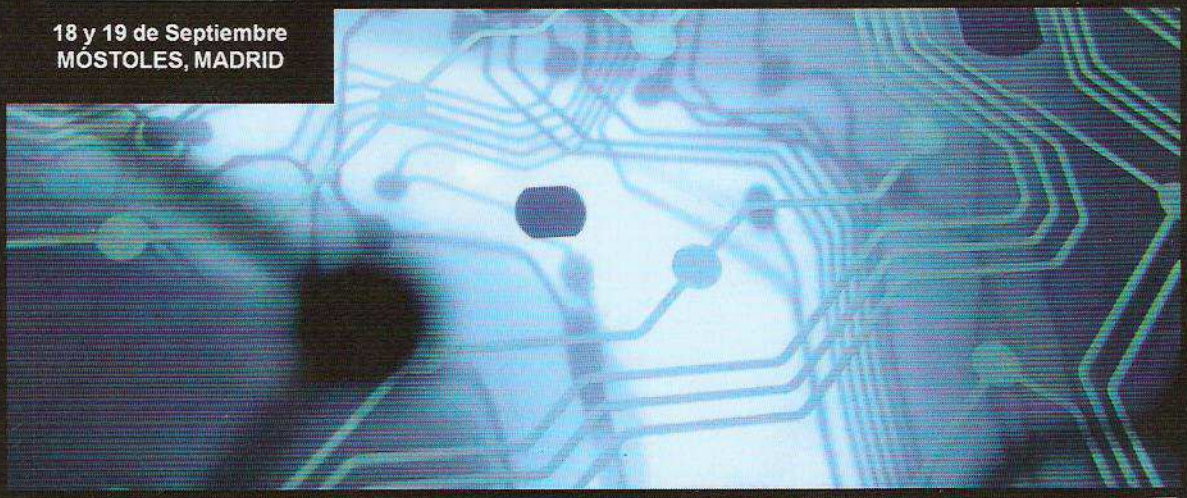
VIII Jornadas de Computación Reconfigurable y Aplicaciones 08

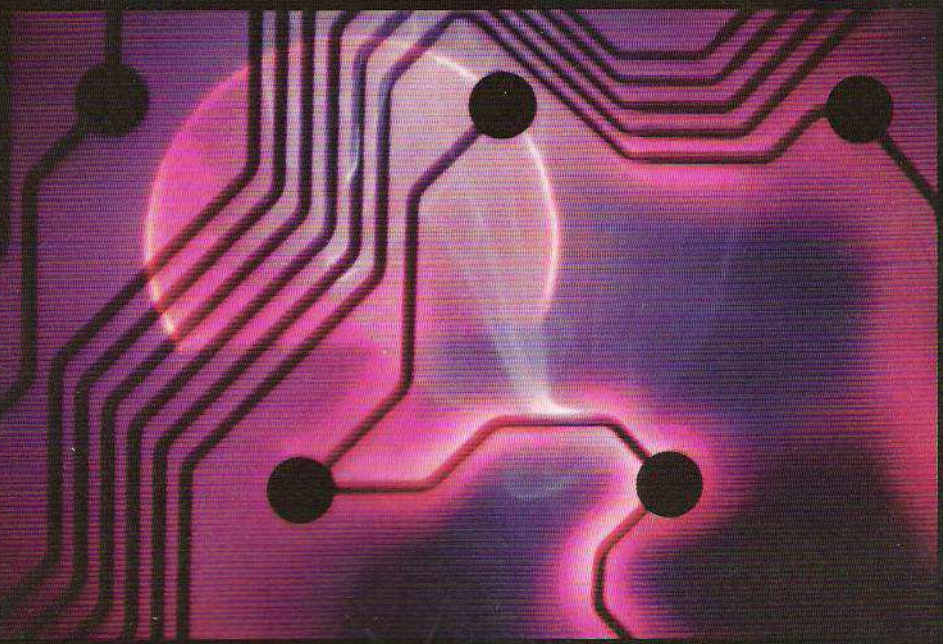
www.jcraconf.org

18 y 19 de Septiembre
MÓSTOLES, MADRID



EDITOR: José Ignacio Martínez Torre

Grupo de Diseño Hardware-Software



Universidad
Rey Juan Carlos

ISBN
978-84-612-5635-8

Sesión 7

Metodología de Diseño de SoC basada en OpenRisc sobre FPGA con Cores y Herramientas Libres

Villar de Ossorno J. I.[1], Bellido Díaz M. J.[1], Ostúa Arangüena E.[1], Guerrero Martos D.[1], Juan Chico J.[1], Muñoz Rivera A.[1]

[1] Dpto. de Tecnología Electrónica, E.T.S.I. Informática, Universidad de Sevilla, Sevilla, España, {jose, bellido, ostua, guerre, jjchico, amrivera}@dte.us..es
http://www.dte.us.es/

Abstract. El aumento de la complejidad de los System-on-Chip (SoC) basados en microprocesador ha hecho necesaria la adopción de metodologías de codiseño con un alto nivel de abstracción. Éstas metodologías, altamente dependientes del procesador utilizado y de la tecnología de implementación, son cada vez más populares debido al creciente auge del hardware libre, que ha hecho que la labor de diseño sea accesible a grandes grupos de usuarios debido a su bajo coste. En este trabajo proponemos una metodología de alto nivel para sistemas basados en el procesador OpenRisc 1200 de Opencores sobre FPGA, que permita minimizar el coste y tiempo de desarrollo. Se ha reducido el coste gracias al empleo de cores y herramientas libres y el tiempo de desarrollo utilizando estos cores como bloques de construcción para elevar el nivel de abstracción.

1 Introducción

En las últimas décadas hemos asistido a una revolución en el mundo de la informática debida al auge que ha cobrado el uso de software libre [1]. Desde que comenzara a vislumbrarse el concepto de software libre hasta hoy, se han producido profundas transformaciones en el modo de desarrollar software, en el tejido de la industria y en los modelos de negocio, que están afectando no solo al mundo de la informática sino a la industria en general y, muy particularmente a aquella que basa su negocio en los derechos de propiedad intelectual. Sirvan como ejemplo las licencias "Creative Commons" [2], desarrolladas para facilitar la distribución y el uso de contenidos de muy diferentes tipos para el dominio público. Otro ejemplo muy significativo y relacionado con el mundo del software es el mundo del hardware.

Quizás, debido a su carácter tangible y a su mayor coste de producción y replicación, esta revolución ha tardado algunos años más en extenderse a este mundo, sin embargo, desde que hace algunos años se popularizasen los dispositivos programables (FPGA) a la par que aumentaron sus prestaciones, se han gestado multitud de comunidades dedicadas a la generación de módulos funcionales bajo licencias libres, generalmente adaptando del modelo de licencia GPL/LGPL [3]. Ante este nuevo movimiento y con el precedente del software, el mundo de la empresa no se ha mantenido al margen. De estas comunidades de usuarios, han surgido iniciativas empresariales como Gaisler Research o Beyond Semiconductor, y en sentido inverso, empresas establecidas en el antiguo modelo, han liberado sus desarrollos, en torno a los cuales se han creado comunidades abiertas. Los ejemplos más recientes

de esto último los encotramos en empresas como Lattice con la liberación de su CPU LM32 [4] [5] o Sun Microsystems [6] con su serie de procesadores UltraSparc [7].

Paralelamente al movimiento del hardware libre, y como factor catalizador de éste, el diseño electrónico se ha hecho accesible a nuevos sectores. El desarrollo de chips programables de altas prestaciones y gran nivel de integración ha permitido que en una pastilla de silicio se pueda desarrollar un sistema casi completo del modo que se ilustra en la figura 1. Este incremento en las prestaciones y en el nivel de integración, supone un reto para la labor de diseño, que ha de asumir la adopción y el desarrollo de nuevas metodologías que permitan gestionar de un modo eficiente la creciente complejidad de los sistemas que aborda.

Una de las metodologías más aceptadas actualmente es la llamada System on Chip o por sus iniciales SoC [8]. Fundamentalmente, ésta se basa en la inclusión en un mismo chip, de todas las partes que conforman un computador o un sistema digital complejo.
La metodología de diseño de SoC no supone una ruptura con los procesos de desarrollo previos, pues los integra a modo de bloques constructivos. Esta integración le permite coexistir con otras metodologías subyacentes, para así aprovechar el know-how de los equipos de diseño a la par que da como resultado sistemas más rápidos, energéticamente eficientes y con menor ocupación de área. Todas estas ventajas han hecho que la metodología SoC sea hoy en día la más aceptada dentro del desarrollo de sistemas empotrados.

Los SoC, están compuestos comúnmente por un microprocesador digital, en torno al cual se configura el resto del sistema atendiendo a las características concretas de éste. Con esto queremos hacer notar que la elección de la cpu determinará en gran medida la arquitectura del resto del sistema y por tanto la metodología de diseño a emplear.

En el mercado podemos encontrar multitud de opciones en lo que a microprocesadores se refiere, tanto libres como no libres. Dentro de las alternativas libres podemos optar por Leon 2 o Leon 3 [9], Lattice Mico 32 [5], OpenRisc [10], Simply Risc S1 [11], UltraSparc [7] y aeMB [12] entre otras. De las opciones no libres, que normalmente requieren la adquisición de licencias e incluso pueden vetar su implementación fuera de productos ajenos a la marca que los distribuye (como ocurre con el MicroBlaze [13] de Xilinx), las opciones más populares son la familia ARM [14], MicroBlaze, PowerPC [15] o el NIOS de Altera [16].

Desde hace algunos años, el grupo de investigación ID2 de la Universidad de Sevilla [17] está dedicado al desarrollo de sistemas digitales basados tanto en hardware como software abierto, empleando FPGAs como nucleo central para integrar los sistemas diseñados. En particular, se ha desarrollado una Plataforma hardware-software basada en el microprocesador Leon 3 sobre la que se ha implantado una distribución Debian [18] [19]. Esta plataforma tiene un enorme ámbito de aplicación, fundamentalmente debido a la distribución Debian, ya que además de facilitar significativamente la instalación de software sobre la plataforma, se dispone de una gran cantidad de éste y de todo tipo, listo para usar y sin restricciones de uso. Sin embargo, la empresa Gaisler Research, desarrolladora del microprocesador Leon, en su última versión, Leon 3, ha puesto algunas restricciones a su uso en aplicaciones de carácter industrial. Esto nos ha motivado a desarrollar nuevas plataformas de las mismas características pero con diferentes microprocesadores que si tengan

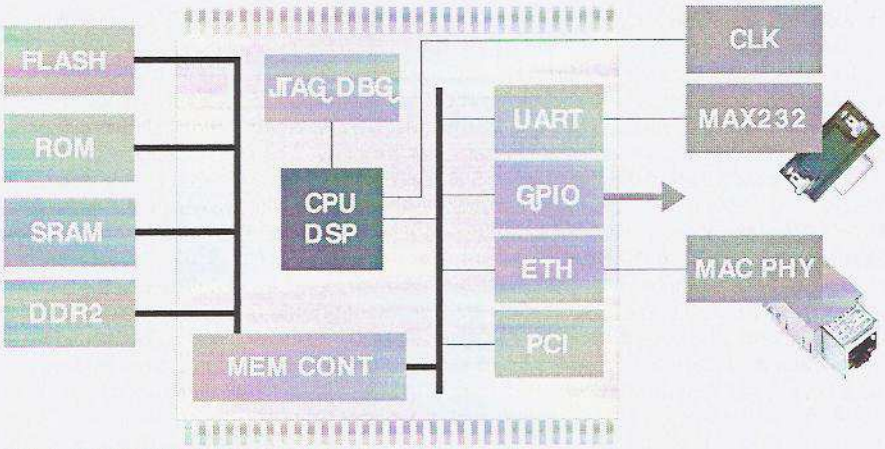


Fig. 1. Concepto de System-on-Chip

como principal característica el ser completamente libres.

Así, en este trabajo se presenta la Metodología de diseño que se está llevando a cabo para desarrollar una plataforma Hardware-Software abierta basada en el microprocesador OpenRisc. En concreto se ha empleado el core libre OpenRisc 1200 cuya implementacion se está llevando a cabo sobre varias familias de FPGAs de la marca Xilinx.

Esta metodología toma como punto de partida el código fuente de todos los cores que compondrán nuestro sistema y tras un proceso de diseño en el que se barajan diferentes posibilidades, obtendremos un sistema plenamente funcional. En nuestro trabajo hemos utilizado la metodología con diferentes plataformas de desarrollo: la placa S3E Starter Kit de Digilent [20] para la familia Spartan 3E y la placa Virtex II Evaluation Kit de Avnet para la familia Virtex II [21]. Con este trabajo deseamos compartir nuestra experiencia en la implementación de OpenRisc sobre FPGA y definir una metodología que minimice el tiempo de aprendizaje de los equipos de desarrollo a la par que se optimice tanto el coste de desarrollo como el time-to-market.

El contenido del artículo se estructura del siguiente modo: en la sección 2 se describen el procesador (OpenRisc 1200) y el bus del sistema (Wishbone [22]) que hemos utilizado, reseñando sus características principales. En la sección 3 se trata a fondo la metodología definiendo el flujo de diseño del hardware y del software respectivamente. Finalmente se presentan las conclusiones sobre el uso de la metodología valorando los aspectos positivos y negativos mas significativos de este enfoque.

2 El Procesador OpenRisc 1200

OpenRisc 1000 es la arquitectura de procesador en la que se basa la familia de procesadores libres OpenRisc 1xxx [23]. Esta arquitectura ofrece un completo abanico de posibilidades de implementación para adaptarse a una gran variedad de objetivos y nichos de mercado. Su diseño ha tenido en cuenta las últimas tendencias en lo que a arquitectura de sistemas se refiere, buscando ofrecer gran estabilidad y fiabilidad a la vez que conseguir unas más que aceptables cotas de rendimiento y bajo consumo. Principalmente está dirigida a

aplicaciones de medio y alto rendimiento en el mercado de redes, el de la automoción y el de sistemas empotrados.

La arquitectura define cinco grupos de instrucciones dirigidas a diferentes tareas. Éstas van desde un conjunto de instrucciones básicas hasta otras dirigidas a aplicaciones de alto rendimiento pero tan sólo aquellas que la especificación de la arquitectura define como básicas, son de obligada implementación por todos los cores. Los grupos son los siguientes: ORBIS32 para instrucciones básicas de 32 bits; ORBIS64 para instrucciones enteras y load/store de 64 bits; ORFPX32 para instrucciones en punto flotante de precisión simple; ORFPX64 para instrucciones en punto flotante de precisión doble y ORVDX64 para instrucciones vectoriales y DSP.

Dentro de la arquitectura OpenRisc nos hemos centrado en una implementación concreta, el OpenRisc 1200. Este core está completamente realizado en Verilog y contempla el conjunto de instrucciones ORBIS32. El OpenRisc 1200 es un procesador segmentado con un pipeline de 5 etapas. Tiene soporte para gestión de memoria virtual e instrucciones básicas para el procesamiento de señales digitales (DSP).

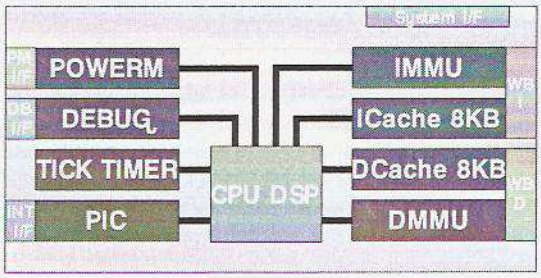


Fig. 2. Diagrama de bloques del OpenRisc 1200

Está compuesto por varios bloques funcionales tal y como se aprecia en la figura 2. Sus características principales las podríamos resumir en que tiene unidad de gestión de memoria (MMU), utiliza una arquitectura Harvard de 32 bits, admite cachés separadas de datos e instrucciones y tiene una unidad de gestión de energía con una capacidad de reducción de la potencia de 100x a 2x. Asimismo, posee una unidad de debug que permite la depuración no intrusiva a través de un interfaz *jtag* externo.

Como hemos mencionado, una de las características más interesantes del OpenRisc 1200 es su capacidad de adaptación y personalización para adaptarse a las diferentes necesidades y tecnologías de implementación que encontramos en el mercado. Podemos parametrizar prácticamente todos sus elementos y sintetizarlo tanto para FPGAs, tecnologías semi-custom y tecnologías full-custom.

El procesador puede ser simulado con diversas herramientas. En nuestro entorno de desarrollo hemos utilizado principalmente Icarus Verilog [24] y Modelsim de Mentor Graphics [25]. Para simplificar la tarea de simulación y pruebas se incluyen testbenchs y tests de regresión para detectar posibles fallos.

Para la implementación podemos optar tanto por FPGAs como por ASICS. De serie, el OpenRisc 1200 está preparado para ser sintetizado por las herramientas Synplify [26], Xilinx XST [27] y Altera QIS [28]. De cualquier modo, no resulta difícil adaptarlo a cualquiera otra herramienta de síntesis del mercado.

En la industria ha sido utilizado con éxito por diversas empresas para desarrollar sus propios SoC, como por ejemplo Flexitronics [29] con su chip Marvin o Vivace [30] con su

Vivid Media Processor [31], en los cuales un core OpenRisc 1200 hace las funciones de cpu principal.

2.1 El bus Wishbone

Los desarrolladores de OpenRisc optarón por utilizar como bus de interconexión con el resto de dispositivos de un sistema el bus Wishbone. Este bus ha sido elegido por Opencores como bus estandar de interconexionado de cores y, de hecho, la mayoria de los cores de Opencores están diseñados para interconexión con dicho bus, que está libre de patentes, royalties y copyright [22], lo que hace que su uso sea gratuito. Wishbone es un bus que se caracteriza principalmente por su simplicidad y flexibilidad.

La simplicidad ha sido entendida como la capacidad de conseguir grandes tasas de datos con una complejidad de hardware mínima. A diferencia de otros estándares que definen uma jerarquía de varios protocolos, en Wishbone sólo existe uno, que es de alta velocidad con capacidad de adaptarse a dispositivos lentos. De este modo, en caso de necesitar conectar dispositivos de alta y baja velocidad es recomendable utilizar dos interfaces Wishbone, una para dispositivos de alta velocidad y otra para dispositivos de baja velocidad en lugar de complicar la gestión del bus.

Desde el punto de vista de la flexibilidad, el estándar nos permite utilizar diversas topologías de bus (bus compartido, crossbar switch, punto a punto) y deja margen para configurar otros parametros, como los anchos de los buses, el significado de las etiquetas de datos y direcciones, endianness, niveles eléctricos, etc...

3 Metodología de Diseño

Para la obtención de una implementación basada en OpenRisc propondremos una metodología de diseño de doble vía tal y como se muestra en la Figura 3. Definiremos dos flujos de diseño diferentes, uno para la implementación del hardware y otro para el desarrollo del software respectivamente. Estos dos flujos, aunque son en gran medida independientes, guardan una intensa relación; los productos de ambos flujos, hardware y software, se imponen mutuamente requisitos tanto funcionales como no funcionales desde el principio del desarrollo.

El flujo de diseño hardware comienza con la configuración del microprocesador OpenRisc 1200 y la selección de las características que queramos implementar en hardware. También se definirá un método para crear el bus de interconexión del sistema, a través del cual conectaremos el microprocesador con otros componentes esenciales, como la memoria, y con el resto de periféricos que vayan a conformar nuestro SoC. Este flujo de trabajo termina con un proceso de simulación tras el cual se procederá a la síntesis y por último a la implementación en una FPGA.

Por otro lado encontramos el flujo de desarrollo del software. En este proceso, se parte de las especificaciones del sistema para generar el software que en una ulterior fase ejecutará nuestra implementación. Este flujo presenta multitud de alternativas en cuanto a su puesta en práctica debido al extenso conjunto de herramientas disponibles. Mostraremos un ciclo de desarrollo genérico mencionando las distintas posibilidades en aquellas etapas en las que se pudieran dar.

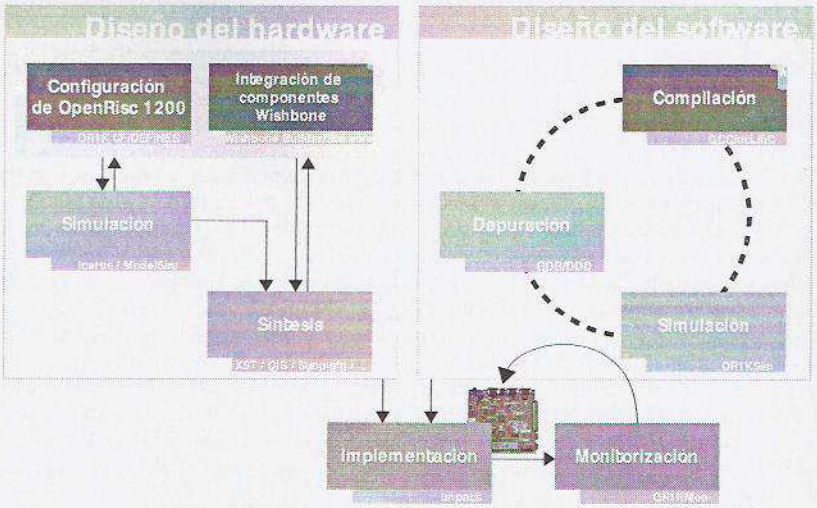


Fig. 3. Flujo de diseño de la metodología.

Estos dos ciclos aunque guardan una estrecha relación, tal y como mencionamos anteriormente, pueden desarrollarse en paralelo prácticamente sin ningún tipo de interferencias. Una vez concluidos, el diseño del hardware y del software, pueden ser implementados en hardware real para proceder a la verificación final de la plataforma.

3.1 Ciclo de Diseño Hardware

El flujo de diseño del hardware con OpenRisc 1200 comienza con el código fuente en Verilog de los distintos componentes que conformarán el sistema. Inicialmente tendremos que personalizar algunos parámetros de configuración del microprocesador. Para realizar esta tarea, se dispone de la herramienta gráfica OR1Kgct cuyo aspecto se ilustra en la Figura 4, que se encuentra disponible en Opencores, en la que se muestran los diferentes parámetros agrupados en varios apartados y de los que se permite modificar sus valores. Alternativamente, para realizar estos cambios podemos prescindir de esta herramienta y configurar directamente los parámetros modificando los ficheros de configuración del OpenRisc, cuyo formato no es más que una serie de sentencias de tipo *'define* para el preprocesador de Verilog.

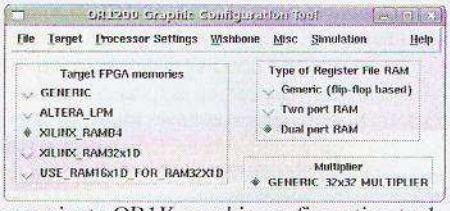


Fig. 4. Captura de la herramienta OR1K graphic configuration tool.

Una vez configurado el microprocesador, debemos crear el bus de interconexión del sistema. Este bus conectará las dos interfaces Wishbone del procesador (datos e instruccio-

nes) con el resto de periféricos Wishbone del sistema. En esta labor podemos optar por varias alternativas en función del bus del sistema que elijamos. La distribución de OpenRisc nos ofrece un core que implementa un bus compartido con siete slots para interfaces Wishbone maestras y ocho para interfaces Wishbone esclavas. .

Acerca del bus del sistema, en Opencores se propone una asignación estándar del mapa de memoria y de los códigos de interrupción.

Para terminar el diseño, debemos crear la entidad superior del sistema, en la que deberán estar instanciados el procesador, el bus de interconexión y los distintos periféricos Wishbone que vayamos a utilizar, entre los cuales al menos debe haber un controlador de memoria. Cada uno de estos componentes deberá estar convenientemente conectado a su correspondiente slot Wishbone del bus central.

Si queremos simular el diseño, podemos utilizar cualquier paquete de simulación Verilog de los que encontramos en el mercado. En este apartado también encontramos una alternativa libre, Icarus Verilog [24], que nos permite analizar el comportamiento del sistema antes de su implementación final en hardware.

Para la etapa final de síntesis necesitamos recurrir a herramientas comerciales en función de la tecnología de implementación que vayamos a utilizar debido a la alta dependencia de este tipo de herramientas de la FPGA utilizada y de la carencia de alternativas libres para esta labor. Algunas como XST de Xilix, QIS de Altera, Synplify de Synplicity o Synopsys Design Compiler entre otras han sido utilizadas para implementar OpenRisc con éxito.

Finalmente, trás el proceso de place&route, obtenemos el fichero que debemos cargar en la FPGA. En el caso concreto del entorno ISE 9.2i de Xilinx, que es el que hemos utilizado para nuestros desarrollos, utilizando la herramienta Impact en unos instantes nuestro sistema queda implementado sobre la plataforma de desarrollo.

3.2 Ciclo de Diseño Software

Para compilar nuestro software disponemos del conjunto de herramientas de GNU, también llamado GNU toolchain. En este set de herramientas encontramos algunas tan conocidas como el compilador GCC [32] o el depurador GDB [33].

Sobre OpenrRisc adicionalmente pueden ejecutarse algunos sistemas operativos. Quizás los mas destacables sean las ramas 2.4 y 2.6 del kernel de Linux y UcLinux [34], una adaptación limitada de Linux para plataformas empotradas sin unidad de gestión de memoria (MMU).

Cuando generemos nuestro toolchain, debemos hacerlo específicamente para cada plataforma para la que deseemos generar software, indicando en el parámetro *target* la plataforma objetivo: linux, uclinux o elf si no vamos a ejecutar el software sobre ningún sistema operativo.

La tarea de generación de un toolchain para una plataforma determinada es una tarea tediosa y pesada que puede automatizarse utilizando una serie de scripts automáticos creados por Mark Jarvin llamados DRP [35]. Con esta herramienta obtendremos imágenes de linux, ucLinux y los toolchains completos para todos los targets.

Durante la escritura del software, el proceso de depuración resulta fundamental. Para ello disponemos del depurador GDB y de un simulador de la arquitectura para PC llamado Or1ksim. De este modo podemos conectarnos mediante GDB al simulador e ir depurando el programa en un entorno equivalente al hardware en el que finalmente lo ejecutaremos. Una vez tengamos disponible el hardware real, podemos utilizar el mismo procedimiento de depuración, pues el OpenRisc 1200 soporta depuración no intrusiva a través de un in-

terfaz JTAG [36]. Para ello tan solo debemos utilizar un interfaz con el PC como el Xilinx Parallel Cable 3 a través del cual GDB se establecerá conexión con el hardware.

4 Ejemplo de implementación

Se han conseguido llevar a cabo implementaciones de SoC basados en el procesador OpenRisc sobre placas de desarrollo con diversas familias de FPGAs de la marca Xilinx.

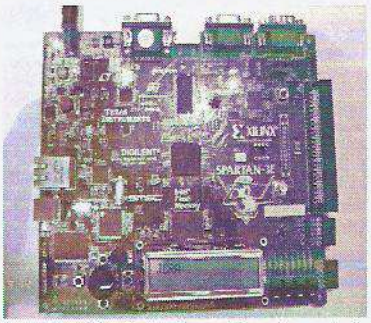


Fig. 5. Plataforma de demostración fruto de aplicar la metodología propuesta.

Dichas implementaciones contemplan la inclusión dentro del SoC de controladores para los periféricos disponibles en cada una de las plataformas de desarrollo como las UARTs para comunicación serie, interfaz de video VGA, interfaz de red 802.3, puertos PS2 para teclado y ratón, etc... Como restricción para el desarrollo de los SoC hemos impuesto utilizar exclusivamente cores de desarrollo propio para su posterior liberación o bien cores existentes con licencias libres disponibles a través de comunidades virtuales de desarrollo como Opencores, o similares. Sobre este punto resulta interesante incidir en que actualmente existen controladores libres para todos los periféricos de que disponemos con la excepción de controladores para memorias tipo DDR, ya sean DDR o DDR-II.

En este ejemplo de aplicación de la metodología presentamos un sistema basado en la placa de desarrollo Spartan 3E Starter Kit rev.D de Digilent. Esta placa monta una FPGA Spartan 3E de 500.000 puertas equivalentes con un reloj a 50Mhz. Adicionalmente dispone de interfaces serie, un PHY ethernet, un puerto VGA, un rotary encoder y diversos interruptores, pulsadores y leds. En este ejemplo se implementa un sistema mínimo en el que a través del puerto serie nos comunicamos con el OpenRisc para que este controle el estado de los leds, los pulsadores y el display de dos lineas que tenemos en la placa. La velocidad de operación es de 33,3 Mhz. y la memoria del sistema la hemos generado utilizando bloques BRAM disponibles en el interior de la FPGA. Esta configuración básica tiene un nivel de ocupación en la FPGA del 65% lo que deja sitio suficiente para la inclusión de otros periféricos que pudieran ser necesarios. Sobre este punto, cabe resaltar que se ha conseguido un nivel de ocupación bastante bajo en comparación con diseños previos basados en el procesador LEON 3, más aun si tenemos en cuenta que hemos usado una placa de desarrollo de iniciación (S3E Starter Kit), con una FPGA que podría ser considerada pequeña para este tipo de aplicaciones en comparación con la que se usa en el S3E Development Kit cuya FPGA tiene 1.600.000 puertas equivalentes.

5 Conclusiones

En un momento de la historia en que el mercado de los computadores empotrados y la computación ubicua comienzan a tener un peso determinante en el mundo de los semiconductores, es fundamental disponer de los medios necesarios para desarrollar sistemas capaces de ofrecer la robustez y flexibilidad que el mercado requiere.

Este mercado, cuyo crecimiento en cuota y en volumen de negocio dentro de los semiconductores se prevee exponencial en los próximos años, hace que sean necesarias prácticas que fomenten la eficiencia y acorten en la medida de lo posible los ciclos de desarrollo para conseguir un *time-to-market* competitivo.

La metodología presentada incide en los cuatro puntos fundamentales de estos requerimientos del siguiente modo: resulta ser altamente productiva en cuanto al tiempo y esfuerzo de diseño, presenta un ciclo de diseño flexible capaz de adaptarse a múltiples entornos y que a la vez afecta al tercer punto, que es la versatilidad de los diseños generados, que pueden adaptarse a requerimientos altamente cambiantes en función de las vías que se tomen. Como última ventaja, está el abaratamiento de costes, pues hemos utilizado cores libres cuyo uso está exento de costes, licencias y royalties. También se han utilizado herramientas libres y gratuitas excepto en las últimas fases de implementación del hardware en las que no existe alternativa a las propietarias.

Los satisfactorios resultados de esta experiencia arrojan buenas expectativas en cuanto a futuros desarrollos se refiere, lo que nos ha motivado a seguir trabajando en esta linea y comenzar nuevos proyectos basándolos en el ciclo de desarrollo que hemos propuesto.

Referencias

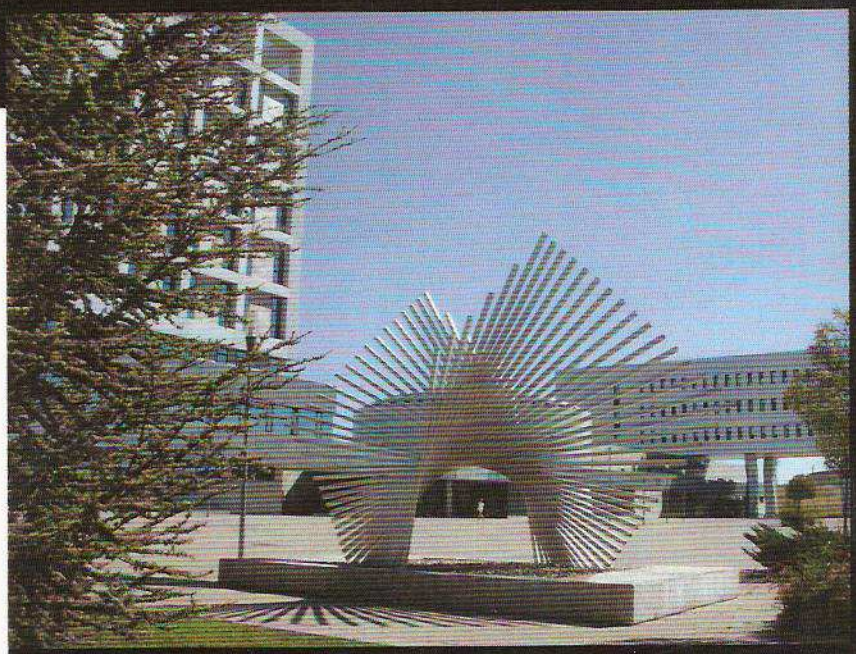
1. Stallman, R, "Free Software, Free Society: Selected Essays of Richard M. Stallman" (2002).
2. Brown, G. "Academic Digital Rights: A Walk on the Creative Commons." *Syllabus Magazine,* April (2003).
3. Free Software Foundation, GPL y LGPL License v3 http://www.gnu.org/licenses/, (2007)
4. Lattice Semiconductor, "LatticeMico32 Open Source Licensing" .
5. Lattice Semiconductor, "LatticeMico32 Processor Reference Manual", 2007.
6. Sun Microsystems, "Free and Open Source Licensing White Paper", 2007.
7. Sun Microsystems, "OpenSPARC T1 Micro Architecture Specification", 2006.
8. Kamath , U., Kaundin , R., "System-on-Chip Designs , Strategy for Success ", Wipro Technologies, 2001.
9. Gaisler, J. "Leon 2 Processor User's Manual", Gaisler Research, 2004 .
10. Lampret, D., "OpenRisc 1200 IP Core Specification", Opencores, 2001.
11. Simply Risc, "Simply Risc S1 Core Specification", 2006.
12. aeMB: http://www.aeste.net/category/cores/aemb/
13. "Microblaze Processor Reference Guide", Xilinx Inc. , 2005.
14. Furber, S., "ARM system-on-chip architecture, 2nd edition"Addison-Wesley, 2000.
15. IBM Corp., "IBM PowerPC Quick Reference Guide", 2004
16. Altera Corp., "NIOS 3 CPU Data Sheet", 2004
17. Grupo de Investigación y Desarrollo Digital, ID2: http://www.dte.us.es/id2/
18. A. Muñoz, E. Ostua, P. Ruiz-de-Clavijo, M. J. Bellido, J. Viejo, A. et al, "Un ejemplo de implantación de una distribución Linux en un SoC basado en hardware libre", 2007
19. Debian project, http://www.debian.org.
20. Digilent Inc. "Spartan 3E Starter Kit Board User Guide", 2006.
21. Avnet Inc. "Xilinx Virtex-II Evaluation Kit User Guide", 2003.

22. Herveille, R. "WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores", Opencores, 2002.
23. Lampret, D., Chen, C., Mlinar, M., Rydberg, J., Ziv-Av, M. et al "OpenRISC 1000 Architecture Manual", Opencores, 2004.
24. Icarus Verilog, http://www.icarus.com/eda/verilog/ .
25. Modelsim, de Mentor Graphics, http://www.model.com.
26. Synplify, de Synplicity Inc., http://www.synplicity.com.
27. Xilinx, "Xilinx Synthesis Technology User Guide 9.2", 2007.
28. Altera, "Quartus II Integrated Synthesis ", 2007.
29. Flexitronics Inc., http://www.flextronics.com.
30. Vivace Semiconductor, http://www.vivacesemi.com.
31. Vivace Semiconductor, "VSP100 Mobile Processor Product Brief".
32. GCC, http://gcc.gnu.org/
33. Stallman, R., Pesch, R., Shebs, S. et al. "Debugging with gdb", Free Software Foundation, 2006.
34. uClinux, Embedded Linux/Microcontroller Project, http://www.uclinux.org.
35. DRP, http://www.eecg.utoronto.ca/~jarvin/or32/
36. Milnar, M., "GDB for OR1k", 2006.

VIII Jornadas de Computación Reconfigurable y Aplicaciones 08

www.jcraconf.org

18 y 19 de Septiembre
MÓSTOLES, MADRID



Patrocinadores:

Universidad Rey Juan Carlos

Ministerio de Educación y Ciencia

Colaboradores:

Sun microsystems

AVNET

Seven Solutions

Organizadores:

Grupo de Diseño Hardware-Software

ISBN: 978-84-612-5635-8

Universidad Rey Juan Carlos