Universitat
Autònoma
de Barcelona

ENTORNO GRÁFICO DE CONFIGURACIÓN PARA EL SOFT-CORE OPENRISC

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per Felipe Pinto Buerba
i dirigit per Marc Moreno Berengué

Bellaterra, Septembre de 2010

ÍNDICE

ÍNDICE DE FIGURAS

El sotasignat, Marc Moreno Berengué

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Felipe Pinto Buerba

I per tal que consti firma la present.

Signat: .....

Bellaterra, 6 de Setembre de 2010

Agradecimientos

En primer lugar, agradecer a mi tutor su tarea de guía y consejo durante todo el desarrollo de este proyecto.

En segundo lugar a mi familia, que me ha ayudado en todos estos años de carrera, sobre todo en los malos momentos.

Y finalmente a mis compañeros de carrera, que sin ellos este camino no habría sido tan agradable como lo ha sido, compartiendo preocupaciones, alegrías, miedos, pero ante todo grandísimos momentos.

Gracias a todos.

Capítulo 1

Introducción

1.1 Motivación

Mi motivación personal para el proyecto final de carrera era el poder construir o realizar un portal web mediante una base de datos, que ofreciera un servicio al demandante. Pero de lo que trataría o se encargaría el portal web tendría que ser medianamente interesante Por este motivo, primeramente mi decisión como proyecto fue la elección de éste, entre otros expuestos, porque tenía la posibilidad de construir un portal web. Ahora bien, había que mirar si el contenido de este proyecto escogido, del que jugaría este portal web fuera interesante. El tema que ofrecía este proyecto, girando alrededor de un entorno gráfico basado en web, me intereso desde el primer momento, ya que no era el simple portal web de agencia de viajes, tienda on-line o basa de datos de un hospital, por poner unos ejemplos. Se centraba en intentar construir un portal para configurar un soft-core, principalmente el soft-core de OpenRISC. Con la motivación extra de que, dado que no existe nada en el mercado que realice esta tarea o funcionalidad, tampoco en Opencores, puede llegar a ser como una especie de primer prototipo. Y por esto sería interesante su realización y dado que como es de especificación abierta, podría ayudar a la comunidad en este primer paso.

Por eso, me pareció bastante interesante entrar en las entrañas de este tema, dado que se adentra por una parte en el tema de base de datos y web, que es lo que yo quería, y por otro lado adentra en el tema hardware y así poder asimilar bastantes conocimientos nuevos al respecto.

Todo esto llevaba a utilizar herramientas que yo deseaba poner en práctica, según mis conocimientos obtenidos durante estos años de carrera, para la realización del proyecto final de carrera, más las añadidas por la otra parte, dando un proyecto robusto y completo.

1.2 Objetivos del proyecto

El objetivo principal del proyecto es poder crear un entorno gráfico para generar SoCs basados en el procesador Soft-Core OpenRISC.

El funcionamiento principal del entorno es poder agregar componentes a tu SoC, de manera dinámica, que el propio entorno te cree de forma automática todo el sistema sin la necesidad de que el programador tenga que ir línea a línea para poder conectar/insertar los módulos al sistema.

El entorno web permitirá mostrar todos los componentes disponibles en sus librerías, unirlos al bus del sistema y combinarlos alrededor del procesador softcore OpenRISC. Las funciones principales que se podrán realizar son:

- Añadir controladores (UART, RAM, etc) y memorias On-Chip al bus del sistema.
- Visualizar el mapa de memoria de forma dinámica.
- Visualizar la librería de componentes disponibles.
- Agregar nuevos componentes a la librería de componentes.
- Generar el sistema confeccionado con el entono web.

Al tratarse de OpenRISC [1], estamos hablando de código abierto, donde el código fuente está en public domain y distribuido bajo una licencia LGPL (*Lesser General Public License)* que permite que este Soft-Core pueda ser manipulado por mucha gente. Por eso este proyecto pretende realizar un trabajo que pueda ser de ayuda para esta comunidad.

[1] ***OpenRISC*** *es un diseño de CPU* RISC *de especificación* libre*, realizado por* OpenCores *y publicado bajo la licencia* LGPL*.*

1.3 Organización de la Memoria

El contenido de esta memoria se divide en 5 puntos, además de la bibliografía y anexos.

El primer capítulo sirve para introducir el tema y poder situar el contexto con el que se va trabajar, con el propósito de situar al lector.

El segundo capítulo trata de informar sobre el método de trabajo y el entorno donde se moverá el proyecto. Se introducen los conceptos y tecnologías básicas como: FPGA's, SoCs e IPs y soft-cores de 32 bits (LEON, MICROBLAZE, NIOS II). Se muestra la variedad de soft-cores que existen en la realidad, para que el lector se familiarice con el tema a explicar, antes de entrar en el tema principal en el que se basa este proyecto. Dicho tema se centra mayormente en el funcionamiento y las características del soft-core de 32 bits, OpenRISC, donde se profundizará en su entorno, su arquitectura, operaciones, registros y bus Wishbone, que es el bus que utiliza este diseño. Al finalizar este capítulo se explicará en detalle el entorno de desarrollo web del proyecto.

En el tercer capítulo entraremos en la parte práctica del proyecto. Diseño del entorno gráfico web, especificación y análisis de requisitos, diseño de los módulos del sistema, base de datos y las herramientas con las que se ha llevado a cabo este entorno.

En el cuarto capítulo se explicarán los diferentes tests que se han podido realizar durante el desarrollo del proyecto y se analizarán los resultados obtenidos, como también la validación del sistema. Finalmente se analizarán también los recursos de la aplicación en su totalidad.

En quinto y último capítulo se explicarán las conclusiones que se han obtenido al terminar este proyecto, junto con la posible evolución futura del sistema realizado y finalmente la experiencia personal y profesional.

Capítulo 2

Metodología de trabajo i entorno

Este capítulo intentará introducirnos en el mundo de las plataformas reconfigurables.

2.1 Estado del Arte

Este proyecto se desarrolla alrededor de los procesadores soft-cores de 32 bits, y especialmente alrededor del OpenRISC. Antes de entrar en materia sobre esta CPU RISC introduciremos los conceptos más importantes relacionados con las plataformas reconfigurables como son: los sistemas empotrados o embedded systems, FPGA's, los SoC, Ip's y procesadores soft-cores 32 bits.

Primero empezaremos a hablar de los *embedded systems o sistemas empotrados. Un sistema empotrado* es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas y que con frecuencia se realizan en tiempo real. Estos sistemas tienen una función específica y no son programables por el usuario. Los usuarios solamente pueden interaccionar a través de una interfaz específica a tal efecto. El hardware de este sistema solamente contiene la arquitectura de procesamiento y la entrada salida necesaria por llevar a cabo la tarea mandada. Este concepto de sistema empotrado es diferente del PC tradicional, entendido como ordenador de propósito general programable.

Las características de los sistemas empotrados son:

- Estos sistemas emplearán una combinación de recursos hardware y software para realizar una función específica.
- Estos sistemas realizan una única función o un conjunto muy limitado de funciones (no suelen ser de propósito general)
- La potencia, el coste y la realizabilidad suelen ser los principales factores de coste. consumos energéticos limitados, sistemas compactos, eficiente en prestaciones, etc.
- El diseño de procesadores de aplicación específica suelen ser un componente significativo de estos sistemas.

Hoy en día, los sistemas empotrados los podemos encontrar en cualquier lugar de nuestra vida, controlando algunas funcionalidades que hace unos cuantos años eran impensables de realizar por una máquina. Además, los sistemas empotrados cada vez

son más complejos debido al gran número de tareas diferentes, que deben realizar. Estos sistemas son utilizados para el control de buena parte de aplicaciones, en los dispositivos electrónicos de consumo (videoconsolas, reproductores de audio/vídeo), en la automoción (control de airbag, climatizador..), en la industria (control de procesos, robótica.), en las comunicaciones (teléfonos móviles, modem...), etc.

2.1.1 FPGAs

El diseño digital ha evolucionado durante todos estos años debido a los avances en las tecnologías de fabricación de circuitos integrados[2]. Hasta la aparición en los años '80 de los primeros dispositivos programables, las opciones viables para diseñar sistemas digitales eran el uso de circuitos integrados estándar de baja o media escala de integración, o el diseño de circuitos integrados a medida conocidos como ASICs (*Application Specific Integrated Circuits*)[1]. Un ASICs es un circuito integrado hecho a la medida para un uso en particular, en vez de ser concebido para propósitos de uso general. Pero estas dos opciones tenían problemas de flexibilidad a la hora de implementarlos por eso una de las alternativas a estos problemas fue la aparición de dispositivos lógicos programables, permitiendo gran flexibilidad a cambios de diseño futuros. Estos dispositivos son llamados FPGAs.

Estructura

Una **FPGA** (*Field Programmable Gate Array*) es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad se puede programar. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip, SoPC, (*System on Programmable Chip*), sistemas embebidos, aplicaciones de procesamiento digital de señales, etc.

Parte de la información de a continuación está extraída de [2]:

Las FPGA son la evolución tecnológica de los PLA's (*Programable Logic Array*) que fueron desarrollados en los años 70. Los PLA's esta constituidos por 2 planos de puertas lógicas, una OR y una AND. Éstos están separados entre sí por una pequeña zona divisoria que es la parte de conexionado, y a la vez tienen por el otro extremo una conexión a dos zonas externas donde se realiza la E/S. El usuario puede realizar cualquier función booleana con puertas AND y OR haciendo el conexionado adecuado.

Los PLA's mayoritariamente utilizan la tecnología de programación OTP (*One Time*

[2] Un **circuito integrado (CI)**, es una pastilla pequeña de material semiconductor, de algunos milímetros cuadrados de área, sobre la que se fabrican circuitos electrónicos generalmente mediante fotolitografía y que está protegida dentro de un encapsulado de plástico o cerámica.

Programable), a pesar de que también existen con EPROM (Erasable Programmable Read-Only) y EEPROM (*Electrically-erasable programmable read-only memory*).

Estos dispositivos todo y ser muy versátiles no disponían ni de flip-flops ni de memoria, elementos que en cambio si incorporan las FPGA.

Las FPGA están formadas por tres elementos:

- Bloques lógicos (CLB's).
- Bloques de entrada / salida (IOB).
- Interconexiones

En la siguiente figura podemos observar su distribución:

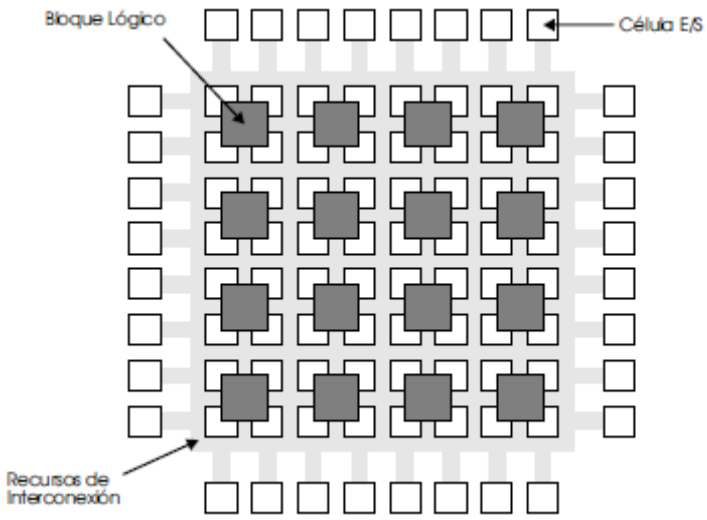


Figura 2.1 Estructura interna de una FPGA.

Los **CLB's**, también denominados LE (Logic Elements) son los bloques principales de una FPGA, y están formados de puertas lógicas, flip-flops y memoria.

Cuando configuramos la FPGA estamos definiendo la funcionalidad de cada uno de los CLB's que intervendrán en el funcionamiento, puesto que es posible que no sea necesario utilizarlos todos, y el conexionado entre CLB – CLB y el CLB – E/S.

Clasificación

Podemos clasificar las FPGA's de muchas maneras, por ejemplo:

- Capacidad de memoria
- Pins E/S
- Velocidad
- Tecnología de configuración

Tecnologías de memoria de programación

Las tecnologías de la memoria de programación de las FPGAs más comúnmente halladas son:

- **SRAM (RAM estática):** En éstas, el contenido se almacena mediante un proceso de configuración en el momento de encendido del circuito que contiene al FPGA.

- **Antifuse (antifusible):** un FPGA que utiliza este tipo de tecnología sólo puede ser programada una vez OTP (*One Time Programed*).

- **Flash:** El avance experimentado en los últimos años en el diseño y prestaciones de las celdas de memoria flash ha permitido su incorporación a los dispositivos programables.

Implementación

Para implementar un diseño en un FPGA se utilizan lenguajes de descripción de Hardware HDL (*Hardware Description Languages*)[3], como por ejemplo, VHDL (Very Hard Description Language) fue desarrollado como un lenguaje para el modelado de sistemas digitales. Proporciona una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento de hardware.
VHDL es un estándar de la IEEE, favoreció su adopción en la industria lo que se ve reflejado en las constantes mejoras en las herramientas. Debido a su estandarización, un código en VHDL puede ser portado a diferentes herramientas y también, puede ser reutilizado en diferentes diseños.

Características

Las características de las FPGA son su enorme flexibilidad, capacidad y volumen de procesamiento, y alta velocidad de reprogramación entre otros. Esto los convierte en dispositivos idóneos para:

- Prototipado rápido de sistemas digitales.
- Procesamiento de señal digital, por ejemplo de vídeo.
- Simulación y depuración en el diseño de ASICS.

Las FPGAs aparecen en el mercado en 1984, con una idea central: permitir realizar un circuito integrado a medida, sin los riesgos económicos asociados a las otras opciones tecnológicas. Hoy las FPGAs están presentes en campos tan diversos como la automoción, la electrónica de consumo, o la investigación espacial. La tecnología

FPGA tiene una aplicación horizontal en todas las industrias que requieren computación a alta velocidad. Algunos ejemplos de su uso podrían ser: arcos de seguridad de bancos, control industrial, equipos de transmisión de TV, Internet, electrónica espacial, seguridad electrónica, equipos de medicina y radiología, comunicaciones por fibra óptica. etc.

Fabricantes de FPGA tenemos por ejemplo: Actel, Altera, Atmel, Xilinx, Motorola, etc.

2.1.2 SoCs e IPs

SoCs

Un SoC (*System-on-Chip*) [4] se basa en la idea de integrar todos los componentes de computación u otros sistemas electrónicos, en un solo chip.

Las características de los SoC son:

- Dimensión reducida: más componentes en el mismo dado de silicio.
- Mejores rendimientos: las comunicaciones entre los componentes del mismo dado de silicio son más rápidas que las comunicaciones entre elementos de diferentes chips.
- Reducción del consumo: puesto que se reduce la utilización de los pads E/S y la longitud de las líneas de transmisión. Los pads son unos de los componentes que consume mucha energía.
- Reducción de los costes: más pads E/S incrementan el coste del chip al igual que tener un PCB más complejo.

Normalmente, un SoC básico está compuesto por un (o varios) procesadores, bloques de memoria, un bus central y diferentes controladores E/S colgados de este bus. Hoy en día, los SoC son cada vez más complejos e integran diferentes tipos de procesadores y controladores en un mismo chip.

La mayoría de SoC se desarrollan a partir de bloques prediseñados, como son los IP Cores (bloques prediseñados de propiedad intelectual), junto con el software de los drivers que controlan su operación. Los bloques hardware se conectan mediante herramientas EDA (*Electronic Design Automation*), mientras que los módulos software son integrados usando un entorno de desarrollo de software.

Dentro de los grupos de los SoC, existe un subgrupo denominado *System on Programable Chip*. Este grupo tiene los mismos componentes que un SoC, pero incorpora una parte de lógica programable para poder implementar componentes a medida o específico, para así, poder aumentar el rendimiento de nuestro sistema.

IPs

Hasta ahora para llevar a cabo el diseño de hardware se estaban realizando mediante capturas de esquemáticos sobre elementos importados de las librerías de la tecnología de fabricación con la que vamos a fabricar nuestro ASIC. Este método, muy ligado a cada proyecto en concreto, compuerta muy poca reusabilidad de módulos entre proyectos. Pero con la aparición de los lenguajes HDL, comentados anteriormente, han permitido realizar diseños de una manera más ágil, llevándonos a obtener una portabilidad donde se pueden rehusar módulos ya desarrollados y testados.

Este diseño basado en la reutilización de módulos permite ensamblar sistemas complejos a través del uso de pequeños componentes. Estos módulos reutilizables se denominan IPs (*Intellectual Propierty*), y permiten la optimización de los recursos debido al reducido tiempo y coste de desarrollo.

Dentro de una FPGA o ASIC se puede incluir la funcionalidad de varios circuitos integrados. Esta funcionalidad puede ser desarrollada por el mismo equipo de trabajo o adquirida a través de un tercero. Debido a que estas funcionalidades son como componentes electrónicos, pero sin su parte física, se los suele llamar componentes virtuales. Por tanto se podrían denominar como bloques prediseñados y testados, para ser incorporados en cualquier proyecto, y pensados para ser reutilizados.

Los IP cores pueden ser muy complejos, como el IP de un microprocesador, o muy simples, como en el caso de un IP core para controlar un GPIO. Alrededor de los IP cores , existe todo un mercado dónde podemos encontrar empresas dedicadas al diseño y soporte de IP, herramientas específicas, y también comunidades de código libre. Los IP cores de estas comunidades de código libre, normalmente están sujetas a licencias de código libre (GPL, etc).

Podemos encontrar 3 tipos de IPs [2]:

- Hard_ IP: la compañía nos da el layout, de una tecnología en particular, para generar la máscara del componente y la documentación asociada (funcionalidad, interfaz, etc). Estos tipos de IP cores no nos permiten cambiar su diseño, pero son componentes muy testados y optimizados por esta tecnología.

- Soft_ IP: la compañía nos da el código HDL que puede ser sintetizado por las herramientas habituales de síntesis. Esta alternativa es más flexible que la anterior, puesto que a través de su código fuente, podemos modificar el IP. Muchos de este IP están escritos en VHDL[3] o Verilog[4].

[3] **VHDL** es un lenguaje definido por el IEEE (*Institute of Electrical and Electronics Engineers*) (ANSI/IEEE 1076-1993) usado por ingenieros para describir circuitos digitales.

[4] **Verilog** es un lenguaje de descripción de hardware (HDL, del Inglés Hardware Description Language) usado para modelar sistemas electrónicos. El lenguaje, algunas veces llamado Verilog HDL, soporta el

- Firm_ IP: la compañía nos da una descripción detallada del componente (netlist) y sus restricciones (retardos máximos, etc). Esta alternativa está en el centro de las dos anteriores.

2.1.3 Procesadores Softcores de 32 bits

En los embedded systems, el elemento más importante es el procesador, puesto que él es quien controla la actividad de todo los componentes del sistema. Por esta razón, se define el procesador como componente principal de toda plataforma arquitectural.

Un procesador softcore es una descripción, en lenguaje de alto nivel hardware (HDL), de un modelo específico de procesador, que es fácilmente personalizable por una aplicación específica y sintetizable en un ASIC o FPGA. Estos procesadores son fácilmente integrables en un SoC.

Antes de mostrar algunos ejemplos de softcores, explicaré la arquitectura computacional RISC (R*educed Instruction Set Computer).*

RISC es un tipo de microprocesador con las siguientes características fundamentales:

1. Instrucciones de tamaño fijo y presentadas en un reducido número de formatos.

2. Sólo las instrucciones de carga y almacenamiento acceden a la memoria por datos.

Además estos procesadores suelen disponer de muchos registros de propósito general.

El objetivo de diseñar máquinas con esta arquitectura es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria. Las máquinas RISC protagonizan la tendencia actual de construcción de microprocesadores.

RISC es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeñas y simples que toman menor tiempo para ejecutarse. El tipo de procesador más comúnmente utilizado en equipos de escritorio, el x86, está basado en CISC en lugar de RISC, aunque las versiones más nuevas

---

diseño, prueba e implementación de circuitos analógicos, digitales y de señal mixta a diferentes niveles de abstracción.

traducen instrucciones basadas en CISC x86 a instrucciones más simples basadas en RISC para uso interno antes de su ejecución.

La idea fue inspirada por el hecho de que muchas de las características que eran incluidas en los diseños tradicionales de CPU para aumentar la velocidad estaban siendo ignoradas por los programas que eran ejecutados en ellas. Además, la velocidad del procesador en relación con la memoria de la computadora que accedía era cada vez más alta. Esto conllevó la aparición de numerosas técnicas para reducir el procesamiento dentro del CPU, así como de reducir el número total de accesos a memoria.

Las características que generalmente son encontradas en los diseños RISC son:

- Codificación uniforme de instrucciones (ejemplo: el código de operación se encuentra siempre en la misma posición en cada instrucción, la cual es siempre una palabra), lo que permite una decodificación más rápida.
- Un conjunto de registros homogéneo, permitiendo que cualquier registro sea utilizado en cualquier contexto y así simplificar el diseño del compilador (aunque existen muchas formas de separar los ficheros de registro de entero y coma flotante).
- Modos de direccionamiento simple con modos más complejos reemplazados por secuencias de instrucciones aritméticas simples.
- Los tipos de datos soportados en el hardware (por ejemplo, algunas máquinas CISC tiene instrucciones para tratar con tipos byte, cadena) no se encuentran en una máquina RISC.

Los diseños RISC también prefieren utilizar como característica un modelo de memoria Harvard, donde los conjuntos de instrucciones y los conjuntos de datos están conceptualmente separados; esto significa que el modificar las direcciones donde el código se encuentra pudiera no tener efecto alguno en las instrucciones ejecutadas por el procesador (porque la CPU tiene separada la instrucción y el caché de datos, al menos mientras una instrucción especial de sincronización es utilizada). Por otra parte, esto permite que ambos cachés sean accedidos separadamente, lo que puede en algunas ocasiones mejorar el rendimiento.

Ejemplos de diseño RISC podrían ser:

El ARM (*Advanced RISC Machines*): Se encuentra en dispositivos PALM, Nintendo DS, Game Boy Advance y en múltiples PDAs, teléfonos móviles y reproductores multimedia (como el iPod).

Actualmente se utiliza en muchos sistemas empotrados en automóviles, routers, etc, así como en muchas consolas de videojuegos, como la Playstation 3, Xbox 360 y Nintendo Wii.

Con esta breve introducción a la arquitectura RISC, comentaré cuatro procesadores softcore:

Las características de los procesadores que se explicarán a continuación son extraídas de [4].

Leon 3

AeroFlex Gaisler, que en los inicios fue un spin-off de la Agencia Espacial Europea (ESA) denominada Gaisler Research, es una compañía que provee IP cores y desarrolla herramientas para procesadores empotrados basados en la arquitectura SPARC. Uno de sus productos es la familia de procesadores sintetizables LEON. Jiri Gaisler, buscaba desarrollar un procesador resistente a radiaciones, modular, fácilmente portable, con interfaces estándar y que ejecutara hasta 100 MIPS. Además, el nuevo diseño, escrito en VHDL, se licenció bajo GPL, lo que permitiría integrarlo en un System-on-Chip (SoC).

Al primer prototipo se le llamó LEON-1. Una vez éste fue verificado se comenzó el desarrollo de un procesador más completo, con unidad de coma flotante y nuevos interfaces, estas fueron las versiones LEON-2 y LEON-3. Finalmente, la ESA encargó a Gaisler Research (hoy integrada en el grupo AEROFLEX) el desarrollo de un LEON-4.

El LEON 3 es un procesador de 32 bit escrito en VHDL y basado con una arquitectura SPARC V8. Este es muy parametrizable a través del uso de VHDL genéricos y una sencilla interfaz gráfica.

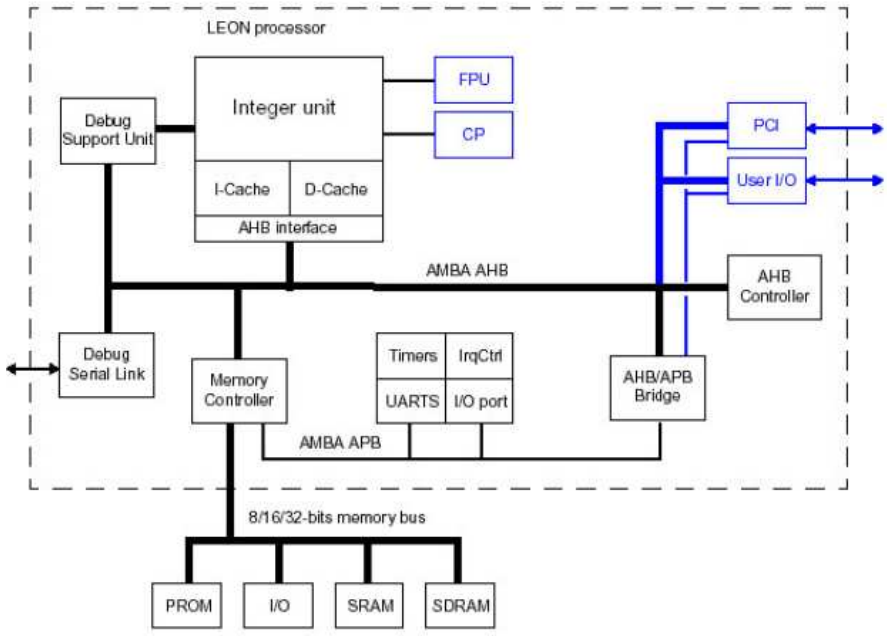


Figura 2.2. Procesador Leon 3.

- Incorpora las extensiones de SPARC v8e.
- Utiliza un pipeline de siete etapas: Fetch, Decode, Register Access, Execute, Memory, Exception,Write.
- Incorpora, de forma opcional, una unidad de coma flotante de alto rendimiento y compatible con el estándar IEE-754.
- Soporte para Multiprocesador.

- Powerdown mode.

MicroBlaze

El MicroBlaze es un procesador RISC de 32 bits optimizado para ser implementado solamente en las FPGA de Xilinx, las Spartan y Virtex . Aún siendo un *soft-processor*, el usuario no tiene acceso a los detalles VHDL del circuito. Sólo puede analizarse a partir de la documentación del fabricante y de los resultados experimentales. Aún así, es una herramienta muy potente para desarrollar proyectos relacionados con arquitecturas paralelas, codiseño y control; y en general, en toda investigación sobre software, puesto que permite una comparación inmediata con otras arquitecturas y metodologías de desarrollo.
El conjunto de características que incluye este procesador está formado por 32 registros de propósito general, palabra de instrucción de 32 bits con 3 operandos y 2 modos de direccionamiento, pipeline y direcciones de 32 bits. Para tal de añadir más características a esta arquitectura predefinida, el MicroBlaze nos permite activar un gran número de funcionalidades.

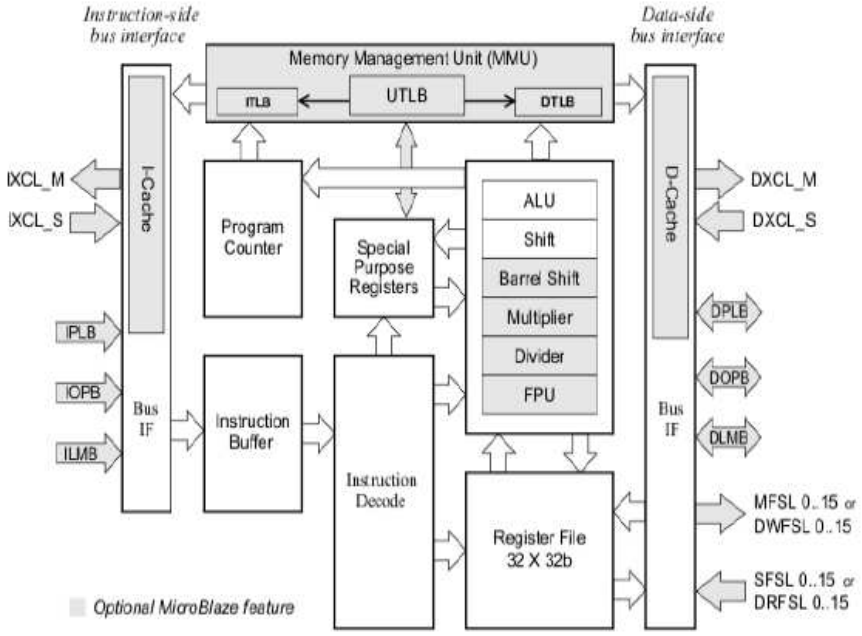


Figura 2.3. Diagrama de bloques del procesador MicroBlaze.

Las principales funcionalidades parametrizables son:

- Dimensiones de las caches de instrucciones y datos.

- Multiplicadores y divisores por hardware.
- Barrel shifter por hardware.
- Memory Management Unit (MMU).
- Floating Point Unit (FPU).
- Fast Simplex Link (FSL) interfaces (0/15).

NIOS II

El procesador NIOS II es similar a los otros microcontroladores softcores. Es un procesador RISC de 32 bits diseñado especialmente por las familias de FPGA de Altera. Las características que podemos destacar de este procesador son: un ISA de 32bits, una palabra de instrucción de 3 operandos, datapath independiente entre instrucciones y datos, 32 registros de propósito general y 256 CI para mejorar el rendimiento de esta arquitectura.

Las herramientas de configuración del NIOS II, nos permiten seleccionar 3 cores diferentes para este procesador:

- NIOS II/e “economy”: El core NIOS II/e está diseñado para ser el más pequeño posible en cuanto al consumo de área. Este core no dispone de caches, pipeline y otros componentes hardware.

- NIOS II/s ”standard” : El core NIOS II/s está diseñado para tener un buen compromiso entre el rendimiento y área utilizada. Este core tiene 5 etapas de pipeline, predictor de saltos estático y posibilidad de cache de instrucciones y datos.

- NIOS II/s “fast”: El core NIOS II/f está diseñado para tener unas altas prestaciones de computación. Como resultado, este core permite ser configurado a través de un amplio abanico de parámetros para tal de mejorar su rendimiento. Básicamente este core está compuesto de 6 etapas de pipline, un predictor de saltos dinámico y la posibilidad de utilizar caches de instrucciones y datos.

Para mejorar las prestaciones del NIOS II/f, podemos seleccionar nuevos elementos como multiplicación y división por hardware, o mejorar la jerarquía de memoria aumentando o disminuyendo la medida de las caches y o/seleccionando las Tightly Coupled Memory (TCM).

Las tres versiones del core, permiten mejorar su set de instrucciones a partir de incorporar hasta 256 *custom instructions*. Éstas pueden aportar un aumento significativo del rendimiento del procesador.

Los cores no incluyen ningún periférico para comunicarnos con el mundo exterior. Puesto que estos cores están pensados para ser conectados del bus Avalon, como un componente más del SoC. Por lo tanto, los cores solamente contienen los circuitos necesarios para implementar una arquitectura NIOS II.

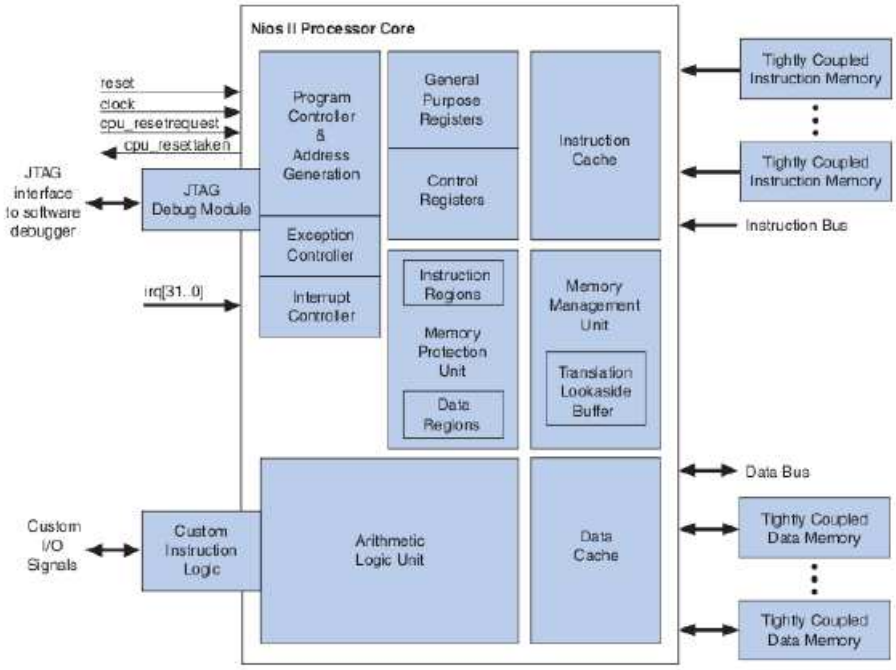


Figura 2.4. Diagrama de bloques del procesador NIOS II.

Las unidades funcionales más destacadas del la arquitectura NIOS II son:

- Interface to custom instruction logic (CI).
- Memory management unit (MMU).
- Memory protection unit (MPU).
- Instruction and data cache memories.
- Tightlycoupled memory interfaces for instructions and data.
- JTAG debug module.

El NIOS II se distribuye como un componente más, dentro del Mega Wizard de Altera SOPC Builde. Como ya hemos comentado, nosotros podemos añadir el NIOS II como un componente más del bus Avalon, y allí sí, podemos incorporar los periféricos necesarios por construir nuestros SoC.

2.2 Entorno de desarrollo OpenRISC

2.2.1 Entorno Hardware

Seguidamente explicaré el softcore que trataremos en este proyecto.

El procesador OpenRisc utilizado para este proyecto es un diseño de CPU RISC de especificación abierta, realizado por OpenCores[5] y publicado bajo la licencia LGPL. El diseño está implementado en el lenguaje de descripción de hardware Verilog y contempla el conjunto de instrucciones ORBIS32 (Instrucciones básicas de 32 bits).

Específicamente el CORE utilizado es el *OpenRisc 1200*:

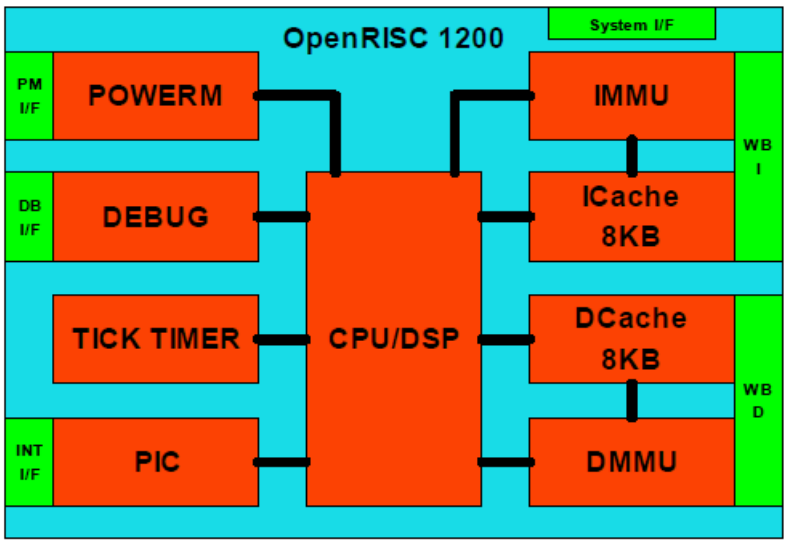


Figura 2.5. Arquitectura del Core OpenRisc 1200.

Este core está completamente realizado en Verilog y contempla el conjunto de instrucciones ORBIS32. El OpenRisc 1200 es un procesador segmentado, con 32 registros de propósito general, con un pipeline de 5 etapas, bus del sistema WishBone, del que luego entraremos en detalle, caches de instrucciones y datos separados, y basadas con la arquitectura Hardvard. Tiene soporte para gestión de memoria virtual e instrucciones básicas para el procesamiento de señales digitales (DSP).

[5] OpenCores es una comunidad de gente que está interesada en el desarrollo de código abierto de hardware digital a través de automatización de diseño electrónico, con un espíritu similar al movimiento del software libre (www.opencores.org).

CPU / DSP es una parte central del procesador RISC OR1200. OR1200 CPU / DSP implementa sólo una parte de 32-bit de la arquitectura OpenRISC 1000. 64 Bit parte de la arquitectura, así como de punto flotante y operaciones de vectores no se implementan en OR1200.

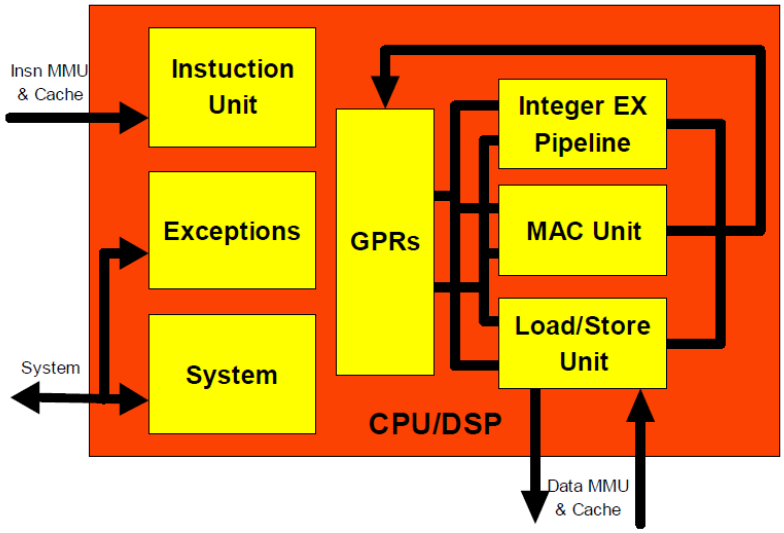


Figura 2.6. Diagrama de Bloque CPU/DSP OpenRisc 1200.

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

- Unidad de Gestión de memoria (*Memory Management Unit* (MMU)).

- Arquitectura Harvard de 32 bits.

- Cachés separadas de datos e instrucciones.

- Hasta 8 Custom Instructions (CI).

- Gestión de energía (Sophisticate Power Management Unit (SPMU).

- 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.

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, Xilinx XST y Altera QIS. De cualquier modo, no resulta difícil adaptarlo a cualquiera otra herramienta de síntesis del mercado.

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, IBM CoreConnect, Altera Avalon y Wishbone.

En nuestro caso, OpenCores utiliza el bus Wishbone, fundamentalmente porque es el único que está libre de patentes, royalties y copyright.

Bus Wishbone

Wishbone intenta ser un bus lógico, no especificando información eléctrica o de topología. Por eso es útil para comunicar diferentes componentes diseñados en lenguajes de descripción de hardware, ya sea Verilog, VHDL u otros.

En el diseño de Wishbone [6] 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 parámetros, como los anchos de los buses, el significado de las etiquetas de datos y direcciones, endianness, niveles eléctricos, etc.

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.

Ejemplo de interfaz e interconexiones en una arquitectura punto a punto [7]:

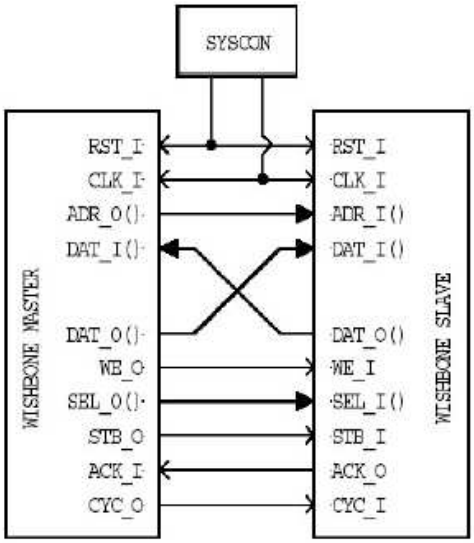


Figura 2.7. Interfaz e interconexiones en una arquitectura punto a punto.

Características técnicas del bus:

- Arquitectura de un bus para todas las aplicaciones.
- Arquitectura simple.
- Multi master.
- Espacio de direcciones de 64 bits.
- Bus de datos de 8 a 64 bits.
- Basado en protocolos estándares de transferencia de datos.
- Soporta varios tipos de interconexiones.
- Protocolo de handshake para regular la velocidad de transferencia de datos.

- Soporta varias terminaciones de ciclos.
- TAGs definidos por el usuario para identificar el tipo de transferencia de datos.
- Basado en arquitectura Maestro/Esclavo.

Comunes a Master y Slave

- CLK_I: entrada de reloj.
- RST_I: reset del diseño.
- DAT_I() y DAT_O(): buses de entrada y salida de datos.
- TGD_I() y TGD_O()
    - Opcional.
    - Lleva información asociada al bus de datos.
    - Validas cuando STB_O activo.
    - Ej: paridad.

MASTER	**SLAVE**
ADR_O(N..n) - N: Límite superior dado por el ancho del bus de direcciones - n: límite inferior dado por la granularidad del bus de datos.	ADR_I(N..n) - N: Límite superior dado por el ancho del bus de direcciones - n: límite inferior dado por la granularidad del bus de datos.
- CYC_O: Indica que se está llevando a cabo un ciclo de bus válido. Se activa al comienzo del ciclo y permanece hasta el final.	- CYC_I: Indica el comienzo de un ciclo.
WE_O: Indica si el ciclo es de lectura o escritura.	WE_I: Indica si la transacción es de lectura o escritura.
STB_O: indica que se está llevando a cabo un ciclo valido de transferencia de datos.	STB_I: indica que se está llevando a cabo un ciclo valido de transferencia de datos.
ACK_I: recibe la confirmación de una transferencia	ACK_O: recibe la confirmación de una transferencia
ERR_I: recibe la indicación de un error en la transferencia.	ERR_O: recibe la indicación de un error en la transferencia.
RTY_I: recibe pedido de re-transmisión de datos.	RTY_O: recibe pedido de re-transmisión de datos.
SEL_O(): asociado con la granularidad, indica donde hay o donde espera datos validos en el bus.	SEL_I(): asociado con la granularidad, indica donde hay o donde espera datos validos en el bus.

WISHBONE facilita el trabajo a los diseñadores, ya que estandariza la interfaz.

Está basado en una arquitectura de Maestro/Esclavo y se comunican a través de una interfaz usualmente denominada INTERCON.

Funcionamiento general:

- Reset

El reset se produce en forma síncrona.

Todas las interfaces WISHBONE deben de inicializarse con el primer flanco de subida en el que RST_I este activo.

- Inicialización de ciclo de transacción

Se indica que hay un ciclo válido si CYC_O está activo. Cuando CYC_O es 0 ninguna de las otras señales del MASTER tienen sentido.

- Protocolo de Handshake

Handshake permite adecuar la velocidad de transferencia de datos, e indicar errores y reintentos.

El hanshake para la transferencia de datos es sumamente sencillo, el MASTER activa STB_O y lo mantiene así hasta que el esclavo activa alguna de las señales ACK_I, ERR_I, RTY_I. Al ocurrir esto el Maestro desactiva la señal.

Las señales de respuesta ACK_O, ERR_O, RTY_O deben generarse solo si están activas CYC_I y STB_I.

Las interfaces del esclavo deben de ser diseñadas para que ACK_O, ERR_O, RTY_O se activen y desactiven respondiendo a STB_O.

Las señales permiten tres ciclos básicos: Read, Write y RMW.
Las señales no son bidireccionales, siempre son entradas o salidas. Esto es así debido a que muchas veces el diseño puede llegar a querer implementarse en hardware que no soporta internamente señales bidireccionales, como por ejemplo las FPGAs de Altera.

Ciclos READ/WRITE simples

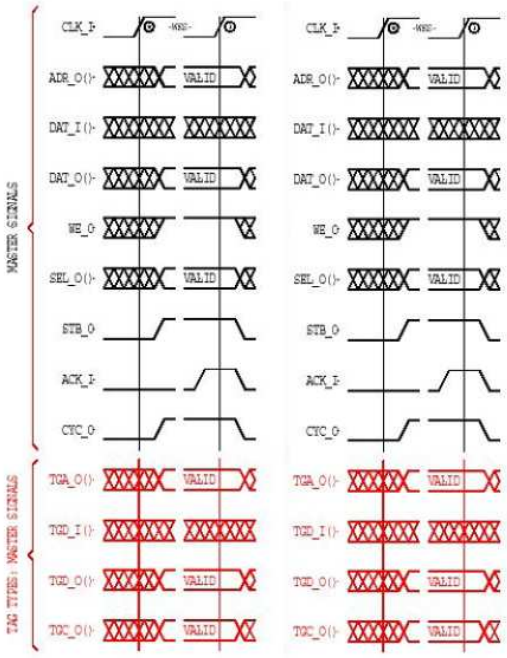


Figura 2.8. Read/Write simples bus Wishbone.

Ciclos READ/WRITE en bloque

Durante las transferencias en bloque, la interfaz básicamente realiza transferencias simples Read/Write.

Este tipo de transferencias es útil, sobre todo en sistemas con varios Master.

Interconexiones de módulos:

Conexión punto a punto:

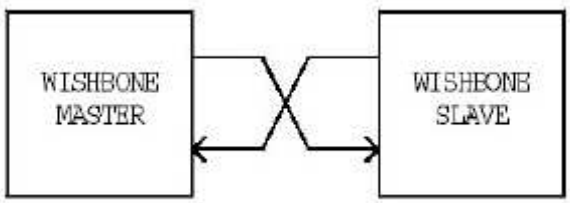


Figura 2.9. Conexión Wishbone punto a punto.

Bus Compartido:

Todos los dispositivos están conectados sobre un mismo bus y cualquier Master puede iniciar una transacción con cualquier esclavo.
Un árbitro es el que determina cuando le toca el turno a cada uno de los Master.

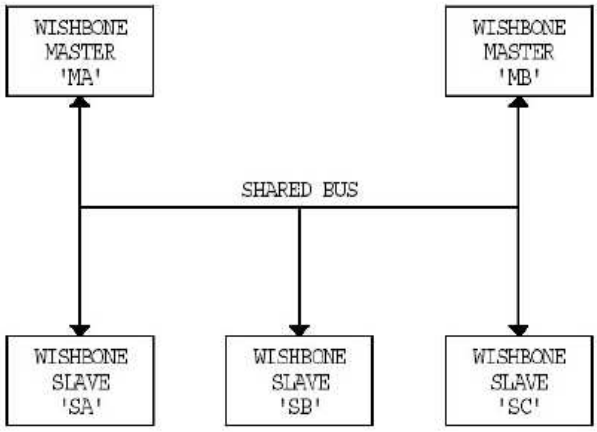


Figura 2.10. Conexión Wishbone bus compartido.

Switch de interconexiones:

Consume más recursos, pero permite varia comunicaciones entre pares Maestro/Esclavo a la misma vez.

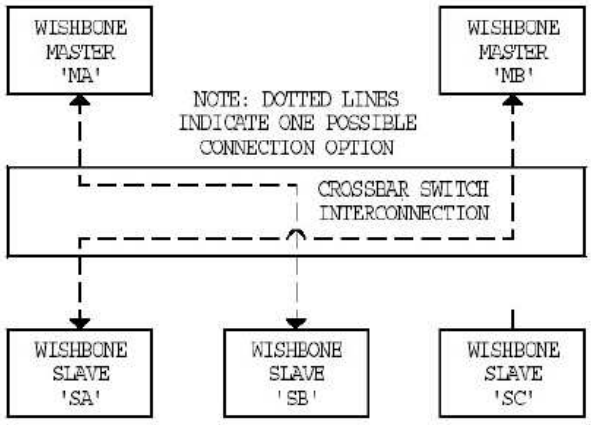


Figura 2.11. Conexión Wishbone con switch de conexiones

Flujo de Datos (Pipeline)

Es una posible arquitectura para implementar diseños que procesen datos en pipeline.

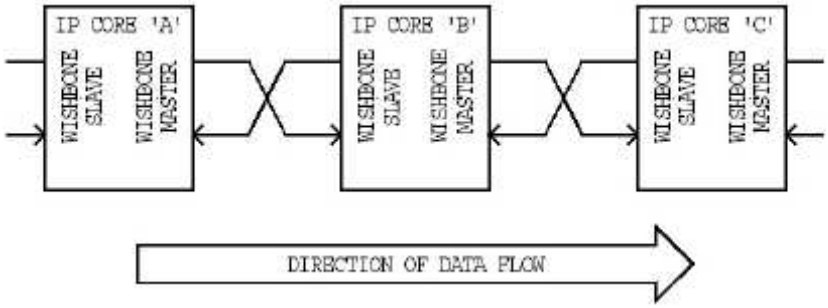


Figura 2.12. Conexión Wishbone con pipeline.

2.2.2 Entorno Software para OpenRISC

Para compilar nuestro software disponemos del conjunto de herramientas de GNU, también llamado GNU toolchain. La GNU toolchain ha sido portada a OpenRISC para permitir el desarrollo en distintos lenguajes. Linux y µCLinux han sido también portados a este procesador.

El *GNU toolchain* es un término que agrupa a una serie de proyectos que contienen las herramientas de programación producidas por el proyecto GNU. Estos proyectos forman un sistema integrado que es usado para programar tanto aplicaciones como sistemas operativos.

El GNU toolchain es un componente vital en el desarrollo del núcleo Linux, el desarrollo del BSD y software para sistemas embebidos. Partes del toolchain de GNU también son usadas en Solaris Operating Environment y la programación de Microsoft Windows con Cygwin y MinGW/MSYS.

Este proyecto aspira a mantener un conjunto de herramientas, de fácil uso, de desarrollo para la plataforma de procesador de OpenRISC.

Las herramientas incluidas son actualmente [12]:

- GNU binutils 2.18.50 (with binutils 2.20.1 available experimentally): *una colección de herramientas binarias.*

- GNU GCC 4.2.2: *colección de compilación que incluye el ANSI C compiler, C++, Java, Fortran y otros compiladores.*
- GNU Make: *proyecto dedicado a la automatización de la estructura y de la compilación.*
- GNU GDB 6.8 (with GDB 7.1 available experimentally): *depurador interactivo.*
- uClibc 0.9.29.
- newlib 1.18.0: *librería destinada al uso en sistemas embedidos.*
- Linux 2.6.24.
- BusyBox 1.7.5.
- or1ksim 0.4.0 (or1skim 0.3.0 for precompiled tool chains).

2.3 Entorno de desarrollo Web

En este apartado se explicará los procedimientos y herramientas para el desarrollo de la página web.

Se ha escogido el uso de WAMP, sistema de infraestructura de internet que usa las siguientes herramientas:

- Windows, como sistema operativo;
- Apache, como servidor web;
- MySQL, como gestor de bases de datos;
- PHP (generalmente), Perl, o Python, como lenguajes de programación.

El uso de un WAMP permite servir páginas html a internet, además de poder gestionar datos en ellas, al mismo tiempo un WAMP, proporciona lenguajes de programación para desarrollar aplicaciones web.

El servidor Apache/2.2.11 fundamentalmente para gestionar un servidor web HTTP.

Lenguaje PHP/5.3.0 para poder interactuar con la Base de Datos.

MySQL como servidor de Base de Datos.

Capítulo 3

Diseño e Implementación

Este capítulo explicará que elementos se necesitarán para crear nuestra aplicación.

3.1 Arquitectura del sistema

Descripción General

Dentro de este apartado explicaremos todo lo necesario para poder llevar a cabo esta aplicación web. Primeramente habrá que diferenciar los principales usuarios que interactuarán con la aplicación. Principalmente sólo hay dos:

- **Administrador:** será la persona encargada del buen funcionamiento de la aplicación, y sobre todo del correcto funcionamiento del código de generación de proyectos para obtener un buen resultado, sin fallos para su comprobación más tarde.

- **Usuario:** el usuario será la persona que mediante sus ficheros .v (verilog) y .vhd (VHDL) pueda generar el proyecto deseado. Con libre acceso para poder subir los ficheros que deseen, borrarlos, etc.

Seguidamente hablaremos de los requerimientos del sistema que serían:

Requerimientos Técnicos:

Servidor:

- *Apache/2.2.11*
- *PhpMyAdmin 3.2.0.1*
- *MySQL 5.1.36*

Requerimientos a nivel de página web:

- Métodos de autenticación para acceso a zonas privada del usuario.
- Detección de subida de ficheros correctos al servidor.

Requisitos funcionales de la aplicación web

La función principal de la aplicación es poder unir módulos insertados por el usuario o de la librería principal de la aplicación, y generar una vista principal del sistema en donde se puedan ver los elementos insertados correctamente para su comprobación y uso posterior. Por tanto se necesitará de un sistema de subida de ficheros, un repositorio de módulos para la librería principal, un entorno de unión de componentes al sistema y el visualizador del sistema.

Otra función de esta aplicación es poder ver el mapa de memoria del sistema según se vayan insertando los módulos de manera dinámica y poder ver las interrupciones.

Diagrama de casos de uso

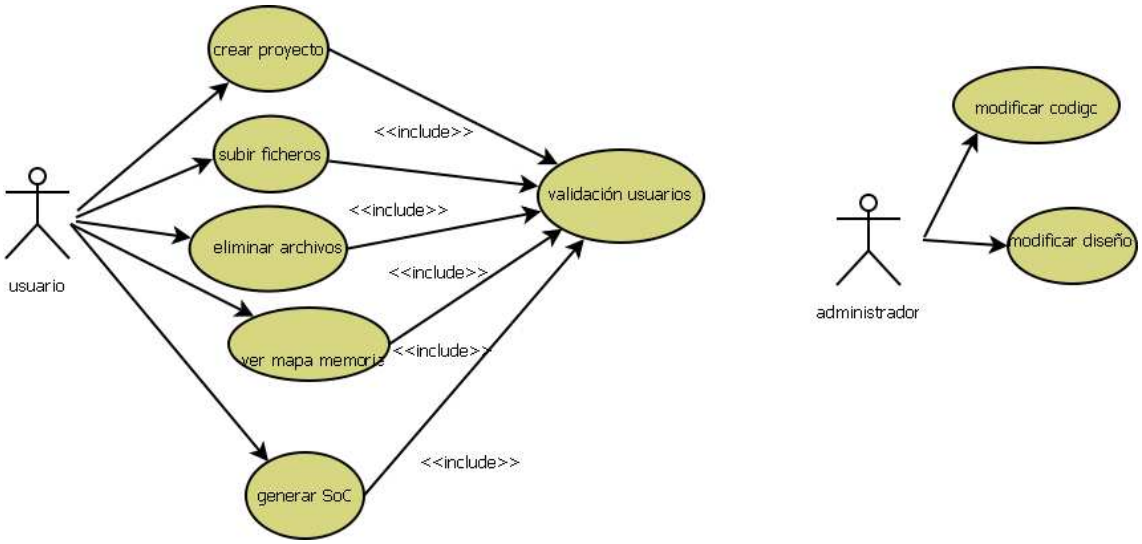


Figura 3.1. Diagrama de casos de uso.

Requisitos de la aplicación

Interface:

- **Conexión:** La aplicación es totalmente on-line y necesita un acceso permanente a la base de datos, implicando un uso del servidor las 24h.

- **Navegadores:** intentar que la aplicación pueda ser visualizada correctamente en cualquiera de los navegadores más comunes que existen a día de hoy (Internet Explorer, Firefox, Google Chrome, Opera, Safari).

- **Sistema Operativo:** La Aplicación esta realizada sobre Windows. Si el usuario usará otro S.O como por ejemplo Linux, se necesitaría adaptar mínimamente la aplicación por ejemplo en el acceso a directorios.

Requisitos a nivel de diseño

- **Seguridad:** El sistema debe disponer de un sistema de autentificación de usuarios.

- **Usabilidad:** Se ha intentado crear una aplicación fácil de usar, con un aspecto sencillo.

- **Mantenimiento:** El administrador debería ser capaz de realizar ampliaciones o modificaciones de los aspectos de la aplicación sin que eso conllevase ningún problema para el correcto uso por parte de los usuarios.

Requisitos de Software externo

Se necesita el uso de un servidor de base de datos y de aplicaciones. El servidor de base de datos debido a que la información debe estar accesible en cualquier momento, y el servidor de aplicaciones, en este caso Php, para poder ejecutar aplicaciones dinámicas.

Bases de Datos

Todos los sistemas gestores que mencionaremos utilizan SQL (*Structured Query Language*), puesto que hoy en día es el lenguaje de acceso a bases de datos relacionales más usado. SQL es un lenguaje declarativo de alto nivel que permite especificar diversos tipos de operaciones sobre las bases de datos, permitiendo recuperar información de una forma efectiva y muy eficiente

Los posibles gestores de base de datos podrían ser Oracle, MySQL, PostgreSQL o el Microsoft SQL Server. El sistema gestor de base de datos escogido para esta aplicación ha sido el MySQL, (*Structured Query Language*), uno de los lenguajes de acceso a base de datos relacionales más utilizado y además con licencia libre, y que no ha hecho falta conocer muy a fondo dado que ya era conocido.

SQL es un lenguaje declarativo de alto nivel que permite especificar diversos tipos de operaciones sobre las bases de datos, permitiendo recuperar información de una forma efectiva y muy eficiente.

MySQL

MySQL [13] es un software de código abierto, licenciado bajo la GPL de la GNU, aunque MySQL AB distribuye una versión comercial, en lo único que se diferencia de la versión libre, es en el soporte técnico que se ofrece, y la posibilidad de integrar este gestor en un software propietario, ya que de otra manera, se vulneraría la licencia GPL.

Características principales:

- El principal objetivo de MySQL es velocidad y robustez.
- Soporta gran cantidad de tipos de datos para las columnas.
- Gran portabilidad entre sistemas, puede trabajar en distintas plataformas y sistemas operativos.
- Cada base de datos cuenta con 3 archivos: Uno de estructura, uno de datos y uno de índice y soporta hasta 32 índices por tabla.
- Aprovecha la potencia de sistemas multiproceso, gracias a su implementación multihilo.
- Flexible sistema de contraseñas (passwords) y gestión de usuarios, con un muy buen nivel de seguridad en los datos.
- El servidor soporta mensajes de error en distintas lenguas

La versión del servidor utilizada es la 5.1.36.

Para la creación de las tablas de la base de datos se ha utilizado la herramienta phpMyAdmin, que explicaremos a continuación.

phpMyAdmin

phpMyAdmin [14] es una herramienta escrita en PHP, de libre distribución con la intención de manejar la administración de MySQL a través de páginas web, utilizando Internet.

La aplicación en si no es más que un conjunto de archivos escritos en PHP que podemos copiar en un directorio de nuestro servidor web, de modo que, cuando accedemos a esos archivos, nos muestran unas páginas donde podemos encontrar las bases de datos a las que tenemos acceso en nuestro servidor de bases de datos y todas sus tablas.

Actualmente puede crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 50 idiomas. Se encuentra disponible bajo la licencia GPL.

Como esta herramienta corre en máquinas con Servidores Webs y Soporte de PHP y MySQL, la tecnología utilizada ha ido variando durante su desarrollo.

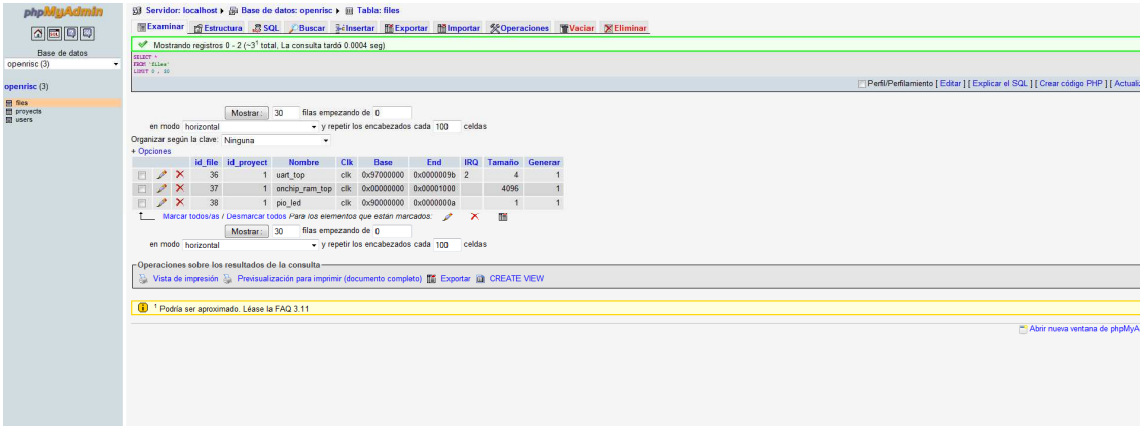


Figura 3.2. Herramienta phpMyAdmin.

Servidor de aplicaciones

Un *servidor de aplicaciones* [15] usualmente se trata de un dispositivo de software que

proporciona servicios de aplicación a los equipos o dispositivos cliente, por lo general a través de Internet y utilizando el protocolo http (*HyperText Transfer Protocol*). Generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones. Los servidores de aplicación se distinguen de los servidores web por el uso extensivo del contenido dinámico y por su frecuente integración con bases de datos.

La característica principal es que permiten la interactividad del usuario con la web,

Por tanto podemos distinguir 2 tipos de Scripts, o instrucciones que darán esa interactividad:

Client Side Scripts: scripts que se ejecutan en el Cliente, en el Navegador

Server Side Scripts: donde los Scripts (instrucciones) se procesan en el Servidor WEB.

Los Scripts Server Side se ejecutan en el servidor y, por tanto, deberemos asegurarnos que nuestro servidor web los soporta si queremos aplicarlos. Para su funcionamiento, el programa se ejecutará en el servidor con los datos o peticiones que el usuario manda desde su navegador y el servidor muestra los resultados del programa en una página HTML que el usuario verá normalmente en su browser.

Los más usados hoy en día son:

CGI Scripts. Abreviatura de "Common Gateway Interface" , protocolo de comunicación entre el servidor web y el cliente, el navegador. Este protocolo puede implementarse en cualquier lenguaje de los utilizados para Internet: PERL, C, Java, Visual Basic, PHP… Se utiliza habitualmente para contadores, formularios, chats, buscadores.

ASP. Abreviatura de Active Server Pages y tecnología propietaria de Microsoft. Se utiliza a menudo para la gestión de Bases de Datos ya que puede conectarse a SQL, Access, Oracle u otras. Requiere por parte del servidor un Microsoft Web Server , el navegador es indiferente pues el trabajo se realiza del lado del Server. Muestra páginas con extensión ASP, que se montan "on the fly" - al vuelo- según unas plantillas y personalizadas según petición de usuario.

PHP. Lenguaje similar al ASP pero código abierto y gratuito. Su gran potencia se encuentra en la interacción con bases de datos más utilizadas: Oracle, Sybase, MySQL.

JSP. Abreviatura de "Java Server Pages". La respuesta de SUN al ASP. No hemos entrado en detalles técnicos pues no era el objeto de este documento.

Finalmente nos hemos decidido por el servidor basado en PHP, debido al rápido aprendizaje, a su facilidad de uso y a la calidad de la documentación asociada, así como al hecho de ser código abierto.

PHP

PHP [16] es un acrónimo recursivo que significa ***P****HP* ***H****ypertext* ***P****re-processor* (inicialmente PHP Tools, o, *P*ersonal *H*ome *P*age Tools) y es un lenguaje de programación de alto nivel interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor (server-side scripting), independiente del navegador que use el visitante, pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica usando las bibliotecas Qt o GTK+.

Una de las características más importantes de PHP es que permite manejar bases de datos desde una interfaz web. Dado que PHP es un lenguaje de código abierto, se puede usar libremente sin necesidad de pagar licencia.

Generalmente se ejecuta en un servidor web, tomando el código en PHP como su entrada y creando páginas web como salida. Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno.

Las principales ventajas de los servidores de aplicaciones que trabajan con PHP son:

- Facilidad de aprendizaje.
- PHP puede funcionar bajo casi todos los sistemas operativos (Unix, Windows, Mac OS).
- Soporta conexiones con casi cualquier tipo de bases de datos.
- PHP tiene licencia open-source, lo cual implica un coste nulo en la adquisición del software del servidor.
- Es fácil y rápido ampliar las funcionalidades del lenguaje PHP y solventar errores, gracias a numerosas extensiones que surgen a menudo gracias a tener licencia abierta.
- PHP permite leer y manipular datos desde diversas fuentes, incluyendo datos que pueden ingresar los usuarios desde formularios HTML.
- Permite técnicas de programación orientada a objetos.

El servidor utilizado para dar soporte al servidor de aplicaciones es el servidor Apache.

Apache

El servidor HTTP Apache [17] es un servidor Web de tecnología Open Source sólido y para uso comercial desarrollado por la Apache Software Foundation (http://www.apache.org) para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.

Es un servidor web que se carga como un servicio más del sistema operativo. Cuando está activo, convierte tu máquina en un servidor capaz de enviar contenido a cualquier navegador.

La versión utilizada es la 2.2.11.

Arquitectura implementada

La arquitectura implementada está formada por: MYSQL para el uso de la base de datos, PHPMyAdmin para la creación de las tablas de la BD y la administración de esta, y el servidor Apache+PHP para dar servicio.

En el siguiente dibujo podemos observar la arquitectura utilizada:

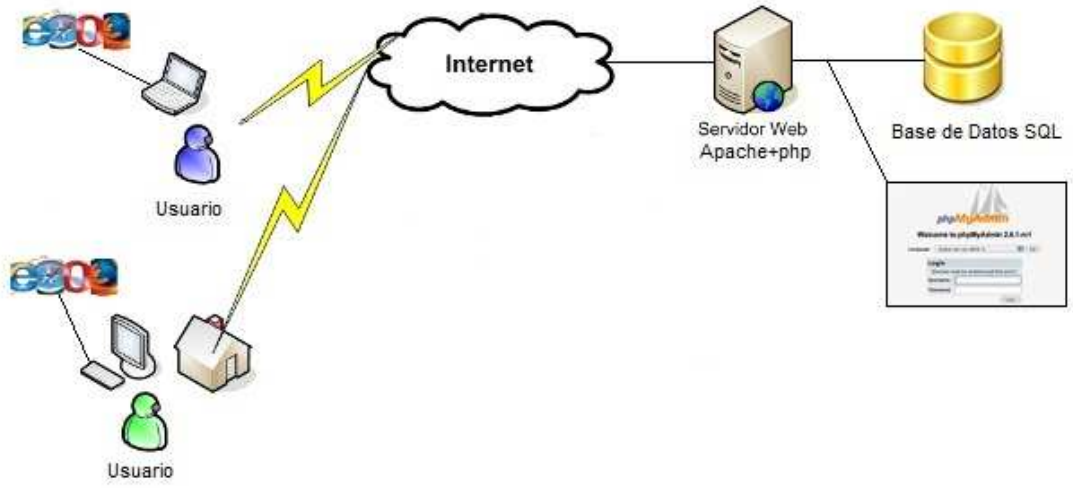


Figura 3.3. Arquitectura del Sistema.

Para el diseño de la página también se han usado hojas de estilo, CSS.

CSS

CSS [16] es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

La idea que se encuentra detrás del desarrollo de CSS es separar la *estructura* de un documento de su presentación.

También se ha utilizado javascript para validar los form de <html> y concretamente se ha usado una librería javascript, overLIB, para insertar pequeñas ventanas "popup" informativas.

Javascript

JavaScript [17] es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

3.2 Diseño de la aplicación

Las funciones principales de la aplicación son:

- Poder subir archivos (módulos) al server.
- Eliminar archivos (módulos) del server.
- Crear proyectos OpenRISC.

- Insertar componentes de la librería al sistema.
- Poder observar el mapa de memoria del Sistema a generar.
- Visualizar la librería de componentes disponibles.
- Generar un SoC basado en el procesador OpenRisc.

Diagrama de la aplicación web

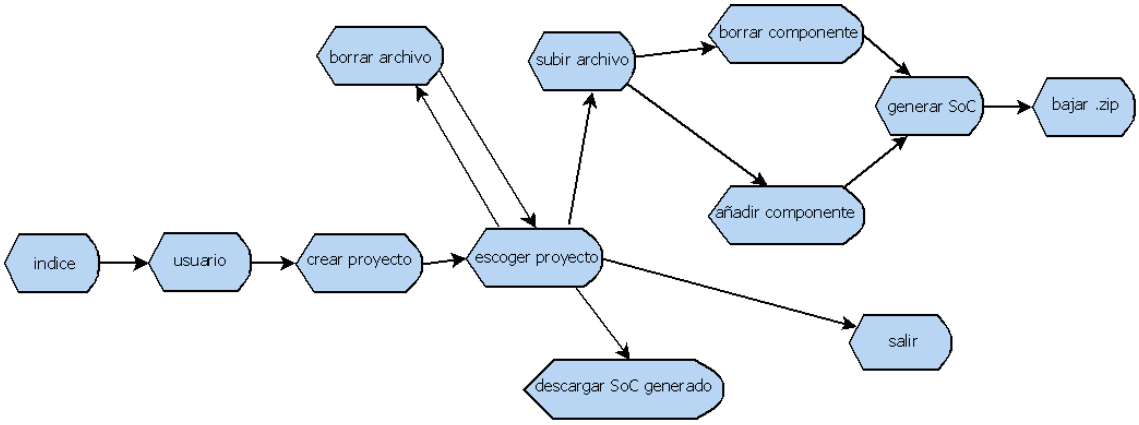
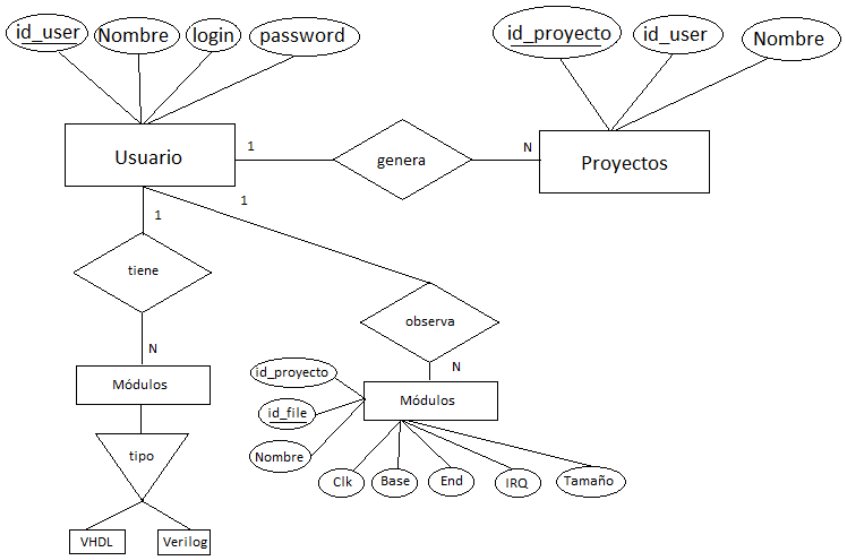


Figura 3.4. Estructura de la Aplicación.

Diseño de la Base de Datos

Mostraremos a continuación el diagrama de entidad-relación que modela la pequeña base de datos con la que trabajará nuestra aplicación. Para guardar la información de los módulos se usará una base de datos, no muy extensa, solamente el uso de 3 tablas.



.

Figura 3.5. Modelo Entidad-Relación.

Implementación de la base de datos

La base de datos contiene 3 tablas como podemos observar en la siguiente imagen:

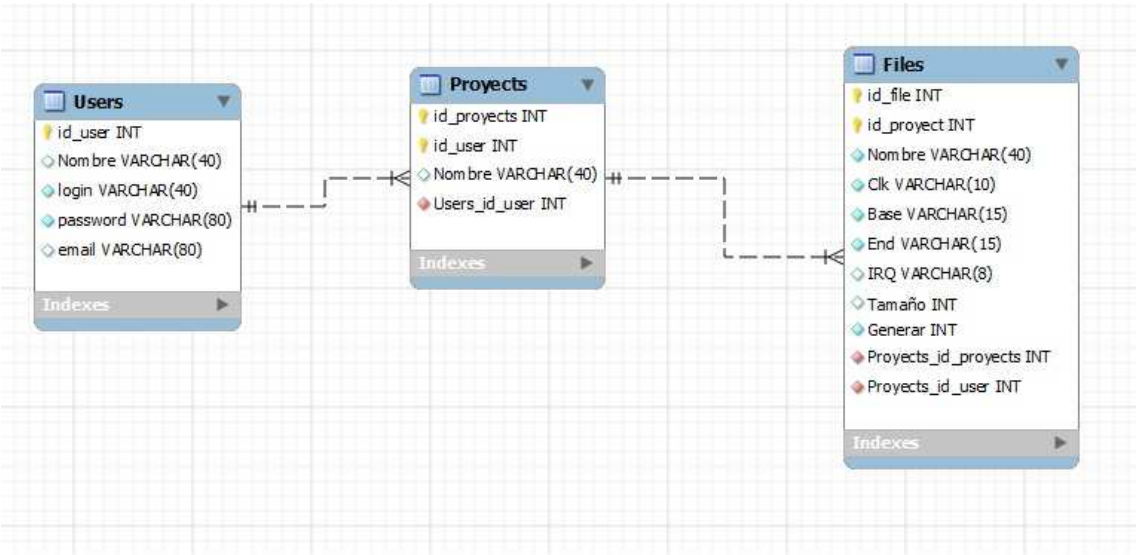


Figura 3.6. Esquema general de la base de datos.

Tablas de la Base de Datos

Tabla *Users*

Esta tabla es la responsable de controlar el acceso de los usuarios a la aplicación.

Campos:

- id_user: identificador del usuario.
- Nombre: Nombre del usuario.
- login: nombre para poder entrar a la aplicación.
- password: palabra de paso del usuario.
- e-mail: dirección de correo electrónico del usuario.

Tabla *Proyects*

Esta tabla es la responsable de almacenar los distintos proyectos del usuario.

Campos:

- id_proyect: identificador del proyecto.
- id_user: identificador del usuario.
- Nombre: Nombre del proyecto.

Tabla F*iles*

Esta tabla es la responsable de guardar los datos de los módulos que los usuarios irán insertando.

Campos:

- id_file: identificador del archivo.
- id_proyect: identificador del proyecto.
- Nombre: Nombre del archivo.
- Clk: ciclo de reloj del SoC.
- Base: dirección de inicio del módulo en el sistema.
- End: dirección de final del módulo en el sistema.
- IRQ: interrupción del módulo.
- Tamaño: tamaño en bytes del módulo.
- Generar: indica si el módulo ha sido insertado.

3.3 Diseño de los módulos

Diagrama de scripts

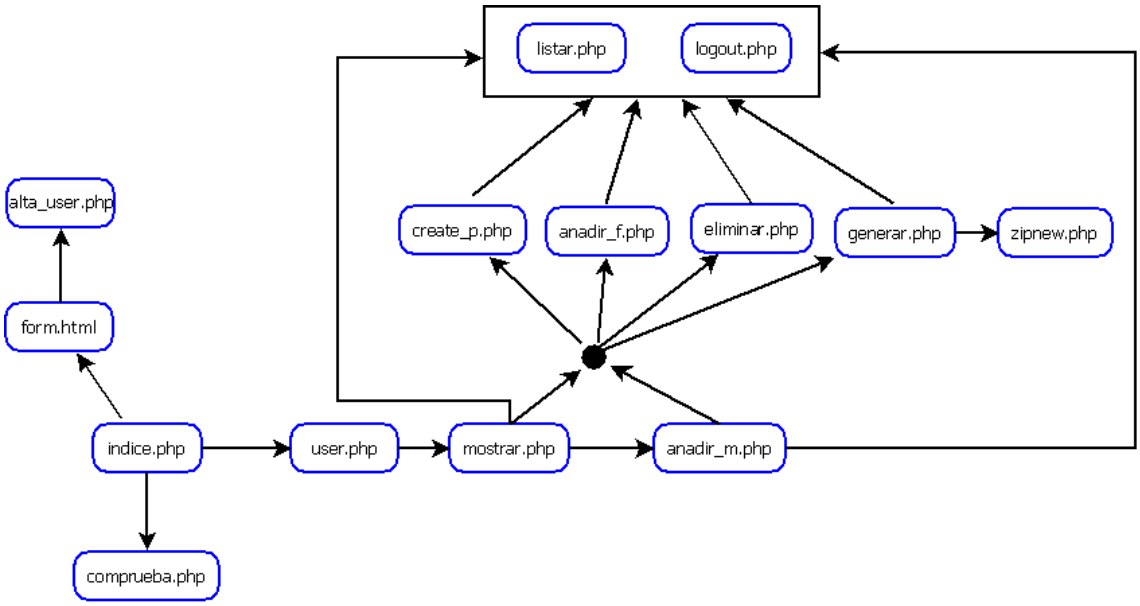


Figura 3.7. Diagrama de Scripts del portal web.

A continuación explicaremos cada script que componen esta aplicación. Casi todos los scripts contienen una función para conectarse a la base de datos para realizar las distintas consultas que se requieren en cierto momento.
También comentar que todos los scripts tienen un menú en la parte superior, que muestra unas opciones como son: escoger proyecto, listar proyectos y salir de la aplicación.

indice.php

Esta es la página inicial de la aplicación. A la izquierda de la página estará la parte de login para poder entrar en ella. Justo debajo hay un link a form.html para poder darse de alta.

comprueba.php

El encargado de validar los datos introducidos para acceder a la aplicación.

form.html

Este script es el encargado de obtener los datos de usuario para su registro mediante un formulario que una vez completado será validado por alta_user.php

alta_user.php

Es el encargado de validar los datos del usuario introducidos en el formulario, accediendo a la base de datos y mirando si el usuario existe o se puede crear.

user.php

La página que da la bienvenida al usuario dando la opción de insertar sus componentes al sistema.

mostrar.php

Este script, en la parte izquierda, mostrará la opción de escoger proyecto o de crear uno nuevo. Una vez escogido el proyecto a generar, se le abrirá la librería de componentes separados por módulos. Al clicar en el módulo, se le desplegará una lista de componentes disponibles para insertar. Previamente el usuario habrá tenido que subir el fichero al servidor. Si no ha subido nada, los módulos estarán vacios.

También en esta parte el usuario tiene las opciones de subir los archivos al server, de eliminarlos o de bajar el proyecto ya generado.

En la parte central de la aplicación se le mostrarán los componentes que tiene insertados, con la descripción del mapa de memoria de cada uno de ellos.

En la parte inferior derecha, el usuario tiene la opción de borrar de la lista de componentes el que él desee, donde rápidamente se actualizará la lista de componentes insertados. Justo a la izquierda de esta opción el usuario podrá generar el archivo principal, donde se insertarán todos los componentes de esta lista.

Una vez aceptada la opción de generar este script nos dirigirá a generar.php.

anadir_m.php

Al clicar como he mencionado antes, en el módulo, se abrirá la lista de componentes. Si ahora clicamos en el componente, éste es insertado en el sistema, dando a generar el script anadir_m.php. Este script es casi lo mismo que mostrar, llevando a cabo las mismas funcionalidades pero pasando las variables del componente por la url. Concretamente el nombre del fichero y el identificador de proyecto. Cada vez que se clique en un componente se llamará si mismo este script.

create_p.php

Aquí se crea un proyecto, dándole un nombre y creándose una carpeta en el servidor con este nombre.

anadir_f.php

Este script es el encargado de subir los ficheros al servidor en una carpeta existente o la que el desee crear. Hay una opción para insertar el tamaño del componente a subir. Normalmente se usa para los componentes tipo Memoria (flash, sram, onchip, etc.). Al mismo tiempo de subir el fichero, este es insertado en la base de datos.

eliminar.php

Encargado de eliminar los ficheros del servidor.

listar.php

Script que muestra todos los proyectos del usuario con los componentes que han sido insertados.

logout.php

Script encargado de cerrar la sesión de la aplicación.

generar.php

Este script es el más importante ya que es el encargado de insertar los módulos en el sistema.

zipnew.php

El encargado de generar el zip final del proyecto insertando las carpetas propias de OpenRISC. En este apartado se ha utilizado la librería pclzip-2-6, para poder crear el zip.

funciones.php

Script que contiene todas las funciones utilizadas en la aplicación. Mostramos por ejemplo las más importantes:

```
function conectar_bd(){

include("config.php");
```

```
$link=mysql_connect($server,$dbuser,$dbpass); //conectamos a BD
 if (!$link)
{
        die ('No se ha podido conectar con la Base de Datos<br>'. mysql_error());
}
$db_selected = mysql_select_db($database,$link);
if (!$db_selected)
{
        die ('No puedo usar la Base de Datos<br> ' .mysql_error() );
}

return $link;
}
```

Esta primera función nos conecta con la base de datos correspondiente.

```
function folders ($ruta,$num,$folder,$idproyect){

$dir = opendir($ruta) or die('Error');
?>
<div style="display:none" id="<?php echo $folder?>">

<?php
$j=0;
while ($elemento = @readdir($dir))
{
  if(is_dir($ruta) && $elemento != '.' && $elemento != '..') //es un directorio
 {
   if (($elemento != '.') && ($elemento != '..'))
   {

        if  (is_file($ruta.'/'.$elemento)  && ($elemento != '.') && ($elemento != '..')) //es un fichero
        {
                 if ($j < $num)
                {

                        $modulo = @explode(".",$elemento);
                        ?>
                        <letra2>
                <?php
                        echo '<ul><li><a
                        href=anadir_m.php?file='.$elemento.'&proj='.$idproyect.'
                        onfocus = "this.blur ()"; >'.$modulo[0].'</a><br></li></ul>';
                        $j++;
                        ?>
                        </letra2>
                <?php
                }
                if ($j == $num) //último archivo
                {?>
                        </div>
                <?php
                }
        }
   }
  }
}
```

```
closedir($dir);
}
```

Esta segunda función, nos imprime por pantalla los componentes disponibles en la librería.

```
function BuscaFiles($cpt, $registrod)
{
$file='nofile';
$dir = opendir($cpt) or die('Error');
$e=0;
while ($elemento = @readdir($dir) )

{
        if ($e==0)
        {
                if (($elemento != '.') && ($elemento != '..'))
                {
                        $ele=explode(".",$elemento);
                        if ($ele[0] == $registrod)
                        {
                                $file=$elemento;
                                $e=1;
                        }
                                else
                        {
                                 $file='nofile';
                        }
                }
        }
}
return $file;
}
```

Y esta tercera función busca los ficheros en el servidor, si lo encuentra devuelve su nombre.

Proceso de generación del SoC

Seguidamente explicaremos el proceso para generar el SoC. El softcore OpenRisc contiene dos archivos importantes donde se encuentra la información sobre los componentes. Los archivos son *cyc_or12_defines.v* y cyc_*or12_mini_top.v*.

cyc_or12_defines.v

Contiene las direcciones de memoria de los componentes insertados en el SoC.

En el siguiente código podemos observar su contenido:

```
`define CYCLONE
```

Primeramente tenemos las interrupciones del sistema:

```
//
// Interrupts
//
`define APP_INT_RES1       1:0
`define APP_INT_UART       2
`define APP_INT_RES2       3
`define APP_INT_ETH        4
`define APP_INT_PS2        5
`define APP_INT_RES3       19:6
```

Seguidamente tenemos el mapa de memoria, que nos muestra qué elementos tenemos insertados en el sistema:

```
//
// Address map
//
`define APP_ADDR_DEC_W          8
`define APP_ADDR_ONCHIP  `APP_ADDR_DEC_W'h00
`define APP_ADDR_UART  `APP_ADDR_DEC_W'h04
`define APP_ADDR_DECP_W  4
`define APP_ADDR_PIO  `APP_ADDR_DEC_W'h9
// `APP_ADDR_DEC_W'h97
// `APP_ADDR_DEC_W'h92
// `APP_ADDR_DEC_W'h9d
// `APP_ADDR_DEC_W'h90
// `APP_ADDR_DEC_W'h94
// `APP_ADDR_DEC_W'h9e
// `APP_ADDR_DEC_W'h9f
`define APP_ADDR_FAKEMC        4'h6
```

Aquí podemos observar que en el sistema tenemos insertados 3 componentes: un módulo onchip_ram, una UART, y un módulo pio led.

Para generar este fichero cuando el usuario inserta un componente y genera el proyecto, tenemos que ir leyendo el fichero, y averiguar qué dirección está libre para poder añadírsela. Como también buscar si el componente ya esté insertado y no ofrecerle otra dirección distinta. Por tanto la dirección que le indicaremos será la dirección base, la dirección de inicio del componente que después nos servirá para mostrar en el mapa de memoria.

Justo con esta dirección de inicio y el tamaño del componente se calculará la dirección end, la dirección de final.

cyc_or12_mini_top.v

Este archivo es el que contiene toda la información del sistema. Aquí estarán todas las señales de los componentes insertados y dónde también se juntarán con las líneas del bus Wishbone.

Para la creación de este archivo:

1- Se obtiene una búsqueda en la base de datos de los componentes que se quieren insertar en el SoC. Esto nos lo mostrará el campo Generar cuando esté activado a 1.
2- Se busca en el server que los archivos existan físicamente.
3- Si el archivo se encuentra, diferenciamos si el componente está escrito en verilog o en vhdl, dado que no es la misma sintaxis.
4- Se busca la ruta del archivo a insertar y se abre en modo lectura.
5- Se leen todas las variables necesarias para la inclusión del componente en el Soc.
6- Una vez leídas las variables se abre el fichero cyc_or12_mini_top.v en modo lectura. Gracias a unos tags en este fichero sabremos donde situar cada variable donde le corresponda.
7- Al mismo tiempo se crea un cyc_or12_mini_top_temp.v para ir añadiendo las variables que hemos guardado anteriormente.
8- Una vez acabado de leer el cyc_or12_mini_top.v principal, el fichero temporal se reescribe por éste para así seguir con el siguiente componente que se quiera insertar, haciendo que se vaya actualizando el fichero principal.
9- Una vez acabado con la inserción de todos los componentes, se llevará a cabo la creación del zip, que incluirá todos los archivos propios del Soc OpenRISC. para poder comprobar después, por ejemplo con el Quartus [6] que todo se ha realizado correctamente.

[6] **Quartus II** es una herramienta de software producida por Altera para el análisis y la síntesis de diseños realizados en HDL.

En esta imagen podemos observar como es un fichero en verilog, concretamente este es el fichero del componente UART:

uart_top.v

```
144
145  `include "uart_defines.v"
146
147  module uart_top (
148      wb_clk_i,
149
150      // Wishbone signals
151      wb_rst_i, wb_adr_i, wb_dat_i, wb_dat_o, wb_we_i, wb_stb_i, wb_cyc_i, wb_ack_o, wb_sel_i,
152      int_o, // interrupt request
153
154      // UART signals
155      // serial input/output
156      stx_pad_o, srx_pad_i,
157
158      // modem signals
159      rts_pad_o, cts_pad_i, dtr_pad_o, dsr_pad_i, ri_pad_i, dcd_pad_i
160  `ifdef UART_HAS_BAUDRATE_OUTPUT
161      , baud_o
162  `endif
163      );
164
165  parameter                           uart_data_width = `UART_DATA_WIDTH; //32
166  parameter                           uart_addr_width = `UART_ADDR_WIDTH; //32
167
168  // INTERRUPT
169  output                              int_o;
170
171  // WISHBONE interface
172  input                               wb_rst_i;
173  input                               wb_clk_i;
174  input [31:0]          wb_adr_i;
175  input [31:0]          wb_dat_i;
176  output [31:0]         wb_dat_o;
177  input                               wb_we_i;
178  input                               wb_stb_i;
179  input                               wb_cyc_i;
180  input [3:0]                         wb_sel_i;
181  output                              wb_ack_o;
182
183  // SIGNALS UART
184  input                               srx_pad_i;
185  output                              stx_pad_o;
186  output                              rts_pad_o;
187  input                               cts_pad_i;
188  output                              dtr_pad_o;
189  input                               dsr_pad_i;
190  input                               ri_pad_i;
191  input                               dcd_pad_i;
```

Figura 3.8. Estructura fichero verilog.

Aquí podemos observar primeramente que el módulo uart_top contiene variables de la propia uart y después le siguen las variables que permitirán conectar este componente con el bus Wishbone.

Estructura ficheros usuario

Los ficheros que el usuario suba al servidor tendrán que seguir una serie de reglas:

En esta figura podemos observar un fichero .vhd. Para poder detectar el bus wishbone se insertará en el código un tag - - *WISHBONE interface:* y seguidamente las variables del bus Wishbone. Si no es así el sistema lanzará un error.

Lo mismo se tendrá que hacer con otras variables que pueda haber, con el tag - - *SIGNALS:*

```
--Comment: Módulo LED.

library ieee;
use ieee.std_logic_1164.all;

entity pio_led is
port(
      -- WISHBONE interface:
      WB_ACK_O:         out   std_logic;
      WB_CLK_I:         in    std_logic;
      WB_DAT_I:         in    std_logic_vector( 31 downto 0 );
      WB_DAT_O:         out   std_logic_vector( 31 downto 0 );
      WB_ADR_I:         in  std_logic_vector( 31 downto 0 );
      WB_RST_I:         in    std_logic;
      WB_STB_I:         in    std_logic;
      WB_WE_I:          in    std_logic;
      -- SIGNALS:
      PRT_O:          out   std_logic_vector( 7 downto 0 )

   );
```

Figura 3.9. Fichero .vhd.

3.4 Entorno Gráfico

Seguidamente mostramos el entorno gráfico desarrollado:

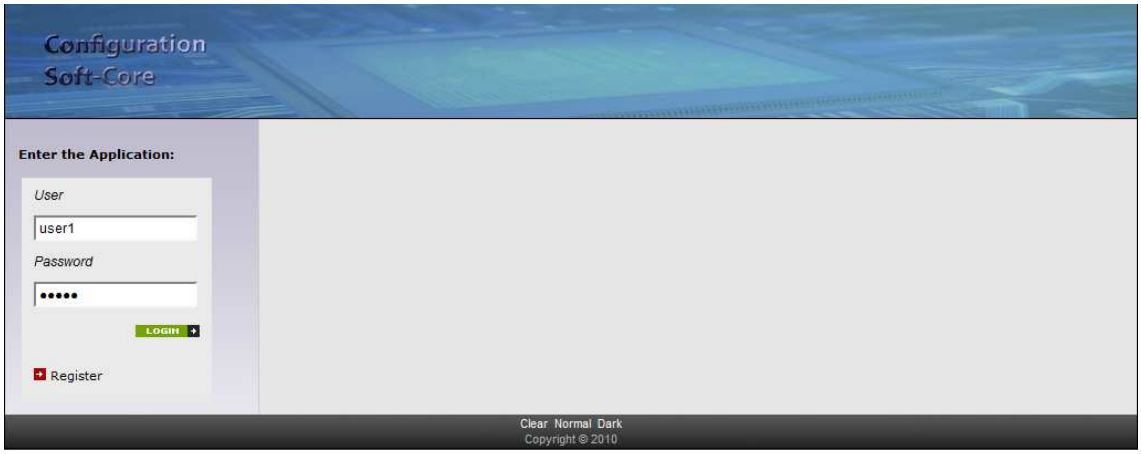


Figura 3.10. Entorno Gráfico (Portada).

Esta es la página principal del entorno gráfico y la encargada de validar a los usuarios. Esta validación se hará mediante correspondencia en la tabla '*users'* de la base de datos. Para ello tendrán que introducir su nombre de usuario y su contraseña.

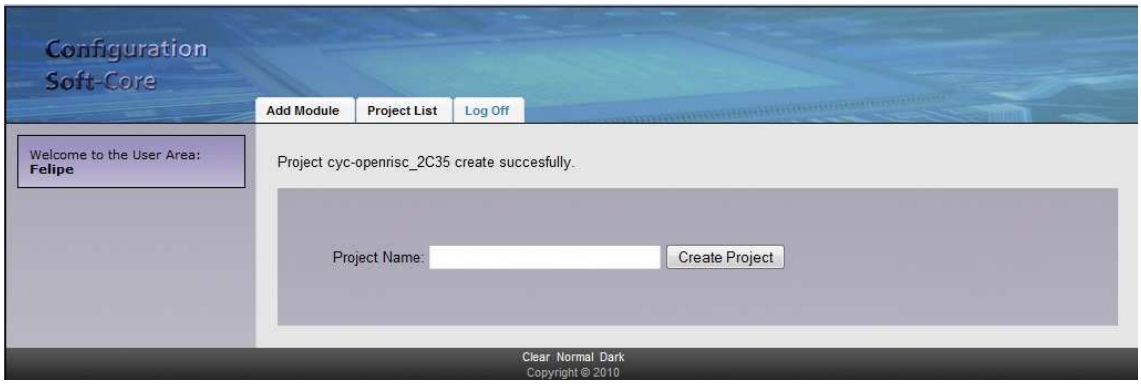


Figura 3.11. Entorno Gráfico (Creación proyecto).

Una vez dentro de la aplicación, el usuario tendrá la opción de crear un proyecto. Escribiendo el nombre y dando a crear proyecto, se creará un proyecto en el servidor.

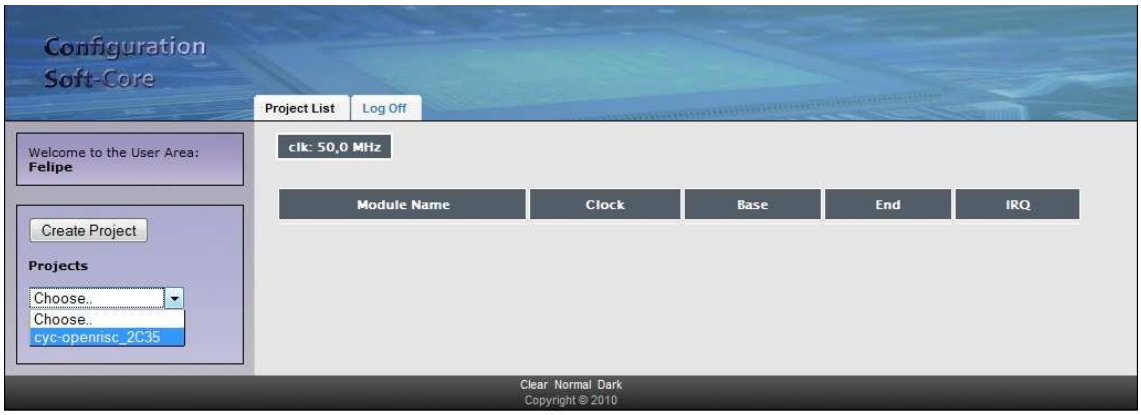


Figura 3.12. Entorno Gráfico (Selección proyecto).

Una vez creado el proyecto, lo seleccionaremos de la lista de proyectos que hay en el margen izquiero de la Figura 3.12. De esta forma podremos empezar a generar un SoC.

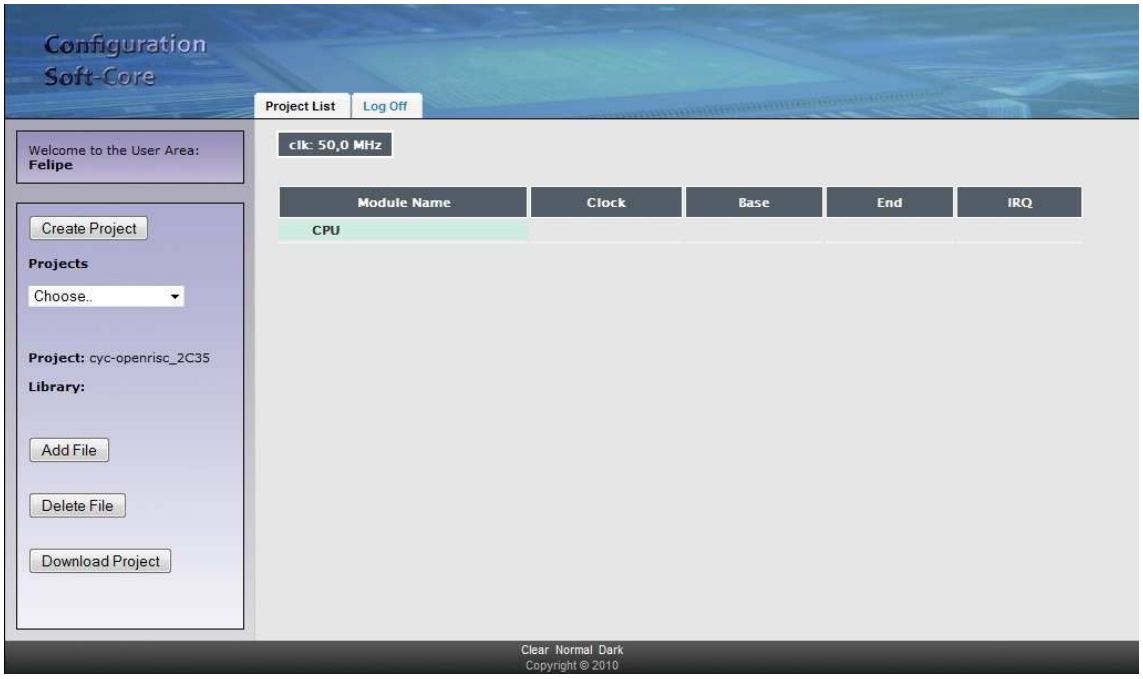


Figura 3.13. Entorno Gráfico (Librería Componentes).

Con el proyecto escogido, podemos ver que el proyecto está vacío. En la parte izquierda vemos que la librería de componentes está vacía y en la parte derecha, vemos que no existe ningún componente insertado. En la parte de abajo del entorno, el usuario podrá subir sus archivos, borrarlos o descargar el proyecto si ya ha sido generado.

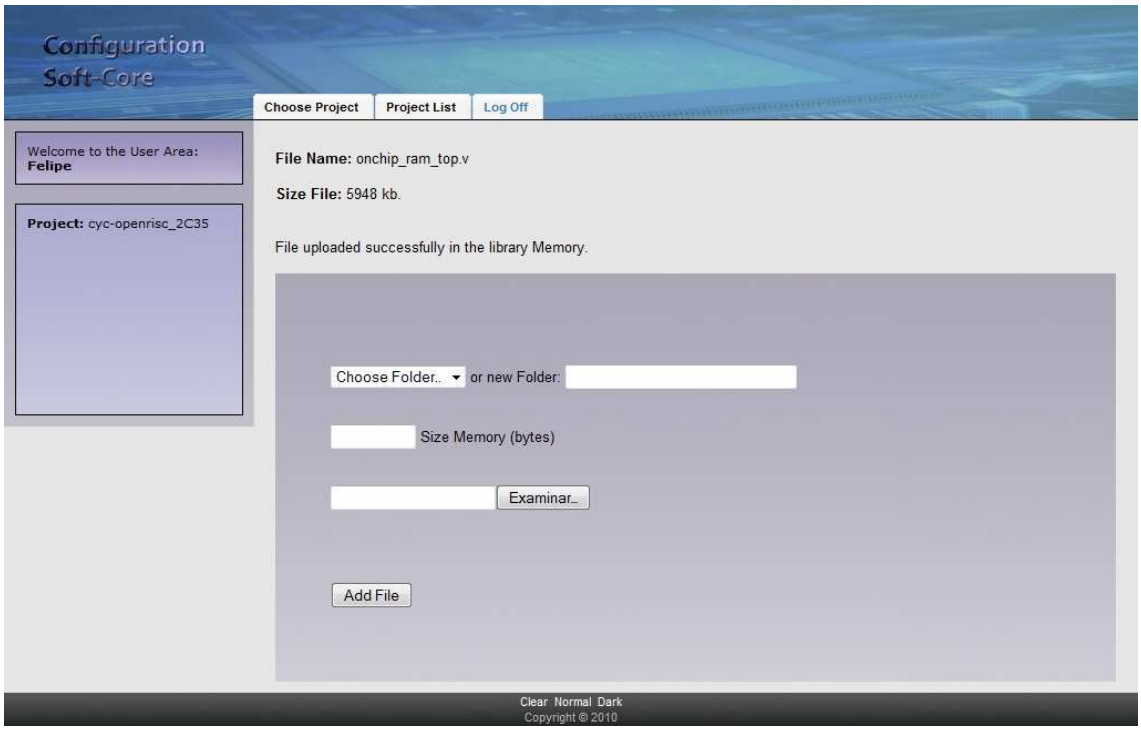


Figura 3.14. Entorno Gráfico (Subir archivos).

Una vez creado el proyecto, el usuario podrá subir sus ficheros (módulos RAM, UART, etc) al servidor. Podrá crear subcarpetas dentro del proyecto para poder tener separados sus módulos. La subida de archivos podrá ser múltiple. En caso que el fichero top de nuestro dispositivo no se adapte formato de bus Wishbone, se mostrará un mensaje de error informando que el modulo no es compatible con este bus.

Con respecto a las memorias tendrá una opción para poder insertar su tamaño.

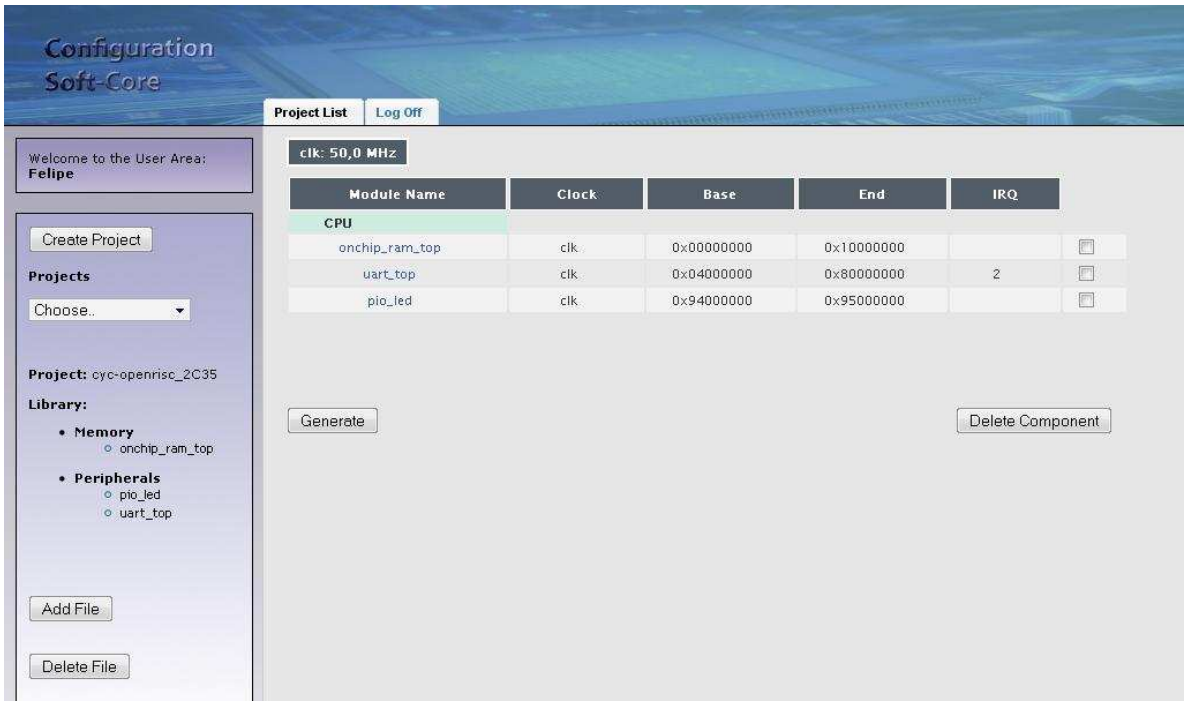


Figura 3.15. Entorno Gráfico (Componentes insertados).

Una vez subidos los archivos al servidor el usuario podrá observar que en la librería de componentes existen sus archivos. Vemos en la imagen que existen dos subcarpetas, una llamada *Memory* y otra llamada *Peripherals*, que permitirán separar sus componentes por tipo. Dándole un clic en cada una de estas subcarpetas se le mostrarán los componentes que tiene a su disposición.

Luego al clicar sobre el componente, éste será añadido al sistema, que se puede observar en la parte derecha del entorno. Aquí podemos ver el nombre del módulo insertado, su frecuencia, la dirección de memoria de inicio y la dirección de memoria de final y sus interrupciones.

Situando el ratón en el nombre del módulo, se abrirá un pequeño pop-up mostrando información de éste. Si el usuario quiere quitar algún módulo, deberá seleccionar el checkbox y dar al botón de Delete Component. Automáticamente se borrará el módulo añadido, pero no así el fichero en particular que seguirá en el servidor.

Después de tener los módulos añadidos que el usuario ha querido, clicará en el botón generar para generar su SoC.

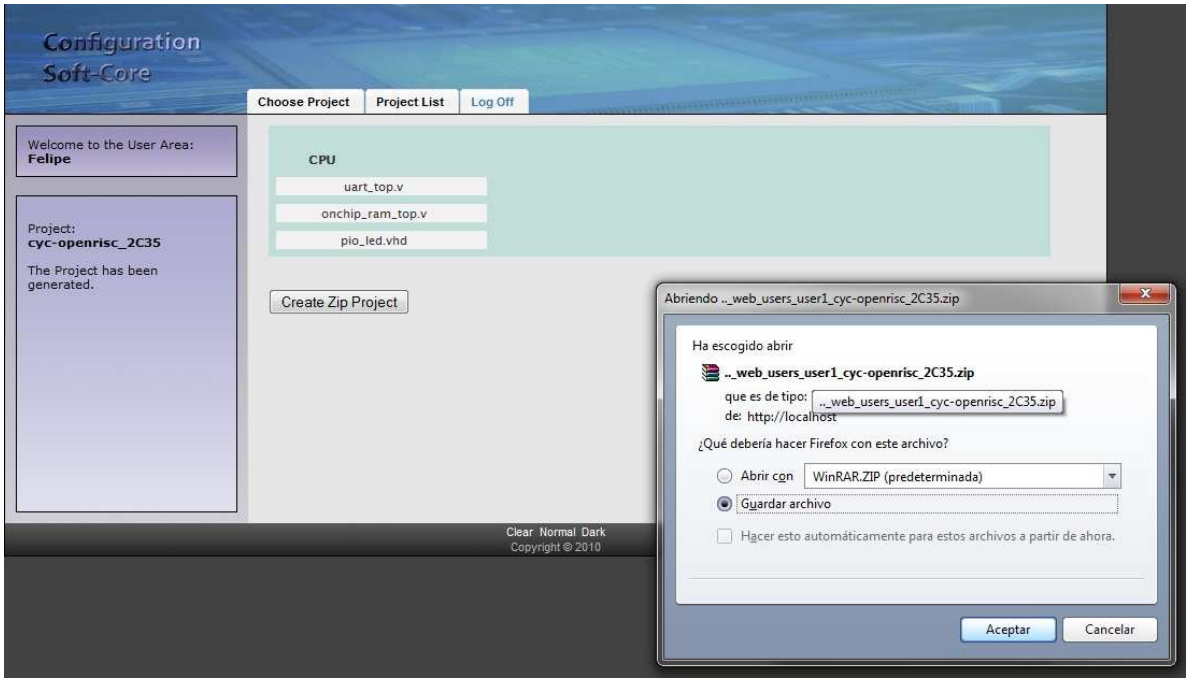


Figura 3.16. Entorno Gráfico (SoC generado).

Finalmente podemos observar que los tres módulos insertados por el usuario han sido conectados satisfactoriamente.

Entonces para poder obtener su SoC, la aplicación le ofrecerá un descarga de su proyecto en formato .zip.

Capítulo 4

Test y Resultados

En este capítulo comentaremos los test que hemos realizado para comprobar el funcionamiento de la aplicación.

4.1 Entorno de verificación

El entorno hardware escogido para realizar los test hardware de nuestra aplicación ha estado formado por la placa de desarrollo DE2 de Altera.

Para el entorno Software el programa utilizado ha sido el proporcionado también por Altera, el QUARTUS II.

Entorno de desarrollo Hardware

Descripción de la placa DE2 ALTERA

La placa de desarrollo académico DE 2 de ALTERA es una herramienta que tiene características que permiten el diseño de proyectos y el desarrollo de sofisticados sistemas digitales.

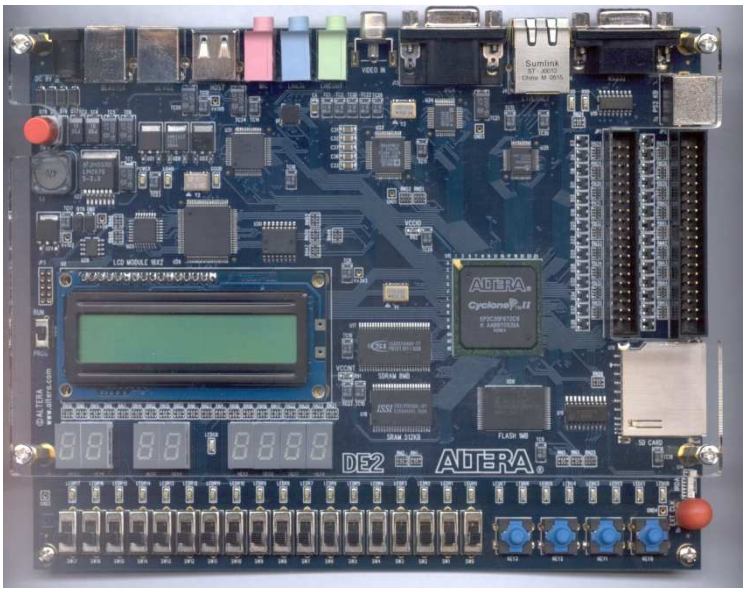


Figura 4.1. Placa DE2-ALTERA.

Los componentes que integran esta palca son [8]:

Característica	Descripción
FPGA	1 FPGA Cylone II EP2C35F672C6 y un dispositivo de configuración en serie EPCS16.
Dispositivos E/S	Configuración incorporada USB Blaster FPGA. 10/100 Ethernet RS-232, Puerto Infrarojo. Salida de Video (VGA 10 bit DAC). Entrada de Video (NTSC/PAL/Multi-formato). USB 2.0 (tipo A y tipo B). Puerto para ratón o teclado PS/2. Line in, Line out, entrada micrófono. CODEC audio 24 bits. Headers de expansión (76 terminales).
Displays	16 x 2 Display LCD 8 Displays de 7 segmentos
Switch y LEDs	18 interruptores basculantes. 4 interruptores pulsadores. 18 LED rojos, 9 LED verdes.
Clocks	Osciladores de 27MHz y 50 MHz, entrada de reloj externa SMA.
Memorias	SDRAM 8 MB, SRAM 512KB, Flash 4 MB

La placa de desarrollo académico DE2 contiene un dispositivo FPGA Cylone II EP2C35F672C6 con 672 terminales, las cuales se encuentran conectadas a los diferentes componentes integrados en la placa para permitir al usuario controlar todos los aspectos de operación de ésta.

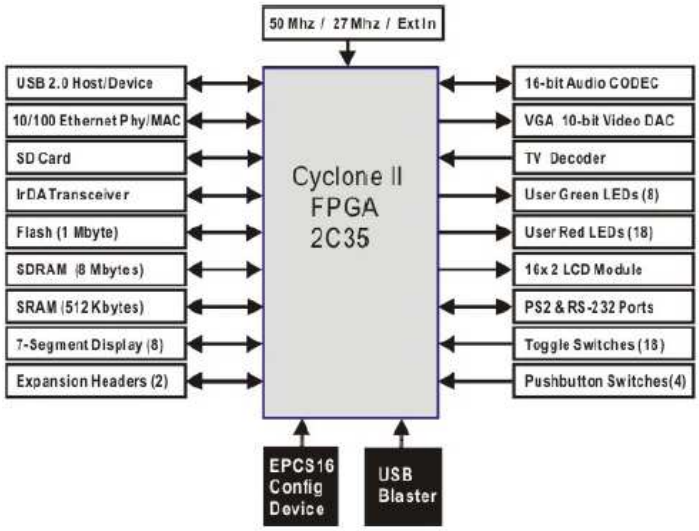


Figura 4.2. Diagrama Bloque de la placa DE2.

Single-Chip USB to UART Bridge

Para poder conectar el puerto serie que controla la UART de la placa con el PC y poder observar las diferentes pruebas realizadas, hemos utilizado el bridge USB a UART, que simplifica el upgrade de RS-232C / RS-485 a USB brindando una solución completa con drivers disponibles que evitan la necesidad de software adicional.

Figure 4.3. Bridge USB-a-UART.

Servidor Web

El server utilizado ha sido un Sony Vaio, con CPU Intel Core 2 Duo Processor T5500, 1.66GHz, memoria 1GB/Go SDRAM (533MHz) con S.O Windows Xp.

Entorno de desarrollo Software

Para poder probar la placa anteriormente mencionada se necesitará el uso de un programa para sintetizar nuestro diseño.
Como ya comentamos en el apartado 2.2.2 también necesitaremos el compilador de archivos compatibles con el procesador or1200, para poder compilar nuestro software.

QUARTUS II

Quartus II es una herramienta de software, gratuita en su versión Web Edition, desarrollada por Altera para el análisis y la síntesis de diseños realizados en HDL.

Quartus II permite al desarrollador o desarrolladora compilar sus diseños, realizar análisis temporales, examinar diagramas RTL y configurar el dispositivo de destino con el programador.

El Quartus II incluye todas las herramientas necesarias para realizar diseños completos para todas las familias de FPGA y CPLD de Altera. Las funcionalidades de las que dispone son:

- Edición de esquemáticos y diseños basado en HDL.
- Trae integrado el compilador de VHDL y Verilog y también soporta incorporar de nuevos.
- Integra el SOPC Builder, un software para generar sistemas SoCs.
- Place-and-route, verificación y síntesis.
- Optimizadores temporales y de recursos.

Una de las comunicaciones principales del Quartus II con el hardware es:

- USB-Blaster™.

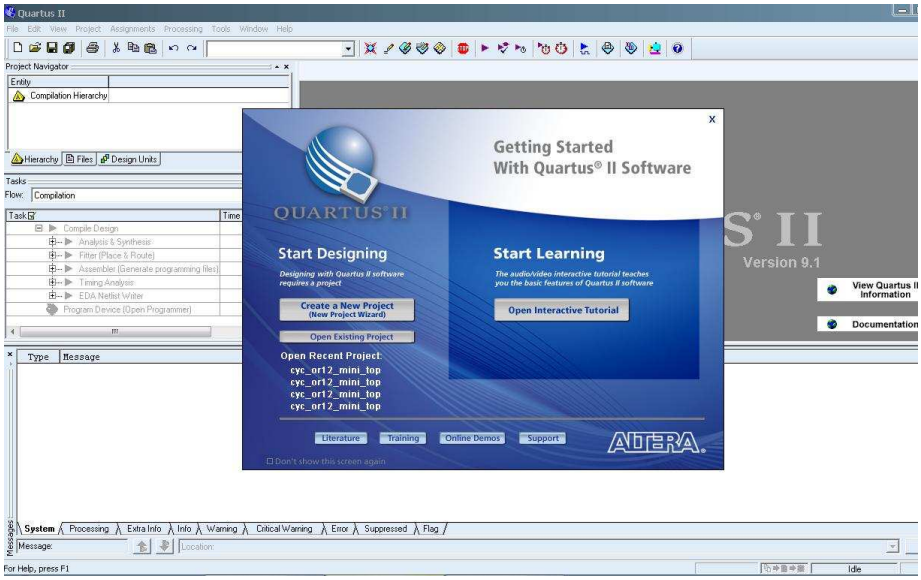


Figura 4.4. Programa QUARTUS II.

ToolChain para OpenRISC

Para el uso del compilador, al ser utilizado bajo Linux, se ha utilizado *VMware Player* con la imagen de Ubuntu 9.10 para poder instalarlo y para su uso posterior.

VMware

VMware es un sistema de virtualización por software. Un sistema virtual por software es un programa que simula un sistema físico (un ordenador, un hardware) con unas características de hardware determinadas. Cuando se ejecuta el programa (simulador), proporciona un *ambiente de ejecución* similar a todos los efectos a un ordenador físico (excepto en el *puro acceso físico* al hardware simulado), con CPU (puede ser más de una), BIOS, tarjeta gráfica, memoria RAM, tarjeta de red, sistema de sonido, conexión USB, disco duro (pueden ser más de uno), etc.

4.2 Test y validación del sistema

Para probar que nuestra aplicación ha creado satisfactoriamente un SoC basado en el procesador OpenRISC, se ha generado uno con tres componentes añadidos: OnChip-RAM, que es donde lanzaremos nuestro programa, un Pio Led para poder interactuar con los leds y una UART para poder visualizar el mensaje y recibir comandos.

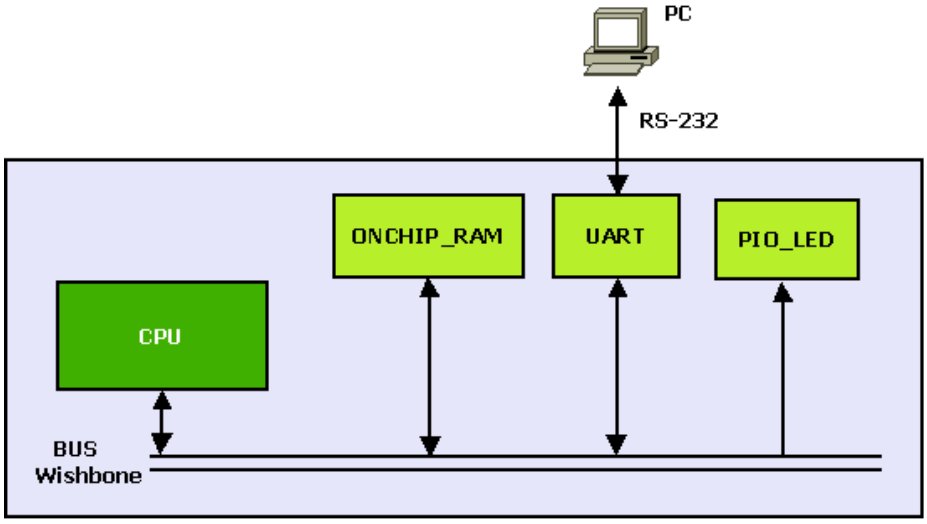


Figura 4.5. SoC generado.

Después se ha creado un código en .c, llamado test_uart.c, que propone enviar unas palabras por la UART de la placa y recibir comandos para encender y apagar el led:

```
#include "board.h"
#include "uart.h"

#define BOTH_EMPTY (UART_LSR_TEMT | UART_LSR_THRE)
#define WAIT_FOR_XMITR \

        do { \
              lsr = REG8(UART_BASE + UART_LSR);
           } while ((lsr & BOTH_EMPTY) != BOTH_EMPTY)

#define WAIT_FOR_THRE \

        do { \
             lsr = REG8(UART_BASE + UART_LSR); \
           } while ((lsr & UART_LSR_THRE) != UART_LSR_THRE)

#define CHECK_FOR_CHAR (REG8(UART_BASE + UART_LSR) & UART_LSR_DR)
#define WAIT_FOR_CHAR \

         do { \
                lsr = REG8(UART_BASE + UART_LSR); \
            } while ((lsr & UART_LSR_DR) != UART_LSR_DR)
```

Primeramente se incluye la librería board.h donde se encuentran las direcciones de los módulos que hemos insertado en el SoC. Luego vemos que incluimos la librería de la uart, donde contiene toda la información de ésta, ya que es lo que vamos a probar.

Y en el main() creamos el código que queremos probar:

```
int main (void)
{
  uart_init ();
  uart_print_str("\n\tTest Soft-Core");
  uart_print_str("\n\tUART streem b");
  uart_print_str("inary       \n\t");
  uart_print_str("by Felipe Pinto\n");

  op=uart_getc();
  op=uart_getc();

  if (op == ‘c’)
          REG32(PIO_BASE)=0x0001;
  else   REG32(PIO_BASE)=0x0000;

  return 0;
}
```

En la figura 4.6 podemos observar el diagrama de flujo de nuestro sistema:

1- Añadimos componentes con el entorno gráfico.

2- Generamos el SoC.

3- Creamos archivo de test *.c, y junto con el archivo board.h generado por el entorno, compilamos los archivos.

4- Al compilar estos archivos se crea un archivo *.intel.hex que será insertado luego en el módulo onchip.

5- Creamos un proyecto con un programa de síntesis (ej: QUARTUS II) junto con el SoC generado.

6- Compilamos nuestro proyecto con Quartus II.

7- Descargamos proyecto en placa.

8- Observamos los resultados.

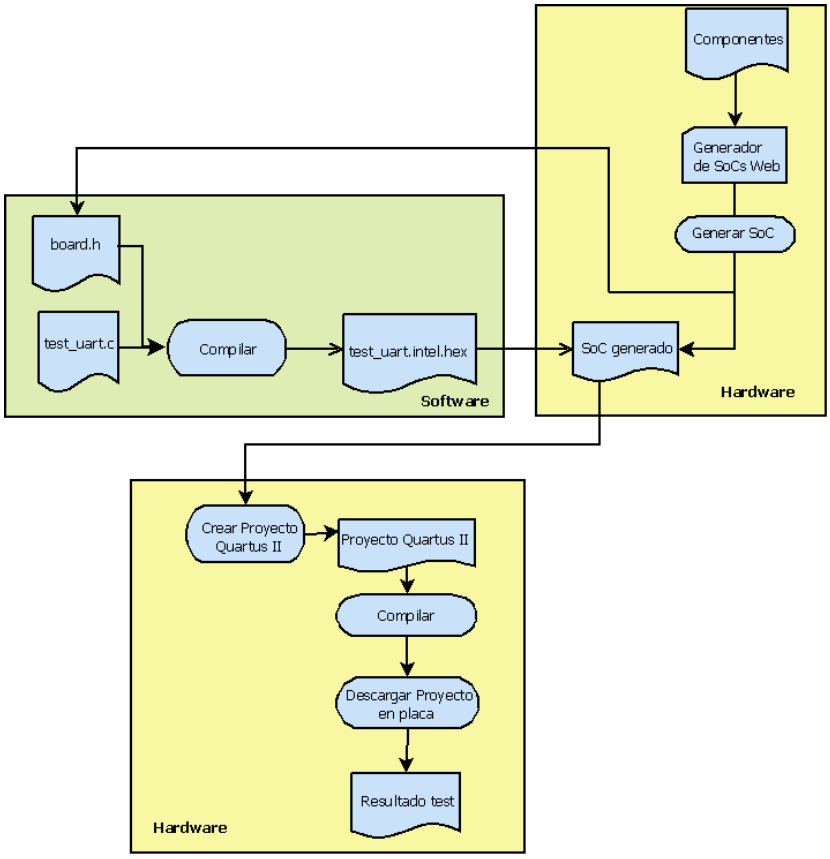


Figura 4.6. Diagrama de Flujo del Sistema.

Como he comentado, este archivo se tiene que compilar con el compilador obtenido de la toolchain de GNU.

Al compilar el código correspondiente se crea un archivo llamado test.intel.hex (en la siguiente página explicamos que significa esta extensión) que nos permitirá grabar lo que hemos generado en la memoria OnChip-RAM. La dirección física de este archivo se tendrá que indicar en el componente Onchip-RAM mediante la línea:

altsyncram_component.init_file = "D:/UAB/PFC/ultimo/cyc-openrisc_2C35/sw/test.intel.hex",

Seguido estos pasos, lo siguiente es crear un proyecto, por ejemplo con el Quartus II y compilar el proyecto entero. Una vez compilado y con el fin de no tener ningún error, se descarga el proyecto a la placa para poder comprobar que funciona correctamente.

En las siguientes imágenes podemos observar que funciona satisfactoriamente:

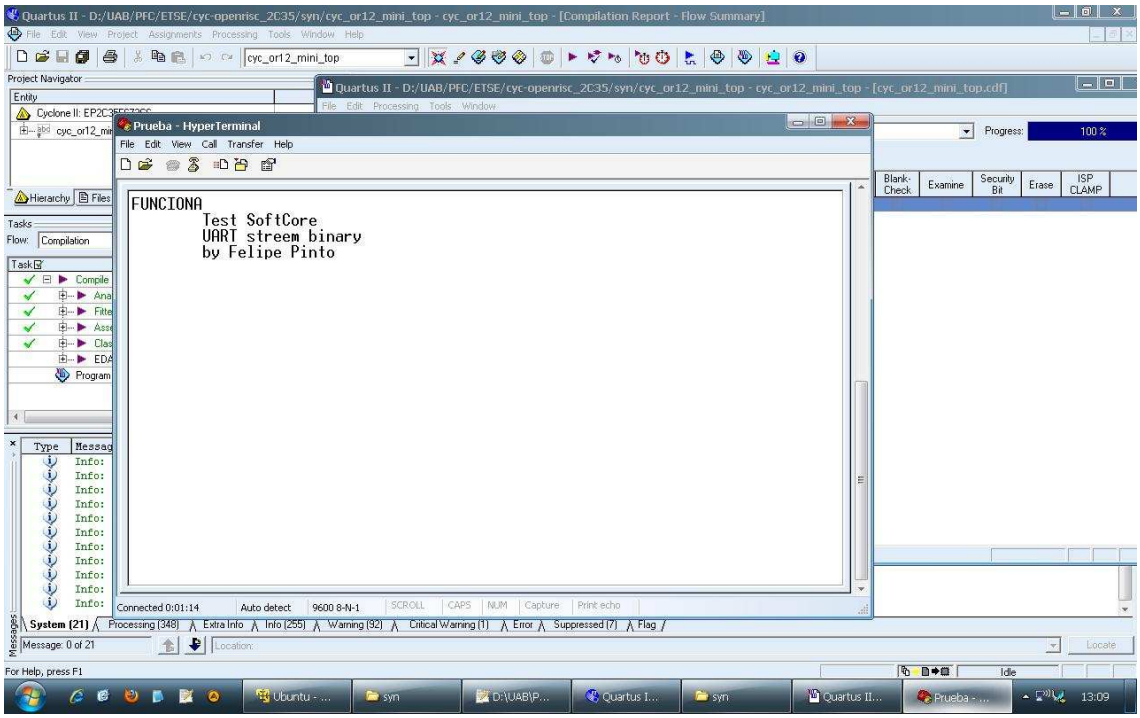


Figura 4.7. Prueba test_uart.c.

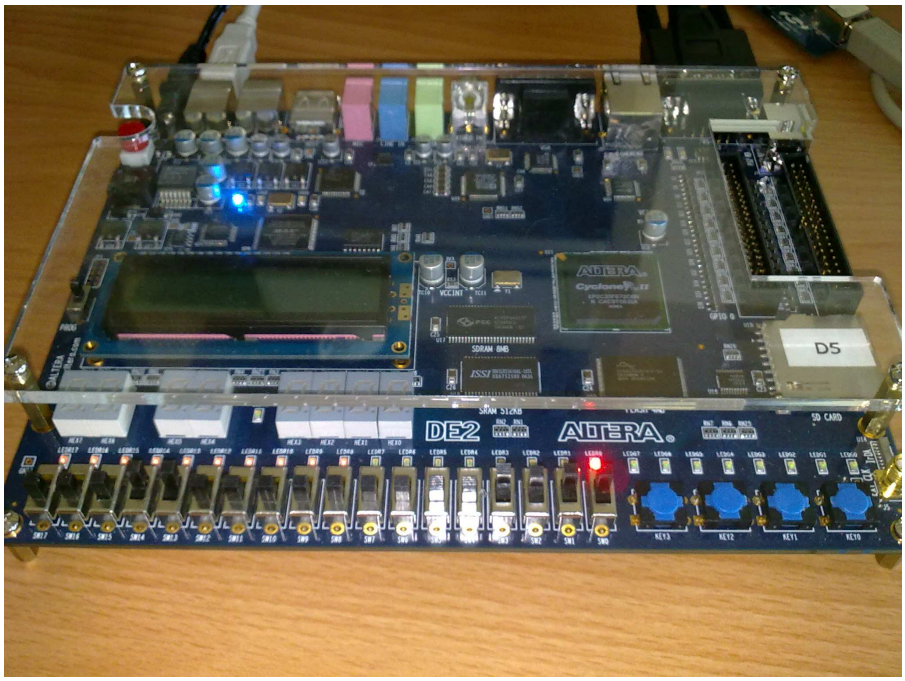


Figura 4.8. Prueba Pio Led.

Formato Intel HEX

Intel HEX es un formato de archivo para la programación de microcontroladores, EPROMs y otros circuitos integrados. Datando de los años 70, está entre los formatos más viejos con esta finalidad.

Consiste en un archivo de texto cuyas líneas contienen valores hexadecimales que codifican los datos, y su offset o dirección de memoria.

Ejemplo:

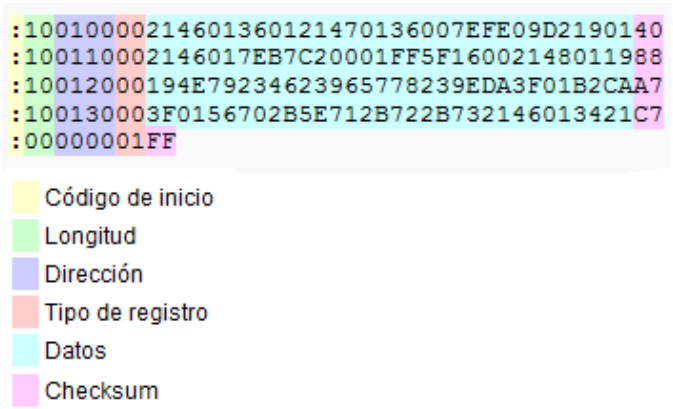


Figura 4.9. Características Intel HEX.

4.3 Análisis de recursos de la aplicación

En este apartado analizaremos los recursos de nuestra aplicación i del código generado:

En la siguiente figura podemos observar el total de LEs que tiene nuestro SoC,

- De los 33.216 logic elements disponibles utiliza unos 8.725, lo que es un 26% de total.
- De los 33.216 registros lógicos disponibles usa unos 4.619, lo que es un 14% del total.

Cabe mencionar que nuestra aplicación no alterará de forma elevada el número de LEs, ya que ésta solamente tiene la función de unir componentes en el sistema.

Flow Status	Successful - Thu Sep 02 21:38:02 2010
Quartus II Version	9.1 Build 222 10/21/2009 SJ Web Edition
Revision Name	cyc_or12_mini_top
Top-level Entity Name	cyc_or12_mini_top
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	No
Total logic elements	8,725 / 33,216 (26 %)
Total combinational functions	6,667 / 33,216 (20 %)
Dedicated logic registers	4,619 / 33,216 (14 %)
Total registers	4619
Total pins	59 / 475 (12 %)
Total virtual pins	0
Total memory bits	215,680 / 483,840 (45 %)
Embedded Multiplier 9-bit elements	6 / 70 (9 %)
Total PLLs	1 / 4 (25 %)

Figura 4.10. Resumen de Síntesis y Análisis de nuestro SoC.

El uso de nuestra aplicación en cuánto a uso de memoria, ocupa unos 4.972KB. El servidor ocupa 18.292KB y MySQL ocupa 14.9126KB.
Antes de generar el SoC el uso de CPU es de 1% y en el momento de generarlo el uso de CPU es de 44%. El PF (*page file*) antes de generar es de 450MB y después aumenta a 542 MB.

El uso de CPU durante la generación del SoC lo podemos ver en la siguiente imagen:

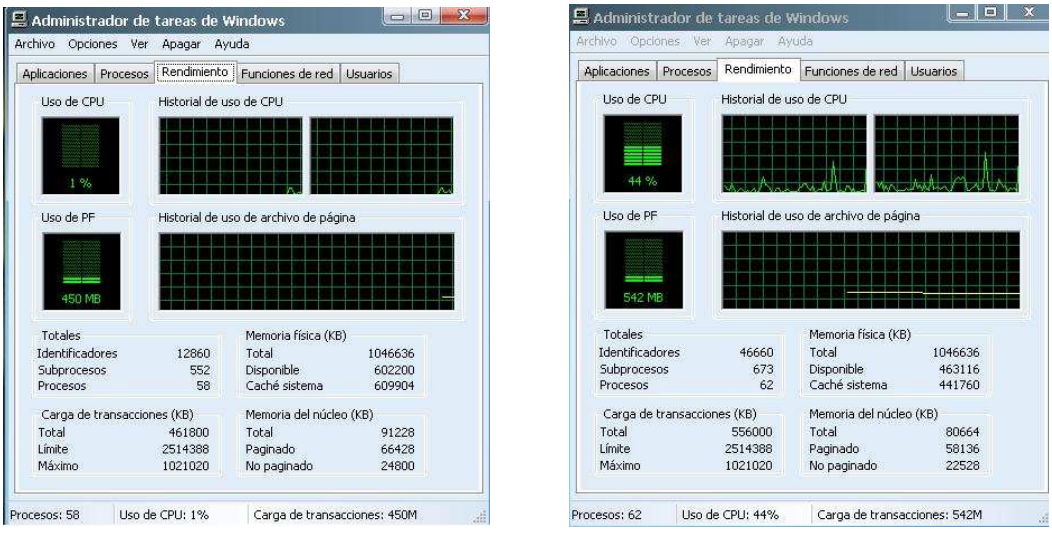


Figura 4.11. Uso CPU del sistema.

Capítulo 5

Conclusiones

Llegados a este punto, con la aplicación ya implementada, es el momento de hacer un análisis del trabajo realizado y como se podría mejorar para un futuro.

5.1 Conclusiones

Cuando un trabajo llega a su finalización, el principal punto que hay que observar es si éste ha llegado a los objetivos marcados desde un principio.

Podemos decir que los objetivos principales se han cumplido:

- Realizar un entorno en el que podamos:
    - Insertar componentes al sistema.
    - Visualizar el mapa de memoria.
    - Agregar nuevos componentes a la librería.
    - Generar el sistema.

Podemos comentar que la parte principal del proyecto se ha podido completar con éxito. En un principio se había hablado de poder insertar *customs instructions* al sistema, pero dado que la primera parte y principal no surgía del todo bien, se optó por dejarlo apartado.

En un período corto de tiempo, este proyecto estará accesible en la web de OpenCores para que los desarrolladores de SoC basados en OpenRISC puedan empezar a desarrollar sus trabajos con este software. Por parte de mi director de proyecto y mía, intentaremos seguir mejorando este software según el feedback que nos den los posibles desarrolladores de OpenRISC y también nosotros como usuarios.

5.2 Evolución futura

Como se ha dicho en el apartado anterior se han cumplido los objetivos marcados inicialmente. Pero esto no cabe que la aplicación tenga muchos puntos en los que poder mejorar:

- En cuanto al entorno, la interfaz podría ser más vistosa, a la hora, por ejemplo de subir los ficheros al server, o al eliminarlos, podríamos decir que con menús más detallados.

- Inserción de customs instructions.

- Se podrían cambiar las opciones del clk (*clock*) para poder cambiar su frecuencia o poder añadir un clk nuevo. Otra podría ser cambiar los nombres de los componentes dinámicamente.

- Gráficamente se podría mostrar un gráfico de cómo queda el sistema después de haber añadido ciertos componentes, con la opción de poder borrar los componentes directamente desde el gráfico. De esta manera todo parecería más simple e intuitivo para el usuario final.

- Habría que mejorar la seguridad a la aplicación, como por ejemplo, referente a los archivos subidos al server, o como controlar de forma un poco más sofisticada las sesiones de cada usuario.

- Otra parte sería ajustar el código para usuarios de otros S.O como Linux o OS X.

- La observación del funcionamiento de la aplicación frente a muchos códigos diferentes y ver su correcta generación.

5.3 Experiencia personal y profesional

Una vez finalizado el proyecto, podríamos decir que llevar a cabo este trabajo ha sido satisfactorio para mí y para mis conocimientos. El haber podido aplicar algunos o muchos de los conceptos que he estudiado durante la carrera, el haber podido manejar herramientas web, o el haber aprendido nuevos conocimientos y de poder ponerlos en práctica, referentes todos ellos en el mundo de los sistemas digitales, me llena de una gran satisfacción.

Lo que me llenaría más alegremente es que el trabajo realizado sirviera su uso para otras personas que se dediquen a esto de los sistemas digitales, y que este fuera el principio de un gran proyecto.

Bibliografía

[1] "Lógica Programable: FPGA", Universidad de Buenos Aires, Facultad de Ingeniería, Sistemas Digitales, 2007.

<http://cactus.fi.uba.ar/6617/Descargas/fpga.pdf>

[2] Guillem Nadal Vicente, "Sistemes Operatius sobre Processadors Virtuals en FPGA's", PFC 2005.

[3] Moisés Pérez Gutiérrez, "Introducción a la Tecnología FPGA".

<http://ccc.inaoep.mx/fpgacentral/pdf/introfpga.pdf>

[4] Marc Moreno, "Optimitzacions de sistemes orientats al tractament intensiu de dades en plataformes reconfigurables", Tesina 2009.

[5] Damjan Lampret, "OpenRISC 1200 Specification", 2001.

<http://opencores.org/svnget,or1k?file=/trunk/or1200/doc/openrisc1200_spec.pdf>

[6] Metodología de Diseño SoC con OpenRISC sobre FPGA, 2008.

<http://www.dte.us.es/id2/publicaciones/congresos/2008-07-fie-3.pdf>

[7] WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores Revision: B.3, 2002.

<http://www.mondueri.com/iiepci/docs/08-DocWishbone-8.pdf>
<http://cdn.opencores.org/downloads/wbspec_b3.pdf>

[8] Altera Corporation, "DE2 Development and Education Board (User Manual)", 2006.

<ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf>

[9] OR1200 OpenRISC processor.

<http://opencores.org/project,or1k>

[10] GNU toolchain (OpenCores).

<http://opencores.org/openrisc,gnu_toolchain>

[11] “MySQL”.

<http://www.uaem.mx/posgrado/mcruz/cursos/miic/MySQL.pdf >

[12] “Wikipedia.org, phpMyAdmin”.

<http://es.wikipedia.org/wiki/PhpMyAdmin>

[13] “Wikipedia.org, Servidor de aplicaciones”.

<http://es.wikipedia.org/wiki/Servidor_de_aplicaciones>

[14] “Desarrollo web, php a fondo”.

< http://www.desarrolloweb.com/php/ >

[15] “Wikipedia.org, Servidor HTTP Apache”.

<http://es.wikipedia.org/wiki/Servidor_HTTP_Apache>

[16] “Introducción a CSS”.

<http://www.librosweb.es/css/index.html >

[17] “Introducción a JavaScript”.

<http://www.librosweb.es/javascript/index.html>

[18] Jorge Chávez “Manual de Verilog”, 1999.

<http://www2.elo.utfsm.cl/~lsb/elo211/labs/docs/verilog-chavez.pdf>

Firmado:

Bellaterra, 06 Septiembre de 2010

Resum

Aquest projecte consisteix en la realització d'un entorn gràfic que serveixi per generar SoCs basats en el processador soft-core OpenRISC. Aquest entorn permetrà afegir diferents components de manera dinàmica a un repositori d'IPs, mostrar i sel·leccionar qualsevol component disponible dins d'aquest repositori, amb la finalitat d'unir-los al bus del sistema i fer-los accessibles al processador OpenRISC. L'entorn també mostrarà en tot moment com va evolucionant el nostre SoC, guardarà cadascún dels projectes que es realitzen amb aquest entorn i finalment permetrà generar el SoC dissenyat.

Resumen

Este proyecto consiste en la realización de un entorno gráfico que sirva para generar SoCs basados en el procesador soft-core OpenRISC. Este entorno permitirá añadir diferentes componentes de manera dinámica a un repositorio de IPs, mostrar y seleccionar cualquier componente disponible dentro de este repositorio, con la finalidad de unirlos al bus del sistema y hacerlos accesibles al procesador OpenRISC. El entorno también mostrará en todo momento como va evolucionando nuestro SoC, guardará cada uno de los proyectos que se realizan con este entorno y finalmente permitirá generar el SoC diseñado.

Abstract

This project involves the implementation of a graphical environment that serves to generate processor-based SoCs soft-core OpenRISC. This environment will add different components dynamically to a repository of IPs, display and select any component available within this repository, in order to connect them to the system bus and make them accessible to the processor OpenRISC. The environment also displayed at all times as our SoC is evolving, keep each of the projects undertaken with this environment and eventually will generate the SoC design.