterior, pero utilizando una memoria externa DDR en lugar de SRAM. Con este sistema se pretende evaluar cómo afecta a un sistema SMP el uso de una memoria más grande y menos rápida.

#### 9.2.6.1 Arquitectura

El controlador de memoria que proporciona Xilinx para interactuar con memoria DDR es el Multi-Channel OPB DDR Controller (Figura 73). Este controlador se caracteriza por tener dos tipos de interfaz con el procesador. Por una parte tiene una interfaz OPB, y por otra tiene un número parametrizable de interfaces XCL para conectar a las memorias caché del procesador Microblaze.

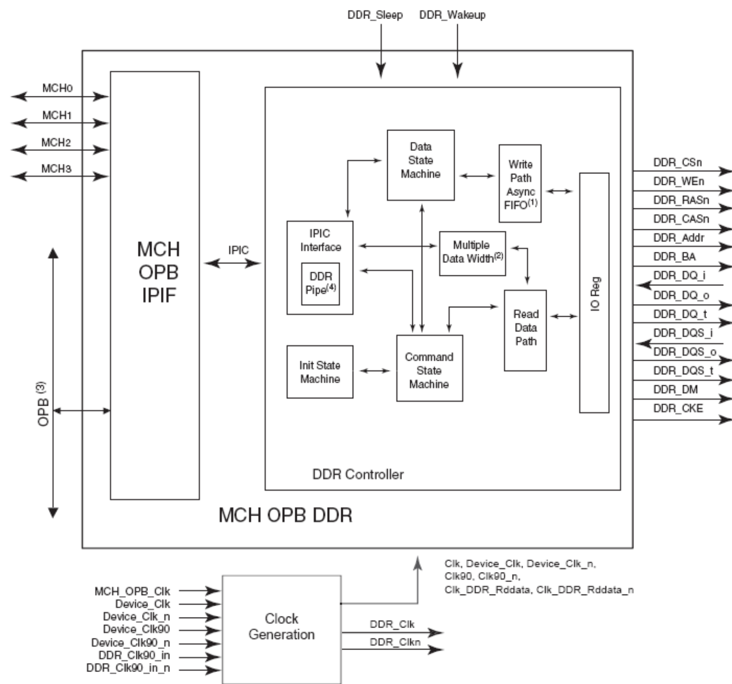


Figura 73: controlador de memoria MultiChannel DDR de Xilinx (86).

Utilizando este nuevo controlador de memoria, se ha implementado un sistema como el mostrado en la Figura 74, que incluye:

- 4 procesadores Microblaze versión 6.0a, con la interfaz de instrucciones Xilinx Cache Link habilitada y 4 Kbytes de caché de instrucciones.
- 4 módulos, uno por procesador, de 2 Kbytes de memoria de bloque. Esta memoria hará las funciones de zona de memoria privada que necesita el sistema operativo para funcionar correctamente.
- Un bus OPB compartido por todos los procesadores y al que están conectados:
    - 64 Megabytes de memoria DDR.
    - Un periférico opb_uartlite para entrada/salida.
    - Un periférico opb_timer, para ser usado como fuente de interrupción de todos los procesadores.
    - Otro periférico opb_timer que se utilizará para medir el tiempo que tardan en ejecutarse los distintos programas.
    - Un hardware-mutex, necesario para poder ejecutar el sistema operativo.
- 4 registros con interfaz FSL, uno para cada procesador, que contendrán el identificador del procesador.

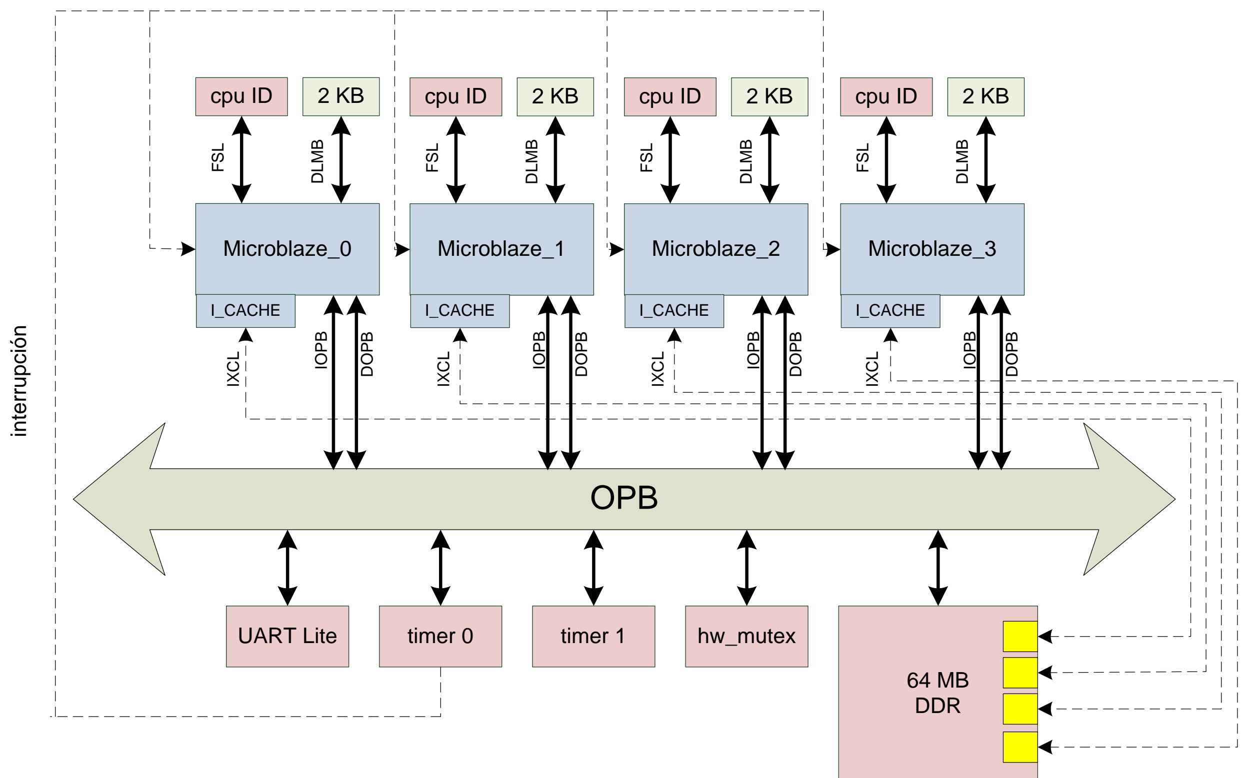


Figura 74: arquitectura del sistema 4.

#### 9.2.6.2 Resultados de síntesis

A continuación se presentan los resultados de síntesis del sistema. Se realizaron síntesis para el mismo sistema con 1, 2, 3 y 4 procesadores, con el objetivo de ver las diferencias de área utilizada al usar más o menos procesadores y poder así calcular la eficiencia del sistema respecto del área ocupada.

La FPGA para la que se realizaron las síntesis es una Virtex4 XC4VLX25-FF668-10, incluida en la placa de desarrollo Xilinx ML401 sobre la que se probaron posteriormente los distintos sistemas. La Tabla 26, muestra los resultados totales de síntesis para el sistema completo con los 4 procesadores, y Tabla 27 muestra las relaciones entre el número de *slices, LUTS* y puertas equivalentes para 1, 2, 3 y 4 procesadores.

```
Design Summary:
Logic Utilization:
  Number of Slice Flip Flops:        6,693 out of  21,504   31%
  Number of 4 input LUTs:            9,280 out of  21,504   43%
Logic Distribution:
  Number of occupied Slices:         7,921 out of  10,752   73%
Total Number of 4 input LUTs:       12,199 out of  21,504   56%
Number of bonded IOBs:                69 out of     448   15%
Number of BUFG/BUFGCTRLs:              5 out of      32   15%
Number of FIFO16/RAMB16s:             32 out of      72   44%
Number of DSP48s:                     12 out of      48   25%
Number of DCM_ADVs:                    2 out of       8   25%
Number of BSCAN_VIRTEX4s:              1 out of       4   25%

Total equivalent gate count for design:  2,359,680
```

Tabla 26: resultados de síntesis para el sistema con 4 procesadores.

| | SLICES | | LUTS | | NAND GATES | |
|---|---|---|---|---|---|---|
| | Número | Razón | Número | Razón | Número | Razón |
| 1 CPU | 3493 | 1 | 5071 | 1 | 830773 | 1 |
| 2 CPUs | 4948 | 1,4165 | 7392 | 1,4577 | 1340085 | 1,6131 |
| 3 CPUs | 6400 | 1,8322 | 9760 | 1,9247 | 1849729 | 2,2265 |
| 4 CPUs | 7921 | 2,2677 | 12199 | 2,4056 | 2359680 | 2,8403 |

Tabla 27: comparativa de la ocupación de *slices*, *LUTS* y puertas equivalentes para diferentes números de procesadores.

### 9.2.6.3 Resultados de los tests software

Sobre este sistema se probaron una serie de configuraciones partiendo de la configuración base. Al igual que en los otros sistemas descritos, se ha utilizado la relación entre el número de *slices* a la hora de calcular la relación de área de los distintos diseños. En la Tabla 28 se muestran los resultados obtenidos para los distintos tests en el sistema base.

| | | Matrices | | | |
|---|---|---|---|---|---|
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 97904 | 1 | 1 | 1 |
| 2 | 1,4165 | 65744 | 1,4892 | 0,7446 | 1,0513 |
| 3 | 1,8322 | 57614 | 1,6993 | 0,5664 | 0,9275 |
| 4 | 2,2677 | 51312 | 1,9080 | 0,4770 | 0,8414 |
| | | Encriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 124049 | 1 | 1 | 1 |
| 2 | 1,4165 | 87885 | 1,4115 | 0,7057 | 0,9964 |
| 3 | 1,8322 | 80366 | 1,5436 | 0,5145 | 0,8424 |
| 4 | 2,2677 | 80015 | 1,5503 | 0,3876 | 0,6837 |
| | | Desencriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 195090 | 1 | 1 | 1 |
| 2 | 1,4165 | 130053 | 1,5001 | 0,7500 | 1,0590 |
| 3 | 1,8322 | 114456 | 1,7045 | 0,5682 | 0,9303 |
| 4 | 2,2677 | 112647 | 1,7319 | 0,4330 | 0,7637 |

Tabla 28: resultados de los tests para la configuración base.

Como se puede apreciar en las gráficas que se muestran en la Figura 75, el sistema obtiene una aceleración significativa al incluir más procesadores, aunque la mejora es cada vez menor al aumentar el número de procesadores debido a la mayor utilización concurrente del bus del sistema. Para 2 procesadores se obtienen unos resultados muy buenos, siendo la eficiencia relativa al área ocupada muy cercana al 100%. Las diferencias de aceleración entre 3 y 4 procesadores son apenas apreciables ya que el bus del sistema en esas condiciones está ya saturado.

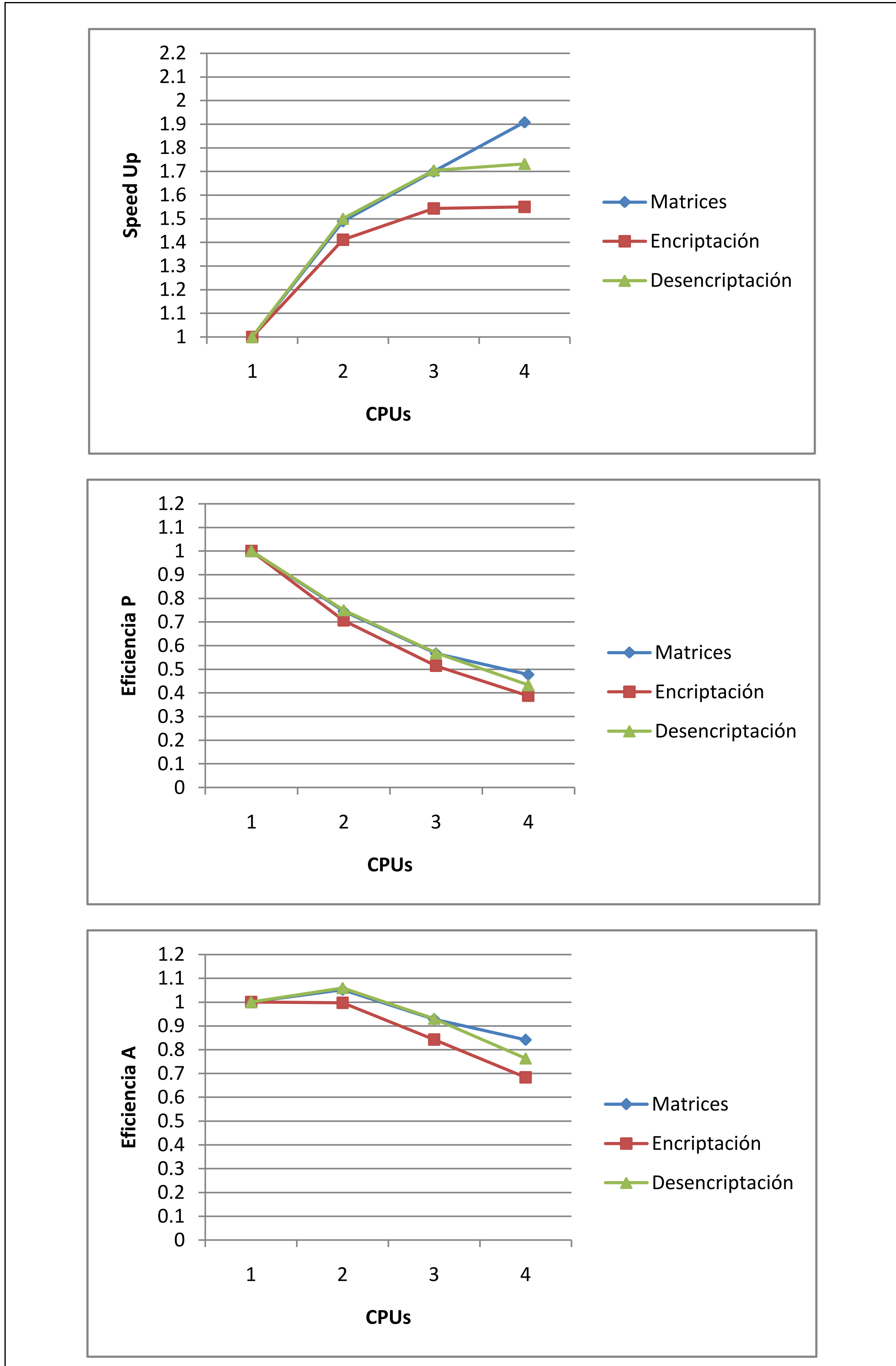


Figura 75: resultados de speedup y eficiencia para la configuración base.

La Tabla 29 muestra los resultados de ejecución de los tests software en las distintas configuraciones del sistema que se han evaluado. Dichas configuraciones son:

- Configuración 1: time-slice 100 ms.
- Configuración 2: optimizaciones gcc –O
- Configuración 3: caché de instrucciones 8 KB.
- Configuración 4: caché de instrucciones 1 KB.

Las gráficas de la Figura 76 muestran una comparativa del speedup de todas las configuraciones en sus versiones de 2 y 4 procesadores. De los resultados presentados se puede concluir que:

- Para las dos duraciones de *time-slice* probadas, apenas hay diferencias en el rendimiento, siendo un poco mejor el rendimiento con un *time-slice* más grande.
- El no usar optimizaciones de compilación hace que el rendimiento relativo sea menor, al tener el código compilado más instrucciones de acceso a datos, que son las que colapsan el bus al no poder usarse una caché de datos.
- El uso de una caché de instrucciones más grande, apenas afecta al *speedup* del sistema para el tipo de aplicaciones que se han probado.

Una caché de instrucciones pequeña hace que el sistema pierda mucho rendimiento, principalmente porque al producirse más fallos de caché, hay más accesos al bus haciendo que haya más bloqueos al competir los distintos procesadores por el uso de éste.

| | | Configuración 1 | | | | Configuración 2 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Matrices | | | | Matrices | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 97904 | 1 | 1 | 1 | 112628 | 1 | 1 | 1 |
| 2 | 1,4165 | 65744 | 1,4892 | 0,7446 | 1,0513 | 78813 | 1,4291 | 0,7145 | 1,0088 |
| 3 | 1,8322 | 60218 | 1,6258 | 0,5419 | 0,8873 | 69189 | 1,6278 | 0,5426 | 0,8884 |
| 4 | 2,2677 | 51312 | 1,9080 | 0,4770 | 0,8414 | 66182 | 1,7018 | 0,4254 | 0,7505 |
| | | Encriptación | | | | Encriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 123356 | 1 | 1 | 1 | 323876 | 1 | 1 | 1 |
| 2 | 1,4165 | 91083 | 1,3543 | 0,6772 | 0,9561 | 244569 | 1,3243 | 0,6621 | 0,9349 |
| 3 | 1,8322 | 82560 | 1,4941 | 0,4980 | 0,8155 | 244288 | 1,3258 | 0,4419 | 0,7236 |
| 4 | 2,2677 | 81447 | 1,5146 | 0,3786 | 0,6679 | 241389 | 1,3417 | 0,3354 | 0,5917 |
| | | Desencriptación | | | | Desencriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 193970 | 1 | 1 | 1 | 454184 | 1 | 1 | 1 |
| 2 | 1,4165 | 134148 | 1,4459 | 0,7230 | 1,0207 | 326883 | 1,3894 | 0,6947 | 0,9809 |
| 3 | 1,8322 | 117307 | 1,6535 | 0,5512 | 0,9025 | 320398 | 1,4176 | 0,4725 | 0,7737 |
| 4 | 2,2677 | 113957 | 1,7021 | 0,4255 | 0,7506 | 316876 | 1,4333 | 0,3583 | 0,6321 |

| | | Configuración 3 | | | | Configuración 4 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Matrices | | | | Matrices | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 81031 | 1 | 1 | 1 | 178581 | 1 | 1 | 1 |
| 2 | 1,4165 | 52972 | 1,5297 | 0,7648 | 1,0799 | 125728 | 1,4204 | 0,7102 | 1,0027 |
| 3 | 1,8322 | 46689 | 1,7355 | 0,5785 | 0,9472 | 120717 | 1,4793 | 0,4931 | 0,8074 |
| 4 | 2,2677 | 44743 | 1,8110 | 0,4528 | 0,7986 | 115002 | 1,5528 | 0,3882 | 0,6848 |
| | | Encriptación | | | | Encriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 124139 | 1 | 1 | 1 | 125996 | 1 | 1 | 1 |
| 2 | 1,4165 | 85880 | 1,4455 | 0,7227 | 1,0204 | 108057 | 1,1660 | 0,5830 | 0,8231 |
| 3 | 1,8322 | 79561 | 1,5603 | 0,5201 | 0,8516 | 103015 | 1,2231 | 0,4077 | 0,6675 |
| 4 | 2,2677 | 78999 | 1,5714 | 0,3928 | 0,6930 | 102664 | 1,2273 | 0,3068 | 0,5412 |
| | | Desencriptación | | | | Desencriptación | | | |
| CPUS | Área | K Ciclos | Speedup | Eficiencia P | Eficiencia A | K Ciclos | Speedup | Eficiencia P | Eficiencia A |
| 1 | 1 | 194601 | 1 | 1 | 1 | 196466 | 1 | 1 | 1 |
| 2 | 1,4165 | 128261 | 1,5172 | 0,7586 | 1,0711 | 171061 | 1,1485 | 0,5743 | 0,8108 |
| 3 | 1,8322 | 113531 | 1,7141 | 0,5714 | 0,9355 | 157541 | 1,2471 | 0,4157 | 0,6806 |
| 4 | 2,2677 | 111509 | 1,7452 | 0,4363 | 0,7696 | 157194 | 1,2498 | 0,3125 | 0,5512 |

Tabla 29: resultados de los tests para las distintas configuraciones.

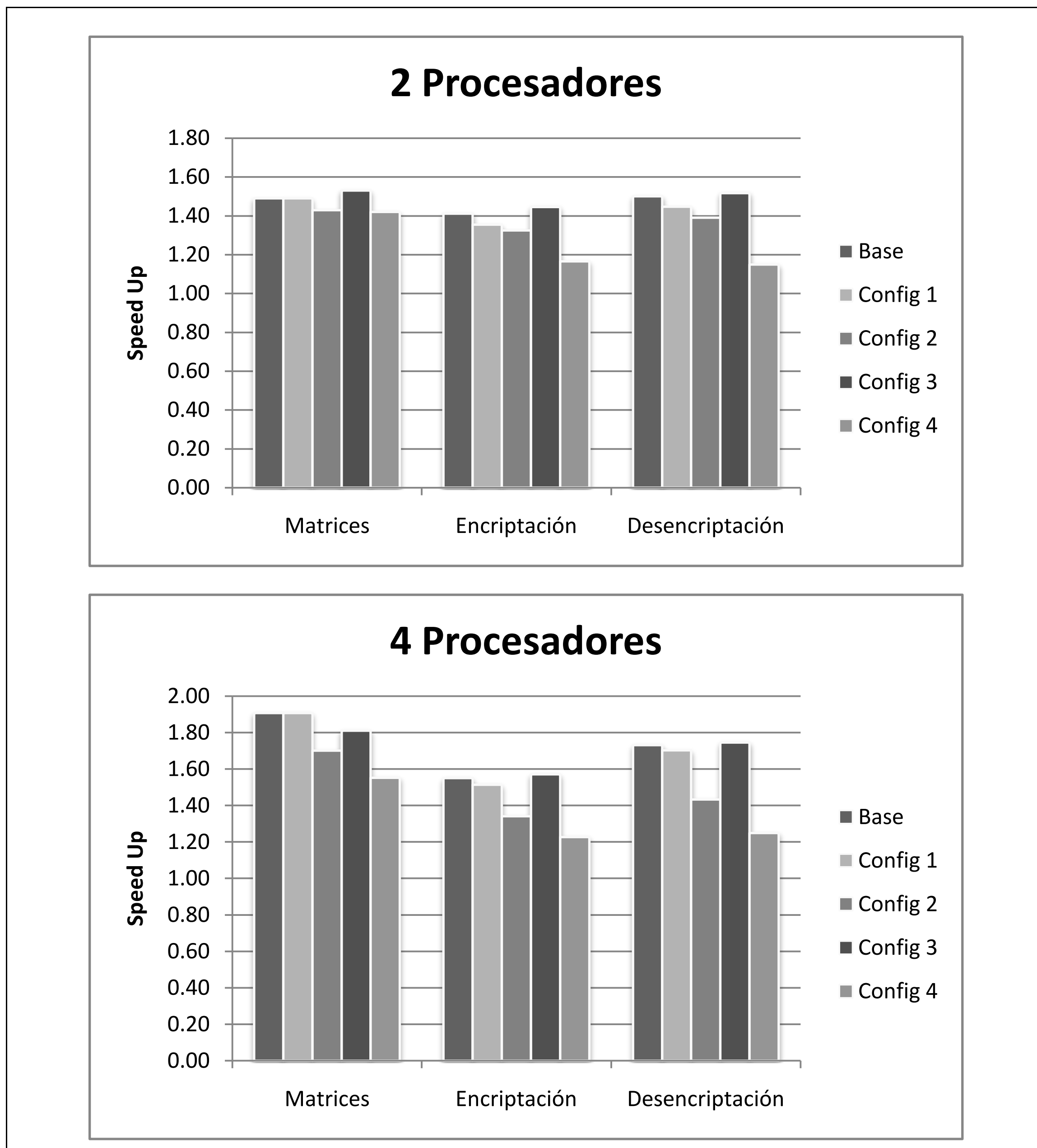


Figura 76: comparativa de speedup para las distintas configuraciones.

#### 9.2.6.4 Conclusiones

El sistema evaluado obtiene unos rendimientos razonables, y para 2 procesadores presenta unos resultados de eficiencia relativa al área cercana al 100 % para las aplicaciones que se han probado.

El controlador de memoria que proporciona Xilinx para interactuar con la memoria DDR sólo tiene 4 interfaces XCL, lo que hace que el número máximo de procesadores que pueden compartir la memoria sea 4, en el caso de utilizar sólo caché de instrucciones. Si se

utilizara también caché de datos, lo cual aún no es posible, sólo podrían implementarse sistemas con 2 procesadores, ya que cada uno consumiría 2 interfaces del controlador: una para la caché de instrucciones y otra para la caché de datos.

## 9.3 Conclusiones

En este capítulo se han presentado varios sistemas SMP que hacen uso de diferentes arquitecturas de memoria compartida. Para todos los sistemas se han realizado experimentos que han permitido evaluar el rendimiento, que va desde un rendimiento casi ideal en el caso del Sistema 1 hasta rendimientos menos ideales en los otros sistemas.

También se han identificado los principales problemas que surgen a la hora de implementar este tipo de sistemas en FPGA, siendo el más importante la ausencia de soporte para coherencia de cachés y de un controlador de interrupciones preparado para repartir las interrupciones entre los distintos procesadores. Otro problema que se ha identificado es la política de acceso a memoria presente en la versión 4 del procesador MicroBlaze, que lo hace poco apropiado para ser usado en sistemas multiprocesador, por lo que el resto de sistemas se han realizado utilizando la versión 5 del procesador.

# 10 CONCLUSIONES Y TRABAJO FUTURO

La evolución en capacidad y prestaciones de las FPGAs y la popularidad de los procesadores *soft-core* ha propiciado la aparición de sistemas multiprocesador en FPGA muy variados.

En esta tesis se han presentado diversos sistemas multiprocesador en FPGA existentes, tanto de propósito general como específico, y se han identificado algunos de las carencias existentes en el desarrollo de este tipo de sistemas.

El trabajo experimental de la tesis ha consistido en la implementación de diversos sistemas multiprocesador con el objetivo de llegar a una implementación final de un sistema multiprocesador de propósito general, flexible, escalable y con el soporte de un sistema operativo que permita ejecutar aplicaciones multihilo de forma eficiente sobre él.

## *10.1 Principales aportaciones de la tesis*

Las principales aportaciones de esta tesis son:

- **Estudio teórico de los problemas y necesidades que se presentan a la hora de implementar un sistema multiprocesador sobre una FPGA:** se han estudiado las necesidades de comunicación, memoria y soporte software que requieren los sistemas multiprocesador en FPGA.

- **Propuesta de una arquitectura de propósito general basada en el concepto de multiprocesamiento simétrico:** se han presentado los requisitos hardware mínimos necesarios para construir un sistema SMP sobre FPGA.

- **Desarrollo de un sistema operativo para sistemas SMP:** se ha desarrollado un sistema operativo con soporte para multiprocesamiento simétrico a partir del S.O *Xilkernel* que proporciona Xilinx. Este sistema permite la ejecución de aplicaciones multihilo de forma eficiente sobre sistemas SMP basados en el procesador *soft-core* MicroBlaze.

- **Evaluación del rendimiento de 4 sistemas SMP diferentes:** se han implementado 4 sistemas SMP que hacen uso de diferentes arquitecturas de memoria y se ha evaluado el rendimiento de distintas aplicaciones paralelas sobre ellos.

## 10.2 Trabajo actual y futuro

La principal carencia de los sistemas que se han desarrollado es la imposibilidad de utilizar la caché de datos de los procesadores debido a la ausencia de un mecanismo que garantice la coherencia de las cachés de los distintos procesadores. Actualmente se está trabajando en una solución a este problema, que permitiría utilizar cachés de datos en el sistema aumentando considerablemente el rendimiento de éste. Una vez se puedan usar cachés de datos se hará necesario también desarrollar nuevos controladores de memoria que soporten más canales XCL, ya que con los controladores actuales sólo se podrían desarrollar sistemas con dos procesadores.

Otra de las limitaciones de los sistemas que se han presentado es la gestión de las interrupciones, que actualmente no están completamente soportadas. Se está trabajando en el desarrollo de un controlador de interrupciones que permita manejarlas de forma más eficiente, interrumpiendo solamente a un procesador cada vez en lugar de a todos simultáneamente.

También está previsto portar el sistema operativo presentado en el capítulo 8 para ser utilizado con los 2 procesadores PowerPC que incluyen algunas FPGAs de las familias Virtex 2 Pro, Virtex 4 y Virtex 5.

## 10.3 Publicaciones

Durante el desarrollo de esta tesis doctoral se han publicado en congresos y revistas una serie de trabajos, algunos relacionados directamente con la tesis y otros no, que se enumeran a continuación.

### 10.3.1 Publicaciones relacionadas con la tesis

- Huerta, P.; Castillo, J.; Martínez, J.I.; de La Lama, C: "Operating System for Symmetric Multiprocessors on FPGA", 2008 International Conference on ReConFigurable Computing and FPGAs, ReConFig'08, 2008

- Huerta, P.; Castillo, J.; Martínez, J.I.; de La Lama, C: "Sistema Opeartivo para SMPs basados en MicroBlaze", Jornadas de Configuración Reconfigurable y Aplicaciones 2008 (JCRA 08), 2008

- Huerta, P.; Castillo, J.; Pedraza, C.A.; Martínez, J.I.: "Exploring FPGA Capabilities for Building Symmetric Multiprocessor Systems", IEEE III Southern Conference on Programmable Logic, SPL07, 2007

- Huerta, P.; Castillo, J.; Martínez, J.I.: "Sistemas MPSoC en FPGAs", Jornadas de Computación Reconfigurable y Aplicaciones 2006, JCR'06, 2006

- Huerta, P.; Castillo, J.; López, V; Martínez, J.I: "A MicroBlaze based MultiProcessor Soc", WSEAS Transactions on Circuits and Systems, 2005

- Huerta, P.; Castillo, J.; López, V; Martínez, J.I: "Multi Microblaze System for Parallel Computing", 9th CSCC Intenational Conference on Computers CSCC'05, 2005

### 10.3.2 Otras publicaciones

- Pedraza, C.A.; Castillo, J.; Huerta, P.; de la Lama, C.; Martínez, J.I: "Self-Reconfigurable Secure File System for Embedded Linux", IET Computer & Digital Techniques, 2009

- Castillo, J.; Huerta, P.; Martínez, J.I.: "Secure IP Downloading for SRAM FPGAs", Microprocessors and Microsystems, 2007

- Castillo, J.; Huerta, P.; Martínez, J.I.: "An Open-Source Tool for SystemC to Verilog Automatic Translation", Latin American Applied Research, 2007

- Castillo, J.; Huerta, P.; Martínez, J.I.: "SystemC Design Flow for a DES/AES CryptoProcessor", WSEAS Transactions on Information Science, 2004

- Cano, J.; Castillo, J.; Huerta, P.; Pedraza, C.A.; Martínez, J.I: "Red de Interconexión de placas de prototipado rápido para computación de altas prestaciones", Jornadas de Computación Reconfigurable y Aplicaciones JCRA'08, 2008

- Castillo, J.; Huerta, P.; Pedraza, C.A.; de la Lama, C.; Martínez, J.I: "Parallel Implementation of the Shortest Path Algorithm on FPGA", IEEE IV Southern Conference on Programmable Logic SPL'08, 2008

- de la Lama, C.; Huerta, P.; Castillo, J.; Pedraza, C.A.; Martínez, J.I: "GNU autotools para proyectos hardware", VII Jornadas de Computación Reconfigurable y Aplicaciones JCRA'07, 2007

- Pedraza, C.A.; Castillo, J.; Huerta, P.; de la Lama, C.; Martínez, J.I: "Sistema de Archivos Seguros Autoreconfigurables para Linux Embebido", VII Jornadas de Computación Reconfigurable y Aplicaciones JCRA'07, 2007

- Castillo, J.; Huerta, P.; Martínez, J.I.: "An Open-Source Tool for SystemC to Verilog Automatic Translation", : II Southern Conference on Programmable Logic SPL'06, 2006

- Castillo, J.; Huerta, P.; Pedraza, C.A.; Martínez, J.I.: "A Self-Reconfigurable Multimedia Player on FPGA", IEEE Reconfig'06, 2006

- López, V.; Huerta, P.; Castillo, J.; Martínez, J.I.: "Fingerprint Minutiae Extraction Based On FPGA and Matlab", XX Design of Circuits and Integrated Systems Conference DCIS'05, 2005

- Castillo, J.; Huerta, P.; López, V.; Martínez, J.I.: "A Secure Self-Reconfiguring Architecture based on Open-Source Hardware", IEEE Reconfig'05, 2005

- Castillo, J.; González, I.; Huerta, P.; Martínez, J.I.: "Auto-reconfiguración sobre FPGAs", V Jornadas de Computación Reconfigurable y Aplicaciones JCRA'05, 2005

- Castillo, J.; Huerta, P.; Martínez, J.I.: "SystemC design flow for a DES/AES CryptoProcessor", 8th CSCC International Conference on Computers CSCC'04, 2004

- Bolado, M.; Posadas, H.; Castillo Villar, Javier; Huerta, P. ; Sánchez Espeso, P.; Sánchez de la Lama, C.; Fouren, H.; Blasco, F: "Platform based on Open-Source Cores for Industrial Applications", Design Automation and Test in Europe DATE'04, 2004

# 11 BIBLIOGRAFÍA


1. *Structured ASICs: Opportunities an Challenges.* **Zahiri, B.** 2003. Proceedings of th 21st International Conference on Computer Design (ICCD'03). pp. 404-409.

2. *Physical Design for System-On-a-Chip.* **Y.-W. Chang, T.-C. Chen, and H.-Y. Chen.** s.l. : Springer, 2006, Essential Issues in SOC Design, pp. 311-403.

3. **Celoxica.** *Handel-C Language Reference Manual.* 2005.

4. **OSCI.** SystemC 2.1. [Online] http://www.systemc.org.

5. *Integrating IP blocks to create a system-on-a-chip.* **Tuck, B.** 1997, Computer Design, pp. 49-62.

6. **Opencores.** WISHBONE, Revision B.3 Specification. [Online] http://www.opencores.org/projects.cgi/web/wishbone/wbspec_b3.pdf.

7. *"Defining platform-based design".* **Sangiovanni-Vincentelli, A.** 2002, EE Times On Line.

8. *Platform based on Open Source Cores for Industrial Applications.* **M. Bolado, J. Castillo, H. Posadas, P. Huerta, P. Sánchez, E. Villar, C. Sánchez, P. Blasco, H. Fouren.** 2004. Design Automation and Test in Europe, DATE 04.

9. **Gottlieb, G.S.Almasi and A.** *Highly Parallel Computing.* s.l. : Benjamin-Cummings publishers, 1989.

10. *Some computer organizations and their effectiveness.* **M.J, Flynn.** 1972, IEEE Transactions on Computers, pp. 948-960.

11. **J.L. Hennesy, D.A Patterson.** *Computer Architecture A Quantitative Approach.* Cuarta edición. s.l. : Morgan Kauffman, 2007.

12. **D.E. Culler, J.P. Singh.** *Parallel Computer Architecture a Hardware/Software Approach.* s.l. : Morgan Kauffman, 1999.

13. **A. Grama, A. Gupta , G. Karypis , V. Kumar.** *Introduction to Parallel Computing.* Segunda edición. s.l. : Addison Wesley, 2003.

14. *Completing an MIMD multiprocessor taxonomy.* **Johnson, E. E.** 1988, ACM SIGARCH Computer Architecture News, Vol. 16, pp. 44-47.

15. *Introduction to Parallel Processing.* s.l. : Springer US, 2006. 1567-7974.

16. **Stokes, J.** ars technica. *Inside the Xbox 360, Part II: the Xenon CPU.* [Online] 2005. http://arstechnica.com/articles/paedia/cpu/xbox360-2.ars/1.

17. *Introduction to the Cell multiprocessor.* **IBM.** 2005, IBM Journal of Research and Development.

18. **IBM.** Power ISA Version 2.04. [Online] 2007. http://www.power.org/resources/downloads/PowerISA_203.Public.pdf.

19. *Synergistic Processing in Cell's Multicore Architecture.* **M. Gschwind, H.P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, T. Yamazaki.** 2, 2006, IEEE Micro, Vol. 26, pp. 10-24.

20. Texas Instruments TMS320DM6467 Digital Media System-on-Chip. [Online] http://focus.ti.com/lit/ds/symlink/tms320dm6467.pdf.

21. Intel IXP2800 and IXP2850 Network Processors. [Online] http://download.intel.com/design/network/datashts/27853717.pdf.

22. *Viper: A Multiprocessor SOC for Advanced Set-Top Box And Digital TV Systems.* **S. Dutta, R. Jensen, A. Rieckmann.** 2001, IEEE Design and Test of Computers, pp. 21-31.

23. **Xilinx.** *XC2000 Logic Cell Array Families.* 1985.

24. *Cramming More Components onto Integrated Circuits.* **Moore, G. E.** 1965, Electronics.

25. **Maxfield, C.** *The Design Warrior's Guide to FPGAs.* s.l. : Elsevier, 2004.

26. **Actel.** *Accelerator Series FPGAs: ACT3 Family.* 1997.

27. **Acte.** *SX Family of High Performance FPGAs.* 2001.

28. *The Chimaera Reconfigurable Functional Unit.* **S. Hauck, T. Fry, M. Hosler y J. Kao.** 2004, IEEE Transactions on VLSI Systems, pp. 206-217.

29. *The NAPA adaptive processing architecture.* **C. Rupp, M. Landguth, T. Garverick, E. Comersall, H. Holt, J. Arnold y M. Gokhale.** 1998. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98). pp. 28-37.

30. *Parallel programmable antifuse field programmable gate array device (FPGA) and a method for programming and testing an antifuse FPGA. 7269814* United States of America, 2007.

31. **Rajsuman, R.** *System-on-a-Chip. Design and Test.* s.l. : Artech House Publishers, 2000.

32. *Configurable Systems-on-Chip (CSoC).* **Becker, J.** 2002. 15th Symposium on Integrated Circuits and System Design (SBCCI2002). pp. 379-384.

33. *Closing the SoC Design Gap.* **Henkel, Jörg.** 2003, IEEE Computer, pp. 119-121.

34. **P. Bricaud, M. Keating.** *Reuse Methodology Manual.* s.l. : Kluver Academic Publisher, 1998.

35. **Semiconductor Industry Association.** *International Technology Roadmap for Semiconductors.* 1999.

36. **G. Martin.** "The Rise of Processor-Centric Design", Electronic News. [Online] http://www.edn.com/index.asp?layout=article&articleid=CA6355153&partner=enews.

37. *Experiences with Soft-Core Processor Design.* **F. Plavec, B. Fort, Z. G. Vranesic, S.D. Brown.** 2005, Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, pp. 167-170.

38. **Xilinx Inc.** PicoBlaze 8-bit Embedded Microcontroller User Guide. [Online] http://www.xilinx.com/support/documentation/user_guides/ug129.pdf.

39. **Xilinx Inc**. MicroBlaze Processor Reference Guide. [Online] http://www.xilinx.com/support/documentation/sw_manuals/edk92i_mb_ref_guide.pdf.

40. **Altera Inc.** Nios II Processor Reference Handbook. [Online] http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf.

41. **Lattice Semiconductor Corporation.** LatticeMico32 Processor Reference Manual. [Online] http://www.latticesemi.com/documents/doc20890x45.pdf.

42. **Lampret, D.** OpenRISC 1200 IP Core Specification. [Online] http://www.opencores.org/tmp/cvsget_cache/or1k/or1200/doc/or1200_spec.pdf.

43. OpenCores. [Online] http://www.opencores.org/.

44. **Gaisler Research.** LEON2 Processor User's Manual. [Online] http://www.freehardwarefoundation.org/documents/leon/leon2-1.0.23-xst.pdf.

45. **Gaisler Research.**. GRLIB IP Library User¡s Manual. [Online] http://www.gaisler.com/products/grlib/grlib.pdf.

46. **Gaisler Research.**. [Online] http://www.gaisler.com/.

47. **Xilinx Inc.** Local Memory Bus (LMB). [Online] http://www.xilinx.com/support/documentation/ip_documentation/lmb.pdf.

48. **IBM.** On-Chip Peripheral Bus. [Online] http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/9A7AFA74DAD200D087256AB30005F0C8.

49. **Xilinx Inc.** Fast Simplex Link (FSL) Bus. [Online] http://www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf.

50. **Xilinx Inc.** Embedded System Tools Reference Manual. [Online] http://www.xilinx.com/support/documentation/sw_manuals/edk10_est_rm.pdf.

51. Eclipse. [Online] http://www.eclipse.org/.

52. **Altera.** Quartus II Version 8.0 Handbook. Volume 4: SOPC Builder. [Online] http://www.altera.com/literature/hb/qts/qts_qii5v4.pdf.

53. **Labrosse, J.J.** *MicroC/OS-II: The Real Time Kernel.* Segunda edición. s.l. : CMP Books, 2002.

54. OpenRISC 1200 Graphic Configuration Tool. [Online] http://www.opencores.org/projects.cgi/web/or1200gct/overview.

55. **eCosCentric.** eCos Reference Manual. [Online] http://ecos.sourceware.org/docs-2.0/pdf/ecos-2.0-ref-a4.pdf.

56. uClinux. Embedded Linux/Microcontroller Project. [Online] http://www.uclinux.org/.

57. **D. Mattsson, M. Christensson.** *Evaluation of synthesizable CPU cores.* Chalmers University of Technology. Master's Thesis in Computer Science.

58. **SPARC International Inc.** The SPARC Architecture Manual, Version 8. [Online] http://www.sparc.org/standards/V8.pdf.

59. *AMBA: Enabling Reusable On-Chip Designs".* **Flynn, D.** 4, 1997, IEEE Micro, Vol. 17, pp. 20-27.

60. **Gaisler Research.** LEON/ERC32 Cross Compilation System. [Online] http://www.gaisler.com/products/leccs/leccs.html.

61. RTEMS. [Online] http://www.rtems.com/.

62. **Garret, Bob.** Multi-Processor Solutions with FPGAs. [Online] 2005. http://www.fpgajournal.com/whitepapers_2005/altera_20050224.htm.

63. *Onchip interconnect exploration for multicore processors utilizing FPGAs.* **G. Schelle, D. Grunwald.** 2006. In 2nd Workshop on Architecture Research using FPGA Platforms.

64. *An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip.* **Kumar, A., et al.** Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07. pp. 1-6.

65. *Network on chip using a reconfigurable platform.* **Sun, Li Ping, Aboulhamid, El Mostapha and David, J.-P.** Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems, 2003. MWSCAS '03. pp. 819-822.

66. *A comparison of five different multiprocessor SoC bus architectures.* **K.K. Ryu, E. Shin, V.J. Mooney.** 2001. Proceedings of Euromicro Symposium on Digital Systems Design. pp. 202-209.

67. *Communication strategies for shared-bus embedded multiprocessors.* **Bambha, N. K. and Bhattacharyya, S. S.** 2005. Proceedings of the 5th ACM international Conference on Embedded Software. pp. 21-24.

68. *A design kit for a fully working shared memory multiprocessor on FPGA.* **A. Tumeo, M. Monchiero, G. Palermo, F. Ferrandi, D. Sciuto.** 2007. roceedings of the 17th ACM Great Lakes symposium on VLSI. pp. 219-222.

69. *Enabling Cache Coherency for N-Way SMP Systems on Programmable Chips.* **A. Hung, W. Bishop, and A. Kennings.** 2004. Proceedings of the 2004 Intl. Conference on Engineering of Reconfigurable Systems and Algorithms.

70. **Fabrizio Ferrandi, Luca Fossati, Marco Lattuada, Gianluca Palermo, Donatella Sciuto and Antonino Tumeo.** Automatic Parallelization of Sequential Specifications for Symmetric MPSoCs. *Embedded System Design: Topics, Techniques and Trends.* s.l. : Springer Boston.

71. *Multi-CPU/FPGA Platform Based Heterogeneous Multiprocessor Prototyping: New Challenges for Embedded Software Designers.* **Senouci, Benaoumeur, et al.** The 19th IEEE/IFIP International Symposium on Rapid System Prototyping, 2008. RSP '08. pp. 41-47.

72. *SoCrates - A multiprocessor SoC in 40 days.* **M. Collin, R. Haukilahti, R. Nikitovic, J. Adomat.** 2001. Proceedings of DATE'01 Conference.

73. **M. Collin, M. Nikitovic, R. Haukilahti.** *SoCrates - A Scalable Multiprocessor System On Chip.* Mälardalen University. 2001. Master Thesis in Computer Engineering.

74. *A Single Program Multiple Data Parallel Processing Platform for FPGAs.* **P. James-Roxby, P. Schumacher, C. Ross.** 2004. Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04).

75. *A Parallel MPEG-4 Encoder For FPGA Based Multiprocessor SOC.* **O. Lehtoranta, E. Salminen, A. Kulmala, M. Hannikainen, T.D. Hamalainen.** 2005. Proceedings of the International Conference on Field Programmable Logic and Applications. FPL'05. pp. 380-385.

76. *Impact of Shared Instruction Memory on Performance of FPGA-based MP-SoC Video Encoder.* **A. Kulmala, E. Salminen, O. Lehtoranta, T.D. Hämäläinen.** Design and Diagnostics of Electronic Circuits and Systems, 2006 IEEE.

77. *Design of Heterogeneous MPSoC on FPGA.* **Zhang, Wen-Ting, et al.** 2007. 7th International Conference on ASIC. ASICON'07. pp. 102-105.

78. *An FPGA-based soft multiprocessor system for IPv4 packet forwarding.* **Ravindran, K., et al.** International Conference on Field Programmable Logic and Applications, 2005. pp. 487-492.

79. **Hung, A.** *Cache Coherency for Symmetric Multiprocessor Systems on Programmable Chips.* University of Waterloo. 2004. Master Thesis in Electrical and Computer Engineering.

80. *Symmetric Multiprocessing on Programmable Chips Made Easy.* **A. Hung, W. Bishop, A. Kennings.** 2005. Proceedings of Design, Automation and Test in Europe Conference DATE'05. pp. 240-245.

81. *The Block Cipher Rijndael.* **J. Daemen, V. Rijmen.** s.l. : Springer-Verlag, 1998. Proceedings of the The International Conference on Smart Card Research and Applications. pp. 277-284.

82. *Rijndael, the advanced encryption standard,.* **Rijmen, J. Daemen and V.** 3, 2001, Dr. Dobb's Journal, Vol. 26, pp. 137-139.

83. **NIST.** *FIPS 197, Advanced Encryption Standard (AES).* U.S. Department of Commerce/National Institute of Standards and Technology. Technical Report.

84. *An Interrupt Controller for FPGA-based Multiprocessors.* **Tumeo, A. Branca, M. Camerini, L. Monchiero, M. Palermo, G. Ferrandi, F. Sciuto, D.** International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2007. IC-SAMOS 2007. pp. 82-87.

85. **Xilinx Inc.** Multi-CHannel OPB External Memory Controller (MCH OPB EMC) (v1.00a). [Online] http://www.xilinx.com/support/documentation/ip_documentation/mch_opb_emc.pdf.

86. **Xilinx Inc.** MCH OPB Double Data Rate (DDR) Synchronous DRAM (SDRAM) Controller. [Online] http://www.xilinx.com/support/documentation/ip_documentation/mch_opb_ddr.pdf.