Sistemas Informáticos
Curso 2004-2005

Desarrollo de una plataforma de emulación de sistemas empotrados multiprocesador

Pablo García del Valle
Fco. Javier García Flores
Esther Andrés Pérez

Dirigido por:
Prof. José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

Agradecimientos:

Una vez concluido este proyecto es imprescindible mostrar nuestro agradecimiento a aquellas personas que no sólo nos han prestado su apoyo en la realización del mismo, sino que también han hecho posible un ambiente inmejorable durante este año de trabajo.

En primer lugar, agradecer a David Atienza Alonso y a Miguel Peón Quirós toda su dedicación y esfuerzo como si de su propio trabajo se tratara así como todos los consejos que nos han dado, en todos los campos, no exclusivamente en lo que se refiere a la realización de este proyecto.

A continuación, agradecer sus continuas contribuciones y apoyo a Iván Magán Barroso, quien siempre ha estado en los momentos más críticos, ayudándonos a superar muchos de los obstáculos que nos hemos encontrado sin importarle el tiempo que dedicaba a ello.

También queremos reconocer que este proyecto no hubiera sido posible sin la colaboración del prof. José Manuel Mendías Cuadros, quien con su palpable entusiasmo por temas de investigación relacionados con el proyecto nos ha hecho involucrarnos en él con todo nuestro empeño e ilusión. También reconocer la importancia de los conceptos que nos ha enseñado y sin los que la realización de este trabajo hubiera sido inviable. Por último, agradecer la sencillez con la que siempre nos ha recibido y tratado.

Dar nuestro agradecimiento también al Departamento de Arquitectura de Computadores y Automática por facilitarnos todos los recursos que hemos necesitado en la realización de este proyecto.

Finalmente, agradecer a todas aquellas personas que nos han prestado su apoyo incondicional durante estos años de carrera, a nuestras familias y amigos sin cuya ayuda nada de esto hubiera sido posible, a Porfirio Noguera Arias por aguantar nuestras ruidosas reuniones mientras trabajaba, y a Carlos Roa Romero por tener siempre esa alegría y prestarnos su ayuda siempre que la hemos necesitado.

No querríamos finalizar sin destacar la acogida que hemos recibido durante este año por parte de todos los que integran la facultad. Fuera del trabajo realizado, hemos encontrado apoyo y comprensión en todos los ámbitos, y esto ha supuesto un hecho muy importante en nuestras vidas.

Los autores de este proyecto autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos no comerciales tanto la memoria, como el código, la documentación y el prototipo de la plataforma desarrollada.

Pablo García del Valle Esther Andrés Pérez Fco.Javier García Flores

Indice:

Lista de palabras clave

Emulación, Codiseño, Hardware, FPGA, VHDL, Memoria, Xilinx, EDK, IP core, sistemas empotrados.

Resumen en Castellano

En los últimos años han aparecido necesidades en la industria moderna y en la electrónica de consumo que no han podido ser satisfechas con los sistemas hardware o software habituales. La mayoría de estas aplicaciones vienen determinadas por el auge del sector multimedia y la tendencia a portabilizar sistemas cada vez más complejos.

Actualmente, resulta de vital importancia desarrollar una metodología de diseño capaz de abordar dicha complejidad dentro de las pequeñas ventanas de tiempo que permite la dinámica empresarial y las restricciones de mercado. Esta metodología de diseño debe soportar la realización rápida de prototipos de sistemas empotrados y tener en cuenta las últimas tendencias aparecidas, en especial la implementación bajo nuevas restricciones como es la minimización de consumo.

Si precisamente nos centramos en la minimización de consumo de energía, es necesario notar que la arquitectura del sistema de memoria de los sistemas empotrados tiene un gran impacto en el mismo.
Además, investigaciones recientes han demostrado la utilidad de pequeñas memorias SRAM adicionales cercanas al procesador y con control totalmente software (scratchpad) lo que ha producido la aparición de este tipo de memorias en todas las arquitecturas de última generación. A pesar de este importante soporte hardware, se ha demostrado que el diseño del subsistema de memoria en un sistema empotrado debe estar de acuerdo con el subconjunto específico de aplicaciones que se desea ejecutar ya que en caso contrario el rendimiento global puede degradarse severamente y, al mismo tiempo, ser causa de un grave derroche energético, que puede llegar hasta un 70% en algunas aplicaciones. [Ref. 2]

Por ello, se plantea en este proyecto el estudio de las necesidades de uso de memoria dinámica de los nuevos sistemas empotrados multimedia de altas prestaciones y para ello, la construcción de una plataforma de emulación hardware que implemente una jerarquía de memoria específica, que permita la extracción de diferentes estadísticas de acceso a los distintos niveles de la jerarquía de memoria propuesta, y la posterior obtención de conclusiones y resultados a partir de los datos obtenidos.

English Summary

In the last years, the requirements of consumer markets in embedded devices have put a lot of pressure in the semiconductor industry. Presently, due to the exponential rise in the silicon available in latest chips, these new consumer embedded systems must provide a large range of multimedia services at a very cheap price and with a very short time-to-market. Furthermore the boom in the multimedia sector to make portable systems adds even more complexity to this design flow. As a result, current design methodologies for the electronic world are not able to efficiently tackle such complex framework. Thus, new solutions need to be found to correctly design the hardware and software components involved in these forthcoming embedded systems. Precisely, it is vital to develop new versatile methodologies to prototype and evaluate new embedded designs, which can easily adjust to the latest tendencies and design metrics within thriving this domain, for instance the minimization of energy consumption.

Within this context, very recently it has been proven that a key issue to achieve the minimization of energy consumed is the correct design of the architecture and management of the memory hierarchy of embedded systems. For example, recent research in this field has shown the benefits of including small software-controlled memories near the processor (i.e. scratchpad memories), which has translated into the inclusion of this type of memory in the recent generations of embedded systems.

In addition, apart from adjusting the hardware of embedded systems, it has been shown that an optimal design of the memory system in a embedded system requires its tuning for the specific subset of applications desired to be executed. Otherwise, the performance can be degraded severely compared to an optimal design, and, at the same time, it can cause a serious waste of energy, namely, up to a 70% in some applications.

As a consequence of all the previous limitations, this project tries to aid designers of embedded systems by providing a complete hardware/software framework for the evaluation and management of the memory subsystem by the embedded software. From the hardware point of view, this framework is built on top of a Xilinx Virtex-II Pro FPGA board to allow the multiple re-synthesis of memory-specific hierarchies, which replace the memory system of the actual embedded system under study and the extraction of a wide range of statistics from the different levels of its memory hierarchy (i.e. scratchpad, data cache and main memory). From the software point of view, the framework includes

a complete tool-chain for the hard-core PowerPC processor present in the employed Virtex-II Pro board, which enables an easy compilation of C code of the different real-life applications to be included in the system. Hence, making feasible the evaluation of the memory hierarchy for each instance of possible input in the system, and the extraction of consistent conclusions according to the data obtained.

Capítulo 1. Introducción

La creciente complejidad de los sistemas electrónicos, que deben dar solución a problemas cada vez más costosos y restrictivos en multitud de aspectos, ha dado lugar a la integración de dos áreas muy dispares de la ingeniería: **el diseño hardware y el diseño software**. Estos condicionantes, que imponen las nuevas aplicaciones, extienden las restricciones tradicionales del diseño hardware como eran el área física del chip y la ejecución de tareas en plazos concretos, hacia novedosas limitaciones como son el consumo de energía, la ejecución en tiempo real o la capacidad de ser utilizados en dispositivos portátiles y económicos con complejidad suficiente para proporcionar el procesamiento requerido en el área de los sistemas empotrados.

En los últimos años han aparecido necesidades en la industria moderna y en la electrónica de consumo que no han podido ser satisfechas con los sistemas hardware o software habituales. La mayoría de estas aplicaciones vienen determinadas por el auge del sector multimedia (convergencia del audio, vídeo e interactividad con el usuario) y la tendencia a portabilizar sistemas cada vez más complejos (ordenadores personales, agendas electrónicas, etc).

Además, debemos destacar el sector del control inteligente, donde los sistemas de control distribuidos, multiusuario y de bajo precio demandan el diseño de esquemas complejos y eficientes que hasta hace poco carecían de viabilidad. Sin embargo, muchos de los avances conseguidos mediante esta conjunción de técnicas no se han visto acompañados de una mejora en los métodos y herramientas disponibles para el diseñador de dichas aplicaciones.

Cuando se habla de procesadores hoy en día, muchas personas piensan casi de forma intuitiva en procesadores de propósito general, aquellos que tienen montados en los ordenadores de casa o en grandes servidores o estaciones de trabajo. Aquellos fabricados por marcas tan conocidas como Intel, y se extienden por todo el mundo, dando soluciones a un amplio espectro de situaciones generales. Estos sistemas formados por dichos procesadores tan conocidos, parecen ser reconocidos de inmediato por la gran mayoría de las personas. Sin embargo hay otro concepto de procesador mucho más presente en nuestra vida cotidiana. Desde hace ya varios años, hemos experimentado un aumento espectacular en el campo de los procesadores empotrados, otro tipo de procesadores que se

encuentran montados en sistemas dedicados a una funcionalidad más o menos específica, con unas limitaciones y requisitos muy concretos. En la actualidad, el mercado de estos procesadores supera al de los procesadores de propósito general en varios órdenes de magnitud. Además se ha producido recientemente una gran evolución y se han creado nuevos horizontes en las prestaciones y capacidades de mercado de los sistemas empotrados que han pasado de ser simples dispositivos de control diseñados para realizar una función o un pequeño conjunto de funciones específicas en entornos más o menos reactivos, a ser sistemas complejos, con una funcionalidad comparable a los sistemas de sobremesa, pero con unos fuertes requisitos, sobre todo de consumo, que satisfacer.

1.1 Situación actual de los sistemas empotrados de altas prestaciones

El sector empresarial, y más concretamente las compañías relacionadas directamente con la producción de dispositivos programables, han potenciado el uso de las nuevas capacidades hardware de sus dispositivos con el desarrollo de sistemas de propósito general totalmente integrables en gran parte de las familias de dispositivos de bajo coste. Este tipo de sistemas, que se estructuran a partir de una descripción hardware realizada a través de alguno de los lenguajes disponibles para tal fin, como VHDL o verilog, reciben el nombre genérico de IP cores (*Intellectual Property Cores*). La naturaleza de su descripción brinda la posibilidad de dinamizar su diseño haciendo posible que un dispositivo pensado inicialmente para ofrecer soluciones genéricas a múltiples problemas pueda especializarse y dar soluciones a problemas muy concretos. No todos los IP cores ofrecidos por el mercado responden a la filosofía del diseño donde se arbitran soluciones de propósito general, también existen diseños específicos que aportan soluciones a problemas concretos (ethernet, buses, periféricos, etc…).

Hoy en día y con las capacidades que brindan los nuevos dispositivos programables, tales como FPGA's , no basta con integrar un IP core de propósito general. El auge del tratamiento digital de la señal debido a la aparición de microprocesadores especializados como los DSP's unido a su elevado coste ha aumentado la demanda de sistemas de procesamiento cada vez más rápidos y baratos. La respuesta del mercado ha sido integrar en los nuevos dispositivos sistemas empotrados completos reduciendo notablemente tiempos de accesos a datos y código y facilitando por tanto, tiempos de ejecución mucho menores.

1.1.1 Volumen de mercado

Para poder realizar un estudio de la situación actual del mercado en cuanto a ventas de microprocesadores, distinguiremos antes brevemente los tipos de éstos que podemos encontrar en los distintos dispositivos que nos rodean así como sus características a grandes rasgos:

- **Procesadores de propósito general** – alto rendimiento
    - Pentium, Alpha, SPARC
    - Ejecución de SW de propósito general
    - SO complejos - UNIX, NT
    - Ejemplos: Estaciones de trabajo, PC's
- **Procesadores empotrados** (Embedded)
    - ARM, Hitachi SH7000, NEC V800, PowerPC 405, Microblaze
    - Aplicaciones ligeras, tiempo real OS
    - Ejemplos: Teléfonos celulares, electrónica de consumo (reproductores de CD,….)
- **Microcontroladores**
    - Motorola familia 68HCxx
    - Muy sensible a coste
    - Tamaño de palabra pequeño – 8-16 bits
    - Alto volumen de producción
    - Integración de elementos analógicos y de control: UART, DAC, PWM, ADC …
    - Automóvil, electrodomésticos, consumo, etc..

La relación coste-rendimiento entre estos tipos de procesadores puede observarse en la siguiente gráfica:

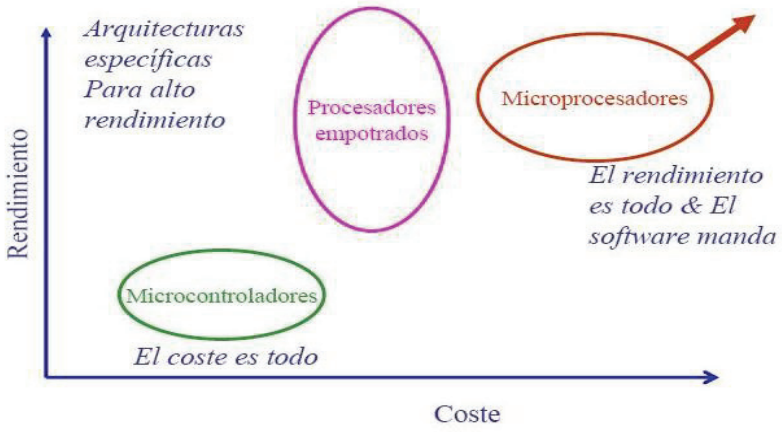


Figura nº 1: Relación coste-rendimiento

Para entender la situación del mercado actual en cuanto a los distintos tipos de procesadores observados anteriormente, se facilitan las siguientes gráficas (datos del año 2001):

- Tamaño del mercado:

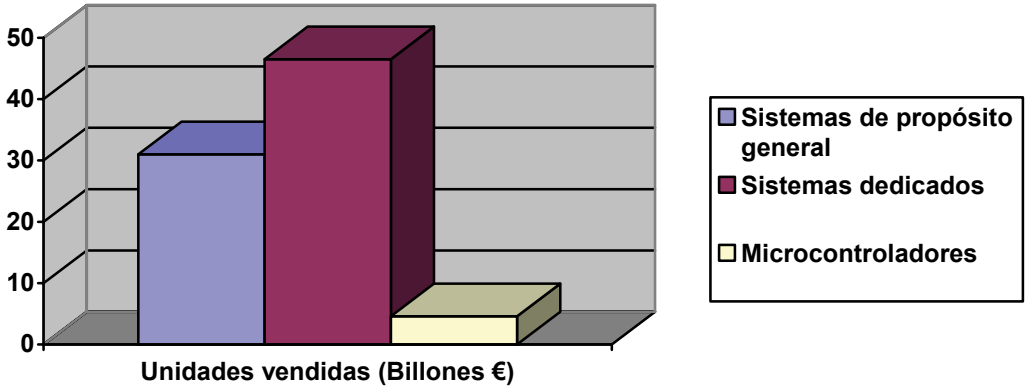


Figura nº 2: Tamaño del mercado en unidades vendidas

- Tasa de crecimiento anual (sistemas dedicados frente a sistemas de propósito general)

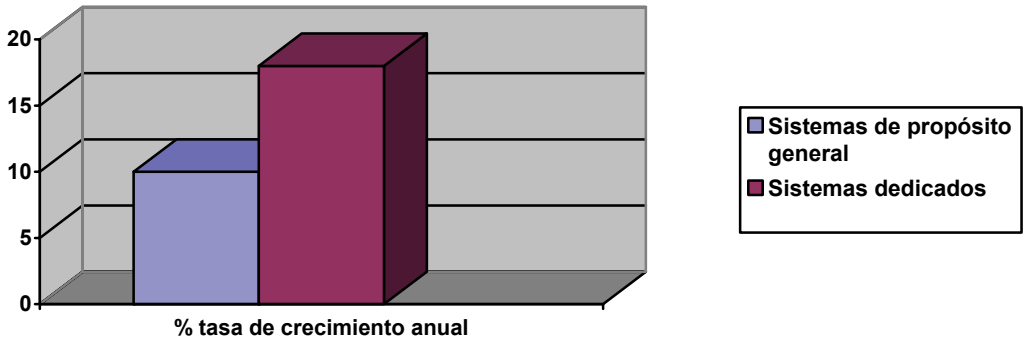


Figura nº 3: Tasa de crecimiento anual del mercado

La producción de sistemas dedicados hace tiempo que es mucho mayor que la producción de otros sistemas: 150 M unidades PC's frente a 7.5 B CPUs empotradas (80% 4 y 8 bits)

Otro dato significativo del mercado y que puede observarse en la siguiente gráfica es que el volumen de venta de procesadores empotrados de 8 bits es enorme y crece rápidamente.

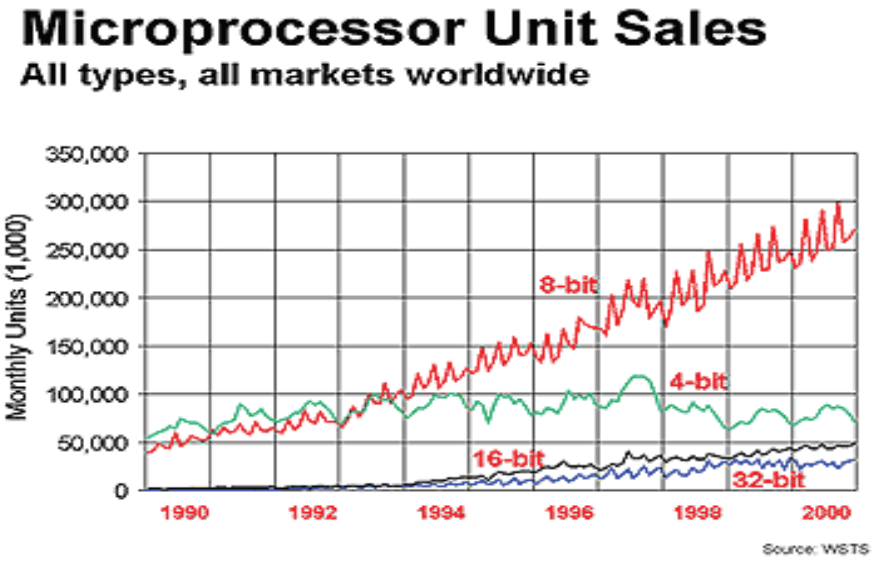


Figura nº 4: Ventas de microprocesadores de 4, 8, 16 y 32 bits

Se estima que cada dispositivo doméstico tiene 40 ó 50 microprocesadores empotrados en él de media. Hay un microprocesador en el microondas, en la lavadora, en la secadora, en el lavavajillas… Y no sólo eso, sino también en dispositivos de audio

vídeo tales como reproductores de DVD, CD… También están presentes en los vehículos en un número realmente abrumador.
Un nuevo coche tiene de media una docena de microprocesadores empotrados, y por poner ejemplos concretos, la serie 7 de BMW tiene 63 microprocesadores empotrados.
En conclusión, la mayoría de los aparatos electrónicos que nos rodean tiene uno ó más procesadores empotrados dedicados a la realización de distintas funciones.

1.1.2 Características especiales de los sistemas empotrados

Antes de enfocar el aspecto teórico y práctico de la plataforma de emulación definiremos lo que actualmente conocemos como un sistema empotrado. Definimos sistema empotrado de la siguiente forma: un sistema empotrado es aquel cuyo control está basado en un microprocesador/microcontrolador de propósito general, y dedicado a realizar una tarea o un conjunto de tareas específicas. Las características más sobresalientes que exigiremos al diseño de nuestro sistema empotrado (y que son una tónica general en el diseño de altas prestaciones de estos dispositivos) son:

- **Concurrencia:** Los componentes del sistema controlado o monitorizado funcionan simultáneamente.

- **Fiabilidad y seguridad:** El manejo de errores se puede hacer a través de "soporte hardware", si el tipo de dispositivo lo permite; o por "vía software", pero en este caso se podría decir que el sistema es menos robusto.
  En nuestro caso, las herramientas utilizadas permitirán depuración de errores mediante soporte hardware.

- **Reactividad y tiempo real:** Ciertos componentes del sistema empotrado deben reaccionar continuamente a los cambios en el ambiente del sistema, y deben computar ciertos resultados en tiempo real (determinado, acotado y predecible). Sobre todo es importante el alto grado de predictibilidad que se exige a este tipo de sistemas.

- **Interacción con dispositivos físicos:** Los sistemas empotrados interaccionan con su entorno mediante diversos tipos de dispositivos que normalmente no son convencionales: teclados, lectoras de datos, convertidores A/D y D/A, PWM, entradas y salidas digitales paralelo y serie, sensores, etc.

- **Robustez:** Es frecuente que este tipo de sistemas estén ubicados en sistemas con movimiento o que pueden ser transportados, sujetos a vibraciones e incluso impactos. No siempre trabajan en condiciones óptimas de temperatura, humedad, limpieza y uso correcto. Pese a ello, hay que garantizar el correcto funcionamiento en este tipo de situaciones.

- **Bajo consumo:** Las deficientes condiciones de ventilación generadas por la ubicación del dispositivo, así como la necesidad de operar con baterías, exigen reducir el consumo de energía y, por tanto, la disipación de potencia que acarrea el sobrecalentamiento de los componentes electrónicos.
  La reducción del consumo de energía es uno de los planteamientos más punteros en la actualidad. De hecho, este proyecto abordará como objetivo la reducción del consumo en los sistemas empotrados, atacando el problema en el punto de mayor importancia, que se encuentra en los accesos del procesador a la jerarquía de memoria.

- **Precio reducido:** Especialmente importante si se plantea la fabricación de gran número de unidades o se pretende sacar al mercado una versión comercial del sistema.

- **Pequeñas dimensiones:** Las dimensiones de un sistema empotrado no dependen sólo de sí mismo sino también del espacio disponible en el sistema que controla y/o monitoriza. Directamente relacionado con el consumo.

- **Diseño especial:** su diseño comprende el desarrollo conjunto de componentes tanto hardware como software
  - hardware ofrece rendimiento
  - software ofrece flexibilidad (para poder cambiar funcionalidad sin un coste excesivo de rediseño)

- **Flexibilidad:** es especialmente sensible a aspectos mercadotécnicos, y deben ser capaces de evolucionar con el mercado de una manera flexible y en un periodo limitado de tiempo.

Las métricas o características que acaban de ser reseñadas típicamente están relacionadas unas con otras, peor aún, compiten una con otra de modo que mejorar una implica la degradación de otras (como ejemplo claro tenemos que, cuando disminuimos el tamaño del

sistema empotrado, se puede estar mejorando en tamaño y peso, pero afectando de manera negativa la funcionalidad al no contar con algún soporte hardware de control de errores). A fin de optimizar estas métricas, el diseñador debe estar en conocimiento de una variedad de tecnologías de implementación hardware y software, con el objetivo de encontrar la mejor implementación para una aplicación y restricciones dadas. De este modo, el diseñador debe ser un experto tanto en el diseño software como en el diseño hardware, donde la validación de las metodologías mediante la selección apropiada de diversos casos de estudio se hace indispensable.

Un sistema empotrado realiza una única función o un pequeño conjunto de funciones específicas diferentes a las realizadas por un sistema de propósito general y fijas una vez que el producto está en el mercado. En los sistemas dedicados actuales el rango de aplicaciones es bastante extenso, pero siempre acotado. Normalmente, forma parte de un sistema mayor que no suele ser un computador como ocurre en el caso de los sistemas de propósito general.

Su funcionamiento suele ir ligado a entornos reactivos que imponen requisitos de tiempo más o menos estrictos y una fuerte penalización si no se cumplen los plazos.

Los requisitos de fiabilidad y seguridad en general son mucho más estrictos para sistemas empotrados que para otro tipo de sistemas basados en computador. Si, por ejemplo, un programa de cálculo científico falla, basta con abortar la ejecución del programa, resolver el error y a continuación lanzarlo de nuevo. En cambio, el sistema de control de una central nuclear, nunca debe dejar que el reactor llegue a una situación fuera de control, ya que esto puede provocar que el reactor se funda y emita una gran cantidad de radiación. Lo mismo ocurriría con el sistema de control de un avión, en este caso el sistema, en caso de fallo grave, debe permitir que el piloto sea lanzado en paracaídas antes de que el avión se estrelle.

Es importante notar que, estos tipos de sistemas deben satisfacer un conjunto estricto de ligaduras consumo, disipación, coste, tamaño, etc, tal y como se ha desplegado anteriormente.
El rendimiento es importante, pero hasta un cierto grado.

La mayoría de los procesadores que se emplean en el diseño de los sistemas empotrados existentes en la actualidad se pueden clasificar de la siguiente forma:

Procesadores de 4 bits: todavía existen procesadores de 4 bits funcionando en sistemas muy sencillos.

Procesadores de 8 bits: los desarrolladores de sistemas empotrados todavía los emplean. Destacan los microcontroladores 8051de Intel, el Zilog Z180 (descendiente del Z80) y los de la familia 68xx de Motorola

***Procesadores de 16 bits*:** destacan los procesadores 80188, 80186 y 8096 de Intel, y el 68HC11 de Motorola (muy empleado en el campo de la robótica).

***Procesadores de 32 bits*:** la oferta dentro de los 32 bits es muy amplia debido a la reducción en los costes del *hardware* y al crecimiento de la complejidad de los sistemas empotrados (que requieren mayor potencia). Ejemplos son los x86 de Intel, el Am29K de AMD, el SH RISC de Hitachi, los PowerPC de IBM y Motorola, y los MIPS de Silicon Graphics.

Una comparación rendimiento/coste se puede observar en la siguiente figura:

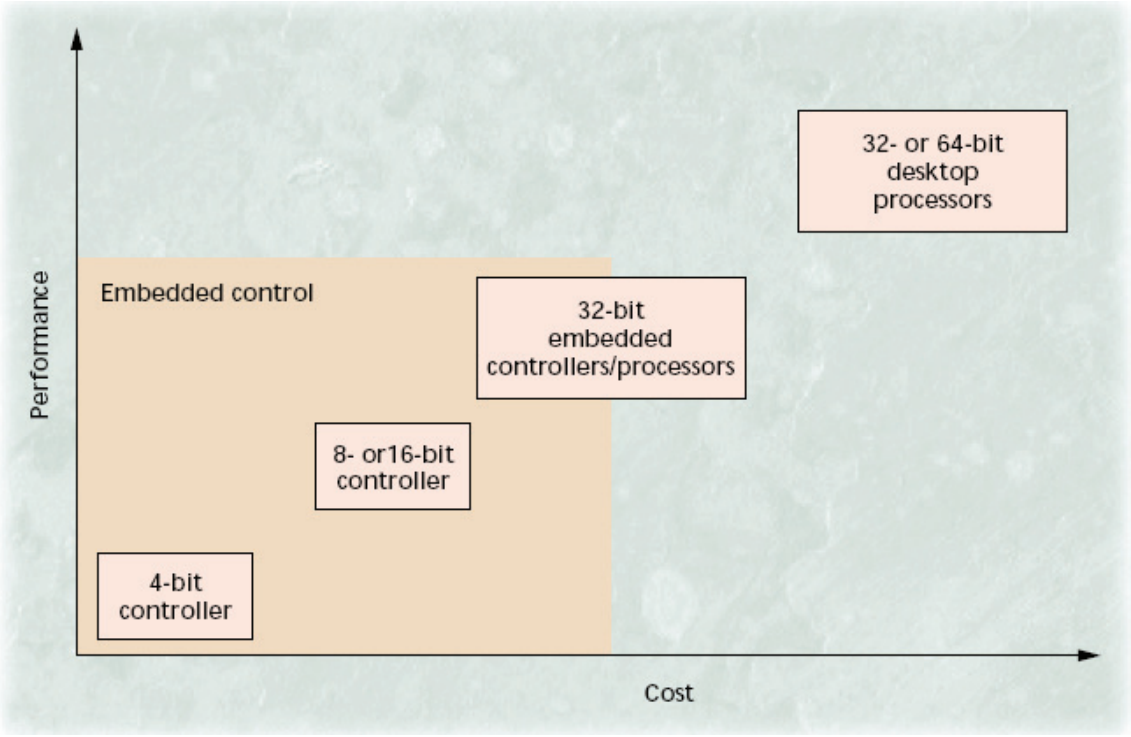


Figura nº 5: Rendimiento-coste procesadores 4,8,16,32 y 64 bits

Aparte del microprocesador, un sistema empotrado tiene otros componentes. Dichos componentes típicos de un sistema empotrado pueden verse en la figura:

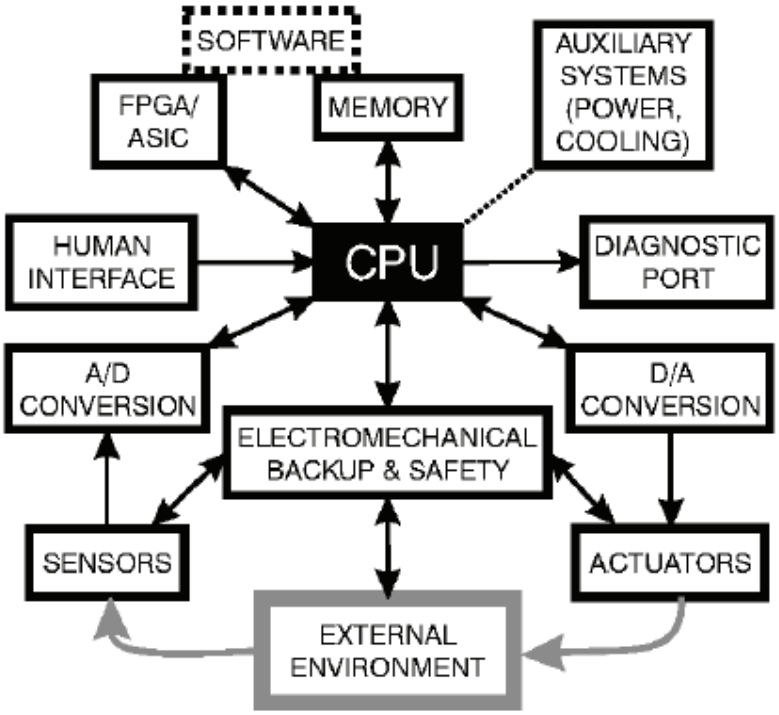


Figura nº 6: Componentes típicos de un sistema empotrado

Entre los componentes más significativos de un sistema empotrado típico cabe destacar la memoria, el puerto de diagnóstico para operaciones de depuración del sistema, y la FPGA como dispositivo reprogramable donde se pueden sintetizar nuevos módulos que realicen funciones nuevas y específicas.

1.1.3 Restricciones y limitaciones en diseño e implementación

Los diseños de los nuevos sistemas empotrados deben ser capaces de suministrar unos ciertos servicios (como por ejemplo, videoconferencia, reproducción de música o vídeo, juegos 3D…) que implican que deben satisfacer un conjunto de ligaduras de diseño/implementación y características comunes que los distinguen de los otros sistemas de computación más generales.

Estas restricciones son debidas principalmente a requisitos específicos de características tales como: energía, rendimiento, tamaño peso…
Si llegamos aún más lejos, no es costoso darse cuenta de que en los nuevos sistemas empotrados, la disminución de la energía consumida tiene tanto peso (o incluso más) que el rendimiento requerido en el sistema, ya que dicho rendimiento debe proporcionar un tiempo de respuesta realista para los usuarios, siempre manteniendo una relación energía/potencia adecuada para el sistema.

1.2 Arquitectura general de los sistemas empotrados de altas prestaciones

Dos componentes fundamentales forman un sistema empotrado de altas prestaciones, y ambas deben estar en perfecta sintonía para conseguir satisfacer los estrictos requisitos que se impongan. Las peculiaridades de cada componente se explican a continuación de forma general.

1.2.1 Componente hardware

La complejidad de diseñar un sistema dedicado de nueva generación comenzando desde cero e intentando rediseñar y optimizar globalmente todos los módulos necesarios para cada estándar, es un trabajo que cada vez se vuelve más inviable. Por ello, actualmente la única alternativa que se considera viable para afrontar el diseño de sistemas heterogéneos de complejidad creciente es mediante la aplicación del **nuevo paradigma de sistemas "on-Chip",** cuya concepción principal es el diseño del sistema global mediante composición y reutilización de módulos o componentes de **Propiedad Intelectual (IPs)** diseñados independientemente.

Con objetivo de reducir el consumo de energía y aumentar al mismo tiempo el rendimiento del sistema, el subsistema de memoria actual de los sistemas dedicados de altas prestaciones incluye una estructura multinivel con al menos un nivel de cache (cache L1 para datos e instrucciones) y una memoria principal compartida (por los distintos procesadores que formen el sistema empotrado).

A pesar de este importante soporte hardware, en los últimos años se ha demostrado que el diseño del subsistema de memoria en un sistema empotrado debe estar de acuerdo con el subconjunto específico de

aplicaciones que se desea ejecutar ya que en caso contrario el rendimiento global puede degradarse severamente y, al mismo tiempo, ser causa de un grave derroche energético. Es necesario destacar que **más de un 40% de la potencia total se consume en el subsistema de memoria**, cantidad que se puede incrementar hasta el 70% en ciertos tipos de aplicaciones (como las de renderizado de imágenes).

Debido a la importancia del subsistema de memoria, el problema de su optimización ha sido considerado desde hace más de una década uno de los problemas más importantes dentro del entorno de sistemas empotrados de alto rendimiento

1.2.2 Componente software

En los nuevos sistemas dedicados, la componente software tiene un peso muy significativo desde el punto de vista del éxito comercial del producto final y de la reducción de costes de diseño en un nuevo sistema.

La componente software debe ser capaz de utilizar eficientemente las optimizaciones que la componente hardware pone a disposición de los diseñadores de aplicaciones empotradas para lograr disminuir el consumo de energía del sistema y aumentar su rendimiento. Por ello debe ser perfecta la sintonía entre componente hardware y software y de esta forma permitir aprovechar cada característica de cada componente para reducir al máximo el consumo del sistema.

1.3 Necesidad de nuevas técnicas de diseño

La complejidad de los diseños electrónicos actuales no puede ser satisfecha por soluciones sólo hardware o sólo software, por lo que aparece la convergencia de ambos planteamientos en un nuevo paradigma de diseño, el **codiseño hardware-software**. Por tanto, la implementación de sistemas empotrados debe estructurar el empleo de ambas metodologías en un nuevo flujo de diseño que distribuya y comunique los componentes hardware y software de forma eficiente para alcanzar la funcionalidad especificada y satisfaciendo las restricciones de tiempo de desarrollo, coste, consumo y prestaciones típicas de la industria moderna.

Todos somos conscientes de que cuando un cierto producto pasa a estar de moda (por ejemplo: teléfonos con videoconferencia, ordenadores de bolsillo…) y a alguna empresa no le ha dado tiempo a desarrollar su producto para ese momento y sufre un retraso de varios meses, cuando el producto llegue al mercado puede tener unas pérdidas muy importantes respecto a sus expectativas de venta iniciales.

Además, algunos estudios han demostrado que las pérdidas en la ganancia total de un producto se ven más afectadas por su aparición tardía en el mercado que por el incremento de su coste total de fabricación y consecuente precio de venta. En este contexto, se hace evidente la necesidad de metodologías y herramientas de diseño e implementación que permitan el desarrollo de los nuevos sistemas dedicados multimedia de altas prestaciones de manera que se logre acortar al máximo su tiempo de desarrollo y, a la vez, que aseguren la corrección del diseño.

Por otra parte, la complejidad de diseñar un sistema dedicado de nueva generación comenzando desde cero e intentando rediseñar y optimizar globalmente todos los módulos necesarios para cada estándar, es un trabajo que cada vez se vuelve más complejo. Por ello, actualmente la única alternativa que se considera es mediante la aplicación del nuevo paradigma de sistemas "on-chip", cuyo objetivo es el diseño del sistema global mediante composición y reutilización de módulos o componentes de Propiedad Intelectual (IPs) diseñados independientemente.

Actualmente, resulta de vital importancia desarrollar una metodología de diseño capaz de abordar dicha complejidad dentro de las pequeñas ventanas de tiempo que permite la dinámica empresarial y las restricciones de mercado. Esta metodología de diseño debería soportar la realización rápida y versátil de prototipos de sistemas empotrados digitales y tener en cuenta las últimas tendencias aparecidas, en especial la implementación bajo nuevas restricciones como es la minimización de consumo.

En la actualidad, se están realizando esfuerzos considerables en la automatización del ciclo de diseño de sistemas empotrados.
Es necesario un conocimiento profundo del flujo de trabajo para poder manejar convenientemente las decisiones que se le plantean al ingeniero al abordar las diversas restricciones del sistema.

1.3.1 Situación actual de las herramientas

Las nuevas metodologías de optimización de sistemas dedicados implican un cambio de mentalidad a la hora de diseñar sistemas empotrados ya que cobra vital importancia, por primera vez en el dominio de los sistemas empotrados, el concepto de codiseño hardware/software.
Debido a las dependencias e interacciones de las dos componentes de los sistemas dedicados de altas prestaciones (hardware y software), las nuevas metodologías de diseño para dichos sistemas poseen un papel fundamental.
Hay que cambiar el punto de vista y adquirir una visión global del sistema, de forma que se relacionen entre sí, y que se aprovechen de las características específicas de diseño.

1.3.2 Emulación vs. Simulación

Actualmente existe un espectro amplio de entornos de simulación en el ámbito académico (Dinero IV, MPARM, SimpleScalar) y también en el ámbito comercial que presentan distintos grados de precisión y velocidad de simulación.

Aun así no son suficientes y es posible afirmar que existe una gran necesidad de mecanismos eficientes de evaluación de soluciones que utilicen modelos detallados de las arquitecturas hardware/software de los sistemas dedicados de altas prestaciones.
Dentro de estos mecanismos eficientes de evaluación de soluciones, el uso de la emulación hardware sobre plataformas reconfigurables de bajo coste (como son las FPGAs) se presenta como una alternativa muy prometedora, ya que permite un gran aumento en la velocidad de evaluación de soluciones con respecto a entornos eficientes de simulación a nivel RT.

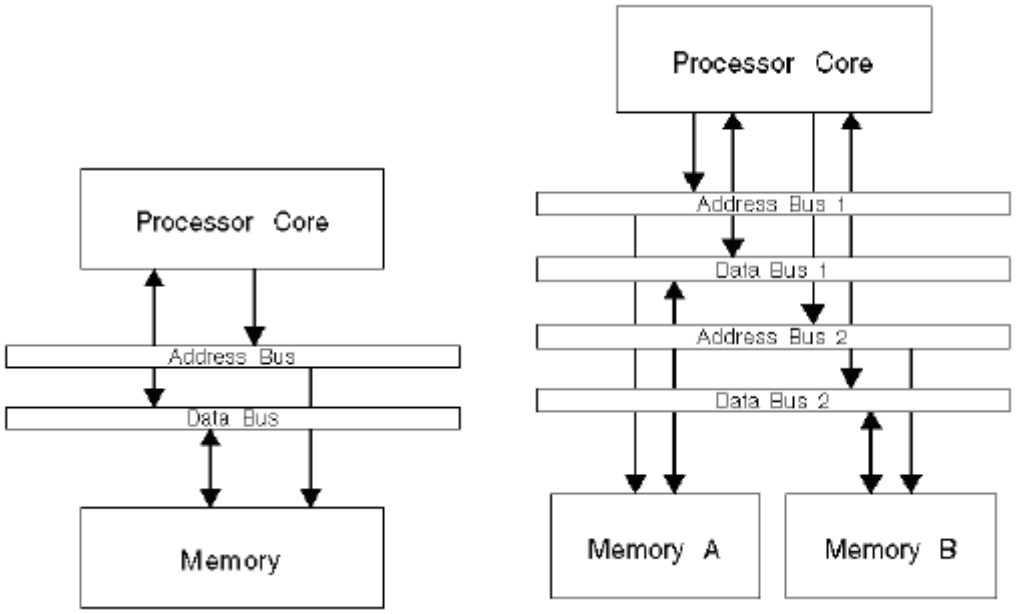
Más aún, la **emulación hardware** todavía adquiere más interés en el contexto de la investigación sobre la gestión de memoria dinámica en entornos multiprocesador, ya que permite mantener la precisión de la evaluación de soluciones a nivel de ciclo de reloj en estos entornos tan complejos, pero sin apenas pérdida de velocidad respecto al caso monoprocesador, lo que no es posible en los simuladores de precisión RT, ya que su rendimiento se deteriora de una manera dramática a

medida que se incrementa el número de procesadores y, por tanto, de señales que deben mantener sincronizadas al mismo tiempo dentro de la misma simulación.

1.3.3 Objetivo del proyecto

La arquitectura del sistema de memoria de los sistemas empotrados tiene un gran impacto en el rendimiento. Es por esta razón por la que los procesadores empotrados tienen una arquitectura del sistema de memoria diferente a la de los procesadores de propósito general.

Como puede observarse en las siguientes figuras, un procesador de propósito general accede a una memoria simple para programa y datos (arquitectura Von Neumman) y de esta forma el procesador no puede leer ni escribir datos mientras que una instrucción está siendo buscada.
Para mejorar este inconveniente, búsqueda de instrucción y lectura / escritura en el mismo ciclo, los procesadores empotrados implementan una arquitectura de memoria Harvard, donde la memoria se divide en dos partes: memoria de instrucciones o de programa y memoria de datos.



Arquitectura Von Neumman Arquitectura Harvard

Figura nº 7: Tipos de arquitecturas de memoria

Además, los procesadores empotrados usualmente vienen con memoria on-chip, bien ROM o bien RAM o ambas. El tamaño de esta memoria on-chip no suele ser muy grande, puesto que en el mercado de los sistemas empotrados, las necesidades de almacenamiento on-chip son pequeñas. En la siguiente gráfica observamos la diferencia de rendimiento que se produce cuando se insertan memorias cache de pequeño tamaño integradas en el propio chip del procesador:

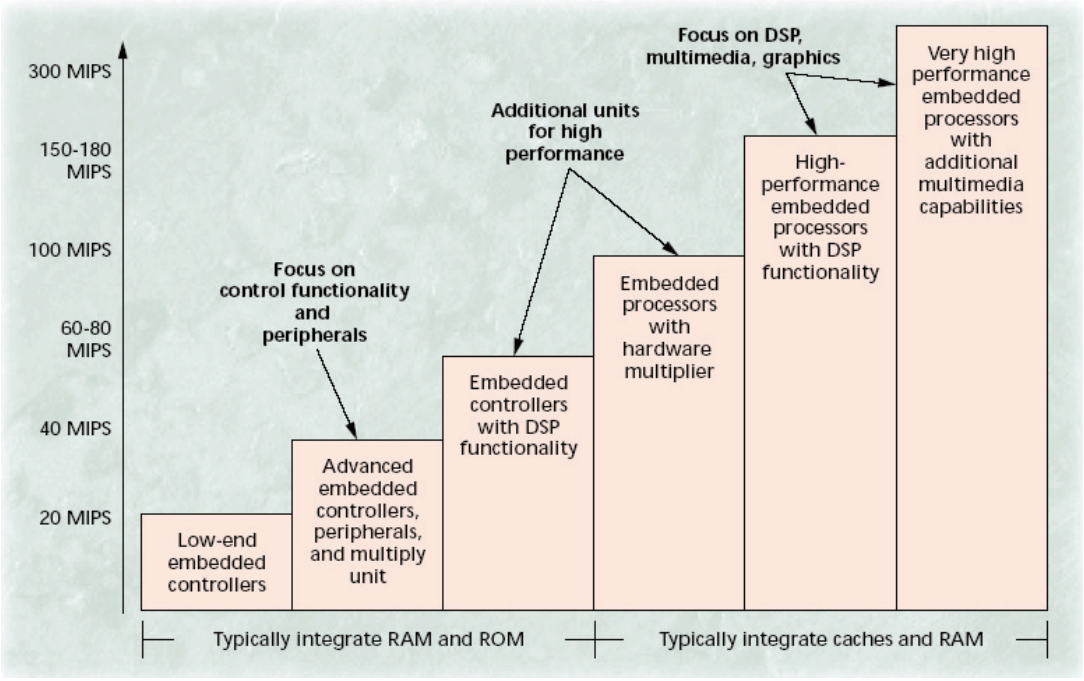


Figura nº 8: Rendimiento en relación con la memoria

Con objetivo de reducir el consumo de energía y aumentar al mismo tiempo el rendimiento del sistema, el subsistema de memoria actual de los sistemas dedicados de altas prestaciones incluye una estructura multinivel con al menos un nivel de cache (cache L1 para datos e instrucciones) y una memoria principal compartida. Incluso, en sistemas dedicados recientes, hay una clara tendencia a ampliar el número de niveles de la jerarquía con un segundo nivel de cache compartido para datos e instrucciones (cache L2). Además, investigación muy reciente ha demostrado la utilidad de pequeñas memorias SRAM adicionales (16, 32 ó 64 KB) cercanas al procesador y con control totalmente software (scratchpad) lo que ha producido la aparición de este tipo de memorias en todas las arquitecturas de última generación.

A pesar de este importante soporte hardware, en los últimos años se ha demostrado que el diseño del subsistema de memoria en un sistema empotrado debe estar de acuerdo con el subconjunto específico de aplicaciones que se desea ejecutar ya que en caso contrario el rendimiento global puede degradarse severamente y, al mismo tiempo, ser causa de un grave derroche energético.

Por ello, el **objetivo de este proyecto** de Sistemas Informáticos es el estudio de las necesidades de uso de memoria dinámica de los nuevos sistemas empotrados multimedia de altas prestaciones y para ello, la construcción de una plataforma de emulación hardware que permita la extracción de diferentes estadísticas de acceso a los distintos niveles de la jerarquía de memoria propuesta, y la posterior obtención de conclusiones y resultados a partir de los datos obtenidos.

1.3.4 Componentes y herramientas utilizadas

En la realización de este proyecto se han utilizado tanto herramientas y entornos software como sistemas hardware:

Hardware:

Virtex-II Pro XC2VP30-xFF896 fabricada por AVNET

- 2 PPCs empotrados
- FPGA Xilinx XC2VP30
- Micron DDR SDRAM 128 MB expandible hasta 1GB
- Micron Mobile SDRAM 32 MB
- Cypress asynchronous SRAM 2 MB
- Intel StrataFlash 16 MB
- Compact FLASH card
- Ethernet connection 10/100/1000 MBit/s
- PCI connector used with PCI-X core
- Puerto VGA

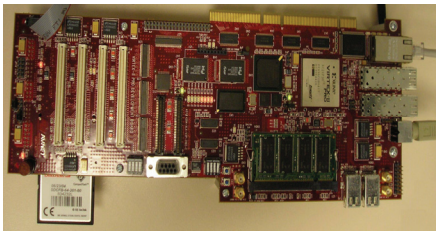


Figura nº 9: Imagen de la placa utilizada

Software:

Xilinx es una compañía líder en tecnología hardware basada en FPGAs. Adicionalmente, Xilinx proporciona distintos paquetes de herramientas de diseño para la gama de FPGAs que fabrica y comercializa. Dichas herramientas ofrecen un amplio abanico de posibilidades de optimización del diseño.
En este proyecto, utilizamos las herramientas de síntesis e implementación que se integran en el paquete ISE v. 6. Por otro lado, empleamos como herramienta de simulación el programa ModelSim, propiedad de Mentor Graphics, que se integra de un modo eficiente y sencillo con el ISE.

En resumen, las herramientas que utilizaremos más frecuentemente son:

- Xilinx EDK v6.3
- Xilinx ISE v6.3
- ModelSim v7.0
- Xilinx ChipScope v6.3
- Xilinx CoreGen v6.3

La herramienta EDK trabaja apoyándose en una arquitectura denominada *Embedded System Tools*, EST, que integra una serie de herramientas en una plataforma denominada *Xilinx Platform Studio*, XPS. La plataforma de estudio de Xilinx para sistemas empotrados, XPS, está fundamentada en cuatro herramientas básicas presentadas a continuación:

- **Herramientas de adaptación Hardware**.

Basadas en una plataforma generadora de hardware para procesadores MicroBlaze o PowerPC, utiliza como elemento principal la herramienta PlatGen. Admite como entrada un archivo de especificación hardware de extensión MHS (*Microprocessor Hardware Specification*). El usuario define la arquitectura del sistema a través de la creación del archivo MHS.

- **Herramientas de adaptación Software**.

Basadas en una plataforma generadora de bibliotecas y herramientas de compilación pertenecientes al proyecto GNU. LibGen es la herramienta encargada de configurar las bibliotecas y drivers de los elementos creados en el sistema empotrado basándose en el archivo

de especificación software MSS (*Microprocessor Software Specification*). A través de la creación del archivo MSS, el usuario especifica los dispositivos estándar de entrada/salida del sistema, las rutinas de atención a las interrupciones y otras características relacionadas con el software. También se incluye una herramienta GNU para compilar, enlazar y ensamblar (mb-gcc).

- **Herramientas de Simulación**.

La herramienta *Simulation Model Generator*, SimGen, crea y configura diferentes modelos de simulación en función de las especificaciones señaladas en el archivo de verificación MVS (*Microprocessor Verification Specification*). A través de este archivo el usuario puede determinar el lenguaje de descripción hardware de salida para los modelos de simulación. Cada ejemplar hardware del diseño puede simularse bajo un modelo concreto de simulación; además SimGen proporciona modelos de simulación y compilación para simuladores comerciales.
También se usa la herramienta de simulación ModelSim que se integra con el entorno de Xilinx ISE y permite realizar la simulación a diferentes niveles.

- **Herramientas de Depuración**.

La plataforma XPS incluye dos herramientas para la depuración del sistema empotrado: *GNU Debugger*, GDB, y *Xilinx Microprocessor Debug*, o también XMD. Al igual que ocurriera con el compilador, GDB está acogida al proyecto GNU.
Proporciona una interfaz unificada para la depuración y verificación de sistemas MicroBlaze y PowerPC varias fases de desarrollo del diseño. XMD ofrece al usuario la posibilidad de elegir la plataforma de depuración entre una tarjeta hardware o un simulador *Cycle Accurate Instruction Set Simulator* (ISS).

Describimos a continuación los pasos efectuados y la metodología seguida para la realización de módulos o periféricos nuevos de la plataforma de emulación.

1.3.5 Creación de un nuevo periférico

La creación de un periférico se desarrolla en varias fases bien definidas, y para ello es necesario utilizar un sintetizador VHDL junto con otras herramientas incluidas en la herramienta EDK

1. Descripción Lógica del Periférico

Esta primera fase consiste en describir el periférico en un lenguaje de descripción de hardware (VHDL en este caso). Dependiendo de la forma de comunicación y la velocidad, se elegirá un tipo de *bus* al que se conectará el periférico.
Para algunas de estas conexiones a los buses, *Xilinx* ofrece un *soft-macro* que maneja el protocolo necesario para la comunicación con el procesador. Estos bloques, denominados IPIF, ahorran al diseñador el estudio detallado de los diferentes protocolos.

2. Compilación del Periférico

La inclusión de un periférico en la biblioteca se realiza a través de una aplicación, denominada *Import Peripheral Wizard*, incluida en EDK, que se encarga de realizar una compilación del diseño. Las bibliotecas que se han utilizado en el diseño deben ser especificadas a esta herramienta. Esta operación se puede realizar seleccionando directamente las bibliotecas en el menú de esta aplicación o bien mediante un archivo con extensión .pao (*peripheral analyze order*). Una vez compilado el diseño satisfactoriamente, se asignan las señales de los *buses* con las del periférico y posteriormente se especifican las diferentes opciones de configuración del periférico. Para almacenar los valores por defecto de estas opciones se requiere la creación de un archivo .mpd (*microprocessor peripheral definition*). Una vez compilado, EDK incluye al periférico dentro de la lista de posibles bloques a utilizar por procesador en cuestión y solo habrá que instanciarlo para incluirlo en el diseño.

Capítulo 2. Arquitectura global de la plataforma de emulación del subsistema de memoria

2.1 Características y requisitos

La arquitectura objetivo que se propone para la implementación de la plataforma de emulación deberá respetar las siguientes características desarrolladas a continuación:

- **Modular.**

Debe ser una arquitectura modular en el mismo sentido que implica la propia metodología IP. No se parte desde cero para construir la plataforma sino que se reutilizan módulos existentes y se diseñan e implementan otros. En el capítulo cinco se describen con alto nivel de detalle tanto los módulos diseñados y sintetizados desde cero, como aquellos módulos IP que han sido reutilizados en la implementación de la plataforma.

- **Flexible.**

La plataforma implementada deberá flexible y permitir a un diseñador incluir un nuevo módulo en ella sin ninguna complicación.

- **Escalable.**

El número de procesadores de la plataforma es escalable.

- **Observable.**

La arquitectura permitirá la extracción de distintas estadísticas mediante la adición de módulos independientes que no interfieran en el comportamiento del sistema de procesamiento.
Esta característica será desarrollada con mayor nivel de detalle en los apartados posteriores.

2.2 Arquitectura objetivo

La arquitectura objetivo a la que se tenderá en el desarrollo de la plataforma de emulación del subsistema de memoria de procesadores empotrados de altas prestaciones estará formada por dos componentes principales, cumpliendo con ello las características generales de la mayoría de los sistemas empotrados:

- Componente hardware:

Se ha considerado un modelo general donde existe un procesador dedicado que ejecuta un reducido número de aplicaciones independientes, y una jerarquía completa de memoria formada por módulos integrados *on-chip* y otros externos *off-chip (en una versión ampliada de la plataforma de emulación)*. La elección de una arquitectura con un solo procesador es fácilmente extrapolable a los sistemas dedicados futuros en los que haya varios procesadores, cada uno de ellos ejecutando varias aplicaciones independientes y donde no existan problemas de acceso concurrente a datos dinámicos compartidos entre distintos procesadores.

En una primera versión de la plataforma de emulación se consideran dos procesadores empotrados. Uno de ellos se dedica al procesamiento, y el otro de ellos se dedica exclusivamente a la extracción de estadísticas.

El esquema general de la arquitectura desarrollada es el siguiente:

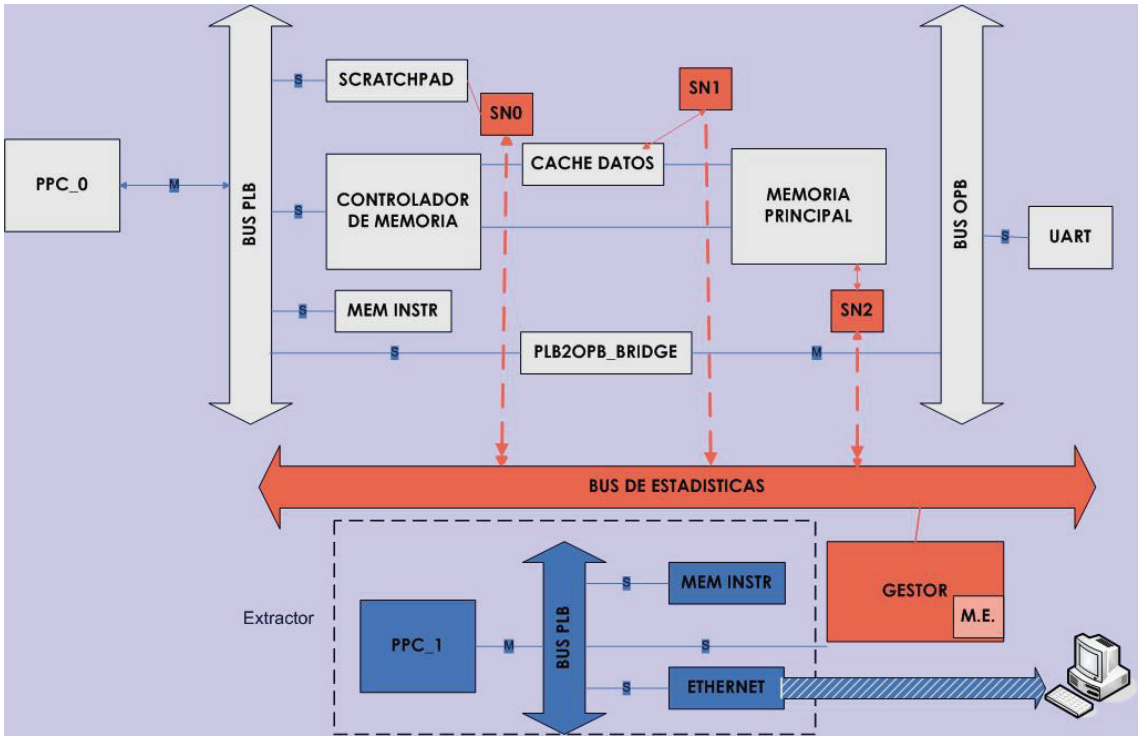


Figura nº 10: Visión global de la arquitectura

Según se observa en este esquema, podemos distinguir tres partes en la arquitectura global de la plataforma de emulación:

- La primera parte (en color gris), llamada a lo largo de este documento subsistema de procesamiento, es la encargada de implementar la jerarquía de memoria de un procesador empotrado. Consta de un procesador Power PC 405, que se encuentra empotrado físicamente en la placa utilizada. Este procesador es maestro del bus PLB (Processor local bus) al cual se conecta la nueva jerarquía de memoria diseñada.
  Esta jerarquía de memoria está compuesta por una serie de memorias de pequeño tamaño.

  La primera de ellas es la memoria **scratchpad,** cuya finalidad es almacenar datos mediante un control totalmente software, es decir, es el programador quien decide mandar un dato determinado a la memoria scratchpad en previsión de un uso muy frecuente. Este hecho afianza el concepto de la gran sintonía que ha de haber entre el software y el hardware. Esta memoria se caracteriza por permitir unos tiempos de acceso realmente rápidos.
  Además, las memorias scratchpad tienen mejor comportamiento en general que las memorias caches: menor consumo de en energía, mayor rendimiento y más predecibles.

  A continuación, en el esquema se puede ver el controlador de memoria, que es un módulo encargado de gestionar las peticiones de lectura y escritura del procesador, y decidir si se accede a memoria cache o bien a memoria principal.

  Por último, el programa software que ejecuta el procesador se encuentra almacenado en la memoria de instrucciones, que se implementa usando la tecnología BRAM.

  Aparte del diseño específico de la jerarquía de memoria, se incorpora al sistema un bus opb y un módulo que permita el control del puerto serie (uart) con el fin de observar cierto funcionamiento en el computador.

  Para realizar el paso de información entre los dos tipos de buses utilizados en el sistema se utiliza un IP de Xilinx llamado plb2opb_bridge que hace de puente entre ambos buses.

- La segunda parte del sistema (en color rojo en el esquema superior), se encarga de la obtención de estadísticas del subsistema de procesamiento. Para ello se incorporan módulos sniffers en aquellas partes del sistema de procesamiento que se quieran espiar y estos módulos enviarán información al bus de estadísticas, que realizará

funciones de sincronización y arbitraje. Este bus es de diseño propio con el fin de satisfacer los requerimientos de tiempo y de no interferencia que se impone al subsistema de extracción de estadísticas.
Los datos enviados por los sniffers a través del bus son recogidos por el módulo gestor, el cual lo almacena en una memoria interna dedicada.

- La tercera y última parte (color azul) es precisamente la encargada de extraer los datos estadísticos de la memoria interna dedicada del gestor. Además de extraer los datos de dicha memoria, construye datagramas UDP y los envía por Ethernet al computador para su posterior estudio. La construcción de los datagramas y el envío de los paquetes por ethernet se realiza mediante software, y para ejecutar dicho software se dedica un procesador Power PC 405 (distinto del anterior), también empotrado físicamente en la placa.

Dentro de la componente hardware, se implementarán módulos en el propio chip, y también se podrá controlar módulos externos tal y como se observa en la siguiente figura:

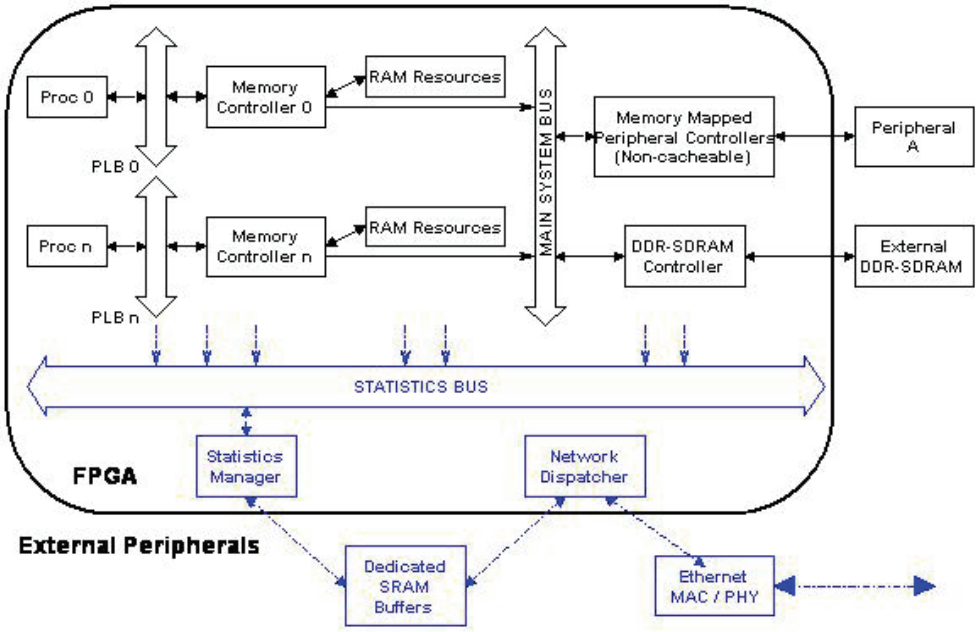


Figura nº 11: Diferentes recursos externos e internos de la placa

- Componente software:

El modelo de arquitectura del software ejecutado por los sistemas dedicados de altas prestaciones también ha aumentado su complejidad en comparación a los sistemas dedicados tradicionales y está formado por dos componentes principales: aplicaciones de usuario y sistema operativo en tiempo real.

En la plataforma tendremos dos conjuntos de programas, los programas software que se ejecutan en el subsistema de procesamiento y los que se ejecutan en la parte del subsistema de estadísticas.
En los siguientes apartados se describirá con mayor precisión cada uno de los tipos de programas implementados.

Además de estos dos conjuntos de programas software ejecutados en procesadores empotrados en la placa utilizada, existe otro tipo de programa que se ejecuta en el computador conectado.
Estos programas diseñados en lenguaje de alto nivel y utilizando para ello entornos de desarrollo actuales como Visual Studio .NET, permiten realizar tareas de análisis de los datos obtenidos y generación de informes.

A continuación, los dos capítulos siguientes, explicarán con más detalle cada uno de los dos subsistemas de importancia en la plataforma.

Capítulo 3. Arquitectura del subsistema de procesamiento

3.1 Componente Hardware

El sistema que emulamos es una arquitectura de tipo Harvard con el mismo bus tanto para datos como para instrucciones por su mayor sencillez a la hora de sacar estadísticas del mismo.

Hemos diseñado un sistema de memoria en el que discriminamos la memoria de instrucciones de la de datos, dando una mayor importancia a esta segunda por existir un mayor interés en el estudio de los datos y del impacto que tienen las distintas posibilidades de organizarlos.

El esquema general de la arquitectura del subsistema de procesamiento es el siguiente:

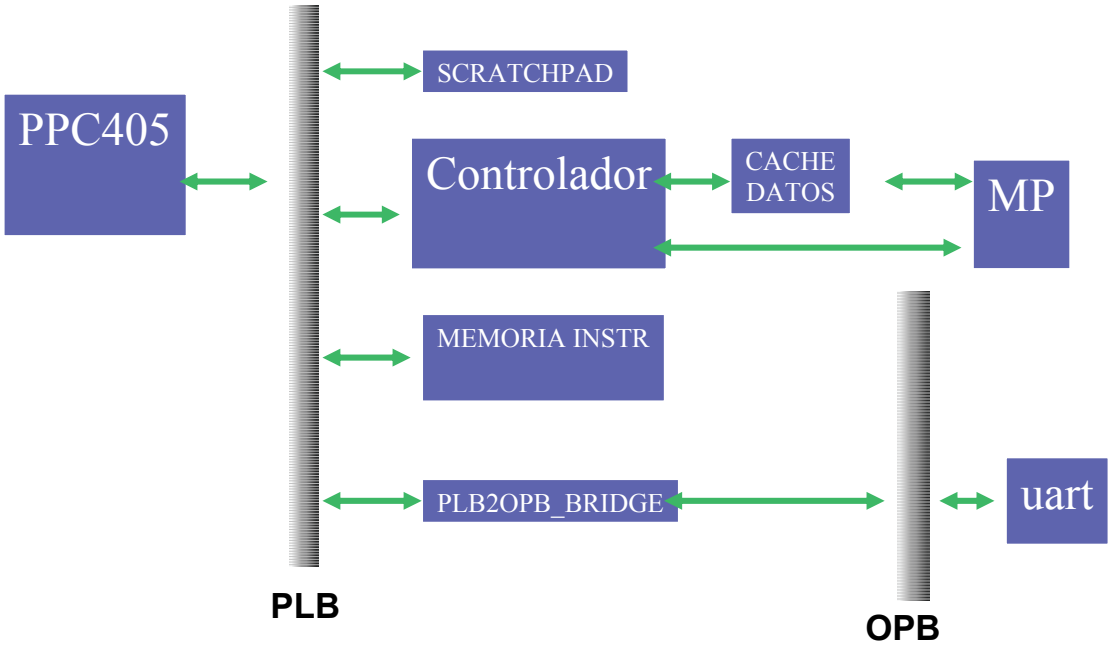


Figura nº 12: Arquitectura subsistema procesamiento

A continuación se explicarán los distintos componentes de la jerarquía de memoria utilizados en la plataforma:

Cache:

En este primer caso existen pequeñas memorias *cache* separadas para datos e instrucciones, locales al procesador e implementadas con tecnología BRAM. Se implementa una cache directa con escritura directa y sin asignación en escritura.

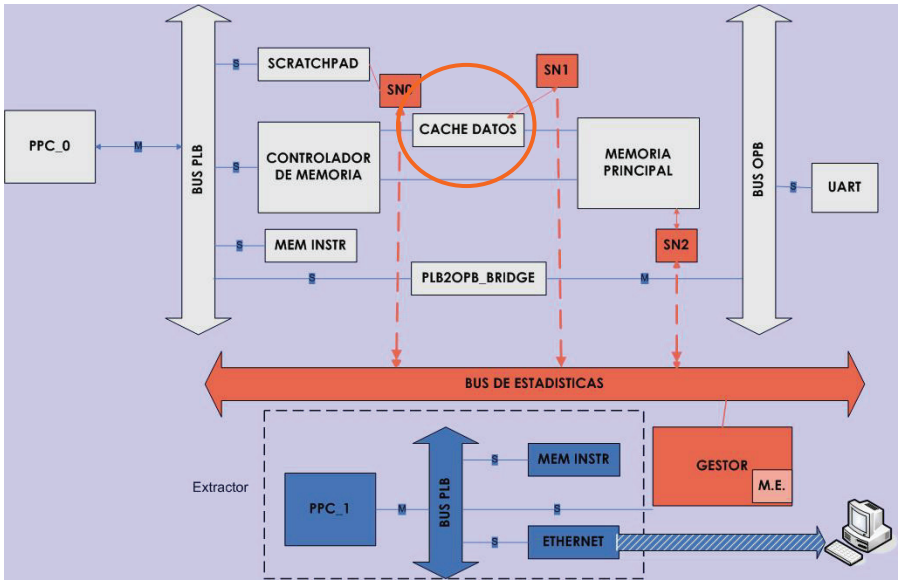


Figura nº 13: Ubicación memoria cache en la arquitectura global

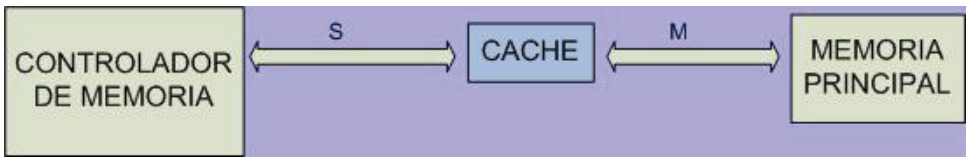


Figura nº 14: Ampliación situación memoria cache

Scratchpad:

En este segundo caso aparte de las memorias *cache* aparece adicionalmente una memoria BRAM local a cada procesador de acceso rápido controlada por software.

El uso de este tipo de memorias suele ser cada vez más frecuente en sistemas dedicados puesto que a pesar de tener una capacidad limitada, su consumo energético es muy bajo, menor aún que el de las memorias cache. La carga de datos en dichas memorias *scratchpad* se puede realizar directamente con el procesador o mediante el uso de un módulo DMA específico.

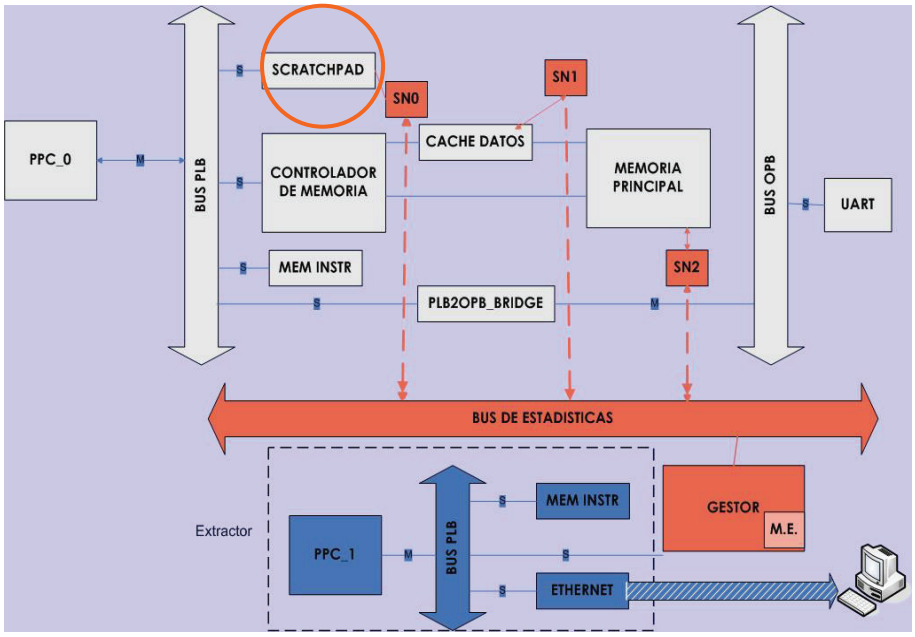


Figura nº 15: Ubicación memoria scratchpad en la arquitectura global

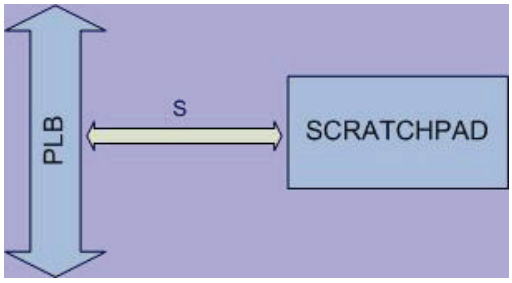


Figura nº 16: Ampliación ubicación de memoria scratchpad

Memoria Principal:

A parte de las memorias cache, existe una memoria principal de mayor tamaño. Esta memoria principal es compartida en el caso de haber

múltiples procesadores en el sistema, y es a la que el procesador accede mediante un bus compartido de alta velocidad

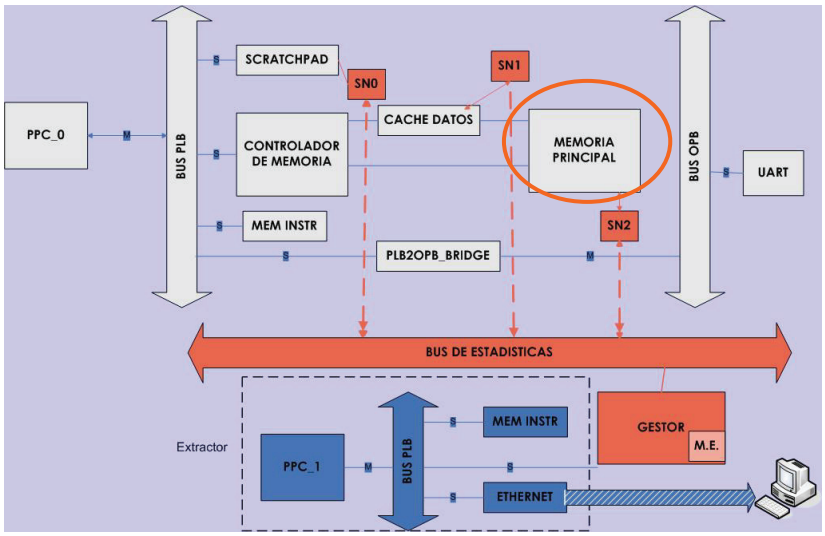


Figura nº 17: Ubicación memoria principal en la arquitectura global

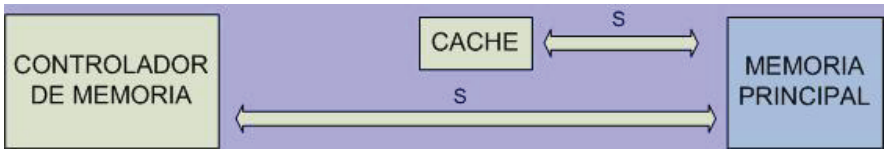


Figura nº 18: Ampliación ubicación memoria principal

Controlador de memoria:

Encargado de gestionar las peticiones de lectura y escritura del procesador, y decidir si se accede a memoria cache o bien a memoria principal.

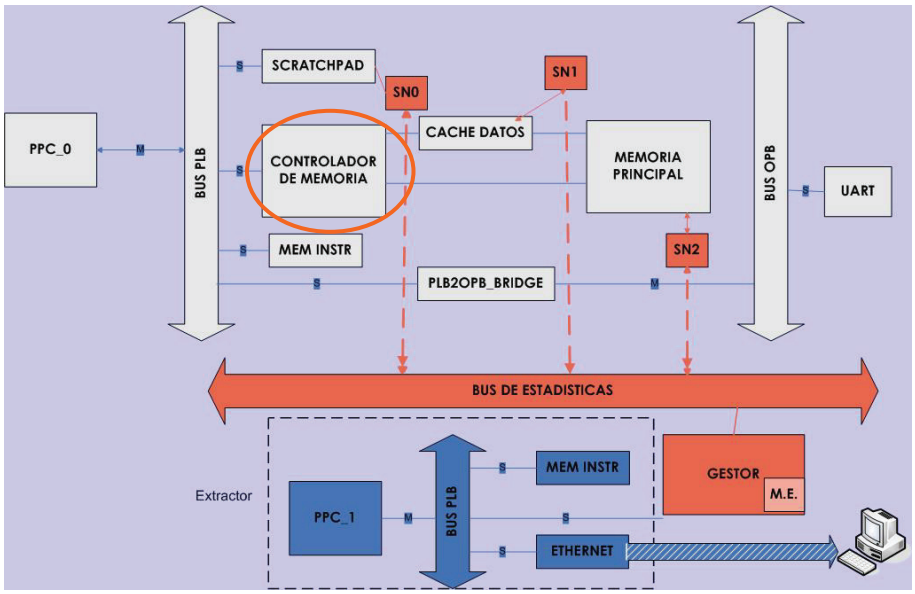


Figura nº 19: Ubicación del controlador de memoria en la arquitectura global

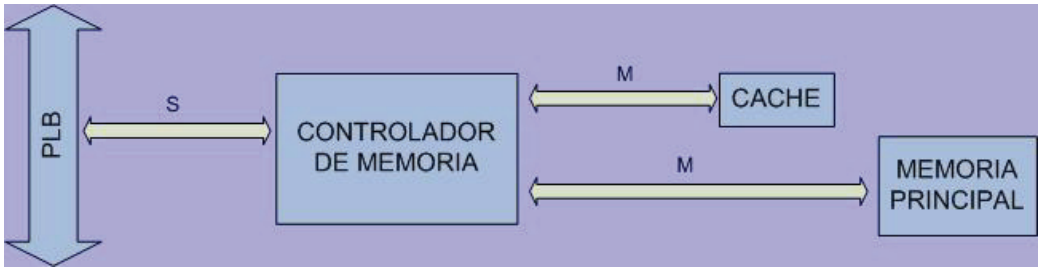


Figura nº 20: Ampliación ubicación del controlador de memoria

Resto del subsistema:

Como procesador hemos seleccionado un PowerPC 405, disponible como hardcore en la FPGA que hemos utilizado para el proyecto.

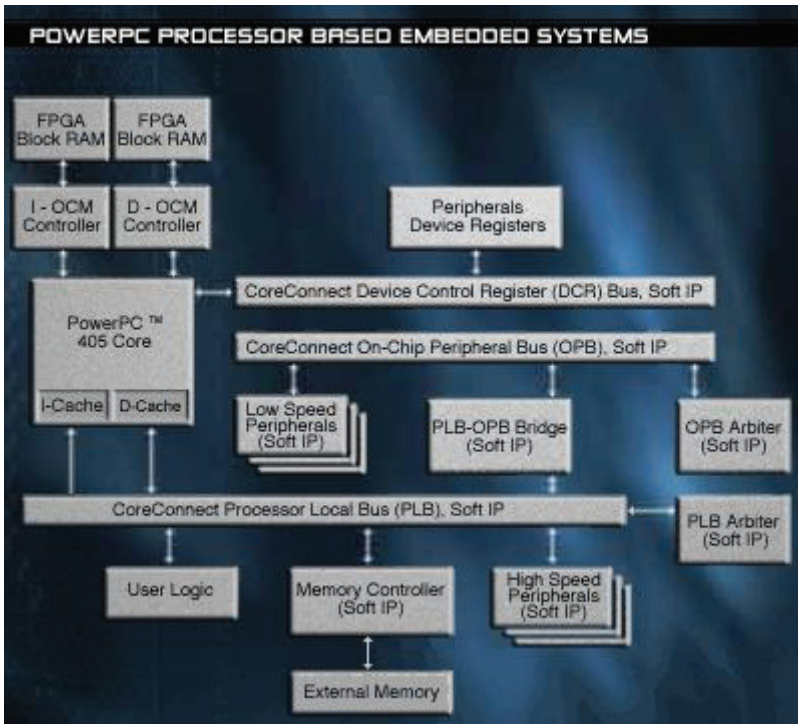


Figura nº 21: Arquitectura teórica de un sistema empotrado basado en procesadores PowerPC

Este procesador está desarrollado por varias compañías de hardware de entre las que destacan Apple, motorola e IBM. El disponible para Xilinx es una versión reducida del que fabrica IBM.

Presenta las siguientes características:

- Proporciona alto rendimiento con un bajo consumo de energía.
- Repertorio de instrucciones UISA (User Instruction Set Architecture) y extensiones para aplicaciones empotradas.
- 32 registros de propósito general de 32 bits.
- Predicción estática de saltos
- Pipeline de 5 etapas, con un ciclo simple de ejecución para la mayoría de las instrucciones, incluidas las de load/store.
- Multiplicadores / Divisores hardware para acelerar las operaciones aritméticas
- Manejo de múltiples palabras
- Little endian
- Soporte para depuración (JTAG)
- Latencia por interrupciones minimizada.
- Avanzado soporte de gestión de memoria.
- MMU (Memory Management Unit)

- Memoria cache (on-chip) para datos e instrucciones (nosotros **deshabilitaremos estas memorias cache**)

Desactivamos la memoria cache on-chip del procesador con el fin de sustituir toda la jerarquía de memoria del sistema por la nueva sintetizada.

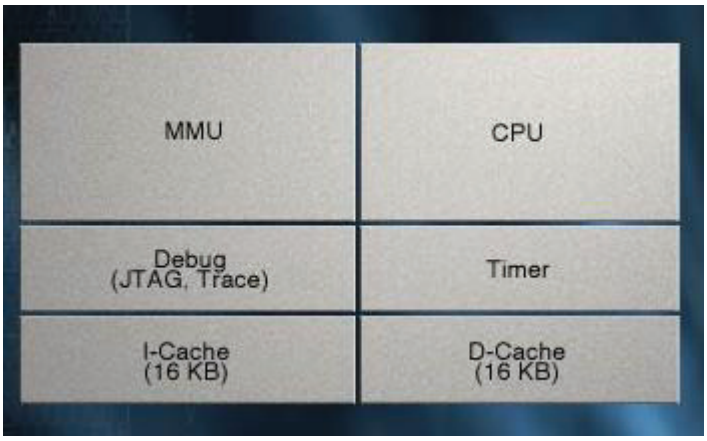


Figura nº 22: Componentes on chip de un core Power PC

PLB (Processor Local Bus)

El core PowerPC accede a los recursos del sistema a través del **PLB** (Processor Local Bus). Este bus es de 64 bits de anchura y tiene las siguientes características destacables:

- Soporte para 16 maestros (El número de módulos maestro conectados al bus se configura por medio de un parámetro)
- Accesos de 32 y 64 bits para maestros y esclavos.

OPB (*On-chip Peripheral Bus*)

Bus síncrono utilizado para conectar periféricos con tiempos de acceso variables. Soporta varios maestros y la conectividad de muchos periféricos es sencilla gracias a su identificación por multiplexación distribuida. Soporta transferencias de tamaño de palabra dinámico.

La memoria de instrucciones es de un tamaño parametrizable, en la que situamos el programa que queremos chequear. Se accede a ella por palabras de 32 bits, ya que está preparada para el código del PowerPC cuyas instrucciones son de este tamaño.

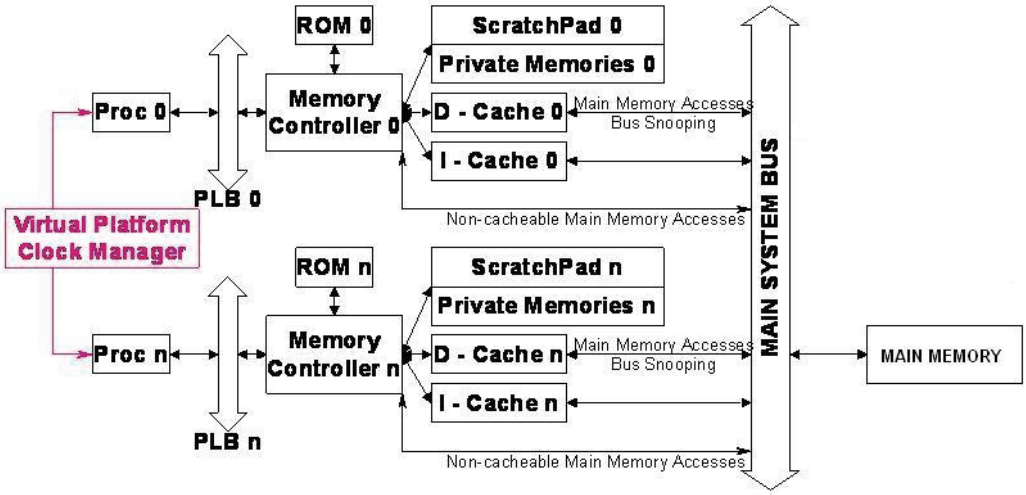


Figura nº 23: Arquitectura general del sistema extensible

3.2 Componente Software

Los programas a ejecutar en el subsistema de procesamiento intentarán ser lo más significativos posibles con el objetivo de realizar un estudio coherente y que refleje distintos accesos a diferentes módulos de la jerarquía de memoria.

Para ello a la hora de elegir los programas a ejecutar se tendrá en cuenta **técnicas de optimización** de código basadas sobre todo en la ordenación de los datos como las técnicas que se describen a continuación:

FUSIÓN DE ARRAYS

- Mejora la **localidad espacial** para disminuir los **fallos de conflicto**
- Colocar las mismas posiciones de diferentes arrays en posiciones contiguas de memoria

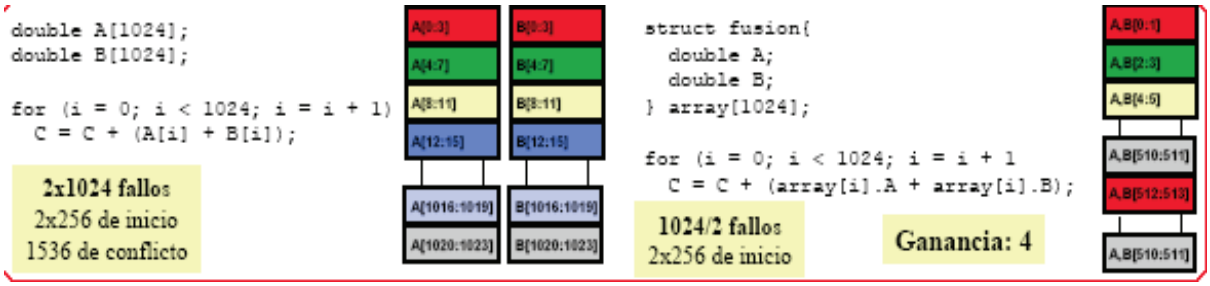


Figura nº 24: Fusión de arrays

ALARGAMIENTO DE ARRAYS

- Mejora la **localidad espacial** para disminuir los **fallos de conflicto**
- Impedir que en cada iteración del bucle se compita por el mismo marco de bloque

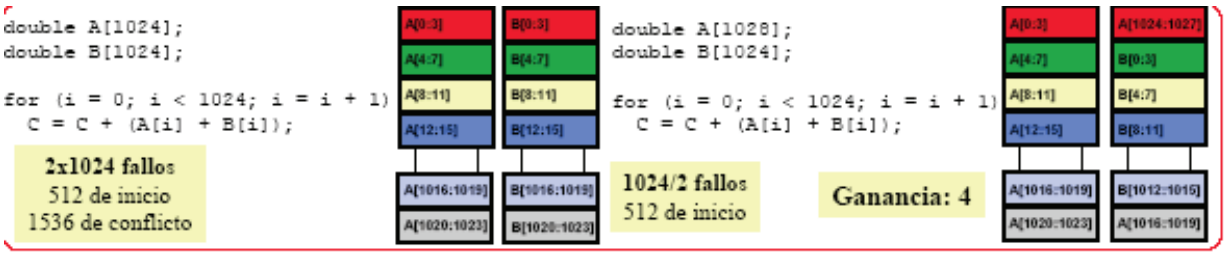


Figura nº 25: Alargamiento de arrays

INTERCAMBIO DE BUCLES:

- Mejora la **localidad espacial** para disminuir los **fallos de conflicto**
- En lenguaje C las matrices se almacenan por filas, luego se debe variar en el bucle interno la columna

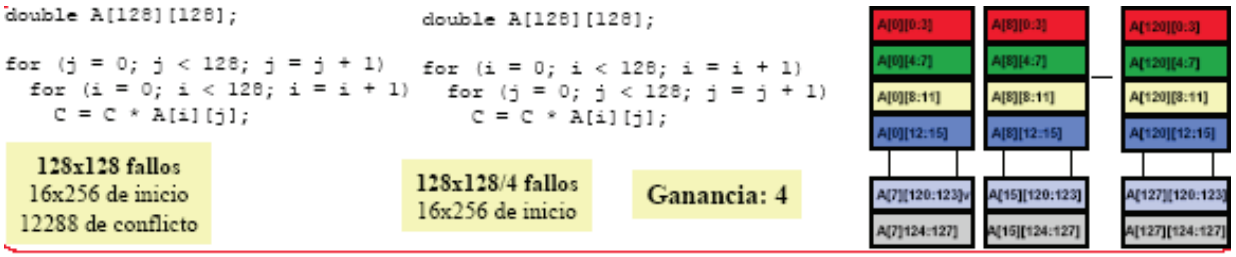


Figura nº 26: Intercambio de bucles

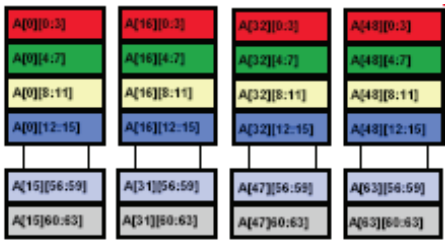
FUSIÓN DE BUCLES

- Mejora la **localidad temporal** para disminuir los **fallos de capacidad**
- Fusionar los bucles que usen los mismos arrays para usar los datos que se encuentran en cache antes de desecharlos

```
double A[64][64];

for (i = 0; i < 64; i = i + 1)
  for (j = 0; j < 64; j = j + 1)
    C = C * A[i][j];

for (i = 0; i < 64; i = i + 1)
  for (j = 0; j < 64 j = j + 1)
    D = D + A[i][j];
```

(64x64/4)x2 fallos
4x256 de inicio
4x256 de capacidad

```
double A[64][64];

for (i = 0; i < 64; i = i + 1)
  for (j = 0; j < 64 j = j + 1)
  {
   C = C * A[i][j];
   D = D + A[i][j];
  }
```



64x64/4 fallos
4x256 de inicio

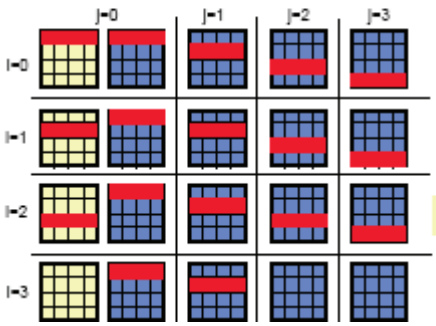
Ganancia: 2

Figura nº 27: Fusión de bucles

BLOQUEO DE MATRICES

- Mejora la **localidad temporal** para disminuir los **fallos de capacidad**
- Operar con submatrices para usar los datos que se encuentran en cache antes de desecharlos

```
double A[4][4];
double B[4][4];

for (i = 0; i < 4; i = i + 1)
   for (j = 0; j < 4; j = j + 1)
      A[i][j] = A[i][j]+B[j][i];
```




```
double A[4][4];
double B[4][4];

for (i = 0; i < 4; i = i + 2)
  for (j = 0; j < 4; j = j + 2
  {
        A[i][j] = A[i][j]+B[j][i];
        A[i][j+1] = A[i][j+1]+B[j+1][i];
        A[i+1][j] = A[i+1][j]+B[j][i+1];
        A[i+1][j+1] = A [i+1][j+1]+B[j+1][i+1];
  }
```

Ganancia: 20/12=1,66

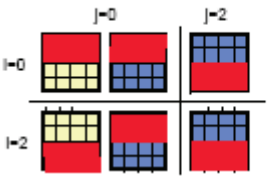


Figura nº 28: Bloqueo de matrices

3.3 Conclusiones

A lo largo de este capitulo se ha explicado la arquitectura del subsistema de procesamiento de la plataforma de emulación, haciendo especial hincapié en la jerarquía de memoria diseñada y sintetizada.

Dentro de este subsistema se han definido las componentes hardware y software del mismo.

Junto con la visión de la arquitectura general de la componente hardware, se han explicado uno por uno la funcionalidad de los diferentes módulos sintetizados dentro de la jerarquía de memoria del sistema empotrado.

En cuanto a la componente software, la elección de programas representativos ha permitido obtener ciertos resultados de interés que serán presentados posteriormente.

Por último, destacar la gran capacidad de reutilización de este sistema creado, puesto que gracias a que cada nivel de la jerarquía de memoria es parametrizable, se puede reutilizar para sacar diversas conclusiones.

También es imprescindible destacar que se ha conseguido crear una plataforma de emulación que funciona con una frecuencia del orden de megahercios, lo cual supone una importante mejora frente a las herramientas de simulación de este tipo de sistemas, que funcionan con una frecuencia del orden de los kilohercios.

Capítulo 4. Arquitectura del subsistema de extracción de estadísticas

4.1 Componente Hardware

La arquitectura general del sistema global puede observarse en la siguiente figura, en la cual, el sistema extractor de estadísticas se encuentra dibujado en color rojo.

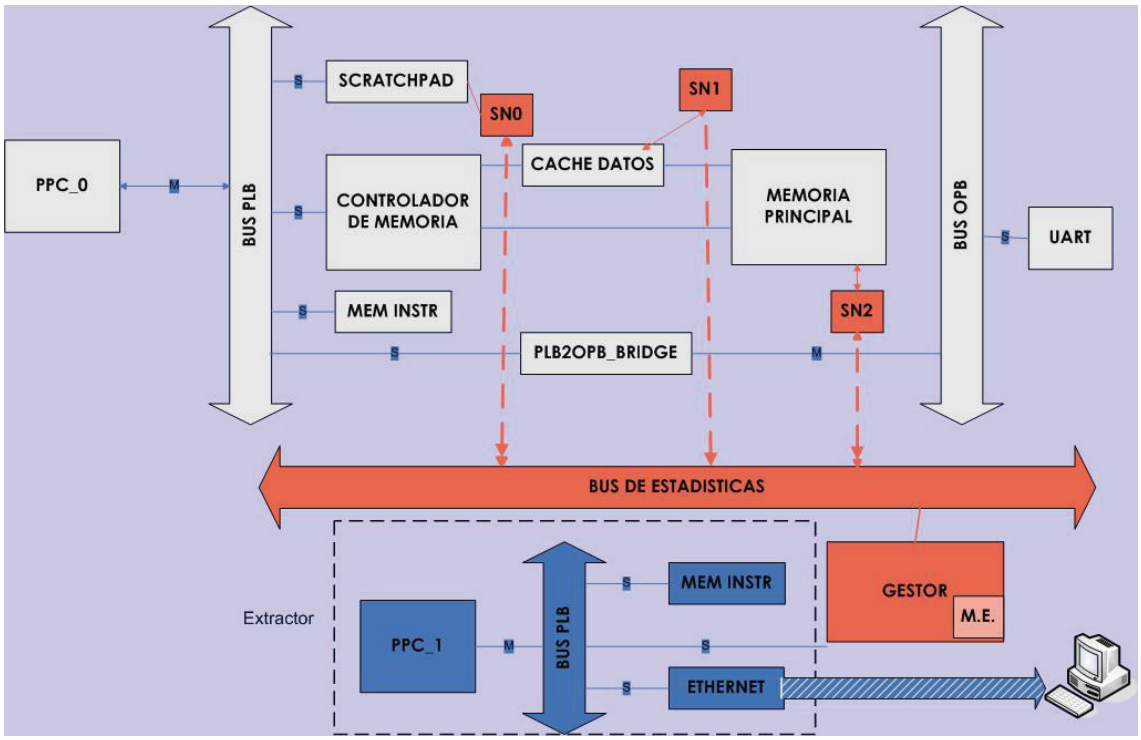


Figura nº29 : Esquemático general del sistema extractor de estadísticas y ubicación en la plataforma global

Como puede observarse en la figura, el sistema extractor de estadísticas, deberá estar completamente integrado con el resto del sistema, pero a la vez, deberá ser independiente para facilitar su escalabilidad.

Se pueden distinguir diversas entidades en el sistema:

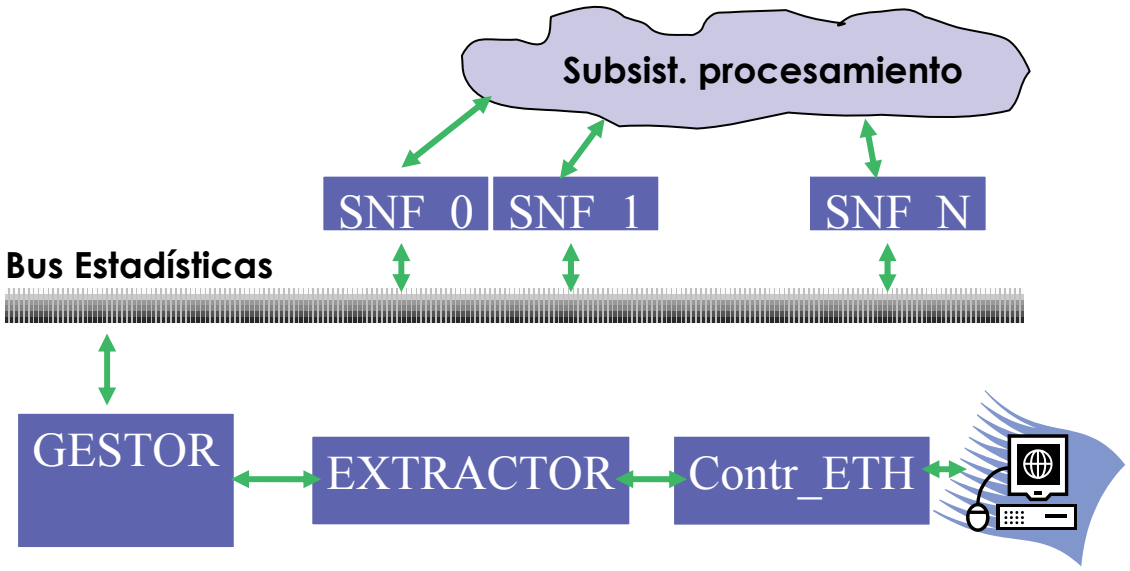


Figura nº30 : Esquemático particular del subsistema de estadísticas

- **Módulo Gestor de Estadísticas**

Este módulo gobierna y controla todo el sistema extractor de estadísticas. Su función consistirá en recibir los datos correspondientes de cada uno de los sniffers existentes en el sistema, interpretarlos, y posteriormente almacenarlos en la memoria dedicada.

Este módulo tendrá que ser capaz de parar el reloj de la plataforma, de forma que tenga tiempo suficiente para procesar la información recibida de los sniffers.

- **Despachador de red (Extractor)**

Esta entidad será la encargada de construir, a partir de la información almacenada en memoria, paquetes UDP, y enviarlos a través del protocolo Ethernet. Para lograr este objetivo, tendrá que existir comunicación entre este módulo y la memoria dedicada.

En el apartado de implementación se profundizará más sobre la opción escogida para el desarrollo de este módulo.

- **Bus de Estadísticas**

Puesto que se necesita una rápida y sencilla comunicación entre sniffers y gestor, se implementará un bus simple y se definirá un protocolo de arbitraje, de forma que se garantice el acceso de los sniffers a él para volcar sus datos, sin conflictos.

- **Memoria dedicada**

Los datos extraídos se almacenarán en una memoria dedicada con dos objetivos primordiales:

  - Permitir un acceso a los contenidos de la memoria mediante la utilidad XMD Debugger que proporciona el entorno EDK.
  - Representa el papel de buffer que permita al extractor construir las tramas y enviarlas a través del protocolo Ethernet.

- **Sniffers**

Estas entidades estarán repartidas por todo el sistema. Su funcionalidad será extraer información de los diferentes módulos existentes y volcar los datos adecuadamente formateados en el bus de estadísticas, para que puedan ser recogidos en su momento por el Gestor de Estadísticas.

A continuación se puede observar un esquema simple del núcleo del sistema extractor de estadísticas en el que se aprecia el acceso mediante JTAG a la memoria para poder consultar sus datos por medio del depurador XMD.

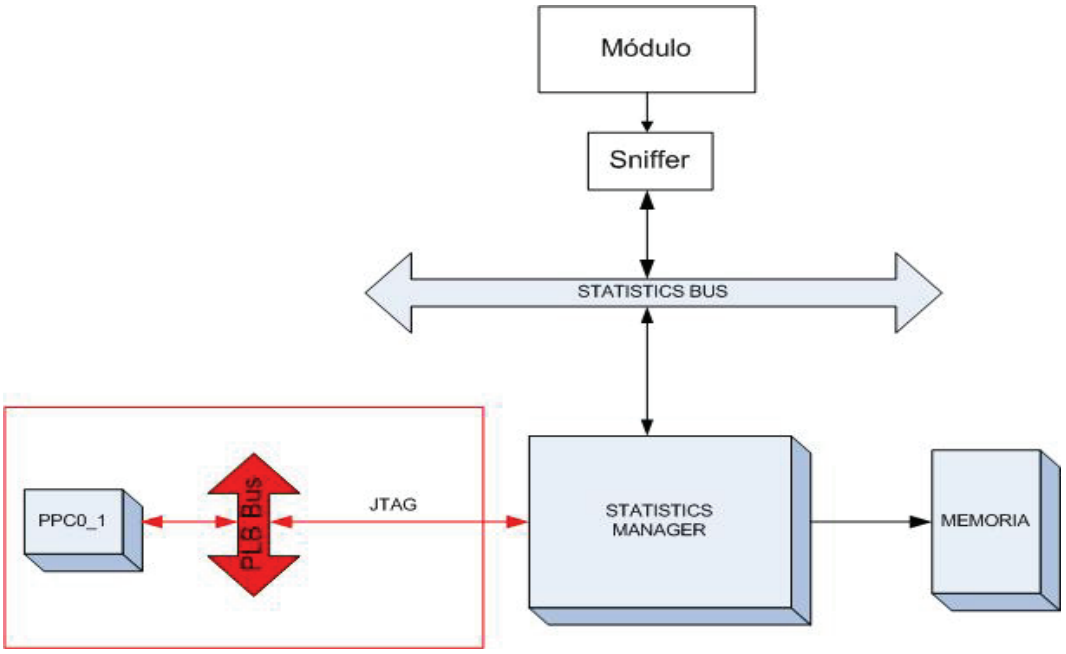


Figura nº 31: Esquemático simplificado del núcleo del sistema extractor de estadísticas

La memoria se implementará de forma interna al gestor de estadísticas.

Especificación del subsistema de estadísticas:

Se pueden distinguir tres tipos de requisitos en este sistema:

- **Requisitos funcionales**

  El sistema extractor de estadísticas tiene que ser capaz de extraer información que interese al investigador de la plataforma o interesado en la emulación del sistema global, y para lograrlo, tiene que ser construido bajo los siguientes principios:

  - Debe ser capaz de **acceder a diferentes tipos de módulos**, y extraer **diferentes tipos de información**, puesto que esto dependerá básicamente del módulo sobre el cuál se esté interesado realizar un análisis.
  - Su desarrollo y funcionamiento **no deben interferir** en funcionalidad del sistema emulado en ningún momento.
  - Deberá ser capaz de **almacenar los datos en la memoria**.
  - Deberá ser capaz de formar paquetes UDP a partir de los datos de la memoria y **enviarlos al computador** a través del protocolo Ethernet.

- **Requisitos temporales**

    - El sistema extractor de estadísticas no debe afectar a otras funcionalidades del resto del sistema, y debe obtener datos "realistas". Para conseguir este objetivo, el sistema extractor de estadísticas **ha de ser capaz de parar el reloj** cuando sea necesario.
    - Puesto que este sistema almacenará los datos extraídos en una memoria, deberá **respetar su temporización** correspondiente que dependerá de la tecnología seleccionada.

- **Requisitos de diseño**

    - El sistema extractor de estadísticas deberá ser **modular e independiente.** Se debe minimizar la interacción entre este sistema y el sistema global.
    - Debe permitir una **rápida y sencilla expansión**. Todo diseñador de un módulo debe ser capaz de desarrollar su sniffer correspondiente e integrarlo dentro del sistema sin complicaciones.

Diseño e implementación:

A la hora de comenzar el diseño de este sistema, dos partes han de ser diferenciadas:

Una **primera parte** que es **dependiente** del resto del sistema. Estará formada por todos los sniffers de la plataforma y será diseñada por el diseñador del módulo correspondiente a cada sniffer. La idea fundamental es que el diseñador del módulo del cual se desea extraer información, es la persona indicada para diseñar el sniffer correspondiente. Por supuesto para que este sniffer se pueda integrar con el gestor de estadísticas y su comunicación sea posible, se fijarán unos estándares en los formatos de los datos que se intercambiarán.

En la figura que se muestra a continuación puede observarse en color rojo los módulos del sistema extractor de estadísticas que dependerán del diseño de otros módulos externos, y que por tanto, es lógico que su implementación se realice a la par que ellos.

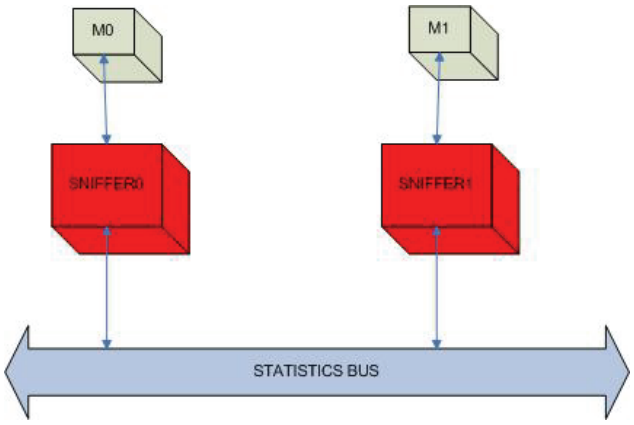


Figura nº 32 : Parte dependiente del resto del sistema

Una **segunda parte** que es complemente **independiente** del resto del sistema, y del número de sniffers que haya en él, puesto que deberá ser parametrizable, por lo menos hasta unos valores máximos.

Esta parte está formada por el gestor de estadísticas, el despachador de red y la memoria. El diseño e implementación de este subsistema puede realizarse de forma totalmente independiente y sin poseer conocimiento alguno de los detalles de la plataforma a examinar.

En el siguiente esquema pueden observarse en color verde aquellas entidades que son completamente independientes:

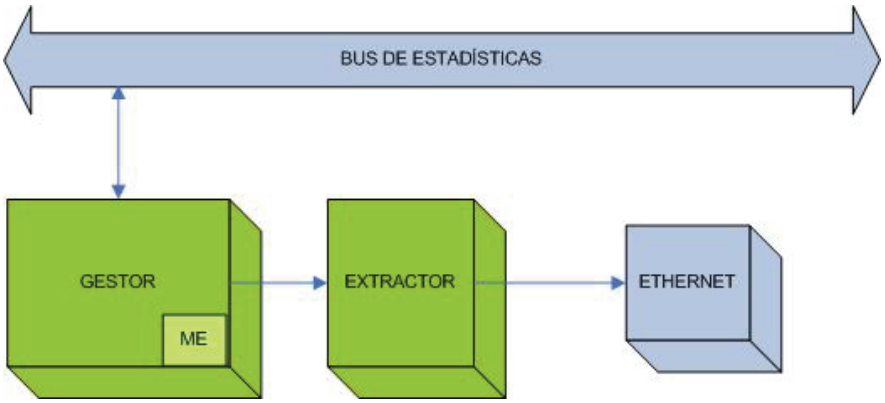


Figura nº33: Parte independiente del resto del sistema

Tendrá que tenerse en cuenta que el módulo extractor es sólo un módulo "conceptualmente" hablando. De hecho, se implementa en parte mediante un programa software, y un procesador dedicado a su ejecución.

A continuación se describirá el diseño de cada una de las entidades a dos niveles. Se explicará el diseño a nivel de bloques detallando aquellos aspectos más cercanos a la implementación que se consideren convenientes.

Descripción de alto nivel de abstracción (nivel de bloques)

Modulo Gestor de Estadísticas

En primer lugar mostramos en la siguiente figura su ubicación dentro del sistema global:

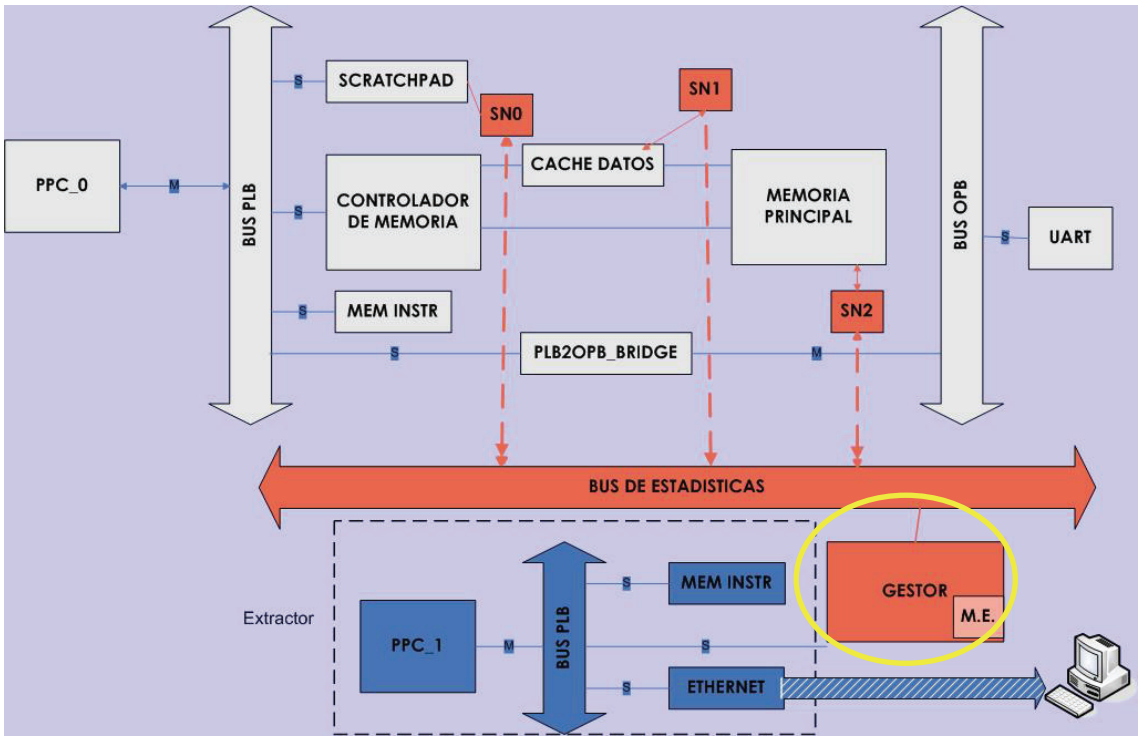


Figura nº 34: Situación del módulo gestor del sistema extractor de estadísticas

Como ya se ha dicho antes, el módulo gestor de estadísticas es el encargado de controlar todo el proceso de extracción de estadísticas y gobernar las señales que en él se utilizan. Además debe ser capaz de parar el reloj de la plataforma para no interferir en la funcionalidad del resto del sistema, ocupando ciclos de ejecución de éste.

Esta entidad estará formada por los siguientes bloques:

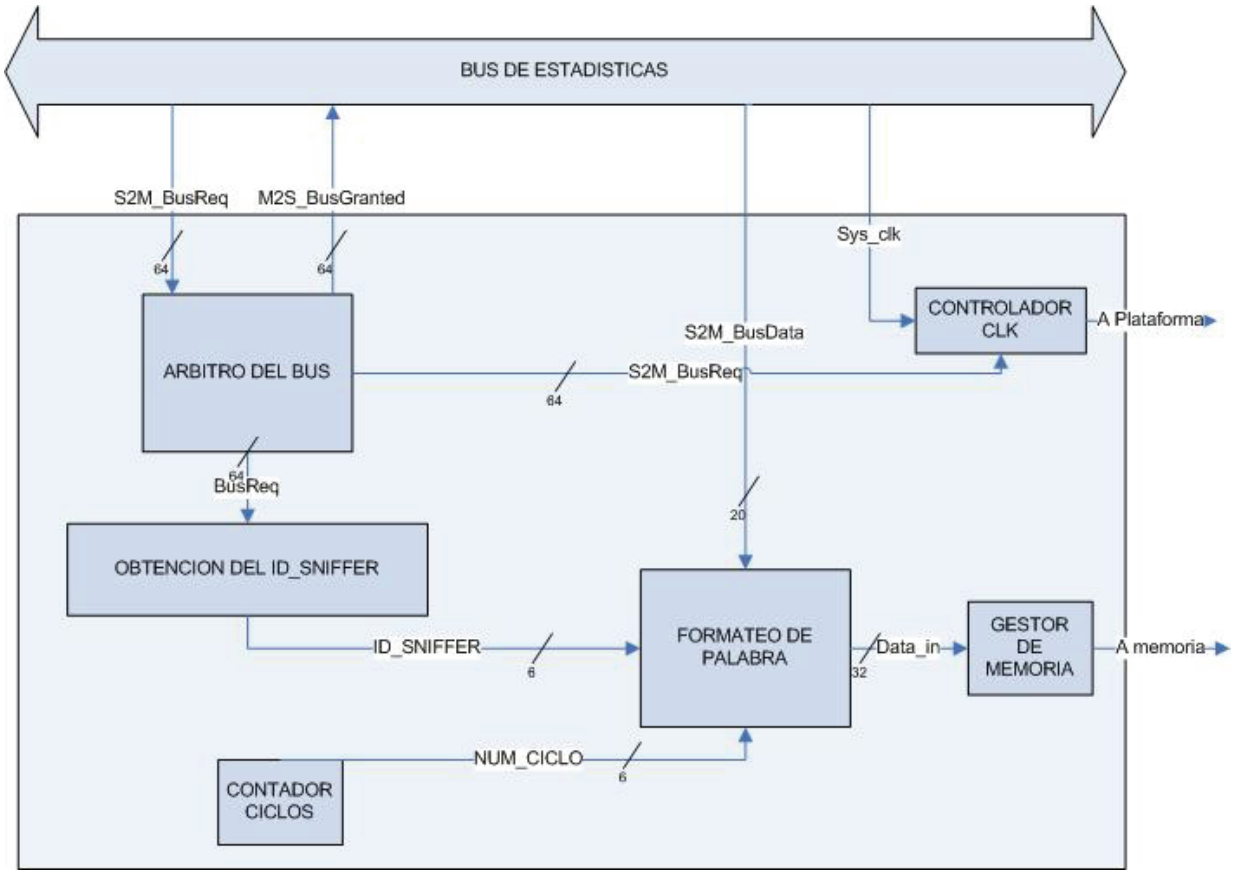


Figura nº35 : Esquema de los bloques del gestor de estadísticas

- **Arbitro del bus:**

  Su misión será gestionar las prioridades cuando varios sniffers tienen datos disponibles para volcar en el bus, en el mismo instante de tiempo.

  El protocolo de arbitraje será el siguiente:
  - Cuando dos sniffers solicitan el uso del bus, se concederá el bus al sniffer con un identificador mayor. Cuando el sniffer correspondiente recibe el mensaje de concesión del bus, bajará su línea de petición correspondiente.

- **Submódulo de obtención del identificador del sniffer**

  La función principal será la obtención del identificador del sniffer a través de los valores de las líneas correspondientes. Para ello

tendrá que realizar una codificación del valor de las líneas de petición del bus.

- **Contador de ciclos virtuales**

  Contará los ciclos de la plataforma, de forma que se pueda añadir al dato estadístico obtenido la información del ciclo en el que se produjo el evento espiado. Para conseguir este objetivo, este contador sólo contará cuando se active el reloj de la plataforma a emular.

- **Submódulo de formateo de la palabra de memoria**

  Se encargará de formatear correctamente todos los datos obtenidos por los sniffer, añadirle las cabeceras que sean oportunas (identificador del sniffer, numero de ciclos...) y de esta forma conseguir una palabra de memoria preparada para ser almacenada.

- **Submódulo gestor de la memoria**

  Será el encargado de interactuar con la memoria, y controlar las señales de lectura, escritura, dirección, byte enable...

- **Controlador reloj del sistema**

  Su misión es generar un reloj virtual, que regirá el funcionamiento de la plataforma global. Deberá de ser capaz de generar una señal de supresión del reloj de la plataforma, en aquellos casos en los que sea necesario.

Obviamente, toda la sincronización entre estos módulos y la comunicación del gestor con los sniffer necesitará señales de control que vendrán generadas y controladas por una **máquina de estados:**

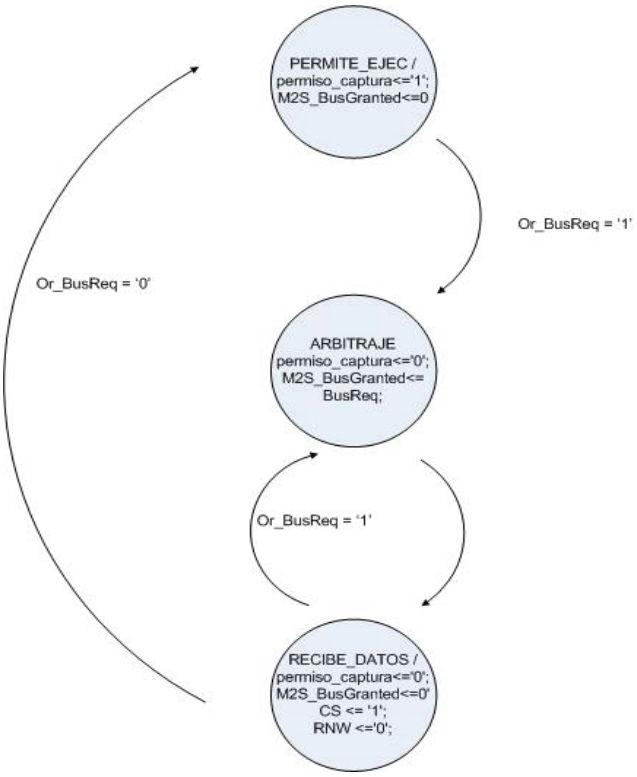


Figura nº 36: Diagrama de la máquina de estados del gestor de estadísticas

Esta máquina será de tipo Moore, y tendrá el control del árbitro del bus.

Su estado inicial (permite_ejecucion) permite al sniffer capturar los datos y a la plataforma funcionar en un modo normal, mientras que cuando se recibe petición por parte de un sniffer de uso del bus para volcar los datos, se pasa a un segundo estado de arbitraje en el cual se decide que sniffer tendrá el permiso para volcar sus datos en el bus. Por último el estado de recibe_datos en el cuál el uso del bus está garantizado para el sniffer correspondiente, y éste procederá a volcar los datos que haya extraído por lo que el manager tendrá que recogerlos del bus de estadísticas y realizar las fases posteriores de formateo de la palabra.

La señal de permiso_captura indica al sniffer que puede capturar los datos del módulo correspondiente. Si está a baja esta señal el sniffer no capturará nuevos datos con el objetivo de evitar capturar los mismos datos en repetidas ocasiones.

Módulo Sniffer

El diseño de un sniffer modelo se puede observar en la siguiente figura. Es importante destacar que se propone un modelo de arquitectura para el sniffer, pero que será el diseñador de él quien tendrá libertad a la hora de su implementación. El diseñador del sniffer deberá ser el mismo que diseñe el módulo a espiar, y sólo tendrá que proporcionar un sniffer que respete los protocolos de sincronización con el maestro y los formatos de mensajes establecidos.

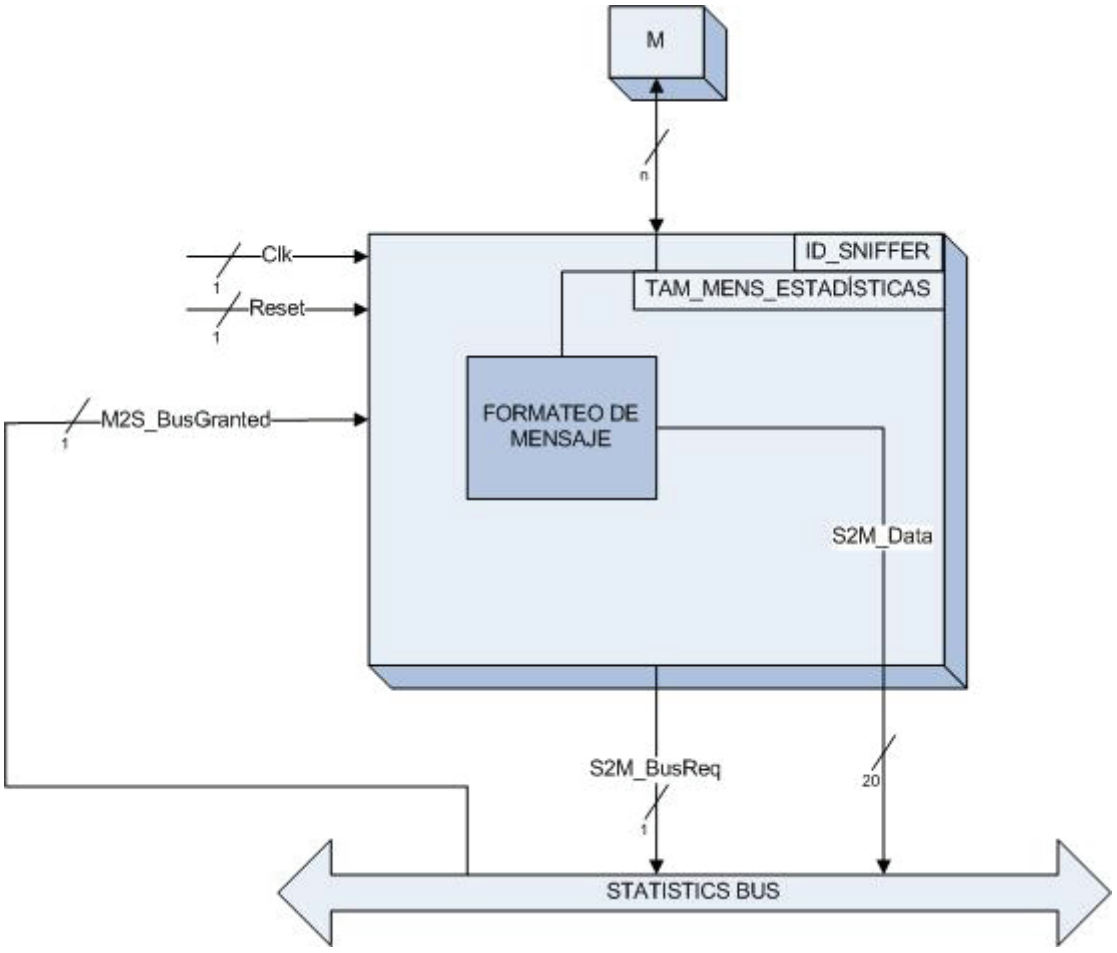


Figura nº 37 : Diagrama de bloques de un sniffer modelo

La máquina de estados que controla el funcionamiento del sniffer es una máquina Mealy de tres estados:

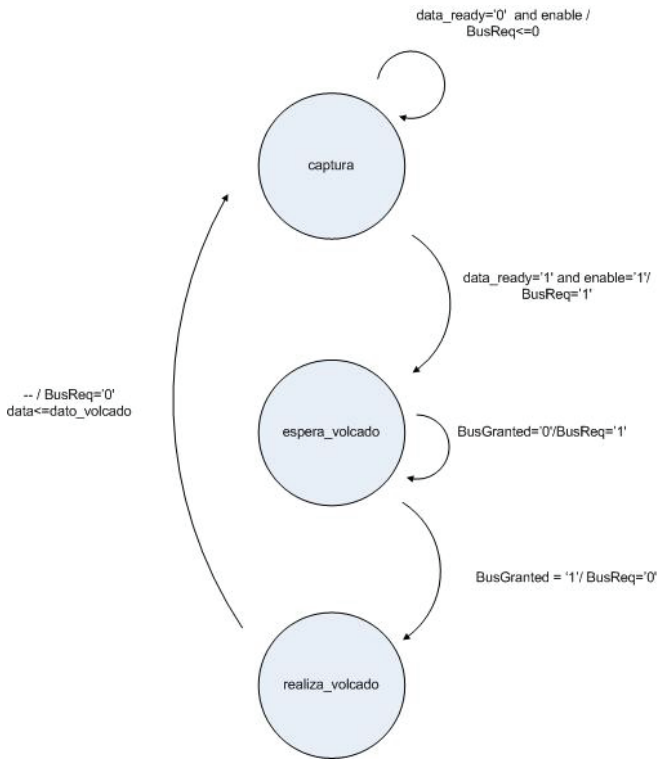


Figura nº 38 : Maquina de estados del sniffer modelo

En el estado inicial (captura) el sniffer accede al módulo del cual quiere extraer información y permanece en este estado hasta que se tienen todos los datos que se quieren obtener. En este modelo de sniffer la señal data_ready es una señal interna que el sniffer activará cuando ya tenga recogidos todos los datos.

En ese momento se enviará al gestor una petición de acceso al bus, y se pasará a un estado de espera (espera_volcado) en el que se permanecerá hasta que se reciba del gestor la señal de acceso al bus concedido. Es en este momento cuando se pasa al estado de realiza_volcado, se baja la línea de petición, y entonces el sniffer accede al bus y vuelca en él los datos extraídos. (salida Mealy). Una vez volcados los datos se vuelve al estado inicial de captura.

Formatos de mensajes en la plataforma:

Se definen formatos específicos para los mensajes de datos intercambiados entre el sniffer y el gestor, y entre el gestor y la memoria.

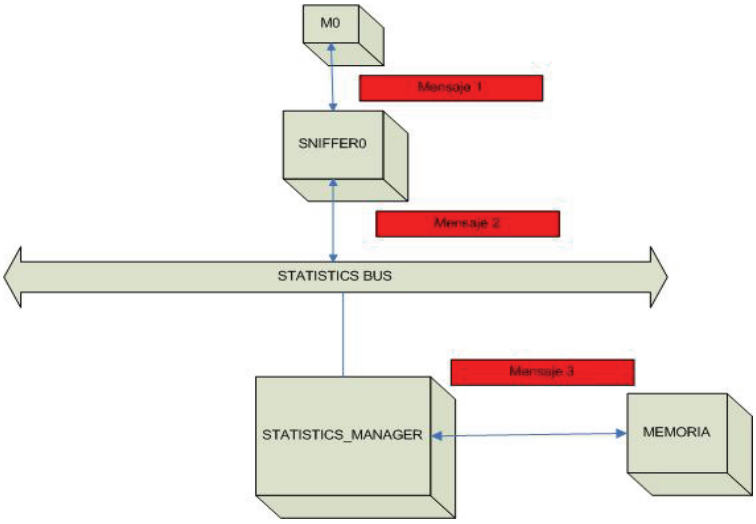


Figura nº 39: Visión global de los tipos de mensajes en el sistema

Mensaje 1:

El formato del mensaje de tipo1 se deja a elección libre del diseñador del módulo y su sniffer debido a que los datos que se desean extraer de un módulo u otro pueden variar sustancialmente. Sólo se instanciarán tantas líneas como datos se quieran extraer del módulo en cuestión y ninguna otra más (ahorro de líneas dedicadas).

Ejemplo:

Entre la memoria cache y su sniffer correspondiente habrá líneas dedicadas a transmitir información sobre el número de fallos o de aciertos, dirección en la que se produjeron….

Mensaje 2:

El sniffer recibirá un mensaje de tipo 1 y analizará la información obtenida hasta identificar un evento que deba registrar. En ese momento construirá un mensaje de tipo dos que enviará al gestor de estadísticas a través del bus dedicado. Este mensaje constará de 20 bits.

Ejemplo:

Si el sniffer correspondiente al módulo de la memoria cache recibe de ésta sólo cuatro líneas dedicadas, y se encarga de formar un mensaje de 20 bits, con el objetivo de pasar un mensaje con formato estándar al módulo gestor del sistema extractor de estadísticas.

Mensaje 3:

El módulo gestor del sistema extractor de estadísticas, recibirá un mensaje de 20 bits (mensaje tipo 2) y su misión será formar un mensaje de 32 bits(mensaje tipo 3) que estará formateado de acuerdo con la siguiente definición:

Bits 31-26: Número identificador del sniffer. Se permiten con este formato de mensaje 64 sniffers en la plataforma.

Bits 25-20: Número de ciclo en el que se ha producido el evento espiado.

Bits 19-0: Mensaje de tipo 2 obtenido directamente del sniffer correspondiente.

En la siguiente figura se puede observar dicho formato:

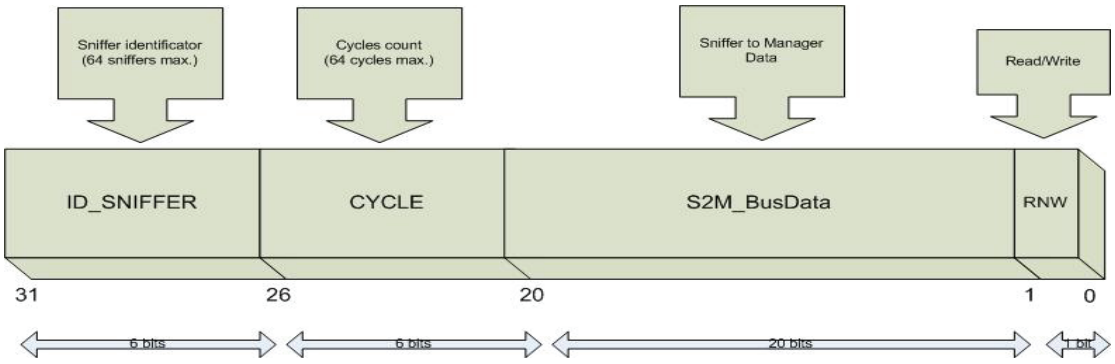


Figura nº 40: Formato de un mensaje de tipo 3

Los mensajes de este tipo son los que se almacenan en las palabras de la memoria dedicada.

A continuación se definen los formatos de los mensajes (de tipo 2) de los sniffers creados en el sistema:

Sniffer para capturar peticiones a la Scratchpad

La scratchpad es una memoria de acceso directo de alta velocidad cercana al procesador. Lo que nos interesa de una petición que haga el procesador a esta memoria es la dirección a la que ha accedido, el tipo de operación que ha realizado y los bytes que han intervenido en esta operación.

Basándonos en el sniffer modelo y sabiendo que el tamaño de la información que un sniffer vuelca por el bus dedicado de estadísticas es de 20 bits definimos el siguiente tipo de mensaje:

Figura nº41: Formato del S2M_BusData para el sniffer de la scratchpad

- Dirección de acceso a la Scratchpad (DIR_WIDTH_SP): No se trata de la dirección completa, ya que en ese caso necesitaríamos 32 bits sino una pequeña parte de la misma que es la que empleamos a la hora de direccionarla: Se trata de los DIR_WIDTH_SP bits más significativos de entre los DIR_WIDTH_SP+2 menos significativos. Los dos bits menos significativos se desprecian ya que las direcciones se refieren a bytes y nuestra memoria se direcciona por palabras (4 bytes corresponden a una palabra).
- Byte enable (4): Como antes hemos dicho nuestra Scratchpad se direcciona por palabras, lo cual no quiere decir que no sea accesible a nivel de byte. Se puede acceder a cualquier byte de la palabra que se desee. Es por esto que en el mensaje que envía este sniffer ha de indicar los bytes que han intervenido en la operación. Es por esto que utilizamos 4 bits y no 2. Con dos bits podríamos indicar cuantos bytes han sido accedidos o modificados mientras que con cuatro podemos especificar cuales han sido marcándolos con unos del byte más significativo al menos. Por ejemplo: BYTE_ENABLE="0001" es que se ha accedido al byte menos significativo de la palabra
- RNW: Indicador del tipo de operación. "0" corresponde a una escritura mientras que "1" corresponde a una lectura.

El uso de este sniffer es limitar el tamaño que puede tener una Scratchpad que esté conectada a él. Necesitamos que el mensaje sea de 20 bits pues así ha quedado definido. De éstos, 4 los necesitamos para el RNW y el Byte enable lo que nos deja un total de 16 para direcciones en la Scratchpad. Esto supone un tamaño 128KB. Teniendo en cuenta que este tipo de memorias no suponen más de un 20% de este tamaño en ningún caso, podemos concluir que esta limitación no afecta al sistema realmente.

Sniffer para capturar peticiones a la Cache

La memoria cache es una memoria de acceso directo de alta velocidad cercana al procesador. Lo que nos interesa de una petición que haga el procesador a esta memoria es la dirección a la que ha accedido, el tipo de operación que ha realizado y si se ha producido un acierto de cache ó bien un fallo.

Basándonos en el sniffer modelo y sabiendo que el tamaño de la información que un sniffer vuelca por el bus dedicado de estadísticas es de 20 bits definimos el siguiente tipo de mensaje:

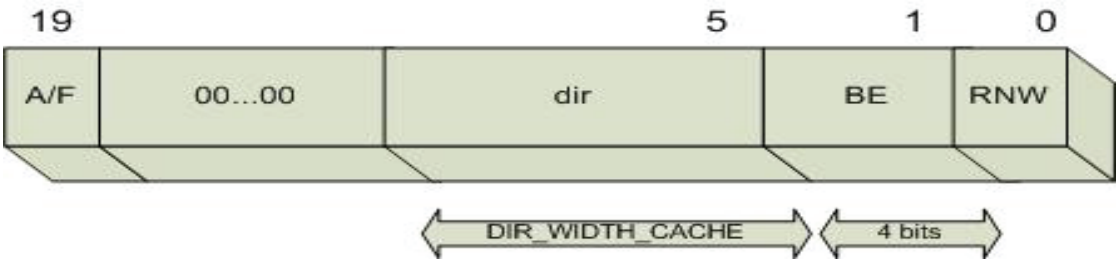


Figura nº42: Formato del S2M_BusData para el sniffer de la cache

Bit 19: A/F indica si se produce un acierto ó un fallo de cache.

- Acierto: 1
- Fallo: 0

Dir: Tendrá DIR_WIDTH_CACHE bits de longitud.
RNW: Indicador del tipo de operación. "0" corresponde a una escritura mientras que "1" corresponde a una lectura.

Sniffer para capturar peticiones a la Memoria Principal / Bus

La memoria principal es la memoria situada en el nivel más alejado del procesador en la jerarquía de memoria implementada. Lo que nos interesa de una petición que haga el procesador a esta memoria es la dirección a la que ha accedido, y el tipo de operación que ha realizado así como si se trae un bloque o una palabra de ella.

Basándonos en el sniffer modelo y sabiendo que el tamaño de la información que un sniffer vuelca por el bus dedicado de estadísticas es de 20 bits definimos el siguiente tipo de mensaje:

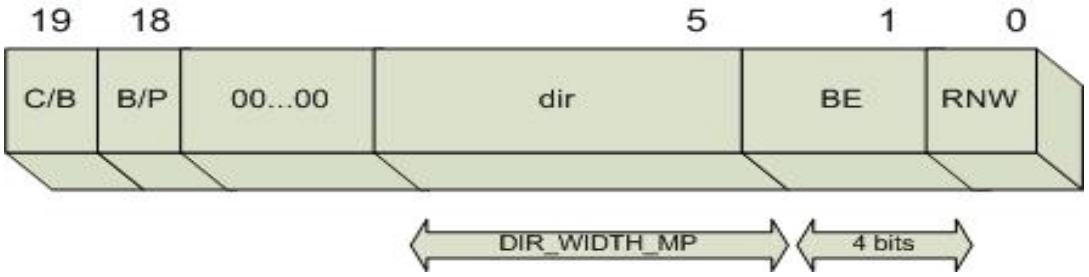


Figura nº43: Formato del S2M_BusData para el sniffer de la memoria principal

Bit 19: C/B indica si la petición la hace la cache por un fallo o bien es una petición desde el bus.
Bit 18: B/P indica si se pidió un bloque o sólo una palabra de un bloque.

- Bloque : 1
- Palabra: 0

Si se pidió un bloque se desecha el bit enable y los LOG2_PALS_PER_BLOCK bits menos significativos de la dirección (estos bits son los que indican la palabra del bloque, pero como se trae el bloque entero no hacen falta)

Dir: Tendrá DIR_WIDTH_MP bits de longitud.
RNW: Indicador del tipo de operación. "0" corresponde a una escritura mientras que "1" corresponde a una lectura.

Para tener una visión global del intercambio de mensajes se propone el siguiente ejemplo:

Mensaje tipo 1

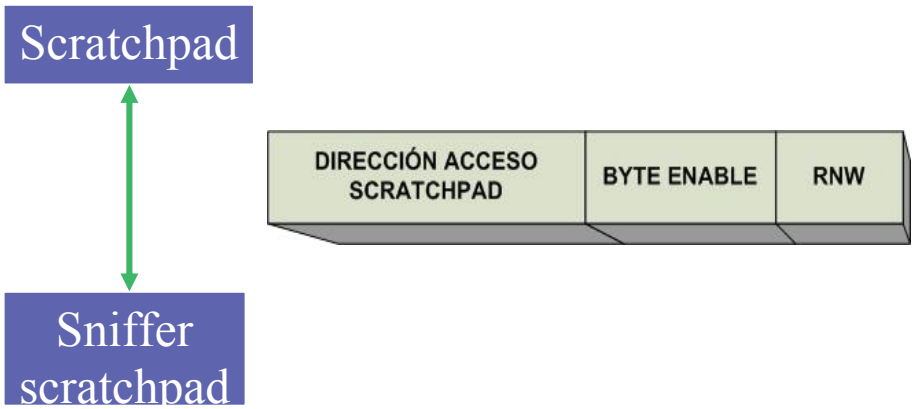


Figura nº44: Ejemplo de intercambio mensaje de tipo 1

Mensaje tipo 2:

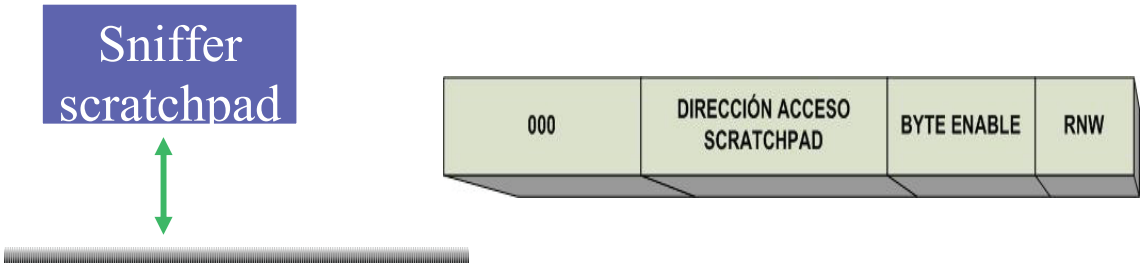


Figura nº45: Ejemplo de intercambio mensaje de tipo 2

Mensaje tipo 3 (palabra de memoria de estadísticas)

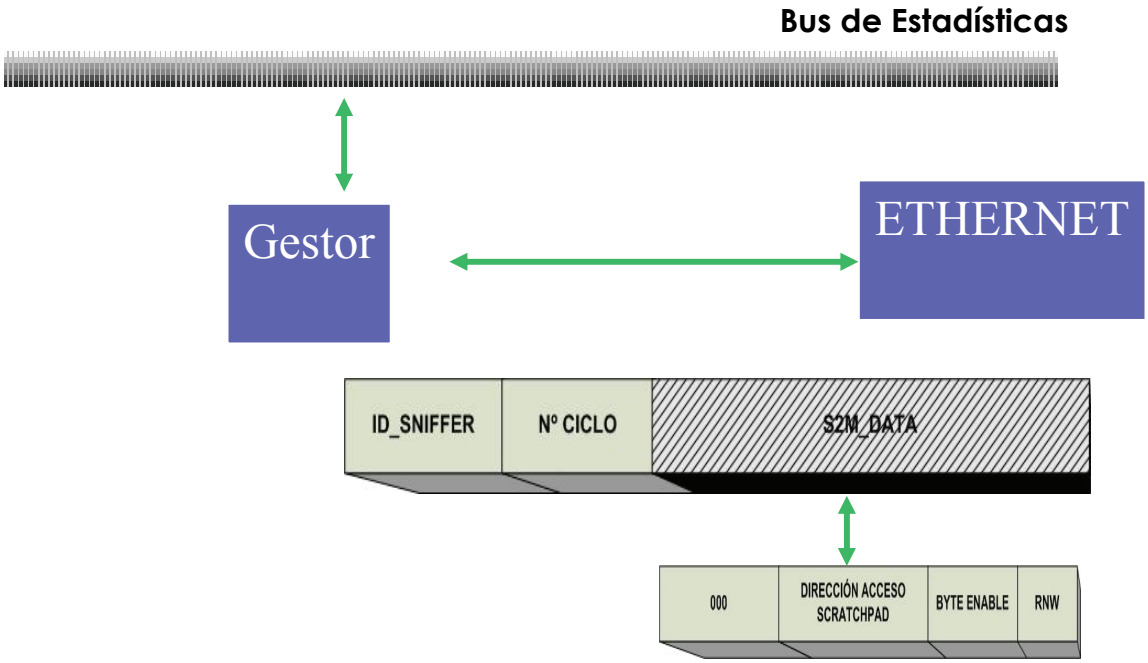


Figura nº46: Ejemplo de intercambio mensaje de tipo 3

Módulo Memoria

La memoria dedicada utilizada en el sistema tiene un ancho de palabra de 32 bits y 1024 palabras. Es decir, se utiliza una memoria de 32K, parametrizable en número de palabras y anchura de palabra.

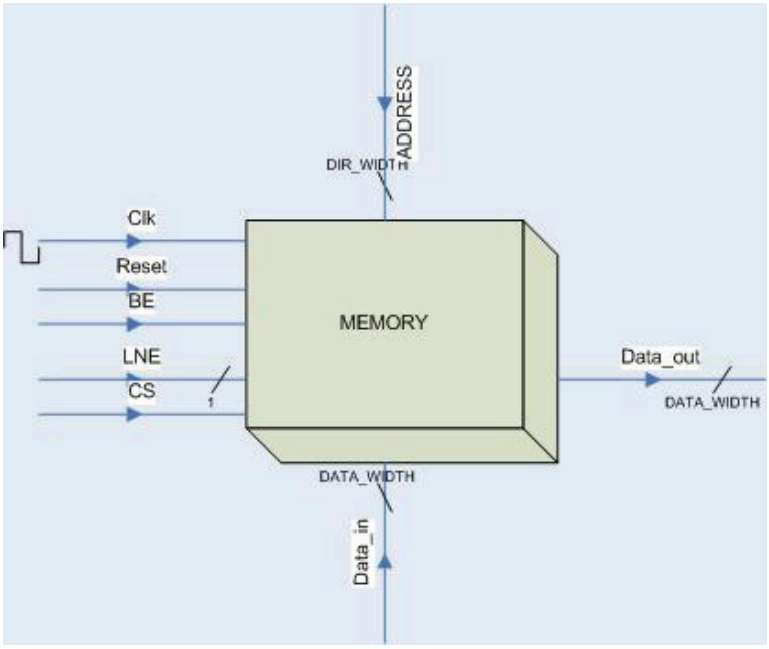


Figura nº 47: Esquema del módulo de memoria dedicada

4.2 Componente Software

La componente software del subsistema extractor de estadísticas consiste en un programa que se ejecuta en la memoria de instrucciones de este subsistema y que realiza las siguientes funciones:

- **Leer de la memoria dedicada e interna del módulo gestor** de estadísticas.
- **Formar paquetes UDP** con los datos obtenidos
- **Enviar estos paquetes vía Ethernet**

Para realizar estas funciones se siguen una metodología concreta que se explica a continuación:

En primer lugar se fijan las direcciones MAC de la placa y del computador. (La dirección local ó fuente es ignorada y sobrescrita por el módulo controlador de ethernet)

```
static Xuint8 LocalAddress[XEM_MAC_ADDR_SIZE] =
{
//Direccion MAC de la placa
0x00, 0x02, 0xb5, 0x01, 0x70, 0x14
};

static Xuint8 DestinationAddress[XEM_MAC_ADDR_SIZE] =
{
0x00, 0x05, 0x1C, 0x1F, 0xB7, 0x8A
};
```

De forma similar se establecen las direcciones IP y los puertos:

```
static unsigned char IPLocal[4]={ 215,13,56,78 };
static unsigned char IPRemota[4]= { 147,96,81,137 };
static unsigned short PuertoLocal = 62017;
static unsigned short PuertoRemoto = 60000;
```

Se define un buffer de transmisión de datos de tamaño totalmente parametrizable:

```
static Xuint32 TxFrame[XEM_MAX_FRAME_SIZE_IN_WORDS];
```

Por ultimo se crea una instancia del driver controlador de ethernet:

```
XEmac Emac;
```

Tras estas declaraciones iniciales y ya en el programa main se realizan lecturas de la memoria y se forman los datagramas para su posterior envío.

El envío del datagrama una vez formado se lleva a cabo llamando a una función (sendFrame) proporcionada por xilinx en la librería xemac.h.

El programa principal realiza una espera activa sobre el registro de estado del gestor de estadísticas que indicará cuando la memoria está llena.

```
int main()
{

        unsigned int volatile *status_register;
        unsigned int *st;
        int i, j;
        unsigned int result;
        int valor;
```

```
        InitEth(); //Inicializa el controlador de ethernet

        // Dirección del status register
        status_register =(unsigned int *) XPAR_ STATISTICS_ MANAGER_ 0_
        BASEADDR;

        //Dirección de inicio de la memoria de estadísticas
        st =(unsigned int *) XPAR_STATISTICS_MANAGER_0_AR0_BASEADDR;
        sleep(10);


        while(1)
        {
                while ( (*status_register) != 0x00000001 );
                result = SendFrame(&Emac, 1024, DestinationAddress, st);
                *status_register = 0;
                for (i=0;i<XEM_MAX_FRAME_SIZE_IN_WORDS;i++)
                                TxFrame[i] = 0;
        }

        return 0;

}


static XStatus SendFrame(XEmac *InstancePtr, Xuint32 Size,
                        Xuint8 *DestAddress, unsigned int *Data)
{
        unsigned int FrameSize;
        unsigned char *FramePtr;
        unsigned char *AddrPtr = DestAddress;
        int i;

        …..

        FrameSize = CrearFrameUDP (FramePtr, Data, Size, IPLocal,
        PuertoLocal, IPRemota, PuertoRemoto);

        // Send
        return XEmac_PollSend(InstancePtr, (Xuint8 *)TxFrame, FrameSize);

        ….
}
```

La creación de paquetes UDP se encuentra englobada en la siguiente función:

```
unsigned int CrearFrameUDP(unsigned char *Frame, unsigned int *Data,
unsigned int Size,  unsigned char *SrcIP, unsigned short SrcPort, unsigned
char *DstIP, unsigned short DstPort)
{
        unsigned char *FramePtr;
        unsigned char *AddrPtr;
        unsigned short *chkPtr;
        unsigned short *hdrPtr;
        unsigned long checksum;
        int i;

        FramePtr = Frame;

        // Create the IP Header
        hdrPtr = (unsigned short *)FramePtr;
        *FramePtr++ = 0x45; // Vers, HLen
        *FramePtr++ = 0x0; // Service Type
        *((unsigned short *)FramePtr) = Size+20+8; // Total Length
        FramePtr+=2;
        *((unsigned short *)FramePtr) = 517; // Identification
        FramePtr+=2;
        *((unsigned short *)FramePtr) = 0; // Flags, Fragment offset
        FramePtr+=2;
        *FramePtr++ = 0xff; // Time To Live
        *FramePtr++ = 0x11; // Protocol
        chkPtr = (unsigned short *)FramePtr;
        *((unsigned short *)FramePtr) = 0; // CheckSum
        FramePtr+=2;
        AddrPtr = IPLocal; // Source IP
        for (i=0;i<4;i++)
                *FramePtr++ = *AddrPtr++;
        AddrPtr = IPRemota; // Destination IP
        for (i=0;i<4;i++)
                *FramePtr++ = *AddrPtr++;

        checksum = 0;
        for (i=0;i<20;i++, hdrPtr++)
                checksum = checksum + *hdrPtr;
        checksum += checksum >> 16;
        checksum = (~checksum) &0xffff;
        *chkPtr = (unsigned short)checksum;

        // Create the UDP Header
        *((unsigned short *)FramePtr) = PuertoLocal; // Source Port
        FramePtr+=2;
        *((unsigned short *)FramePtr) = PuertoRemoto; // Destination Port
```

```
        FramePtr+=2;
        *((unsigned short *)FramePtr) = Size+8; // Length
        FramePtr+=2;
        *((unsigned short *)FramePtr) = 0; // CheckSum
        FramePtr+=2;

        // Data
        for(i=0, AddrPtr = (unsigned char *)Data; i<Size ; i++)
        {
           *FramePtr++ = *AddrPtr++;
        }

        return 20 + 8 + Size + XEM_HDR_SIZE;
}
```

Esta función rellena aquellos campos del paquete udp que no son rellenados automáticamente por el controlador.

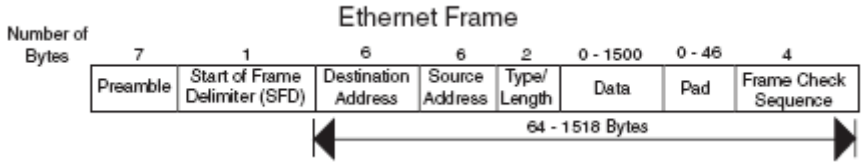


Figura nº 48: Formato trama ethernet

Por otra parte, una vez conseguidas las estadísticas y enviadas vía Ethernet nos encontramos con la necesidad de una aplicación que recogiera dichas estadísticas y nos diera la posibilidad de observar los resultados durante la ejecución a fin de poder evaluar el comportamiento del sistema.

Una posible opción era hacer uso de la herramienta "Ethereal" que captura los paquetes de la red y los puede almacenar en un archivo. Esta opción fue descartada porque no permitía la evaluación en tiempo real y por la cantidad innecesaria de datos que se introducía en el archivo en el que se almacenan las estadísticas pues guarda no sólo la información de los paquetes sino también sus cabeceras completas, información que no deseamos por tener una conexión punto a punto con la placa.

Es por esto que decidimos desarrollar una pequeña aplicación que satisficiera nuestras necesidades, aprovechando para ella las facilidades que proporciona la plataforma .NET para el uso de sockets, hilos y semáforos con la potencia de C pero con un entorno de

desarrollo amigable, haciendo muy simple la creación y uso de formularios.

La aplicación cuenta con dos formularios. En el primero seleccionamos el tipo de sniffer que corresponde con cada identificador y los bits que utilizan para la dirección. Este segundo parámetro no se utiliza en esta versión, aunque queda preparado para una versión siguiente que muestre gráficas en tiempo real.

Al pulsar en el botón de iniciar se llama al segundo formulario. En él se muestran los contadores más interesantes en función del tipo de sniffer, actualizándose sus valores cada medio segundo.

Internamente en el constructor de la ventana de mostrar las estadísticas se llama a un método que abre un socket que lee paquetes UDP en el puerto 6000, que es al que enviamos la información desde la placa, y crea un hilo que cuando hay datos los recibe y almacena en un array, que empleamos a modo de buffer así como en un archivo para procesos posteriores.

Cada cierto tiempo se comprueba si hay nuevos datos en el buffer y si los hay se procesan, organizando la información por sniffers y actualizando sus contadores.

Todos los accesos al buffer están arbitrados por un semáforo con el que evitamos que se produzcan fallos en la coherencia de los datos.

4.3 Gestión del reloj en la plataforma de emulación

La señal de reloj para la plataforma de emulación es generada por el subsistema de estadísticas. Esta característica permite a los controladores de memoria simular las latencias de reloj que los diseñadores deseen emular.
Además facilita el tiempo necesario al gestor de estadísticas para recolectar los datos necesarios y enviarlos.

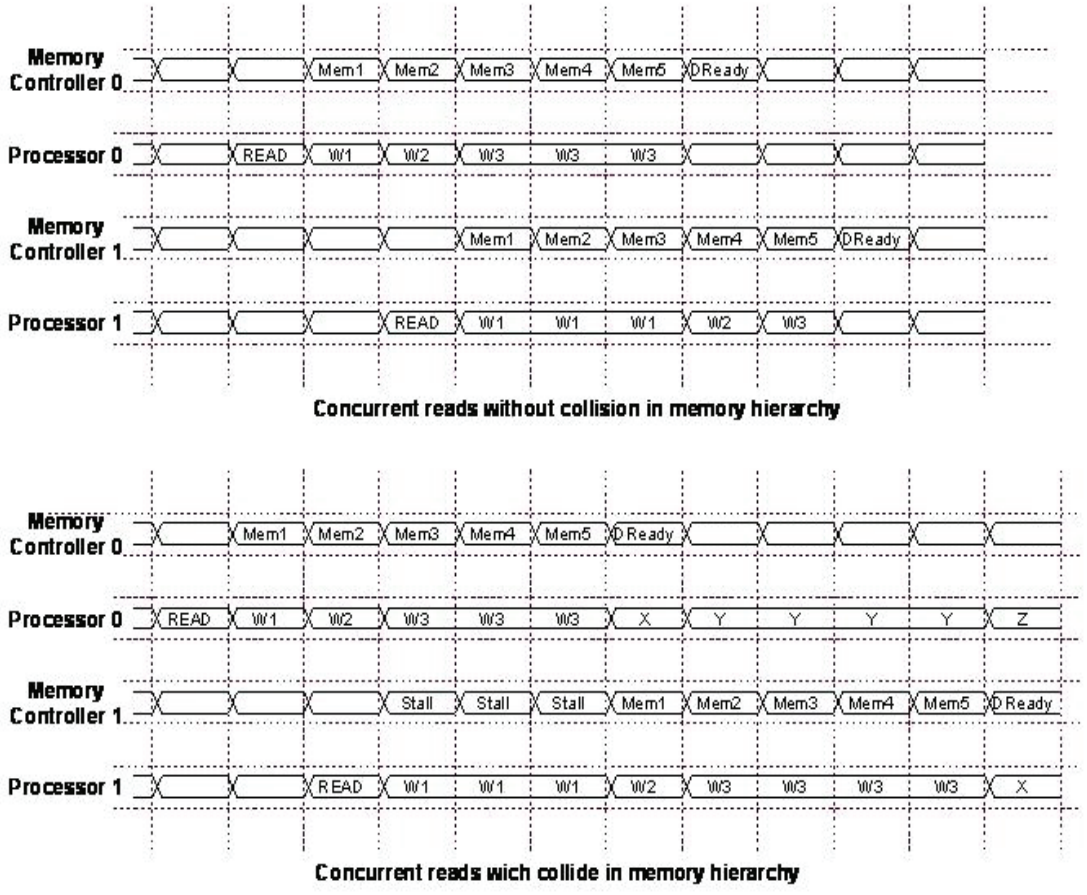


Figura nº 49: Cronograma de accesos concurrentes a la jerarquía de memoria (con/sin colisiones)

La generación del reloj en la plataforma se realiza mediante el siguiente esquema hardware:

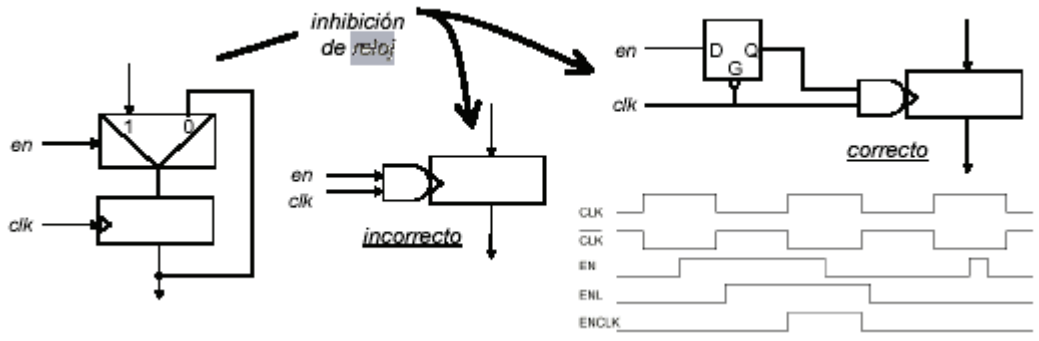


Figura nº 50: Generación reloj plataforma

4.4 Conclusiones

Sobre el subsistema de procesamiento de la plataforma de emulación se construye un subsistema de extracción de estadísticas capaz de observar determinados puntos clave del sistema y extraer datos significativos de ellos.

El principio que se ha seguido en el diseño e implementación de este subsistema de extracción de datos estadísticos es la independencia y facilidad de expansibilidad. Es decir, el sistema extractor de estadísticas puede estar en funcionamiento o no en un momento dado, y su funcionamiento no afectará a la correcta ejecución del resto de la plataforma.

Por otra parte cada uno de los sniffers del sistema pueden seleccionarse de forma que se activen unos u otros según se crea conveniente en cada momento.

En un momento dado, un diseñador de un nuevo módulo del subsistema de procesamiento sería totalmente capaz de construir el sniffer correspondiente para su módulo, sin necesidad de conocer detalles de implementación del subsistema extractor.

Por último destacar que la interfaz de extracción de datos y envío al computador para su posterior estudio elegido, permite una gran velocidad lo que garantiza que la extracción no será un cuello de botella en el sistema. Además, el software que se ejecuta en el computador conectado, permite la observación de los datos estadísticos obtenidos, así como filtros de la información recibida.

En este capítulo se ha explicado la arquitectura del subsistema de estadísticas, destacando los aspectos clave en su construcción y la interacción con el resto de la plataforma.

Capítulo 5. Implementación y verificación de la plataforma de emulación

La plataforma de emulación completa involucra la integración entre el subsistema de procesamiento y el subsistema de extracción de estadísticas.
Esta integración resulta sencilla debido a la característica de alta independencia del subsistema extractor de estadísticas. Tan sólo se mapean puertos de salida de cada uno de los sniffers del sistema con el módulo gestor.

5.1 Implementación

Desde un punto de vista global la plataforma de emulación consta de dos tipos de cores.
Cores diseñados por Xilinx que cubren una funcionalidad concreta y que se encuentran disponibles (bien a nivel de síntesis, ó bien a nivel de código fuente) en la herramienta EDK.
El otro tipo de cores son diseñados y sintetizados en este proyecto, y dan solución a funcionalidades concretas, o bien ofrecen una implementación más particular.

5.1.1 IP cores utilizados

POWERPC (HARD CORE)

Figura nº 51: Chip Power PC

- Proporciona alto rendimiento con un bajo consumo de energía.
- Repertorio de instrucciones UISA (User Instruction Set Architecture) y extensiones para aplicaciones empotradas.
- 32 registros de propósito general de 32 bits.
- Predicción estática de saltos

- Pipeline de 5 etapas, con un ciclo simple de ejecución para la mayoría de las instrucciones, incluidas las de load/store.
- Multiplicadores / Divisores hardware para acelerar las operaciones aritméticas
- Manejo de múltiples palabras
- Little endian / Big endian
- Soporte para depuración (JTAG)
- Latencia por interrupciones minimizada.
- Avanzado soporte de gestión de memoria.
- MMU (Memory Management Unit)
- Memoria cache (on-chip) para datos e instrucciones (nosotros deshabilitaremos estas memorias cache)

PLB_ETHERNET

La interfaz PLB Ethernet es un IP soft-core diseñado para ser implementado en una FPGA como las siguientes, Virtex™-E, Virtex-II™, Spartan™-II, Spartan™-IIE, Spartan™-3 o Virtex-II Pro.

El diseño de este core proporciona una velocidad de 10 Megabits por segundo (Ethernet) o 100 Megabits por segundo (Fast Ethernet). Incluye la mayoría de las funciones y la flexibilidad de la que disponen los controladores de Ethernet disponibles actualmente en el mercado.

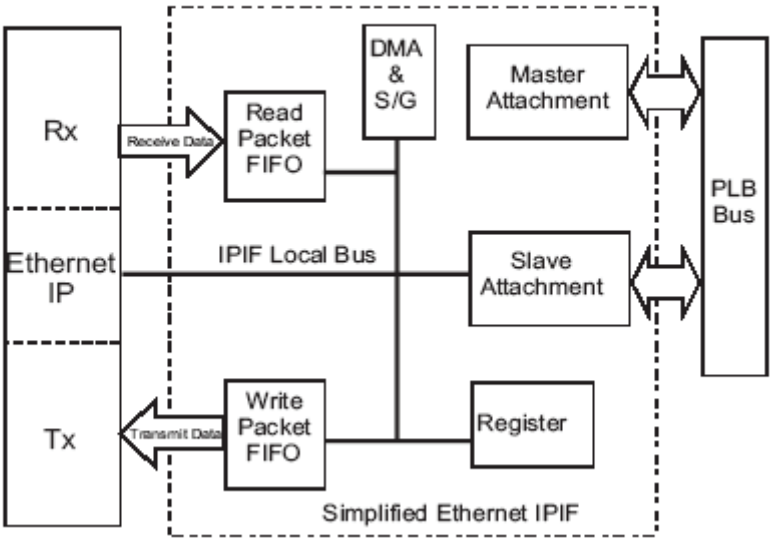


Figura nº 52: Esquema del controlador de ethernet

Principales características:

- 64-bits de interfaz con el bus PLB (maestro o esclavo).
- Interfaz de entrada/salida mapeada directamente en registros de memoria. Se puede elegir modo de funcionamiento: FIFO, DMA y Scatter/Gather DMA.
- TX y RX son buffers internos e independientes para manejar datos de más de un paquete. Su tamaño oscila entre 2K y 32K.
- Soporta modos de transmisión y recepción unicast, multicast, y broadcast.
- Proporciona facilidades de relleno de ciertos campos de la trama a transmitir:

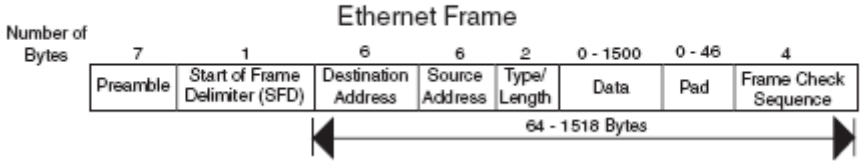


Figura nº 53: Formato trama Ethernet

Los campos siguientes son rellenados de manera automática por el core, siendo posible también su definición de modo manual:

- Preámbulo
- SFD

BRAM_BLOCK

El módulo de BRAM que proporciona Xilinx no es más que la agrupación de los distintos bloques de memoria RAM presentes en la FPGA permitiendo unirlos a distintos controladores, que permiten a los procesadores el acceso a esta memoria.

- Es configurable en tamaño y en anchura de palabra.
- Admite accesos y escrituras de tipo Byte, media palabra, palabra y doble palabra
- Un mismo BRAM puede conectarse simultáneamente a dos controladores independientes.
- Los bloques internos pueden ser de 4kb o 16kb dependiendo de la arquitectura, con una anchura máxima de 32 bits. Para conseguir una anchura de palabra de 64 bits se colocan dos bloques interno en paralelo .

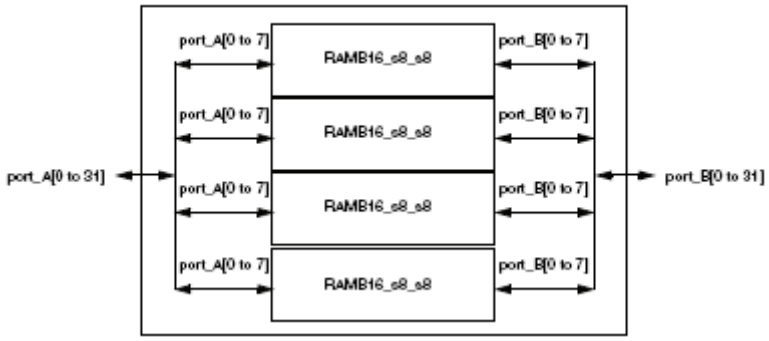


Figura nº 54: Ejemplo bloque BRAM

PLB_BRAM_IF_CNTR

Se trata de un módulo que conecta el interfaz de BRAM anterior al bus PLB. Soporta el uso de los habilitadores de byte (Byte enable) tal y como se contempla en la versión 3.4 de la implementación de este bus.

- Al igual que el interfaz admite transferencias de tamaños byte, media palabra, palabra y doble palabra.
- Admite tanto lecturas como escrituras en modo ráfaga, aunque este tipo de accesos tienen que ser de 64bits cada uno.
- Dispone de un modo de acceso, preparado para estar conectado a una cache, ya que permite acceder/reemplazar bloques, mandando las operaciones una detrás de otra sin esperar a que haya finalizado la primera.

PROC_SYS_RESET

Este módulo permite al diseñador desarrollar un sistema que permita habilitar y deshabilitar los módulos necesarios.

Las características que presenta son:

- Sincroniza la señal principal de reset y otra auxiliar con el reloj.
- Se puede decidir si las señales de reset son activas a alta o a baja. Lo único, ambas deben seguir el mismo criterio.
- Permite la selección de la mínima anchura de pulso necesaria para que se reconozca la señal, con lo que se evitan los glitches.

PLB2OPB_BRIDGE

Este puente realiza en el bus OPB las peticiones que se realizan en el PLB dirigidas a módulos conectados a este primer bus. Para ello se conecta en modo esclavo al PLB y maestro al OPB.

Sus principales características:

- Peticiones de 32 ó 64 bit de datos
- Permite hacer accesibles desde el PLB hasta cuatro rangos de memoria
- Permite accesos de ráfagas, aunque puede eliminarse por medio de los parámetros.

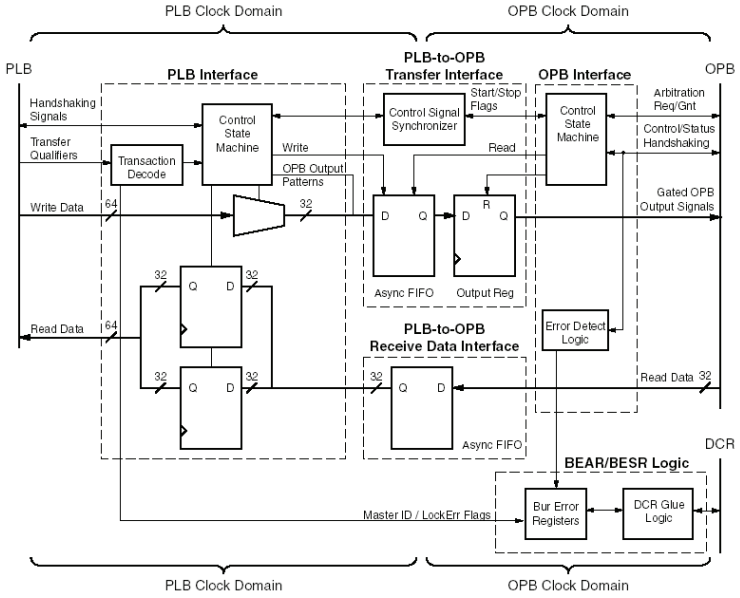


Figura nº 55: Esquema del puente plb-opb

DCM_MODULE

El administrador de reloj digital (Digital Clock Manager) es un elemento ya presente en la FPGA, siendo el módulo empleado, al igual que en el caso del BRAM block, un interfaz que facilita su uso así como su integración en el entorno del EDK. Se emplea para estabilizar la señal de reloj, permitiendo que llegue el mismo reloj a los distintos módulos de nuestro sistema. Así mismo permite desfasar la señal de reloj e insertar

buffers a las distintas salidas de reloj para que sea posible conectarlas a memorias externas.

No va conectado a ninguno de los buses.

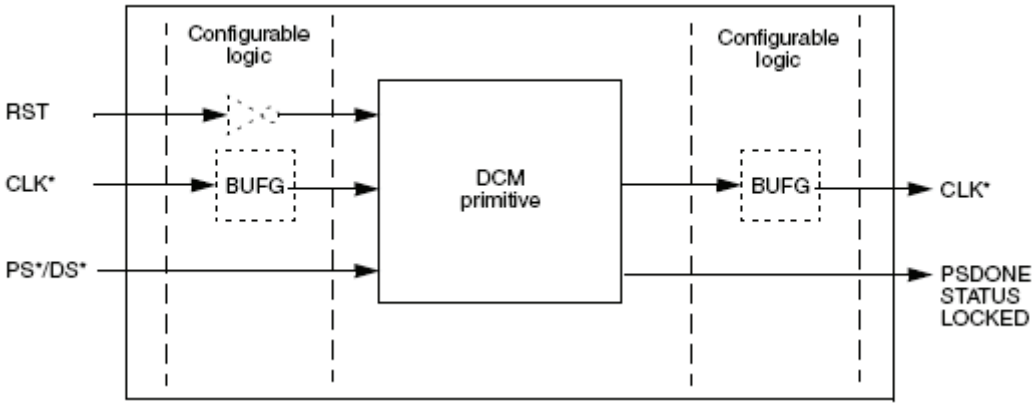


Figura nº 56: Esquema del módulo dcm

OPB_UARTLITE

Este módulo nos proporciona un interfaz para la comunicación por el puerto serie con un pc. Permite visualizar información por la pantalla del mismo.

Sus principales características son:

- Se conecta al bus OPB, respondiendo a las peticiones atendiendo las señales que habilitan los bytes (Byte enable).
- Está diseñado para funcionar conectado a buses de 8 bits de tamaño de palabra.
- Realiza una comunicación full duplex con un canal de transmisión y otro de recepción.
- Tiene dos colas de 16 caracteres, una para la transmisión y otra para la recepción
- El número de bits de los caracteres es configurable y puede ir de cinco a ocho.

5.1.2 Módulos sintetizados

Scratchpad:

Se trata de una memoria accesible desde el bus PLB, cuyos rangos de direcciones no se solapan con los de la memoria principal, de tal manera que estas dos memorias no están conectadas entre sí.
Como parámetros de la scratchpad tenemos el número de bits de dirección que se utilizan para acceder a esta memoria. Con este parámetro además podemos conocer el tamaño, ya que contamos con la siguiente constante:

```
NUM_PALABRAS : integer := 2**BITS_DIRECCION;
```

Esta constante pertenece a un componente "memoria" que empleamos en todos los módulos de memoria.
Otros parámetros son los ciclos que tarda en realizar una operación de lectura o una de escritura.

Como señales de entrada y salida tenemos, además de las propias del bus PLB al que está conectado, una señal de entrada "Vclken" que si está activa permite el uso de la plataforma y Vclksup, de salida, que inhibe los ciclos virtuales, caso que no haya dado tiempo a realizarse la petición en los ciclos que vienen determinados.

Además el módulo cuenta con las señales que se interconectan con el sniffer, pudiendo permanecer al aire siempre y cuando no se quiera realizar un estudio sobre este módulo.

Memory Controller:

Este módulo es el encargado de atender las peticiones a memoria que realiza el procesador.

El módulo que tenemos consta del controlador de memoria y de la memoria cache. El controlador discrimina entre las peticiones cacheables y las no cacheables, mandando unas a la cache y otras directamente a la memoria principal como se ve en el código siguiente. Las direcciones cacheables y no cacheables están diferenciadas por pertenecer a rangos distintos:

```
if ( PLB2C_IsCacheableAddr = '1' ) then
        -- Leemos el directorio
        -- La direccion de la petición aún no ha sido registrada
        D_Addr <= PLB2C_Addr(DIR_WIDTH-DIR_WIDTH_CACHE-2 to
DIR_WIDTH-LOG2_PALS_PER_BLOCK-3);
        D_Req <= '1';
        D_RnW <= '1';
        -- Fin de lectura del directorio
        nextState <= WAIT_READ_DIRECTORY;
else
        -- La petición va directamente a MP
        non_cacheable_access <= '1';
        MPnC_Data2PLB <= '1';
        nextState <= WAIT_MP_NON_CACHEABLE_ACCESS;
end if;
```

En el caso de que la palabra esté en una zona cacheable, se trata como un acceso a cache, consultando el directorio, determinando si ha habido fallo o acierto… Si el acceso fue a una zona no cacheable la petición va directamente a memoria principal.

Cache Controller:

En realidad está integrado en el módulo controlador de memoria, pero conceptualmente tiene más sentido definirlo como un ente independiente.

Se trata de una cache directa, con write-through y write-no-allocate. Consta de la propia memoria cache y de un directorio, donde guarda información de ubicación de los bloques. Cuando recibe un acceso consulta el directorio, determinando así si se produjo un fallo o un acierto.

Main memory controller:

Responde a peticiones tanto de la cache como del controlador de memoria (accesos a zonas no cacheables).
En este fragmento vemos las condiciones para pasar a uno u otro estado.

```
if ( C2MP_IsCacheableAddr = '0' ) then
        nextState <= WAIT_MP_NON_CACHEABLE_ACCESS;

elsif ( C2MP_RnW='0' ) then
        nextState <= WAIT_WRITE;
else
        nextState <= WAIT_READ;
end if;
```

Con la memoria cache intercambiará bloques (lecturas, recordemos que se trata de una cache con write-no-allocate) o palabras (escrituras, cache con write-through), mientras que si el acceso era directo desde el procesador sólo podrá ser a nivel de palabras.

Virtual Platform Clock Manager

Se trata del módulo gestor del reloj virtual de la plataforma. Permite parar el reloj virtual cuando a algún módulo no le ha dado tiempo a responder en el tiempo especificado.

Como entradas tiene las señales de supresión que le vienen de todo módulo con capacidad para parar el reloj: ScratchPad, statistics manager, sniffers...

Como salidas emite una señal de "virtual clk enable" que indica cuándo está habilitado el reloj de la plataforma de emulación (valor '1' de la

señal). También distribuye el "reloj virtual" que no es sino una señal de reloj con el mismo periodo que el reloj del oscilador, pero que no emite pulsos cuando el reloj virtual está inhabilitado.

Una situación de inhibición del reloj es, por ejemplo, cuando se llena la memoria de estadísticas. El virtual platform clock manager detiene el reloj virtual hasta que las estadísticas son extraídas y la ejecución puede continuar.
Un detalle particularmente interesante es la generación del reloj virtual.
En el código VHDL, además de hacer pasar la señal por un "clock buffer" (necesario para asegurar la correcta distribución de una señal de reloj) se buscó un mecanismo que evitara glitches en la señal, pues cualquier glitch generaría un flanco de reloj no deseado. Se muestra un esquema de la solución adoptada:

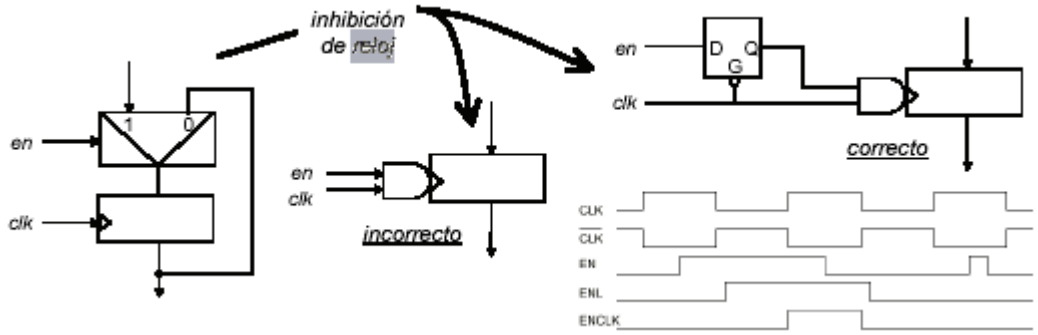


Figura nº 57: Inhibición del reloj

5.1.3 Entorno de desarrollo

Todo el desarrollo de la plataforma hardware se ha realizado utilizando la herramienta EDK, tal y como hemos mencionado anteriormente, que proporciona las bases necesarias para el interconexionado de IP cores y procesadores por medio de los buses que, bajo las especificaciones de IBM, proporciona Xilinx o mediante líneas dedicadas definidas por el diseñador.

Todos estos pasos se realizan Pulsando "Add/edit cores" en el menú de proyecto.

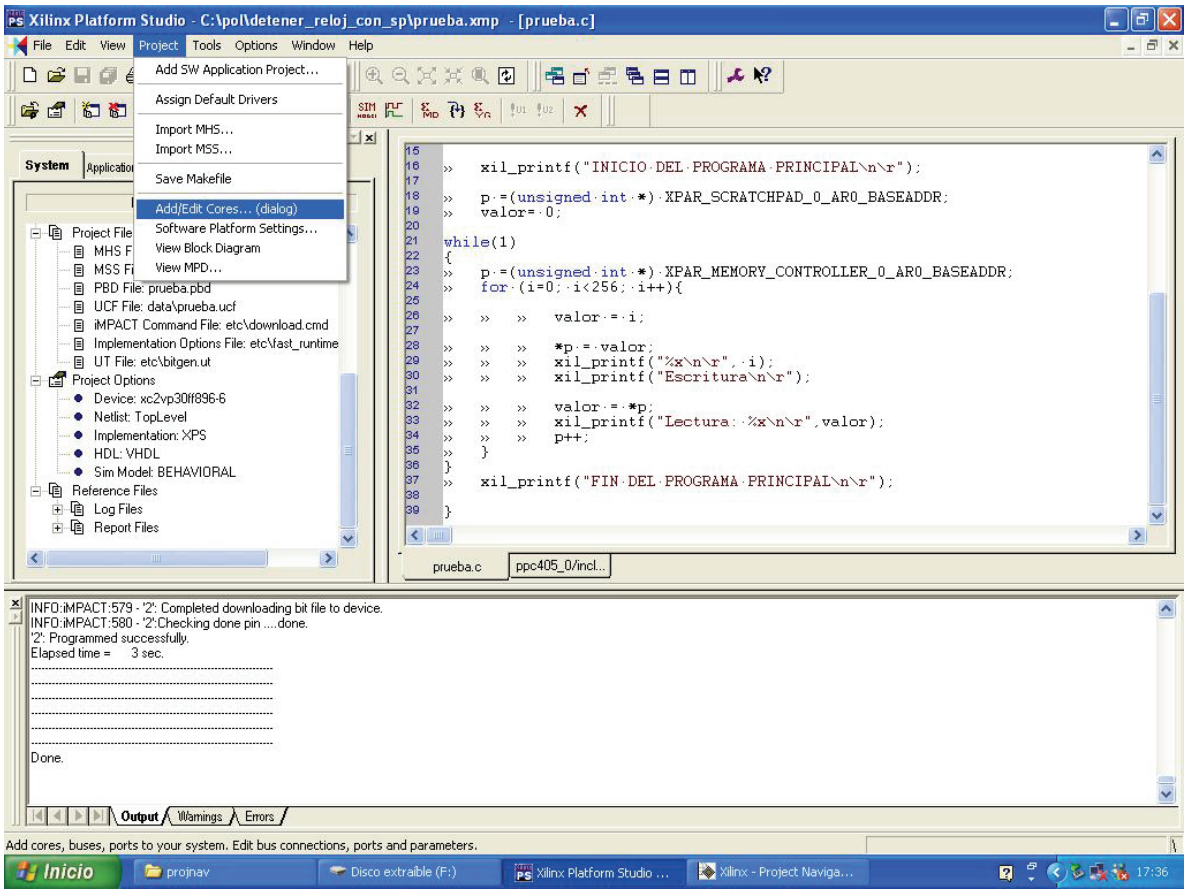


Figura nº 58: Entorno de desarrollo

Una vez pulsada esta opción aparece una pantalla auxiliar con cinco pestañas: "peripherals", "bus connections", "addresses", "ports" y "parameters". Cuando se completen estas fases, tendremos un proyecto listo a falta de los códigos que tengan los procesadores y del archivo de restricciones para bajar a la placa y empezar a funcionar. Añadir que todos estos pasos también se realizan en el marco de esta herramienta.

Pasaremos, a continuación, a explicar brevemente en qué consisten las diferentes pestañas de la opción de "Add/edit cores".

"Peripherals": En un primer momento se deciden los periféricos que va a tener el proyecto, estableciendo qué versión del mismo va a emplearse. Para importar los periféricos diseñados para el proyecto se acude previamente a la opción "Create/import peripheral" accesible desde el menú de herramientas.

Si por modificaciones en el sistema fuera necesario eliminar algún periférico, se realizará la operación desde la ventana que se muestra a continuación:

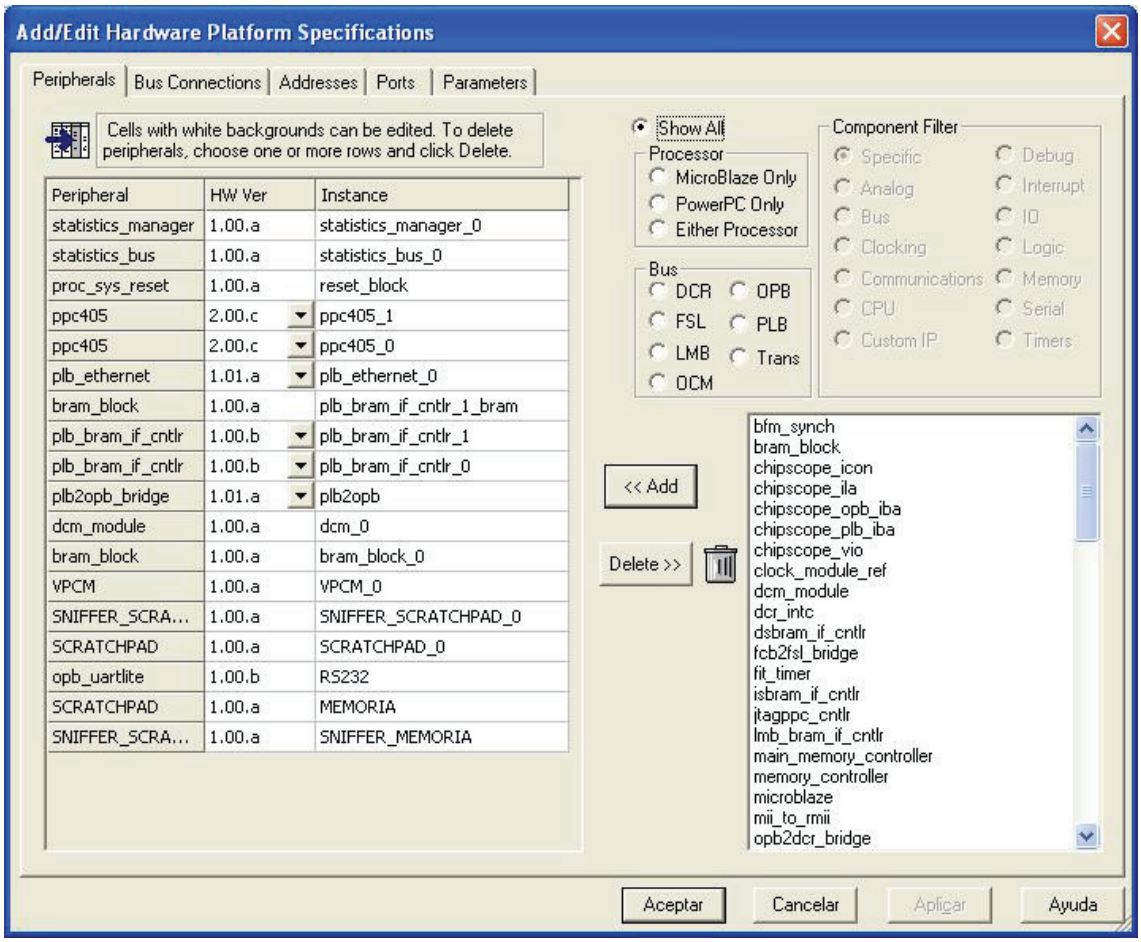


Figura nº 59: Entorno de desarrollo (Periféricos)

Para consultar las conexiones de los módulos existentes en la plataforma generada con los buses basta con visualizar la pestaña de "bus connections". También desde ella incluimos los buses del sistema así como las conexiones entre los bloques de memoria y sus controladores. Por otra parte, también desde esta interfaz podemos habilitar la depuración mediante hardware por medio de la asignación del jtag a los procesadores correspondientes.

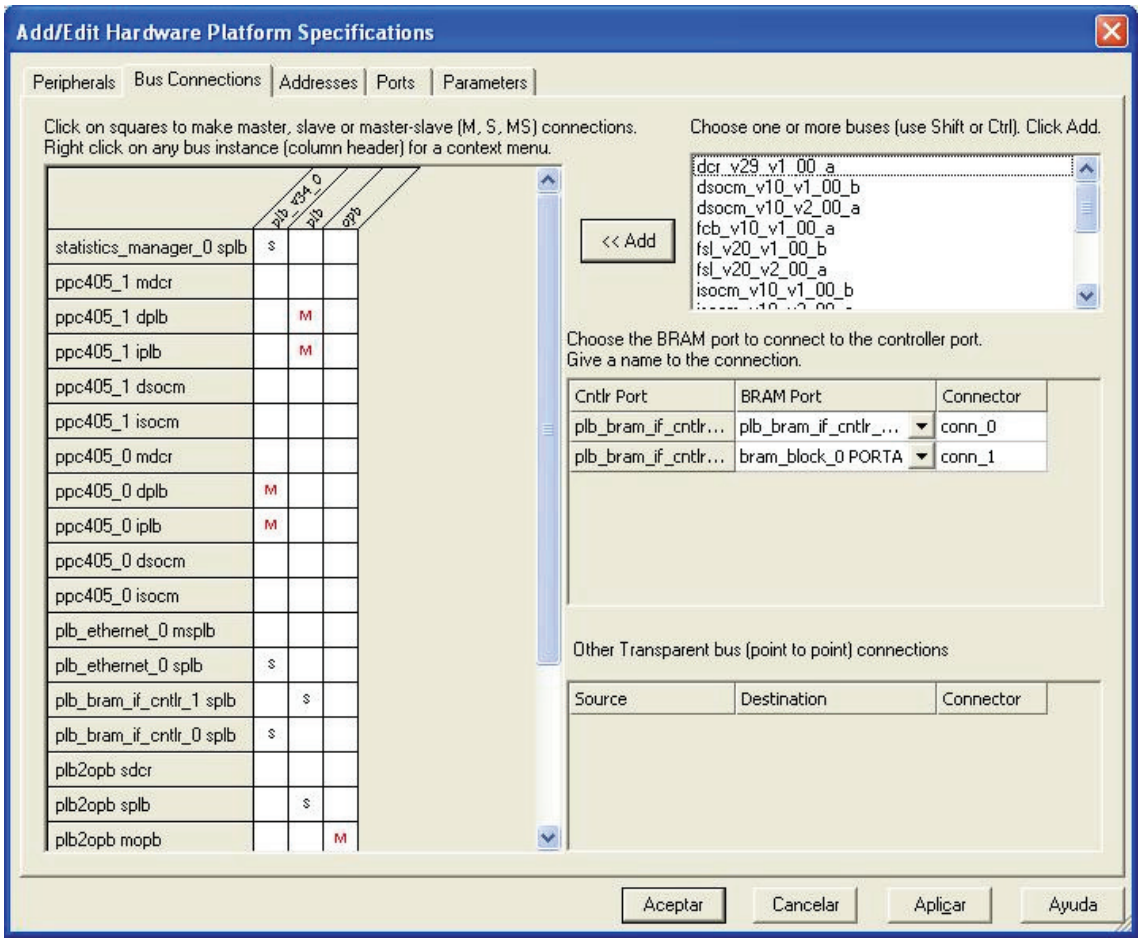


Figura nº 60: Entorno de desarrollo (Conexiones de bus)

Tal y como se puede observar en la captura, la plataforma realizada está compuesta de tres buses (dos buses PLB y un bus OPB).

Para visualizar el mapeo de cada uno de los periféricos en la memoria debemos acceder a la pestaña "addresses" y vemos lo siguiente:

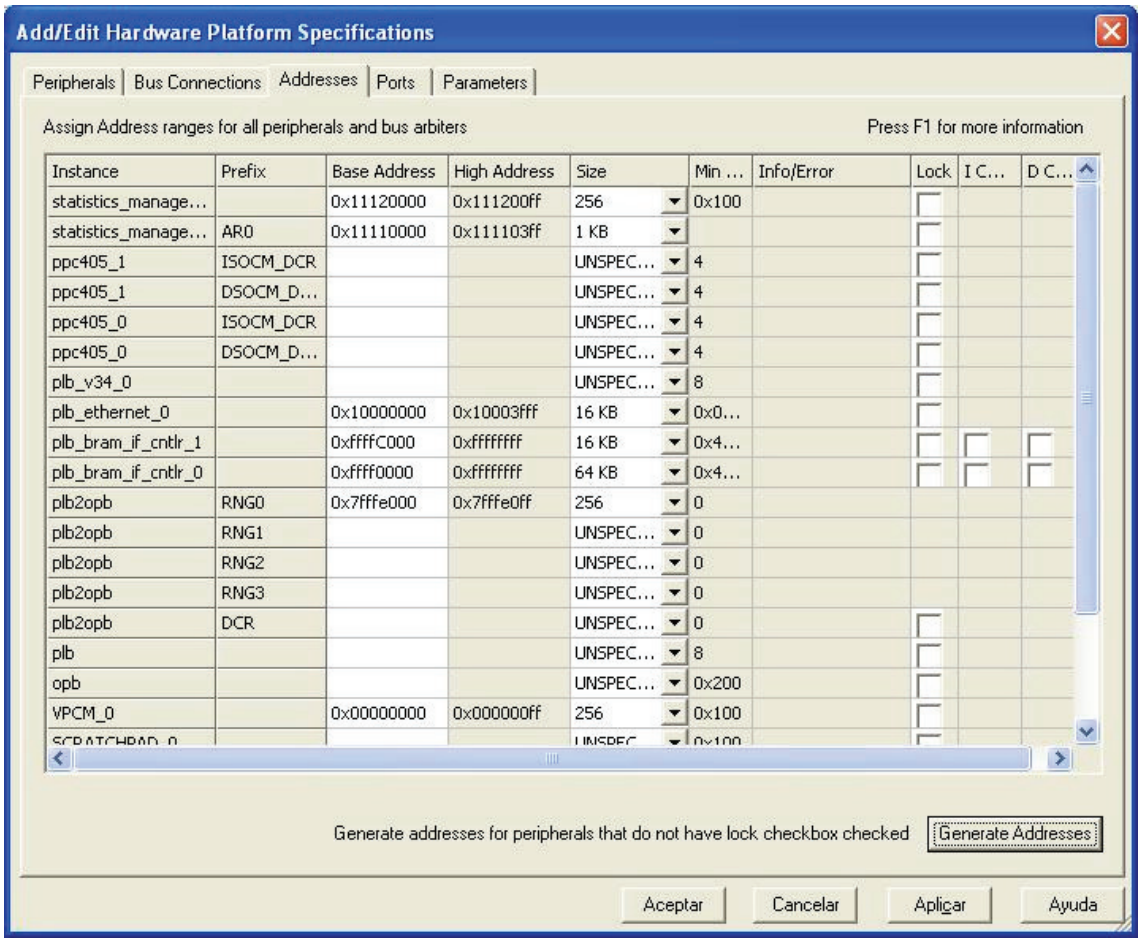


Figura nº 61: Entorno de desarrollo (Direcciones)

Por último se pueden definir y modificar los parámetros de los periféricos así como conectar sus puertos tal y como sea conveniente para el propósito que se desea alcanzar.

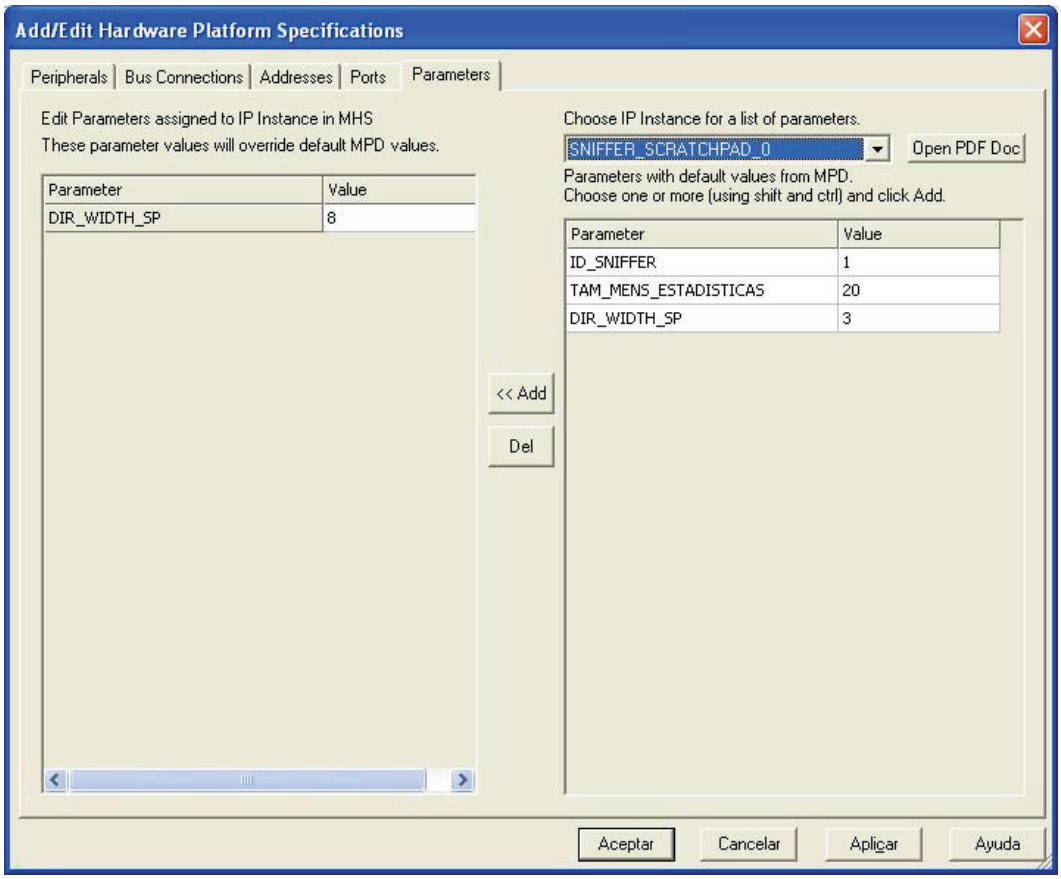


Figura nº 62: Entorno de desarrollo (Parámetros)

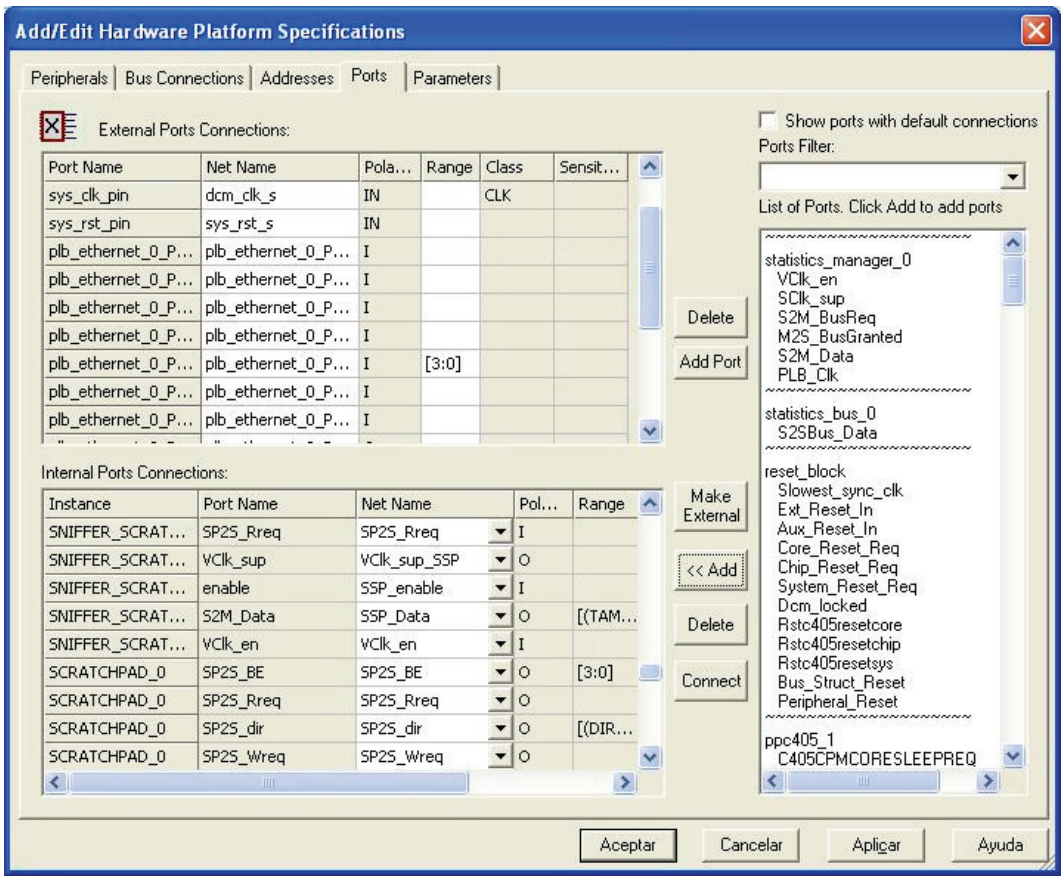


Figura nº 63: Entorno de desarrollo (Puertos)

5.2 Verificación

La verificación de la plataforma de emulación es una fase vital para comprobar el correcto funcionamiento de la misma, así como ponerla en práctica para obtener resultados y poder sacar conclusiones del estudio de la jerarquía de memoria de los procesadores empotrados, que realizamos en este proyecto.

Proceso de simulación

El proceso de simulación puede descomponerse en dos subetapas fundamentales:

- Compilación o análisis de código (comando vcom): se encarga de comprobar que el código VHDL es sintácticamente correcto y de generar una serie de ficheros necesarios posteriormente para el correcto funcionamiento del entorno.
- Simulación de código: que permite simular cualquier sistema descrito en VHDL una vez se haya analizado. Incluye un visor de formas de onda (wave) que presenta de forma gráfica la simulación.

La metodología del proceso de simulación se observa en el siguiente esquema:

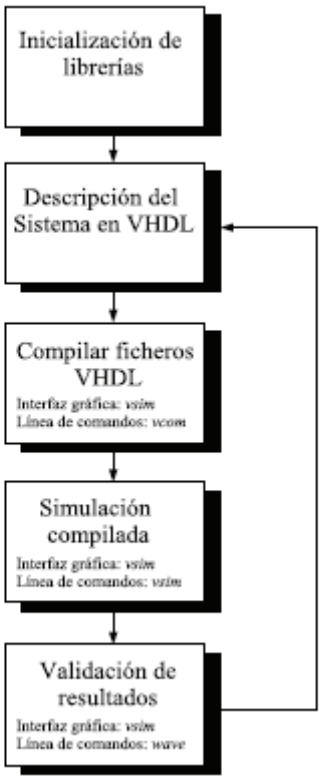


Figura nº 64: Proceso de diseño hw

Proceso de síntesis:

Se entiende por síntesis el proceso de obtención de un circuito realizado en una determinada tecnología de diseño (ASIC, FPGA, etc) a partir de su especificación en HDL. Sin embargo, la nomenclatura difiere ligeramente por parte de cada fabricante. En concreto, Xilinx distingue las siguientes fases en el flujo de diseño:

Síntesis:

Convierte el código HDL en una netlist de puertas lógicas genéricas (incluyendo Flip-Flops). El formato de esta netlist puede ser EDIF o NGC para las herramientas de implementación de Xilinx. El proceso se puede subdividir en 2 pasos:

- Análisis y comprobación de sintaxis del código HDL.
- Síntesis propiamente dicha o compilación: traduce y optimiza el código HDL en una red de puertas.

Traducción:

Realiza las siguientes funciones:

- Procesa las netlists de los distintos módulos del diseño y las integra en una netlist unificada, describiendo toda la lógica del diseño así como las restricciones de ubicación y temporización introducidas por el diseñador.
- Realiza comprobaciones de reglas de diseño lógico y especificación temporal

Mapeado:

En esta fase el diseño es convertido a CLBs e IOBs, que son los recursos físicos de la FPGA.

Asigna recursos CLB e IOB para todos los elementos lógicos que integran el diseño.
Procesa todas las restricciones de ubicación y temporización, realiza optimizaciones específicas del dispositivo y lleva a cabo una comprobación de las reglas de diseño sobre la netlist mapeada.

Colocación y rutado:

Esta etapa se encarga de ubicar los distintos CLBs en posiciones óptimas a lo largo y ancho de la FPGA, así como de encaminar las redes lógicas a través de las rutas físicas pre-existentes en la FPGA. Los algoritmos de colocación y rutado pueden opcionalmente ejecutarse de forma que tengan en cuenta las restricciones temporales incluidas en la netlist.

Generación del bitstream:

Esta es la fase final del diseño, en la cual se genera el archivo binario que habremos de cargar en la FPGA. Almacena la información de configuración para los CLBs y matrices de conexión que utilice nuestro diseño.

El proceso de síntesis de un diseño sobre una FPGA es el siguiente:

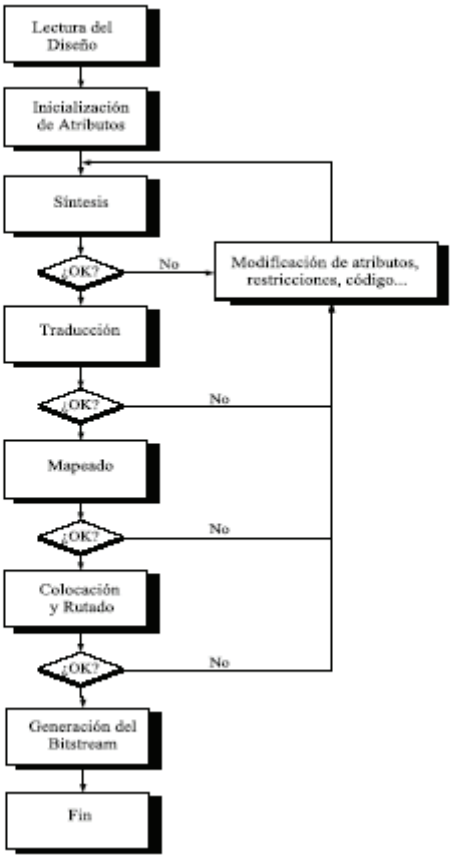


Figura nº 65: Proceso de síntesis

5.2.1 Simulación del sistema completo

Una vez desarrollado el sistema completo, esto es, el subsistema de emulación y el de estadísticas pasamos a realizar su simulación. Para ello empleamos un script que genera el EDK para el ModelSim y que contiene un modelo del sistema especificado, incluyendo procesadores, memorias de instrucciones así como todos los periféricos desarrollados y los IP cores.

En la página siguiente veremos un ejemplo del sistema cuando se realiza una petición de escritura sobre la scratchpad.
Esta petición es recogida por su sniffer que solicita el uso del bus de estadísticas y cuando se le concede envía los datos que ha sacado del módulo.

Una vez llevados los datos al módulo de estadísticas este los procesa y añade los bits que indican el identificador del sniffer que ha enviado el paquete así como el ciclo en que éste se ha recibido.

Vamos a ver cómo todo lo explicado se cumple a nivel de señales.

Scratchpad:

En un momento determinado la señal de Bus2IP_wrReq se pone a alta, lo que significa que hay datos en el bus listos para ser escritos en la dirección especificada en el vector Bus2IP_Addr, en este caso se trata de "FFFC0000". Los datos a escribir vienen determinados por Bus2IPArData ("FFFF0000"), teniendo en cuenta que los bytes que se escriben vienen determinados en la señal Bus2IPArBE, eliminada en esta simulación por ser igual a otra que extrae el sniffer. Una vez realizada la operación la scratchpad activa la señal de IP2Bus_ArWrAck, con lo que se indica al procesador que ha sido exitosa la escritura.

Líneas entre la scratchpad y el sniffer:

El sniffer recibe la señal Sp2s_WrReq a uno en el mismo momento en que se produce la petición de escritura. Simultáneamente en la señal de Sp2s_dir nos indica los bits menos significativos de la dirección excluyendo los dos últimos. El número de bits que enviamos es proporcional al número de palabras que tiene de capacidad la scratchpad y que queda definido por parámetro tanto en el sniffer como en la scratchpad. En este caso recibimos 8 bits, "00000011". Además recibe el Sp2s_BE, que es la señal que antes comentábamos. En esta simulación estamos accediendo a toda la palabra, ya que están activos todos bits.

Sniffer de la scratchpad:

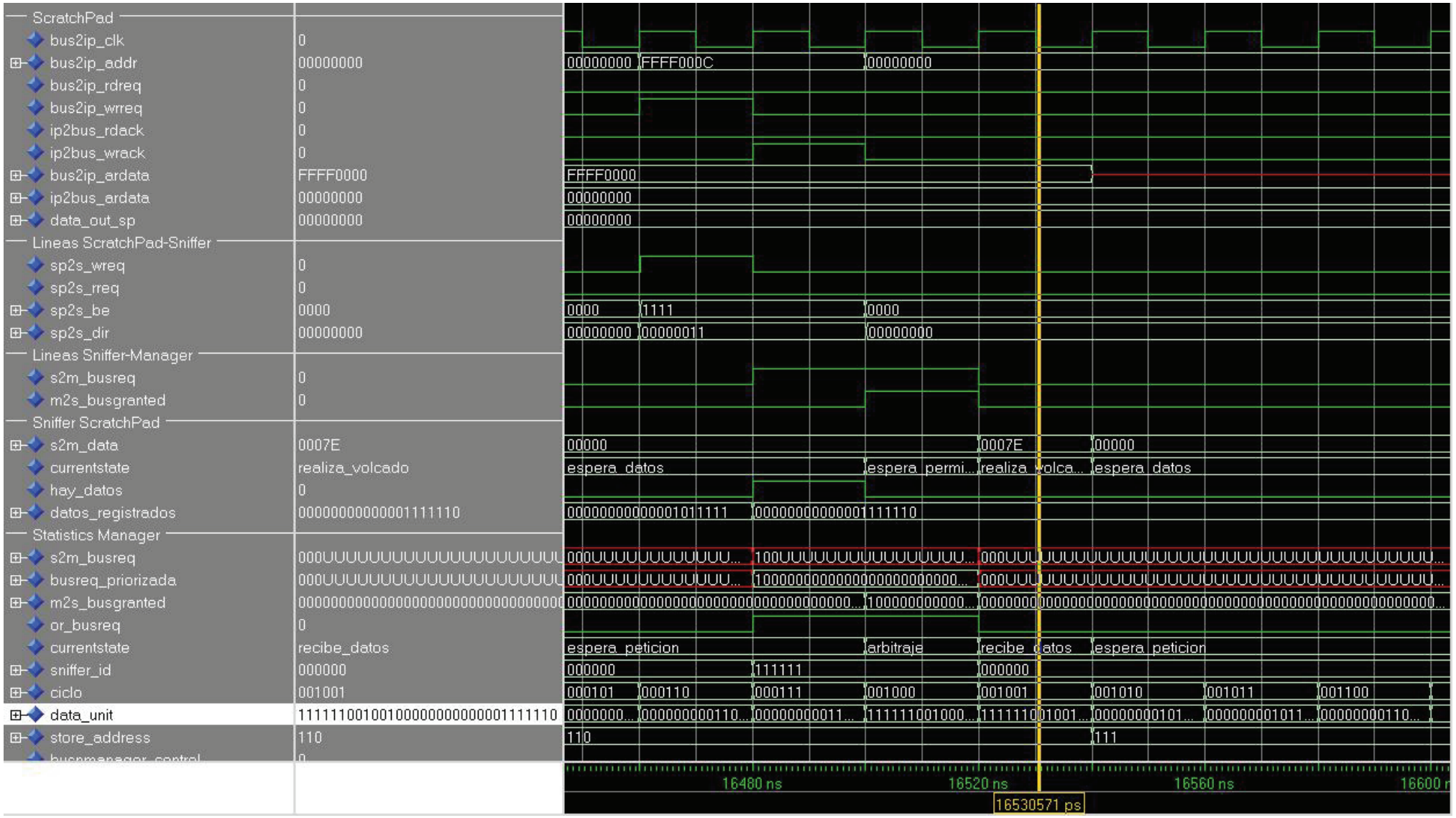
En el ciclo siguiente a haberse realizado la petición de escritura, el sniffer pone a 1 su señal de hay datos durante un ciclo, la cual provoca un cambio de estado de "espera_datos" a "espera_permiso". Una vez se le concede el permiso de bus envía un mensaje de 20 bits concatenando las señales que ha recibido de la scratchpad.

Líneas entre el sniffer y el Statistics Manager:

En el momento en que el sniffer tiene datos activa la señal de "s2m_busReq" que no baja hasta que se le concede el permiso de escritura, con el "m2s_BusGranted". Esto hace que el sniffer pase a "volcado_de_datos" y tras volcar los datos vuelve a "espera_datos"

Statistics manager:

Mientras no se reciben peticiones en el bus de estadísticas el gestor de estadísticas permanece en estado de "espera_peticion". Cuando un sniffer activa su línea de petición de bus, es gestor pasa al estado de "arbitraje" en el que concede el permiso al sniffer con id superior (y se introduce este y el ciclo en el mensaje a enviar) y en consecuencia activa únicamente la señal de Bus_Granted asociada a ese sniffer, pasando al estado de "recibe_datos" donde añade el mensaje del sniffer al mensaje del gestor y lo almacena en su memoria en una dirección propia, determinada por store_address, hasta que su memoria se llene. Es entonces cuando se realizaría el volcado de los datos al exterior del sistema.


ScratchPad
bus2ip_clk
bus2ip_addr
bus2ip_rdreq
bus2ip_wrreq
ip2bus_rdack
ip2bus_wrack
bus2ip_ardata
ip2bus_ardata
data_out_sp
Lineas ScratchPad-Sniffer
sp2s_wreq
sp2s_rreq
sp2s_be
sp2s_dir
Lineas Sniffer-Manager
s2m_busreq
m2s_busgranted
Sniffer ScratchPad
s2m_data
currentstate
hay_datos
datos_registrados
Statistics Manager
s2m_busreq
busreq_priorizada
m2s_busgranted
or_busreq
currentstate
sniffer_id
ciclo
data_unit
store_address
0
00000000
FFFF000C
FFFF0000
0000
1111
00000011
0007E
realiza_volcado
espera_datos
espera_permi...
realiza_volca...
00000
0000000000000101111
00000000000001111110
recibe_datos
arbitraje
recibe_datos
espera_peticion
000000
111111
001001
000101
000110
000111
001000
001010
001011
001100
110
111
16480 ns
16520 ns
16560 ns
16530571 ps

5.2.2 Pruebas realizadas en tiempo real

Para verificar el funcionamiento de la plataforma de emulación del subsistema de memoria de sistemas empotrados de altas prestaciones se realiza primeramente un estudio analítico, para posteriormente comparar los resultados obtenidos con los extraídos de la plataforma.

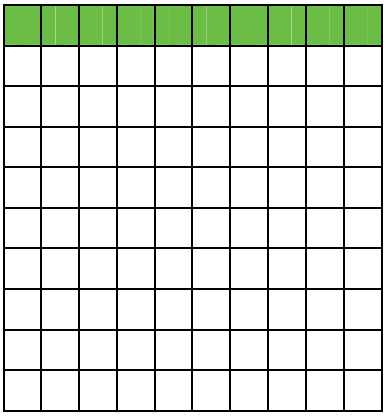
Los programas software que utilizamos para realizar las pruebas tratarán de ser siempre lo más representativos y simples posibles, de modo que sea fácil observar los resultados y sacar conclusiones.

Primer estudio realizado:

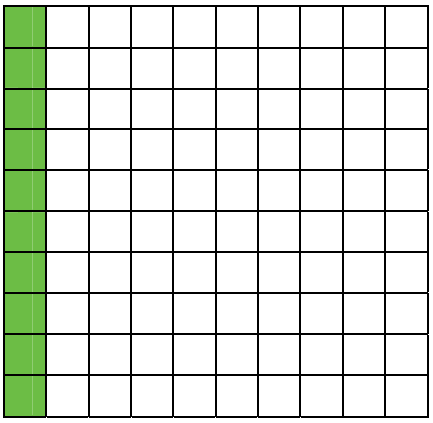
En primer lugar consideramos un sencillo y típico algoritmo de multiplicación de matrices de tamaño 10*10. Realizamos un estudio analítico de los accesos a memoria:

A10x10 * B10x10 = C10x10

A



B



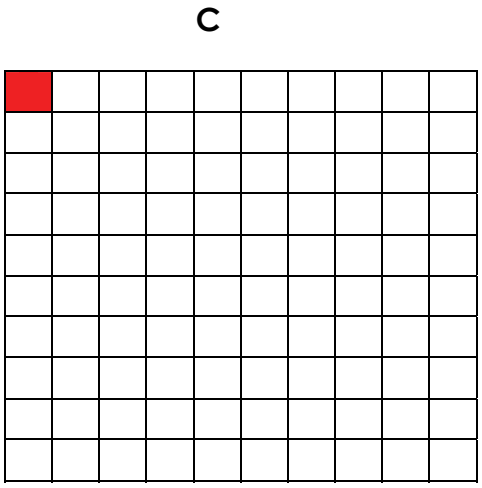


Figura nº 66: Multiplicación de matrices

En color verde se muestran los accesos de lectura a memoria principal y en color rojo se muestran los accesos de escritura. Todo esto sólo para el cálculo de la primera posición de la matriz resultado.

Consideremos primero el caso en el que todos los accesos fuesen accesos a memoria principal y realicemos el estudio sobre esta situación:
Para el cálculo de una posición de la matriz resultante C, se realizan 10+10 = 20 accesos de lectura y 1 escritura. En total 21 accesos a memoria principal para cada posición de la matriz resultado.
La matriz resultado tiene 10*10 posiciones y por lo tanto:

(20 + 1)* 100 = 2100 accesos a memoria principal.

De estos 2100 accesos, 2000 son lecturas y 100 son escrituras.
Una vez tenemos estos datos calcularemos los dos datos característicos de un sistema como éste: energía y tiempo consumido.

Para el cálculo analítico del consumo de energía tomamos valores medios de consumo por acceso cuyas fuentes se muestran a continuación:

e_mp = Energía consumida / acceso a mp = 2000 pJ/acceso.
(Datos para memoria SDRAM. Fuente: Micron Technology)

Por tanto el valor que obtenemos es el siguiente:

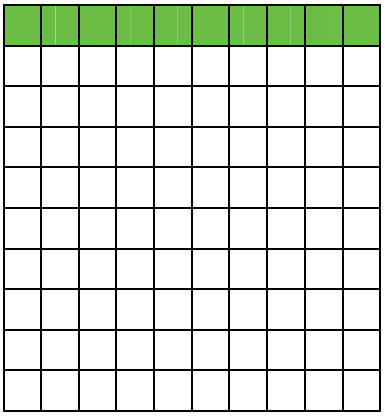
Energía = accesos_mp * e_mp = 2100 accesos_mp * 2000 pJ/acceso= 4.200.000 pJ

En cuanto al tiempo, para realizar su estudio analítico sólo tendremos en cuenta los ciclos invertidos en el subsistema de memoria.
Si se tuvieran en cuenta los ciclos totales, el resultado obtenido sería **ligeramente** superior, debido a que existen instrucciones de procesamiento de datos cuya latencia no queda totalmente oculta por la latencia de las instrucciones de acceso a memoria.

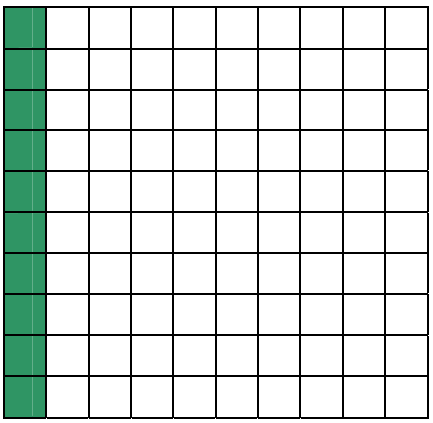
Tiempo = accesos_mp * Latencia = 2100 accesos * 5 ciclos / acceso = 10500 ciclos.

Consideramos ahora el caso en el que la segunda matriz (el multiplicador) es una matriz de coeficientes que se tiene almacenada completamente en la memoria scratchpad. Entonces distinguiremos los accesos realizados a memoria principal y los realizados a memoria scratchpad, y volvemos a realizar el estudio analítico:

A



B



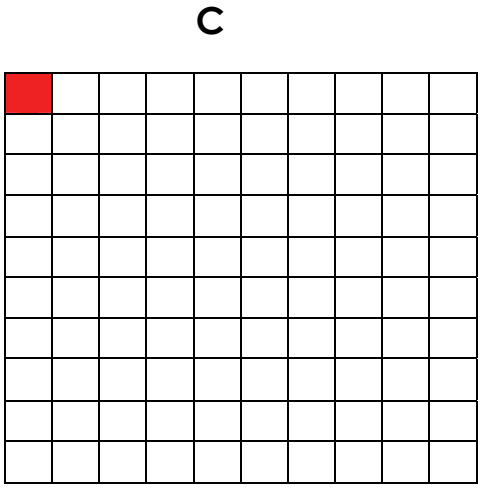


Figura nº 67: Multiplicación de matrices

En verde claro se muestran los accesos de lectura a memoria principal, en verde oscuro los accesos de lectura a la scratchpad, y en color rojo los accesos de escritura a memoria principal, todo ello para el cálculo de la primera posición de la matriz resultado.

Por tanto:

10 accesos de lectura a memoria principal.
10 accesos de lectura a memoria scratchpad.
1 acceso de escritura a memoria principal.

En total, para realizar el cálculo completo de la matriz de resultados:

(10+10+1)*100= 2100 accesos de los cuales:

- 1000 accesos de lectura a memoria principal.
- 1000 accesos de lectura a memoria scratchpad.
- 100 accesos de escritura a memoria principal.

Los cálculos de energía y tiempo se realizan de la forma siguiente:

e_sp = Energía consumida / acceso a sp.

e_sp (lectura) = 55'40 pJ
e_Sp (escritura) = 62'57 pJ
(Datos para scratchpad de simple puerto con tamaño 8KB y 32 bits. Fuente: ST Microelectronics)

Energía = accesos_ mp * e_mp + accesos_sp_lect * e_sp_lect + accesos_sp_escr * e_sp_escr.

(Hay que distinguir entre accesos de lectura y escritura a scratchpad)

Tiempo = accesos_mp * latencia_mp + accesos_sp * latencia_sp

Latencia_mp=5 ciclos
Latencia_sp=1 ciclo

Entonces:

Energía = 1000 accesos_mp * 2000 pJ/acceso + 1000 accesos_sp_lect *55'40 pJ/acceso_sp_lect + 100 * 2000 pJ/accesos_mp = 2.255.400 pJ

Tiempo = accesos_mp * latencia_mp + accesos_sp * latencia_sp = 1100 * 5 + 1000 * 1 = 6500 ciclos

En lo que a tiempo se refiere se obtiene una mejora notable:

10500/6500 = **1.6** (≈ 40 % de mejora)

En cuanto a energía:

4.200.000 pJ / 2.255.400 pJ = **1.86** (≈ 46 % de mejora)

Al ejecutar este software sobre nuestra plataforma se puede comprobar cómo se obtienen los mismos resultados lo que verifica su correcto funcionamiento. Esto se puede observar en las siguientes capturas de pantalla:

Caso en el que las dos matrices se encuentran almacenadas en memoria principal:

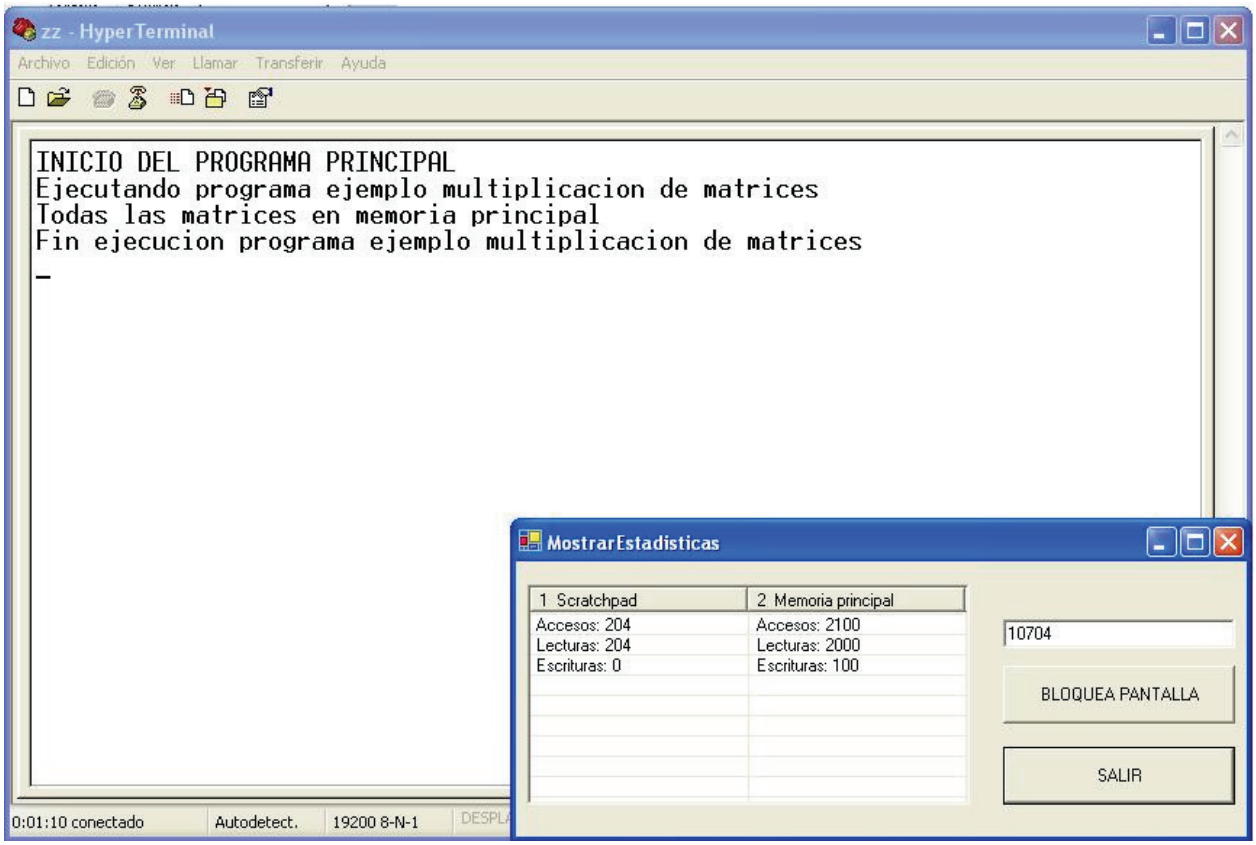


Figura nº 68: Ejecución real (2 matrices en mp)

Caso en el que la matriz B se encuentra almacenada en la scratchpad:

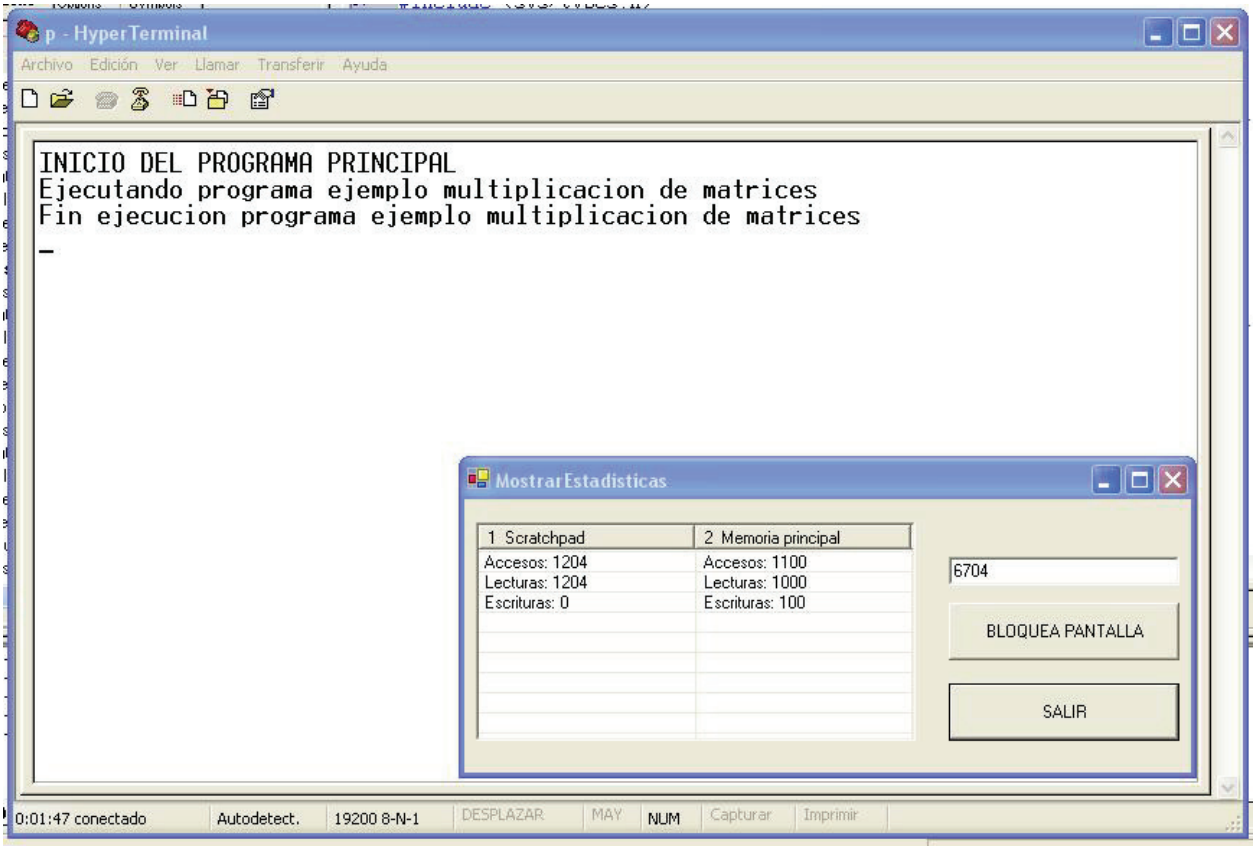


Figura nº 69: Ejecución real (matriz B en mp)

Se puede observar cómo los sniffers capturan 204 accesos de lectura en la scratchpad extras con respecto al estudio analítico. Esto es debido a que el buffer en el que almacenamos las estadísticas antes de mandarlas por ethernet, tiene un tamaño de 256 y por tanto hasta que no se llena el buffer, no se envían los datos.

El código ejecutado sobre la plataforma para esta verificación fue el siguiente:

Caso en el que todas las matrices están en memoria:

```
a =(unsigned int *) XPAR_MEMORIA_AR0_BASEADDR;
b =(unsigned int *) XPAR_MEMORIA_AR0_BASEADDR;
r =(unsigned int *) XPAR_MEMORIA_AR0_BASEADDR;
p =(unsigned int *) XPAR_SCRATCHPAD_0_AR0_BASEADDR;

for (i=0;i<10;i++)
{
        for (j=0;j<10;j++)
        {
```

```
                    acum=0;
                    for (k=0;k<10;k++)
                    {
                            at=i*10;
                            bt=k*10;
                            a_aux=a+(at)+(k);
                            b_aux=b+(bt)+(j);
                            va=*a_aux;
                            vb=*b_aux;
                            acum = acum + (va*vb);
                    }
                    *r=acum;
                    r++;
            }
    }

    //ACCESOS EXTRAS QUE SE EFECTUAN PARA LLENAR EL BUFFER
    //DE ESTADÍSTICAS Y VOLCAR LOS DATOS
    for (i=0;i<204;i++)
            j=*p;

    xil_printf("Fin ejecucion programa ejemplo multiplicacion de
    matrices\n\r");

}
```

El fichero de log registrado por el software de análisis de estadísticas es el que se muestra a continuación:

Simultáneamente al procesamiento que se puede ver por pantalla, el fichero de registro que contiene toda la información recibida de la placa se iría llenando de forma ordenada de tal forma que, con un programa de procesamiento de scripts, sería posible realizar un estudio más en profundidad que incluyera gráficas que nos ayudaran a ver las posiciones de uso más frecuentes de la memoria así como las posiciones no utilizadas en la scratchpad, que nos ayudara en la optimización del código.

El archivo contiene información binaria, siendo este un ejemplo del mismo.

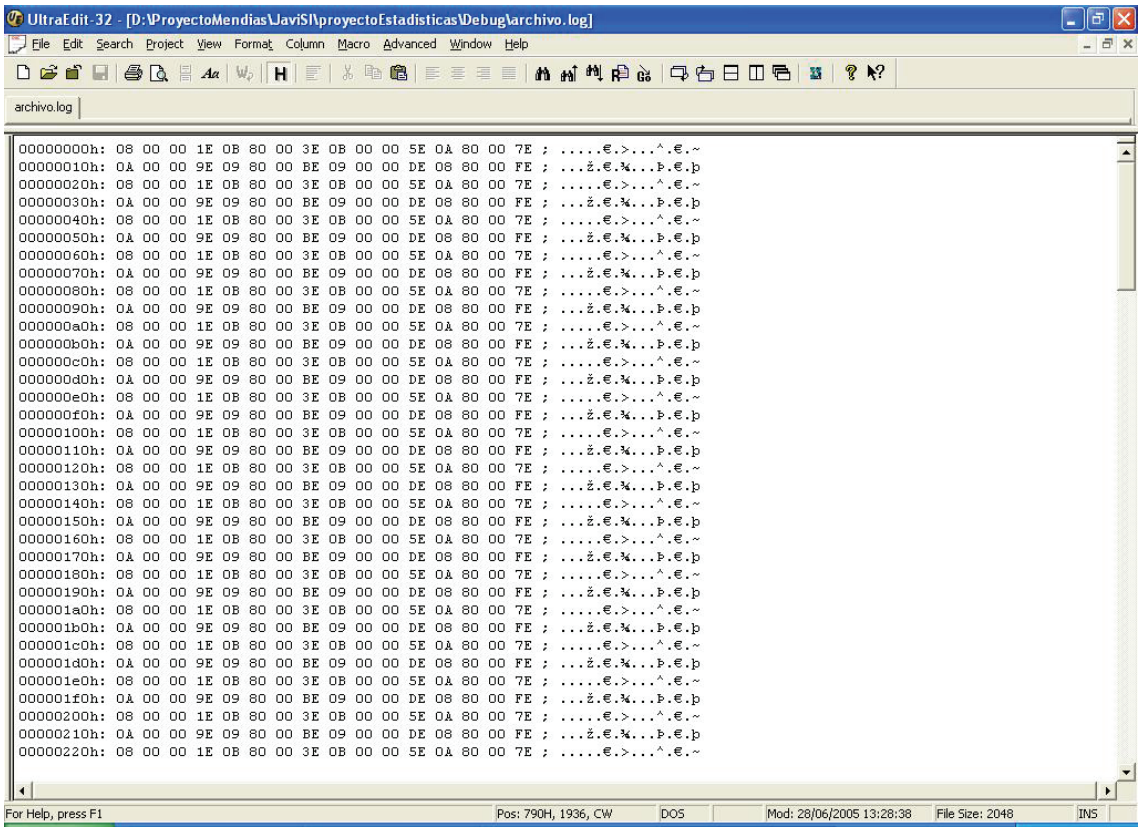


Figura nº 70: Contenido del fichero .log

Caso en el que la matriz B está en la memoria scratchpad:

```
a =(unsigned int *) XPAR_MEMORIA_AR0_BASEADDR;
b =(unsigned int *) XPAR_SCRATCHPAD_AR0_BASEADDR;
r =(unsigned int *) XPAR_MEMORIA_AR0_BASEADDR;
p =(unsigned int *) XPAR_SCRATCHPAD_0_AR0_BASEADDR;

for (i=0;i<10;i++)
{
        for (j=0;j<10;j++)
        {
                acum=0;
                for (k=0;k<10;k++)
                {
                        at=i*10;
                        bt=k*10;
                        a_aux=a+(at)+(k);
                        b_aux=b+(bt)+(j);
                        va=*a_aux;
                        vb=*b_aux;
                        acum = acum + (va*vb);
```

```
                }
                *r=acum;
                r++;
            }
        }

        //ACCESOS EXTRAS QUE SE EFECTUAN PARA LLENAR EL BUFFER
        //DE ESTADÍSTICAS Y VOLCAR LOS DATOS
        for (i=0;i<204;i++)
            j=*p;

        xil_printf("Fin ejecucion programa ejemplo multiplicacion de
        matrices\n\r");

}
```

Segundo ejemplo de estudio:

A continuación ejecutamos un programa con el que se trata de simular la inserción de forma ordenada en una lista enlazada, realizando esto con un algoritmo a sabiendas ineficiente.

De esta forma se realizan en el primer bucle 256 accesos de escritura a la memoria principal, y a continuación en el segundo bucle se realizan 256 accesos de lectura a la lista, lo que implica bastantes accesos de lectura a memoria principal, para realizar las comparaciones oportunas. Por último, se realizará un acceso de escritura a memoria principal que correspondería con la inserción del nuevo elemento a la lista.

El estudio analítico realizado en este caso es el siguiente:

Nº de lecturas:

lecturas = $(\sum_{i=0}^{255} i + \sum_{i=0}^{255} i + 1) = \sum_{i=0}^{255} (2i + 1)$

lecturas = número_términos * (primero + último) / 2

lecturas = 256 (1+ 510 +1) / 2= 131072 / 2 = **65536**

Nº de escrituras = 256.

Tiempo = accesos_mp * Latencia = (65536 + 256) * 5 ciclos / acceso = 328960 ciclos.

Energía = accesos_mp * e_mp = (65536 + 256) accesos_mp * 2000 pJ/acceso= 131'584 μJ

El código ejecutado sobre la plataforma en este caso de estudio es el siguiente:

```
unsigned int *p;
unsigned int valor;
int cont;
int i, j,k;

p =(unsigned int *) XPAR_MEMORIA_AR0_BASEADDR;

//Escribimos en una lista de posiciones consecutivas (simula una
//lista enlazada)
for (i=0; i<256; i++){
        cont=0;
        p  =(unsigned int *) XPAR_MEMORIA_AR0_BASEADDR;
        while (cont<i)
        {
                valor+=*p;
                p++;
                cont++;
        }
        *p=valor;
}

//Accedemos en una lista de posiciones consecutivas (simula una
//lista enlazada)
for (i=0; i<256; i++){
        cont=0;
        p =(unsigned int *) XPAR_MEMORIA_AR0_BASEADDR;
        while (cont<i)
        {
                valor=*p;
                p++;
                cont++;
        }
        valor=*p;
}
```

El resultado capturado por el sistema de estadísticas y mostrado por medio de la aplicación de visualización creada, puede observarse en la siguiente captura:

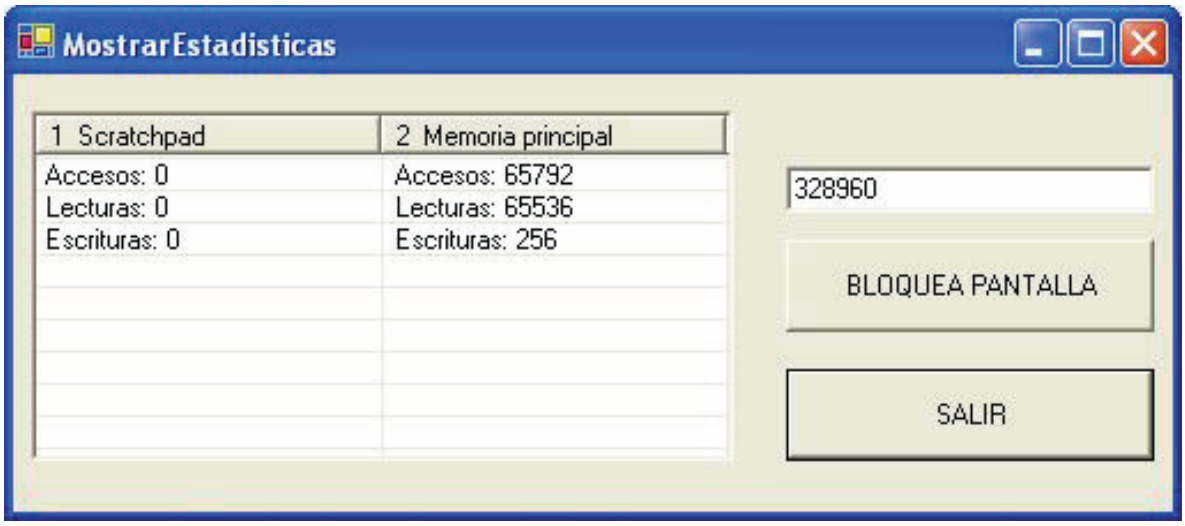


Figura nº 71: Ejecución real del ejemplo

El número de accesos es del orden $O(n^2)$.

Capítulo 6. Conclusiones

6.1 Conclusiones

Del proyecto presentado en esta memoria se extraen una serie de conclusiones y logros que se detallan a continuación:

- Es obvia la necesidad de nuevas metodologías de diseño para satisfacer los requerimientos que exigen los nuevos sistemas multimedia de altas prestaciones. Estos sistemas han de satisfacer fuertes ligaduras de coste y consumo, y todo ello acompañado de un tiempo muy limitado de producción, debido a la actual situación del mercado tecnológico.

- Las herramientas actuales que permiten la simulación de estos tipos de sistemas, carecen de propiedades que comienzan a ser necesarias para alcanzar una visión más detallada de este tipo de plataformas. De este modo, han de mejorar tanto en tiempo de simulación como en frecuencias de funcionamiento requeridas, para dar soporte de este modo a la creciente complejidad de los sistemas empotrados.

- Como se ha comentado en varias ocasiones a lo largo de esta memoria, los nuevos sistemas empotrados tienen que cumplir estrictos requisitos en cuanto a consumo, y es conocido que la mayoría del derroche energético en un sistema empotrado se produce en los accesos que el procesador realiza al subsistema de memoria. Además, para reducir el consumo, es necesario una sintonía entre el hardware y el software en el sistema de forma que las aplicaciones aprovechen al máximo las características de la arquitectura.

- Por ello, se ha construido una plataforma de emulación encargada de sustituir la jerarquía de memoria de un sistema empotrado, por una jerarquía totalmente parametrizable y reconfigurable que permita realizar emulación de sistemas empotrados complejos con frecuencias de funcionamiento del orden de los megahercios, lo que supone una mejora de varios órdenes de magnitud con respecto a las herramientas de simulación de este tipo de sistemas que existen en la actualidad.

- Para obtener conclusiones sobre la plataforma de emulación, se diseña e implementa un sistema totalmente independiente del

anterior, capaz de extraer datos de diferentes puntos clave tales como la memoria cache, la memoria principal, scratchpad…

Estos datos permiten visualizar el comportamiento del sistema a un grano muy bajo (a nivel de ciclo), lo cual facilita el estudio del funcionamiento de los sistemas empotrados y la observación de en qué medida afecta cada recurso al comportamiento global.

Además permite comprobar en tiempo real, el resultado de la ejecución de distintos programas software y su repercusión en cuanto a accesos a memoria, y por tanto en consumo.

- Finalmente, se ha abordado la construcción de un sistema de emulación que permita reproducir el comportamiento de un sistema real independientemente de la tecnología física utilizada para construirlo.

Referencias

1. **Computer architecture: a quantitative approach**
   John L. Hennesy, David A. Patterson

2. **Metodología multinivel de refinamiento del subsistema de memoria dinámica para los sistemas empotrados multimedia de altas prestaciones**

   Tesis doctoral – David Atienza Alonso

3. **Embedded Processors**

   Project Report for ICS 212
   Prof. Rajesh K. Grupta

4. **A Dynamic Memory Management Unit for Embedded Real-Time System-on-a-Chip**

   Mohamed Shalan
   Vincent J. Mooney III

5. **Using Embedded Network Processors to Implement Global Memory Management in a Workstation Cluster**

   Yvonne Coady, Joon Suan Ong and Michael J. Feeley
   Departamento de Ciencia computacional
   Universidad de Columbia

6. **Optimización de microprocesadores.**

   Ignacio Martín Llorente. Apuntes de la asignatura Procesamiento Paralelo.