
FIE' 08
Conferencia Internacional
5ta edición
Santiago de Cuba, Cuba
14 al 16 de Julio del 2008
UNIVERSIDAD DE ORIENTE
CIENCIA Y CONCIENCIA
GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN
CSIC
cigré
ISBN: 978-84-00-08680-0
NIPO: - - - -

FIE ' 08
international conference

Fifth edition
Santiago de Cuba, Cuba
From 14 to 16 July 2008

Santiago de Cuba, Cuba July 14-16 2008

FIE ' 08
International Conference
fifth edition

Home

HOME

THEMATICS

AUTOMATIC CONTROL

TELECOMMUNICATIONS

ELECTRIC

COMPUTER SCIENCE ENGINEERING

PATTERN RECOGNITION
AND BIOMEDICAL ENGINEERING

ENGINEERING LEARNING

topicname

Control Electrónico (Electronic Control)
Ingeniería Telemática. (Telematic Engineering)
Microelectrónica (Microelectronics)
Procesamiento Digital de Señales (Digital Signal Processing)
Sensores y Circuitos. (Sensors and Circuits)
Sistemas de Radiocomunicaciones (Radiocommunications Systems)
Sistemas Electrónicos y de Control de Potencia (Power and Electronic Systems)
Tecnologías e Infraestructura de Redes de Telecomunicaciones (Technologies and infrastructures for telecommunications networks)
Tecnologías para Aplicaciones y Servicios (Technologies for applications and services)
Tecnologías para el Desarrollo de Sistemas Distribuidos (Technologies for the Development of Distributed Systems)

HOME

THEMATICS

AUTOMATIC CONTROL

TELECOMMUNICATIONS

ELECTRIC

COMPUTER SCIENCE ENGINEERING

PATTERN RECOGNITION AND BIOMEDICAL ENGINEERING

ENGINEERING LEARNING

Title	Paper
AMPLIACIÓN DE PERIFÉRICOS PARA APLICACIONES EMBEBIDAS BASADAS EN HARDWARE Y SOFTWARE LIBRE	PDF
Aplicación CTI para consultas a bases de datos Oracle.	PDF
Aplicación de PicoBlaze al diseño de sistemas de control industrial	PDF
Detección de Fibras en la Calibración de un Mazo Incoherente para la Transmisión de Imágenes	PDF
Electro síntesis	PDF
Estrategia para la Migración hacia el Software Libre en la carrera de Ingeniería en Telecomunicaciones y Electrónica	PDF
Metodología de Diseño SoC con OpenRisc sobre FPGA	PDF
Procedimientos para el cálculo de la capacidad de transmisión en GPRS	PDF
SIGACIT: Sistema para la gestión de actividades de ciencia e innovación tecnológica	PDF
SIPNO: SISTEMA PROVEEDOR DE NOTICIAS ONLINE	PDF
Sistemas de localización en interiores: Tecnologías empleadas.	PDF

METODOLOGÍA DE DISEÑO SOC CON OPENRISC SOBRE FPGA

José I. Villar[1], Manuel J. Bellido [2], Enrique Ostúa[3], David Guerrero[4], Jorge Juan[5], Alejandro Muñoz[6]

Abstract *— The increasing complexity of microprocessor based System-on-Chip (SoC) has made necessary the adoption of co-design methodologies with a high level of abstraction. These methodologies, highly dependent on the selected processor and the implementation technology, are becoming increasingly popular because of the growing rise of free hardware, which has made SoC design accessible to new big market niches and user groups due to its low cost.*
In this paper authors propose a specific high-level design methodology that allows to minimize the cost and development time for systems based on the free microprocessor OpenRisc 1200 from Opencores using Xilinx FPGA devices as implementation technology. We have reduced costs through the use of free cores and tools and have been able to reduce development time using these cores as building blocks for raising the level of abstraction of the design process.

Index Terms —FPGA, SoC, OpenRisc, GNU

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 el 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. 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.

[1] José Ignacio Villar de Ossorno, Dpto. de Tecnología Electrónica, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012 Sevilla, Spain, jose@dte.us.es
[2] Manuel Jesús Bellido Díaz, Dpto. de Tecnología Electrónica, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012 Sevilla, Spain, bellido@dte.us.es
[3] Enrique Ostúa Arangüena, Dpto. de Tecnología Electrónica, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012 Sevilla, Spain, ostua@dte.us.es
[4] David Guerrero Martos, Dpto. de Tecnología Electrónica, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012 Sevilla, Spain, guerre@dte.us.es
[5] Jorge Juan Chico, Dpto. de Tecnología Electrónica, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012 Sevilla, Spain, jjchico@dte.us.es
[6]Alejandro Muñoz Rivera, Dpto. de Tecnología Electrónica, Universidad de Sevilla, Av. Reina Mercedes s/n, 41012 Sevilla, Spain, amrivera@dte.us.es

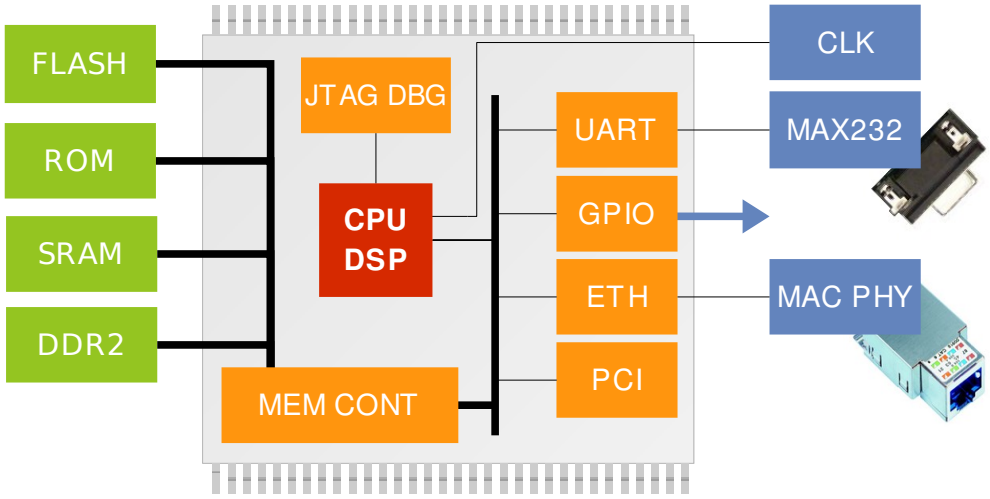

FIGURA. 1
CONCEPTO DE SYSTEM-ON-CHIP

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 como principal característica el ser completamente libre.

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 1 se da una definición básica del hardware libre y de las ventajas que aporta su uso al entorno académico. En la sección 2 se analiza la iniciativa estandarizadora de Opencores. En las secciones 3 y 4 se describen el procesador (OpenRisc 1200) y el bus del sistema (Wishbone [22]) que hemos utilizado, reseñando sus características principales. En la sección 5 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.

Hardware y Licencias Libres

Cuando hablamos de hardware libre, estamos refiriéndonos a aquel que se distribuye con licencias libres. Existen multitud de éstas con diversos caracteres e incluso incompatibles entre sí. De cualquier modo, todas las licencias libres deben cumplir las cuatro libertades esenciales del software libre:

- Libertad 0: Libertad de uso con cualquier propósito.
- Libertad 1: Libertad de estudiar el código y adaptarlo a tus necesidades.
- Libertad 2: Libertad de distribuir copias.
- Libertad 3: Libertad de mejorar el programa y hacer públicas tus mejoras a la comunidad para beneficio común.

Debemos diferenciar en todo caso el hardware libre del hardware abierto. En el primero se debe exigir que utilice una licencia libre del mismo modo que se hace con el software, mientras que para que el hardware sea abierto, sólo debe ofrecer la posibilidad de consultar sus fuentes sin importar cuan restrictiva sea su licencia.

No cabe duda que en estas nuevas reglas del juego, el mundo académico encuentra un fuerte aliado. Desarrollos que hace pocos años estaban restringidos a pequeños grupos de usuarios o a grandes corporaciones, retornan libres para enriquecerse con la sinergia de la colaboración entre todos los agentes de la comunidad y así poder ser estudiados, reutilizados y mejorados por todos, haciendo las veces de transmisor del conocimiento y evitando reinventar la rueda una y otra vez.

La Iniciativa Opencores

Dentro del marco del hardware libre, surge a finales de los 90 la iniciativa Opencores [23], un amplio grupo de usuarios y empresas interesados en el desarrollo de hardware libre. El modelo de desarrollo se basa en la generación de componentes reutilizables que pueden interoperar bajo ciertas circunstancias, como veremos más adelante, y unirse para formar un sistema digital complejo. Estos componentes, también llamados cores o IP cores, pueden ser considerados bloques básicos de construcción, para una posterior implementación del sistema sobre un ASIC o FPGA. Esta metodología fomenta principalmente el abaratamiento de costes y la reutilización. Asimismo cualquier mejora de los componentes por alguna de las partes, revierte automáticamente en toda la comunidad.

Estos componentes son desarrollados en distintos HDLs y se encuentran con licencias diversas (todas libres). La comunidad Opencores se declara agnóstica con respecto al lenguaje y licencia de sus cores, quedando esta decisión a la elección de los creadores. Este hecho provoca que cuando vayamos a utilizar diferentes cores tengamos que tener en cuenta la posible incompatibilidad entre las licencias, pues no todas ellas son compatibles entre sí, más aún cuando combinemos estos cores con otros no libres. Este hecho es considerado como una manifestación más del carácter libre, pues favorece que los nuevos diseños desarrollados, para ser interoperables, sean liberados también.

En el plano funcional, en Opencores se potencia la interoperabilidad entre cores fomentando el uso de un bus estándar llamado Wishbone [22]. Aunque su uso no es obligatorio, más de la mitad de los cores lo usan como interfaz con el exterior, permitiéndo así poder combinarlos en sistemas de gran complejidad a un coste mínimo. La oferta abarca desde CPUs hasta cores criptográficos pasando por gran variedad de coprocesadores, controladores de comunicación, controladores de memoria, DSPs, etc...

El Procesador Openrisc 1200

OpenRisc 1000 es la arquitectura de procesador en la que se basa la familia de procesadores libres OpenRisc 1xxx [24]. 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.

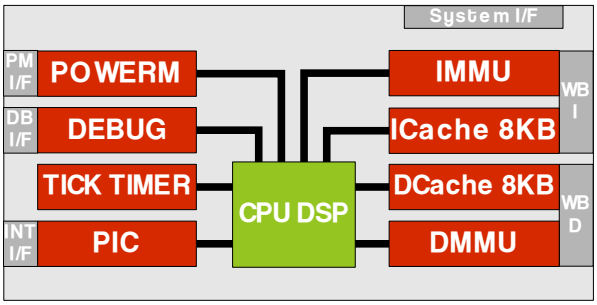

FIGURA. 2
Arquitectura del OpenRisc 1200

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: Instrucciones básicas de 32 bits.
- ORBIS64: Instrucciones enteras y load/store de 64 bits.
- ORFPX32: Instrucciones en punto flotante de precisión simple.
- ORFPX64: Instrucciones en punto flotante de precisión doble.
- ORVDX64: 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).

Está compuesto por varios bloques cuyas características principales listamos a continuación:

- Unidad de Gestión de memoria (MMU).
- Arquitectura Harvard de 32 bits.
- Cachés separadas de datos e instrucciones.
- Gestión de energía (Power Management Unit).
- Reducción de la potencia de 100x a 2x.
- Frecuencia de reloj controlada por software en modos slow e idle.
- Interrupción wake-up en los modos doze y sleep.
- Unidad de depuración avanzada.
- Depuración no intrusiva.
- Acceso y control a la unidad de depuración desde el RISC y el interfaz externo.
- Medición precisa del tiempo.
- Multiplicador Hardware.
- Controlador programable de interrupciones.
- Doble interfaz Wishbone para datos e instrucciones.
- On-chip RAM.
- Controlador uart.
- Controlador Ethernet.
- Variedad de periféricos listos para usar con el OpenRisc 1200.
- Nuevas instrucciones definidas por el usuario.

Como hemos mencionado, una de las características más interesantes del OpenRisc 1200 es su capacidad de adaptación y personalización para acomodarse 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 [25] y Modelsim de Mentor Graphics [26]. 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 [27], Xilinx XST [28] y Altera QIS [29]. 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 [30] con su chip Marvin o Vivace [31] con su Vivid Media Processor [32], en los cuales un core OpenRisc 1200 hace las funciones de cpu principal.

El Bus Wishbone

Dentro del diseño de un SoC, juega un papel fundamental el modo en que vamos a interconectar los distintos componentes. Normalmente se utilizan buses estándares llamados on-chip buses cuya elección vendrá determinada por la cpu y los periféricos, ya que han de ser compatibles.

Dentro de una arquitectura de interconexión, podemos diferenciar dos conceptos: la topología y el protocolo lógico. La topología se refiere a la forma, física o lógica, de los caminos de datos entre los distintos componentes. Por otro lado, el protocolo lógico da las reglas de comunicación para que el sistema funcione a través del bus físico. Cada arquitectura de interconexión estándar, suele definir uno o varios protocolos lógicos interoperables a través de puentes y diferentes posibilidades en cuanto a topología se refiere. Las arquitecturas más utilizadas en el mercado actualmente son: ARM AMBA [33], IBM CoreConnect [34], Altera Avalon [35] y Wishbone.

De las opciones anteriores, en Opencores se ha optado por el bus Wishbone como estándar de interoperabilidad entre cores por varios motivos. Quizás el más importrante no responda al criterio tecnológico, sino a que es el único que está libre de patentes, royalties y copyright [22], lo que hace que su uso sea gratuito. Aun así, hay claros argumentos tecnológicos que hacen de la elección de Wishbone como arquitectura estándar una opción a tener en cuenta.

En el diseño de Wishbone se han buscado dos objetivos principalmente: 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 una 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...

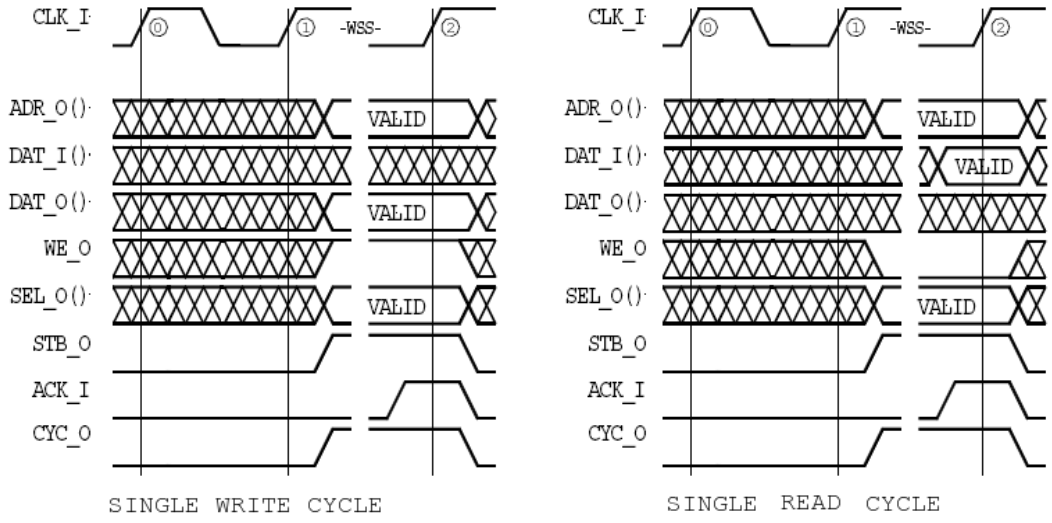

FIGURA. 3
CICLOS CLÁSICOS DE ESCRITURA Y LECTURA EN EL BUS WISHBONE

Las características principales de Wishbone son las siguientes:

- Un solo bus para todas las aplicaciones.
- Arquitectura simple y compacta.
- Soporte multi máster.
- Espacio de direcciones de 64 bits.
- Decodificación parcial o total de direcciones.
- Ancho del bus de datos de 8 a 64 bits expandibles.
- Ciclos de escritura y lectura atómicos.
- Ciclos de evento.
- Soporte para reintentos.
- Soporte para dispositivos lentos.
- Etiquetas definidas por el usuario para identificar los ciclos.
- Arbitrador definido por el usuario.

De lo anterior se desprende que Wishbone encuentra un campo de aplicación prácticamente ilimitado. Abarcando desde simples diseños de baja velocidad a sistemas complejos de alto rendimiento sin ningún coste económico para aquellos que lo implementen.

La metodología de diseño de un SoC basado en Wishbone vendrá determinada fundamentalmente por el tipo de topología que utilicemos y de los parámetros de configuración que establezcamos.

En nuestro caso vamos a utilizar una topología de bus compartido con ocho slots para masters y nueve slots para slaves, con decodificación parcial de direcciones y un bus de datos de 32 bits.

METODOLOGÍA DE DISEÑO

Para la obtención de una implementación basada en OpenRisc propondremos una metodología de diseño de doble vía. 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 éstas se pudieran dar.

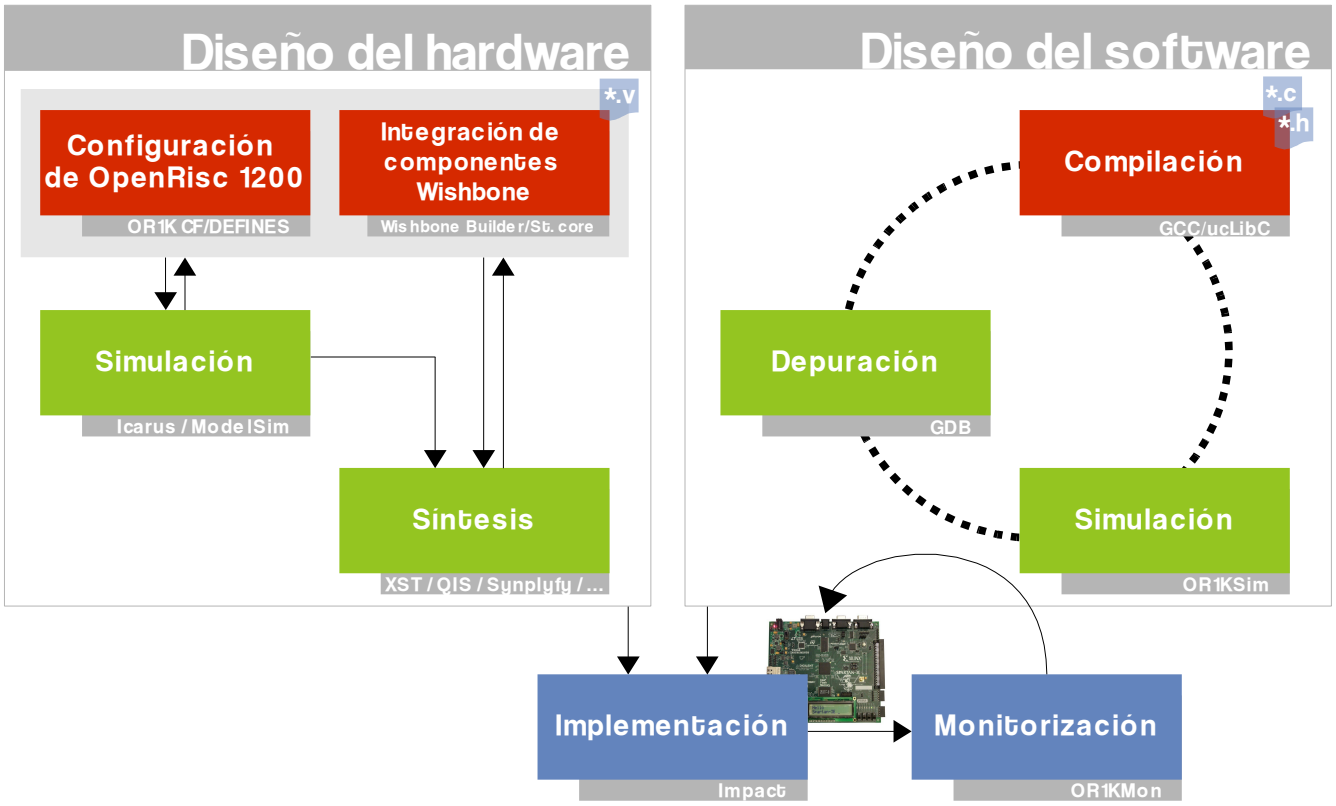

FIGURA. 4
FLUJO DE DISEÑO DE LA METODOLOGÍA

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.

CICLO DE DISEÑO DEL 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 una herramienta gráfica, OR1K Graphic Cconfiguration Tool, que se encuentra disponible en Opencores, a través de la cual se muestran los diferentes parámetros agrupados en varios apartados y se permite modificar sus valores. Alternativamente, para realizar estos cambios podemos prescindir de esta herramienta y configurar directamente los parámetros modificando el fichero de configuración del OpenRisc, cuyo formato no es más que una serie de sentencias de tipo *'define* para el preprocesador de Verilog.

Una vez configurado el microprocesador, debemos crear el bus de interconexión del sistema. Este bus conectará las dos interfaces Wishbone del procesador (datos e instrucciones) 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 ocho slots para interfaces Wishbone maestras y nueve para interfaces Wishbone esclavas. Aunque en la mayoría de las ocasiones este módulo será suficiente, podemos optar por crear uno propio. Para ello, además del método de diseño manual, disponemos de una herramienta que automatizará el proceso: Wishbone Builder; ésta a partir de un archivo con las especificaciones del bus requerido, generará un core sintetizable.

Acerca del bus del sistema, en Opencores se propone una asignación estándar del mapa de memoria y de los códigos de interrupción. La finalidad de esta iniciativa es promover la interoperabilidad del software desarrollado entre las distintas plataformas que utilizan OpenRisc. Esta iniciativa se denomina OpenRisc Reference Platform u OrpSoC y su adopcion es altamente recomendable en nuestros diseños para asegurar así que todo el software disponible sea compatible con las plataformas desarrolladas.

Para terminar el diseño, tenemos que crear la entidad superior del sistema, en la que deberan 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 [25], 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. 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, tras 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.

Ciclo de Diseño del Software

Aunque el ciclo de desarrollo del software del sistema empotrado pueda tener lugar con independencia del ciclo de desarrollo del hardware, debemos tener en cuenta las características de éste, ya que este proceso estará determinado por las características y periféricos que hayamos seleccionado y que más tarde estarán disponibles en la plataforma real.

Para desarrollar el software del sistema debemos tener en cuenta que nuestra arquitectura, OpenRisc, no ha sido nunca antes utilizada por otros procesadores por lo que prácticamente todo el software disponible ha sido creado ad-hoc para este modelo de procesador, el 1200. Aun así, esta circunstancia no ha sido traba para que la comunidad de desarrolladores de Opencores haya portado a esta plataforma multitud de herramientas de desarrollo y sistemas operativos, por lo que actualmente, el ciclo de desarrollo del software no difiere en gran medida del utilizado en otras arquitecturas empotradas como por ejemplo Leon 3 [36].

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 [37] o el depurador GDB [38].

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 [39], 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 [40]. 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 interfaz JTAG [41]. 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.

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 funcion de las vías que se tomen. Como última ventaja, está el abaratamiento de costes, pues hemos utilizado cores libres cuyo uso está exento de pagos, licencias comerciales o royalties. También se han utilizado herramientas libres y gratuitas excepto en las últimas fases de la implementación del hardware en las que no existe alternativa a las propietarias.

Los satisfactorios resultados de esta experienca 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] Opencores project, http://www.opencores.org.

[24] Lampret, D., Chen, C., Mlinar, M., Rydberg, J., Ziv-Av, M. et al "OpenRISC 1000 Architecture Manual", Opencores, 2004.

[25] Icarus Verilog, http://www.icarus.com/eda/verilog/ .

[26] Modelsim, de Mentor Graphics, http://www.model.com.

[27] Synplify, de Synplicity Inc., http://www.synplicity.com.

[28] Xilinx, "Xilinx Synthesis Technology User Guide 9.2", 2007.

[29] Altera, "Quartus II Integrated Synthesis ", 2007.

[30] Flexitronics Inc., http://www.flextronics.com.

[31] Vivace Semiconductor, http://www.vivacesemi.com.

[32] Vivace Semiconductor, "VSP100 Mobile Processor Product Brief".

[33] ARM Limited, "Advanced Microcontroller Bus Architecture (AMBA) Specification, v2.0", 1999.

[34] IBM Corp., "CoreConnect Bus Architecture", 1999.

[35] Altera, "Avalon Bus Specification Reference Manual", 2002.

[36] Ostúa, E., Juan, J.,Viejo, J., Bellido, M., Guerrero, D. et al "A SoC Design Methodology for LEON2 on FPGA", 2006

[37] GCC, http://gcc.gnu.org/

[38] Stallman, R., Pesch, R., Shebs, S. et al. "Debugging with gdb", Free Software Foundation, 2006.

[39] uClinux, Embedded Linux/Microcontroller Project, http://www.uclinux.org.

[40] DRP, http://www.eecg.utoronto.ca/~jarvin/or32/

[41] Milnar, M., "GDB for OR1k", 2006.