



Departament d'Enginyeria Electrònica Elèctrica i Automàtica

Coprocessadores biométricos en una arquitectura SoC Wishbone

Titulació: Ingeniería en Automática y Electrónica Industrial

AUTOR: Eloy Vilella Guiu
DIRECTOR: Enric Cantó Navarro

DATA: 06 / 2007

1 Índice

1 Índice.....	2
2 Introducción.....	5
2.1 Antecedentes	5
2.2 Objetivos.....	5
2.3 Definiciones	6
2.3.1 System on Chip (SoC)	6
2.3.2 Application-Specific Integrated Circuit (ASIC).....	6
2.3.3 Field Programmable Gate Array (FPGA).....	7
2.3.4 Electronic Design Automation (EDA)	9
2.3.5 Hardware Description Language (HDL)	9
2.3.6 Tecnología Open Source (código abierto).....	10
2.3.7 IP Cores	11
2.3.8 OpenCores	11
2.3.9 GNU Project	11
2.3.10 Tecnología RISC (Reduced Instruction Set Computer)	12
2.4 Software Principal.....	12
2.4.1 Xilinx ISE Foundation 8.2i.....	12
2.4.2 ModelSim SE 6.0d.....	17
2.4.3 GNU Toolchain	19
2.4.4 Cygwin	19
3 Memoria Descriptiva.....	20
3.1 Bus Wishbone.....	20
3.1.1 Objetivos.....	20
3.1.2 Principios de funcionamiento	21
3.2 Proyecto ORP.....	23
3.2.1 ORPSoC	24
3.2.2 Topología de Bus.....	28
3.3 Sistema Básico.....	36
3.3.1 Hardware	37
3.3.2 Software.....	40
3.3.3 Simulación Funcional	50
3.3.4 Síntesis.....	75

3.4	Sistema Avanzado con Controlador de Memoria Externa.....	84
3.4.1	Hardware	86
3.4.2	Software.....	95
3.4.3	Simulación Funcional	102
3.5	Sistema Avanzado con Coprocesador.....	111
3.5.1	Coprocesador coPMM.....	113
3.5.2	Hardware	115
3.5.3	Modelo de memoria.....	120
3.5.4	Conexión al bus	121
3.5.5	Software.....	124
3.5.6	Simulación Funcional	132
3.5.7	Síntesis.....	149
3.5.8	Simulación Post-Layout	158
3.5.9	Configuración de la FPGA	163
3.5.10	Depurado	176
4	Conclusiones.....	179
4.1	Resultados Experimentales.....	179
4.2	Propuestas	181
5	Anexos.....	185
5.1	Anexos de Software	185
5.1.1	Anexo 1: Programa hello-uart (software).....	185
5.1.2	Anexo 2: Programa checkmem (software)	192
5.1.3	Anexo 3: Programa checkcopro (software).....	194
5.1.4	Anexo 4: Programa Maio-Maltoni (software).....	195
5.2	Anexos de Hardware	205
5.2.1	Anexo 5: Jerarquía superior del diseño	205
5.2.2	Anexo 6: Simulation Testbench	218
5.2.3	Anexo 7: Modelo de memoria ROM interna.....	219
5.2.4	Anexo 8: Modelo de memoria RAM interna no sintetizable.....	222
5.2.5	Anexo 9: Modelo de memoria RAM interna sintetizable.....	225
5.2.6	Anexo 10: Modelo de memoria RAM externa	227
5.2.7	Anexo 11: Controlador de memoria RAM externa	232
5.2.8	Anexo 12: Coprocesador	251
5.2.9	Anexo 13: DCM	281
5.2.10	Anexo 14: General Purpose I/O	283

5.3	Proyecto ORPSoC	284
------------	------------------------------	------------

2 Introducción

2.1 Antecedentes

Los coprocesadores biométricos están todavía en fase embrionaria, ya que la mayor parte de los sistemas embebidos de identificación por huella dactilar están basados en microprocesadores o *DSPs* de elevadas prestaciones, y donde el tiempo de procesamiento de extracción es del orden de 1 segundo o más. Algunas de las soluciones propuestas son:

- La empresa *IKENDI* ofrece el *ASIC IKE-1* basado en un *ARM7TDMI* con un coprocesador de unas 50K puertas lógicas y que asegura una reducción de 10-20 veces el tiempo de procesamiento al compararse con otras soluciones basadas en *DSP*.
- El prototipo *ThumbPod* está basado en un microprocesador *LEONII* con un coprocesador *DFT*, ambos mapeados en una *FPGA Virtex2*. Los resultados experimentales muestran que el coprocesador permite una reducción del 55% durante la extracción de la minutia, aunque un tiempo de 4 segundos es demasiado elevado para muchas aplicaciones.
- En un trabajo que forma parte del proyecto *TRUST –eS* financiado por el McyT en un programa *PROFIT*, se ha implementado un sistema basado en el microprocesador *MicroBlaze* y el bus *OPB* con un coprocesador de extracción de minutia optimizado. Los resultados experimentales muestran que el coprocesador (ocupa 58K puertas aprox.) permite una reducción del 52% del tiempo durante la extracción de la minutia.

En todas las soluciones mencionadas el sistema está formado por *IP Cores* propietarios, es decir, protegidos bajo diferentes licencias sobre la patente y derechos de *copyright* por propiedad intelectual. Por esta razón, el coste fijo de fabricación resulta elevado.

2.2 Objetivos

El principal objetivo de este trabajo es demostrar la factibilidad de la implementación de un sistema embebido de reconocimiento de huella dactilar bajo licencias *open source*. Con este fin, se definen los siguientes objetivos secundarios:

- Estudio del microprocesador *OpenRISC* y sus herramientas de desarrollo.
- Estudio de la arquitectura *Wishbone*.
- Estudio de diseños de referencia de periféricos *Wishbone* y *OpenRISC* (proyecto *ORPSoC*).
- Estudio de las herramientas de simulación *VHDL (ModelSim)* y de síntesis e implementación (*ISE Foundation*).
- Obtención de un sistema embebido básico funcional basado en el bus *Wishbone*.
- Adaptación del coprocesador biométrico al bus *Wishbone*.

- Simulación funcional, síntesis, implementación, simulación post-layout y verificación del diseño sobre un dispositivo *FPGA*.
- Ejecución de un algoritmo biométrico y comprobación del funcionamiento del coprocesador.

2.3 Definiciones

2.3.1 *System on Chip (SoC)*

Se trata de un concepto de integración de todos los componentes de un sistema electrónico en un mismo chip. Puede contener señales digitales, analógicas, mixtas e incluso funciones de radio-frecuencia. Una aplicación típica se encuentra en el área de los sistemas embebidos. Si la construcción de un SoC para una aplicación particular no fuera factible, una alternativa sería un sistema SiP (*system in package*) que comprima un número de chips en un mismo encapsulado.

Un SoC típico consiste en:

- uno o más microcontroladores, microprocesadores o *DSP cores*
- bloques de memoria, incluyendo una selección de *ROM*, *RAM*, *EEPROM* y *Flash*
- fuentes de temporización
- periféricos (contadores, relojes de tiempo real...)
- interfaces externos (*USB*, *FireWire*, *Ethernet*, *UART*, *SPI*...)
- interfaces analógicos (convertidores *A/D* y *D/A*)
- circuitos reguladores de tensión y de control de potencia.

La mayoría de *SoC* se desarrollan a partir de bloques prediseñados para los elementos de hardware mencionados anteriormente (*IP Cores*), junto con el software de los drivers que controlan su operación. Los bloques hardware se conectan mediante herramientas *EDA*; mientras que los módulos software son integrados usando un entorno de desarrollo de software.

Un punto clave en el flujo de diseño de sistemas *SoC* es la emulación: donde el hardware es mapeado sobre una plataforma de emulación basada en *FPGA* que imita el comportamiento del *SoC*, y donde los módulos software se transfieren a la memoria de la plataforma. Una vez programada, la plataforma de emulación permite que el hardware y el software del *SoC* sean testeados y depurados a una velocidad próxima a la de operación real.

2.3.2 *Application-Specific Integrated Circuit (ASIC)*

Se trata de un circuito integrado customizado para un uso particular. Debido a la reducción del tamaño de los elementos y a la mejora en las herramientas de diseño a lo largo de los años, la complejidad máxima (y, por lo tanto, la funcionalidad) posible ha crecido desde 500 puertas lógicas hasta por encima de los 100 millones. Los *ASIC* modernos a menudo incluyen procesadores de 32 bits, bloques de memoria y otros bloques. Los diseñadores de *ASIC* digital utilizan lenguajes *HDL*, como *Verilog* o *VHDL*, para describir la funcionalidad de los *ASIC*.

Las *FPGA* son la alternativa moderna al prototipado de sistemas *ASIC* ya que, debido a su característica de fácil reprogramación, permiten la utilización del mismo dispositivo para muchas aplicaciones diferentes. Para diseños y/o volúmenes de

producción menores, las *FPGA* pueden ser más coste efectivas que un diseño *ASIC*. El coste de preparación de una fábrica para producir un *ASIC* particular puede oscilar entre los cientos de miles de euros.

El término general *application specific integrated circuit* incluye las *FPGA*, pero la mayoría de diseñadores utilizan *ASIC* solamente para dispositivos no programables en campo.

2.3.3 Field Programmable Gate Array (FPGA)

Se trata de un dispositivo semiconductor que contiene componentes lógicos programables e interconexiones programables. Los componentes lógicos pueden ser programados para imitar la funcionalidad de puertas lógicas básicas (como *and*, *or*, *xor*, *not*) o funciones combinacionales más complejas (como decodificadores o funciones matemáticas simples). En la mayoría de *FPGAs*, estos bloques lógicos incluyen, a su vez, elementos de memoria (como simples flip-flop o bloques de memoria más completos).

Una red de conexiones internas programables permite al diseñador del sistema conectar los bloques lógicos de la *FPGA* según sus necesidades. Tras el proceso de fabricación, estos bloques y conexiones lógicas pueden ser programadas por el cliente/diseñador para que la *FPGA* pueda ejecutar cualquier función lógica que se precise.

Generalmente, las *FPGAs* son más lentas que los circuitos *ASIC*, no pueden manejar diseños tan complejos y consumen más potencia. Sin embargo, tienen muchas ventajas como un *time to market* menor y la posibilidad de reprogramación *in the field* para solventar errores. En el mercado existen versiones más baratas de *FPGAs* que no pueden ser modificadas tras el proceso de fabricación. Otra alternativa son las *CPLDs* (*complex programmable logic devices*).

Arquitectura

La arquitectura básica típica consiste en un array de bloques lógicos configurables (*CLB*) y canales de rutado. El típico bloque lógico en una *FPGA* se compone de una *LUT* (*lookup table*) de 4 entradas y un flip-flop, como se muestra en la figura:

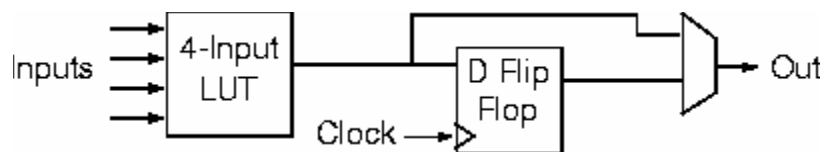


Imagen 1 - Esquema de la CLB típica de una FPGA

El bloque tiene una salida solamente, que puede ser la salida registrada o no registrada de la *LUT*. La señal de *clock*, junto con otras señales, son rutadas por canales dedicados a propósitos especiales en las *FPGAs* comerciales, por lo tanto son gestionadas aparte.

Generalmente, el rutado en una *FPGA* es no-segmentado. Cada segmento de cableado abarca solamente un bloque lógico antes de terminar en una *switch box*. Activando algunos de los interruptores de la caja se pueden construir caminos más largos. Siempre que un canal vertical intersekte con uno horizontal habrá una *switch box*.

Las familias modernas de *FPGA* se expanden en incluir funcionalidades de alto nivel fijadas en la silicón. Esta característica ofrece ventajas como un área requerida menor y un incremento de la velocidad de ejecución de esas funciones comparado con la arquitectura formada a partir de bloques primitivos. Ejemplos de funciones de alto nivel son multiplicadores, bloques *DSP* genéricos, procesadores embebidos, lógica de *I/O* de alta velocidad y memorias embebidas.

Diseño y programación de FPGA

Para definir el comportamiento de un *FPGA* el usuario proporciona un diseño *HDL* (*hardware description language*) o un diseño esquemático. Lenguajes *HDL* comunes son *VHDL* y *Verilog*. Entonces, usando una herramienta *EDA* (*electronic design automation*), se genera una *netlist* de tecnología mapeada. Mediante el uso de software para el *place&route* se corresponde la *netlist* generada con la arquitectura *FPGA* usada. El usuario valida el *layout*, resultado del *place&route*, mediante *timing analysis*, simulación y otras metodologías de verificación. Una vez completados los procesos de diseño y validación, se genera el fichero binario usado para reconfigurar el dispositivo *FPGA*.

En un flujo de diseño típico, un desarrollador de aplicaciones *FPGA* simulará el diseño en múltiples etapas a lo largo del proceso de diseño. Inicialmente la descripción *RTL* en *VHDL* o *Verilog* es simulada funcionalmente mediante la creación de un *test bench* para estimular el sistema y observar los resultados. Entonces, después que el sintetizador haya mapeado el diseño a una *netlist*, ésta se traduce a una descripción a nivel de puerta lógica donde la simulación post-síntesis se realiza para confirmar que la síntesis se produjo sin errores. Finalmente, el diseño se vuelca sobre la *FPGA*, punto en el que se añaden los retardos de propagación y se realiza la simulación post-layout.

Para simplificar el diseño de sistemas complejos en *FPGAs*, existen librerías de funciones complejas predefinidas y circuitos que han sido testeados y optimizados para agilizar el proceso de diseño. Estos circuitos predefinidos se llaman *IP Cores*, y están disponibles de fabricantes de *FPGA* y de proveedores de IP (raramente gratis, y típicamente protegidos bajo licencias de propiedad). Otros circuitos predefinidos están disponibles de comunidades de desarrolladores como *OpenCores* (típicamente gratuitas, y publicadas bajo licencias *GPL*, *BSD* o parecidas).

FPGA con CPU Core

Algunas aplicaciones han usado un solo dispositivo *FPGA* para reemplazar la función de un microcontrolador embebido simple. Posteriormente, el *core* de una CPU completa de 32 bits puede ser implementado mediante la lógica programable de una *FPGA* de alta capacidad. Estos CPU cores se conocen como *soft CPU Cores*, ejemplos de ello son *MicroBlaze™*, *Nios II™* y *LatticeMicro32™* de *Xilinx*, *Altera* y *Lattice* respectivamente.

Más allá, algunos dispositivos *FPGA* contienen cores CPU dedicados (*hard-core*). Una CPU embebida contiene exactamente la lógica y solamente las estructuras lógicas necesarias para las funciones de una CPU. Estas CPU embebidas pueden resultar más fáciles de integrar en una aplicación basada en *FPGA* debido a que sus características de temporización son en este caso predecibles, y a que la complejidad de su equivalente formado a partir de lógica programable consume muchos más recursos de la *FPGA*, complicando el *place&route* del diseño. Sin embargo, el uso de CPU embebidas puede limitar la elección de dispositivos disponibles, tanto de proveedores como de herramientas de diseño.

2.3.4 Electronic Design Automation (EDA)

El acrónimo EDA da nombre a la categoría de herramientas (tanto *software* como *hardware*) utilizadas para el diseño y producción de sistemas electrónicos, abarcando desde placas de circuito impreso a circuitos integrados.

El concepto de CAD (diseño asistido por ordenador, *Computer Aided Design*) significa proceso de diseño que emplea sofisticadas técnicas gráficas por ordenador, apoyadas por paquetes de software para la resolución de problemas de cálculo, de desarrollo, costes, etc. asociados con el trabajo de diseño.

El impacto de las herramientas de CAD sobre el proceso de diseño de circuitos electrónicos y sistemas procesadores es fundamental. No sólo por la adición de interfaces gráficas para facilitar la descripción de esquemas, sino por la inclusión de herramientas como simuladores, que facilitan el proceso de diseño y la verificación de las unidades de diseño.

El diseño hardware tiene un problema que no existe en la producción software. Este problema es el alto coste de *prototipación-testeado-vuelta a empezar* (costes NRE), ya que el coste del prototipo suele ser bastante elevado. El enfoque de las herramientas de diseño ha ido encauzado a la inclusión de la etapa de prototipado solamente al final de las etapas de diseño, evitando la fabricación de varios prototipos a lo largo del proceso de diseño. Para ello se introduce la fase de simulación y comprobación de circuitos utilizando herramientas CAD, de forma que no sea necesario la realización física del prototipo para la comprobación del funcionamiento del circuito.

En el ciclo de diseño hardware las herramientas CAD están presentes en todos los pasos. Inicialmente en la fase de descripción de la idea, que estará constituido por un esquema eléctrico, diagrama de bloques, etc. En segundo lugar en la fase de simulación y comprobación de circuitos, donde diferentes herramientas permiten realizar simulación funcional y digital eléctrica de un circuito atendiendo al nivel de simulación requerido. Por último existen las herramientas de CAD orientadas a la fabricación. En el caso del diseño hardware estas herramientas sirven para la realización de PCBs (*Printed Circuit Boards* o placas de circuito impreso), y también para la realización de ASICs, o circuitos integrados de aplicación específica (*Application Specific Integrated Circuits*). Estas herramientas permiten la realización de microchips, así como la programación de dispositivos programables.

2.3.5 Hardware Description Language (HDL)

Se trata de un lenguaje para la descripción formal de circuitos electrónicos. Puede describir tanto la operación del circuito, su diseño y tests para verificar su operación mediante simulación. Un HDL es una expresión estándar basada en texto del comportamiento temporal y/o la estructura circuital de un sistema electrónico. En contraste con un lenguaje de programación software, la sintaxis y la semántica de un HDL incluyen notaciones explícitas para expresar tiempo y concurrencia, que son atributos principales del hardware. Aquellos lenguajes cuya única característica es la de expresar la conectividad del circuito entre una serie de bloques se clasifican como lenguajes *netlist*. Los lenguajes de descripción hardware más comunes son VHDL y Verilog.

Los HDL son utilizados para escribir especificaciones ejecutables de algunos hardwares. Un programa de simulación, diseñado para implementar la semántica subyacente de las declaraciones del lenguaje, combinado con la simulación del progreso

del tiempo, proporciona al diseñador de hardware la habilidad de modelar un dispositivo hardware antes de ser creado físicamente.

Un programa software llamado sintetizador puede deducir operaciones lógicas hardware de las declaraciones del lenguaje HDL y producir una *netlist* equivalente de primitivas hardware genéricas para implementar el comportamiento especificado. Esto suele requerir que el sintetizador ignore la expresión de cualquier construcción temporal en el texto.

Diseño con HDL

Los lenguajes HDL se utilizan para diseñar dos tipos de sistemas. Primero, son utilizados para diseñar un circuito integrado dedicado (ASIC), como un procesador u otro tipo de chip de lógica digital. En este caso, un HDL especifica un modelo del comportamiento de un circuito antes que el circuito sea diseñado y construido (*built*). El resultado final es un chip de silicio que será producido en una fábrica.

En segundo lugar, son utilizados para la programación de dispositivos lógicos programables (PLD), como las FPGA. El código HDL es introducido en un compilador lógico, y la salida se carga en el dispositivo. La característica especial de este proceso, y de la lógica programable en general, es la posibilidad de alterar el código, compilarlo y cargarlo en el mismo dispositivo múltiples veces para testarlo.

2.3.6 Tecnología Open Source (código abierto)

- **Software de código abierto** - software cuyo código fuente es publicado y se pone a la disponibilidad del público, permitiendo que cualquiera lo copie, modifique y redistribuya sin pagar impuestos ni licencias. Ejemplos de productos de código abierto son:
 - + Linux - sistema operativo basado en *Unix*
 - + Apache - servidor de web http
 - + Moodle - entorno de aprendizaje a distancia
 - + Mozilla Firefox - buscador de web
 - + Mozilla Thunderbird - cliente e-mail
 - + OpenOffice.org — office suite
 - + Mediawiki — software del servidor wiki
- **Hardware de código abierto** - hardware cuya especificación inicial, usualmente en formato software, es publicada y se pone a la disponibilidad del público, permitiendo que cualquiera lo copie, modifique y redistribuya sin pagar impuestos ni licencias. Algunas de las iniciativas de hardware de código abierto son:
 - + Sun Microsystem's OpenSPARC T1 - procesador multicore. Sun afirma en la prensa: "El código fuente será donado bajo una licencia aprobada de código abierto."
 - + Arduino - una plataforma de microcontrolador para aficionados, artistas y diseñadores.
- **Diseño abierto (*open design*)** - consiste en aplicar metodologías de código abierto en el diseño de artilugios y sistemas en el mundo físico. Se trata de una tendencia naciente pero con mucho potencial.

Con la evolución de los dispositivos lógicos reconfigurables, compartir diseños lógicos es también una forma de hardware abierto. En lugar de compartir los esquemas

(*schematics*) es el código HDL lo que se comparte. Esto es diferente al *software de código abierto*. Las descripciones HDL se utilizan normalmente para realizar sistemas SoC (*system on a chip*) tanto en FPGA's o directamente en diseños ASIC. Los módulos HDL, cuando se distribuyen, son llamados IP Cores (*intellectual property cores*).

2.3.7 IP Cores

En diseño electrónico un IP Core (*intellectual property core*) es una unidad lógica, celda o diseño reutilizable que es, a su vez, propiedad de un grupo. El término se deriva de la licencia sobre la patente y de los derechos de copyright por propiedad intelectual del código fuente que subsisten en el diseño. Pueden ser utilizados como bloques de diseño en diseños de chips ASIC o diseños de lógica FPGA. En la industria del diseño electrónico, los IP Cores han tenido un profundo impacto en el diseño de sistemas SoC (*system on a chip*) debido a su característica de reutilizabilidad de los diseños.

2.3.8 OpenCores

Se trata de un colectivo de personas interesadas en el desarrollo de *hardware de código abierto* (*open source*) mediante EDA (*electronic design automation*), con una filosofía similar al movimiento de software libre. Actualmente, se centran en módulos digitales llamados *cores*, comúnmente conocidos como *IP Cores*. Éstos son implementados en lenguajes de descripción de hardware. Los componentes son utilizados para crear sistemas ASIC y sistemas mapeados sobre FPGA.



Los componentes producidos por la iniciativa OpenCores se apoyan en muchas licencias de software diferentes, pero la más común es la GNU LGPL, que determina que cualquier modificación en un componente debe ser compartida con la comunidad, mientras se pueda seguir usando junto con componentes propietarios.

2.3.9 GNU Project

Unix es un sistema operativo no libre muy popular, porque está basado en una arquitectura que ha demostrado ser técnicamente estable. El sistema GNU fue diseñado para ser totalmente compatible con Unix. Su nombre es un acrónimo recursivo para "GNU's Not Unix", que fue escogido porque su diseño es similar al de Unix, con la diferencia de que se trata de software libre. Actualmente, el sistema está siendo activamente desarrollado en lo que se denomina el *GNU Project*.



El proyecto fue iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre. El hecho de ser compatible con la arquitectura de Unix implica que GNU está compuesto por pequeñas piezas individuales de software. Las partes principales del sistema son el kernel, las librerías, las utilidades del sistema, los compiladores y las aplicaciones.

Tratando que el software GNU permaneciera libre para que todos los usuarios pudieran "ejecutarlo, copiarlo, modificarlo y distribuirlo", el proyecto fue liberado bajo una licencia diseñada para garantizar esos derechos al mismo tiempo que evitar restricciones posteriores de los mismos. La idea se conoce en inglés como copyleft (en

clara oposición a copyright), y está contenida en la Licencia General Pública de GNU (GPL).

2.3.10 Tecnología RISC (*Reduced Instruction Set Computer*)

Se trata de un tipo de μ P 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 de datos.
- 3) Disposición de un alto número de 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. RISC es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeños y simples que toman menor tiempo para ejecutarse. Las características que generalmente son encontradas en los diseños RISC son:

- Codificación uniforme de instrucciones, 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
- Modos de direccionamiento simple con modos más complejos reemplazados por secuencias de instrucciones aritméticas simples

Los diseños RISC también prefieren utilizar como característica un modelo de memoria Harvard, donde los buses para el acceso a instrucciones y datos están conceptualmente separados. Esto permite que ambos caches sean accedidos separadamente, lo que puede en algunas ocasiones mejorar el rendimiento.

2.4 Software Principal

2.4.1 Xilinx ISE Foundation 8.2i

El *Integrated Software Environment* (ISE) es el paquete *software* de desarrollo de sistemas de Xilinx requerido para implementar diseños en PLDs (Dispositivos Programables Lógicos) de Xilinx. El programa *Project Navigator* gestiona y procesa el diseño a través de los pasos del flujo de diseño de ISE.

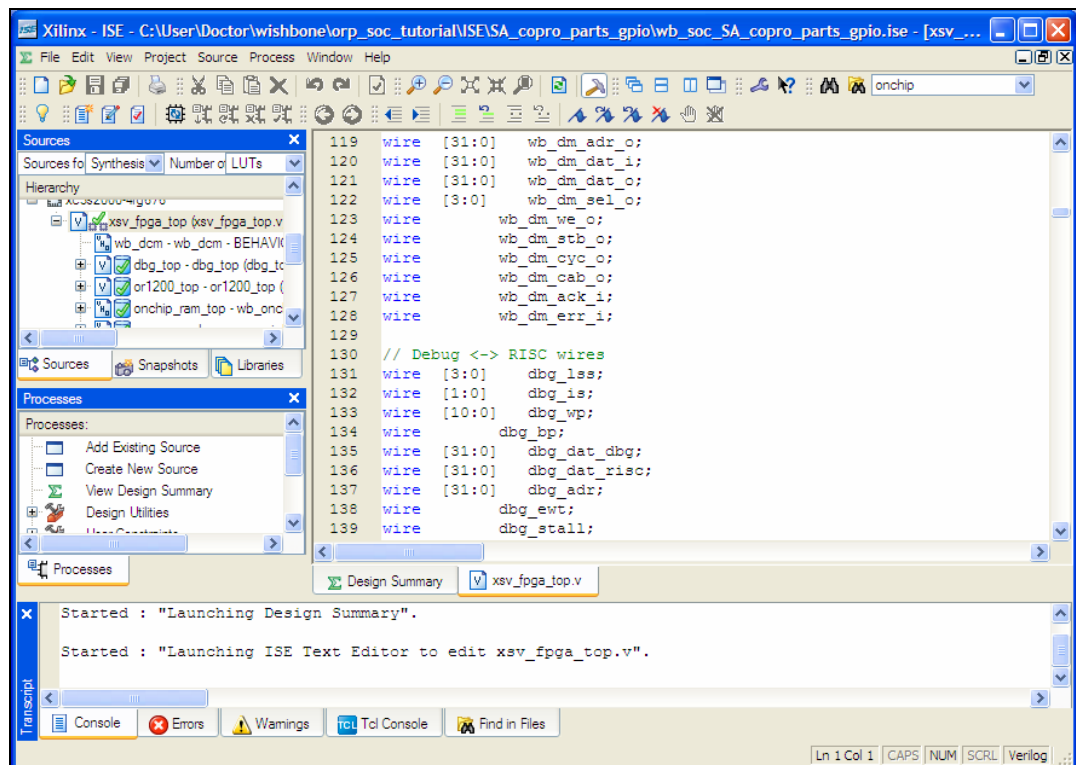


Imagen 2 - Ventana principal del programa *Project Navigator* del entorno *ISE Foundation*

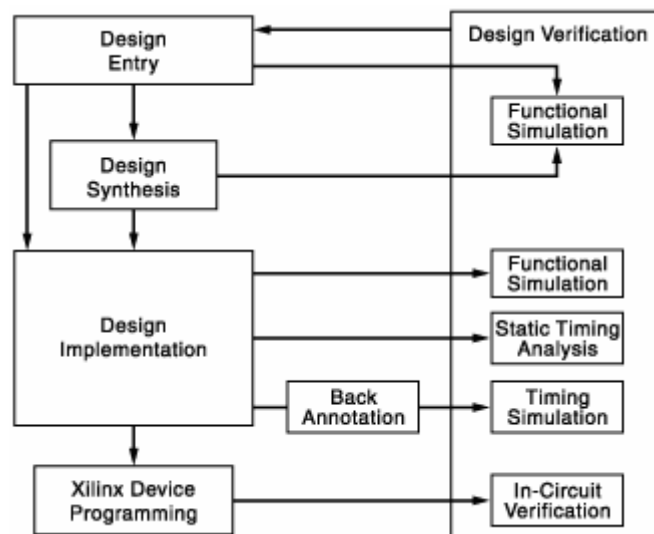











Imagen 3 - Flujo típico de diseño con entorno *ISE*

a) Design Entry

La primera fase del flujo de diseño de ISE se denomina *Design Entry*. Las principales tareas realizadas durante esta etapa son:

- Crear un proyecto de ISE
- Añadir los ficheros fuente existentes al proyecto
- Crear ficheros fuente necesarios y añadirlos al proyecto
- Asignar *constraints* como condiciones temporales, asignaciones de pines o condiciones de área (fichero *ucf*)

Se pueden utilizar múltiples formatos para los ficheros fuente de más bajo nivel. Algunos de los formatos permitidos son los listados a continuación:

Extensión	Icono	Tipo de fichero	Descripción
.bmm		Block Ram Memory Map	Usado en sistemas con procesador embebido para describir la organización de los recursos Block Ram
.edn .edf .edif		Electronic Data Interchange Format	Especifica la <i>netlist</i> del diseño en un formato estándar industrial
.elf		Executable Linkable File	Contiene una imagen codificada de la CPU ejecutable
.ise		Project File	Contiene ajustes y estado del proceso, e información de gestión del proyecto
.mem		Memory Definition	Utilizado para definir el contenido de memorias
.sch		Schematic	Contiene un diseño esquemático
.ucf		User Constraints File	Contiene las condiciones lógicas especificadas por el usuario
.v		Verilog Module	Contiene un diseño en código <i>verilog</i>
.vhd		VHDL Module	Contiene un diseño en código <i>vhdl</i>

b) Design Synthesis

La siguiente etapa en el flujo de diseño se denomina *Design Synthesis*. El software ISE incluye el *Xilinx Synthesis Technology* (XST), que sintetiza diseños *vhdl*, *verilog* o de lenguajes combinados para generar un fichero de *netlist* específica de Xilinx conocido como fichero NGC. Otras herramientas secundarias de síntesis pueden ser utilizadas desde ISE como:

- *Synplify* de Synplicity
- *LeonardoSpectrum* de Mentor Graphics
- *Precision* de Mentor Graphics

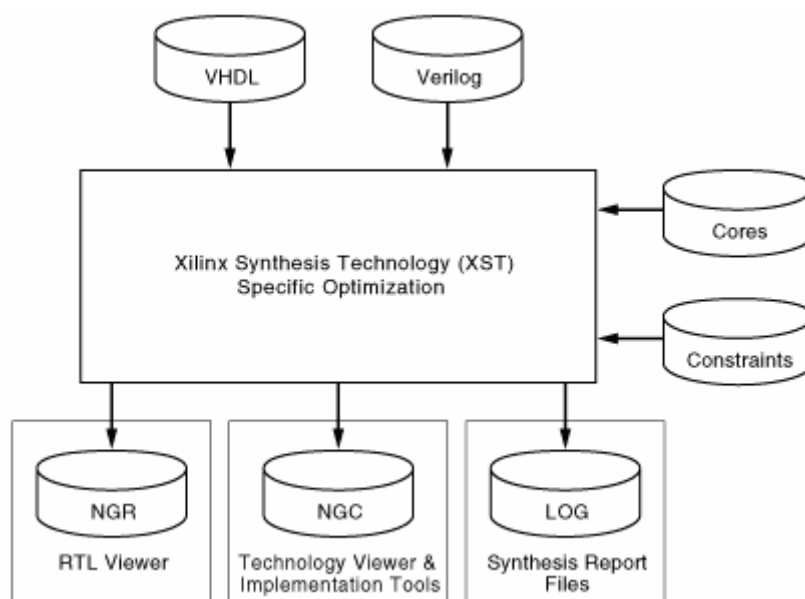


Imagen 4 - Esquema de entrada/salida a la herramienta XST

Los ficheros implicados en el proceso llamado por la herramienta XST han sido definidos en la tabla anterior. Se trata de los formatos siguientes:

Extensión	Tipo de fichero	I/O	Descripción
.vhd	VHDL Module	I	Contiene un diseño en código <i>vhdl</i> .
.v	Verilog Module	I	Contiene un diseño en código <i>verilog</i> .
.ngc .edif	Core File	I	Contiene un diseño previamente sintetizado. XST no modifica los <i>cores</i> , sino que los utiliza para informar de la optimización de área y temporización.
.xcf	Xilinx Constraints File	I	Especifica condiciones de síntesis, implementación y temporización.
.log	Synthesis Report	O	Contiene los resultados de la síntesis. Incluye la estimación de área y temporización.
.ngr	Esquema RTL	O	Contiene la representación esquemática del diseño pre-optimizado mostrado a Nivel de Transferencia de Registros (RTL). Esta representación está en términos de símbolos genéricos, como sumadores, multiplicadores, contadores y puertas lógicas, y es generada tras la fase de <i>HDL Synthesis</i> del proceso de síntesis.
.ngc	Esquema Tecnológico	O	Contiene la representación esquemática de un fichero NGC en términos de elementos lógicos optimizada a la arquitectura destino (LUTs, registros, I/O buffers...). Es generada tras la fase de <i>Optimization</i> del proceso de síntesis..

A continuación se muestra cada uno de los pasos que tienen lugar durante el proceso de síntesis con la herramienta XST:

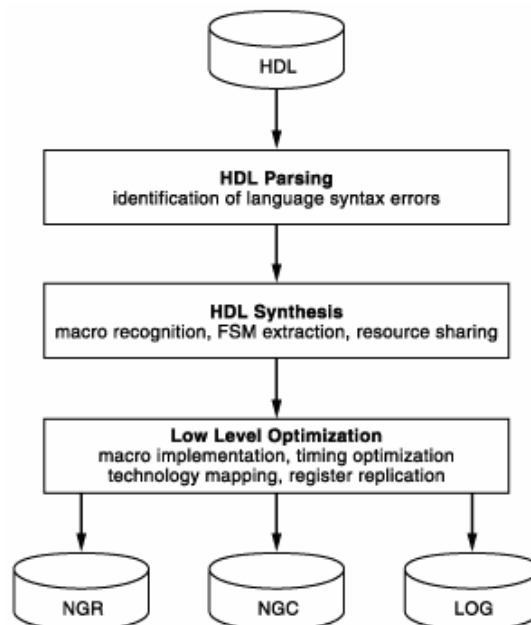


Imagen 5 - Detalle de los subprocesos de la herramienta XST

HDL Parsing:

- Comprueba la sintaxis del código *hdl*.

HDL Synthesis:

- Analiza el código *hdl* y trata de inferir bloques de diseño específicos o macro como multiplexores, bloques de memoria *ram*, sumadores...
- Reconoce los FSM (*Finite State Machine*) y determina el algoritmo de codificación óptimo para lograr los objetivos.

Low Level Optimization:

- Transforma las macros inferidas y la lógica general en implementaciones específicas de la tecnología como lógica *carry*, bloques de memoria, registros de desplazamiento, buffers de reloj, multiplexores...

c) Design Implementation

Tras el proceso de síntesis, la siguiente etapa en el flujo de diseño de ISE se denomina *Design Implementation*. El proceso de implementación se compone de los siguientes pasos principales:

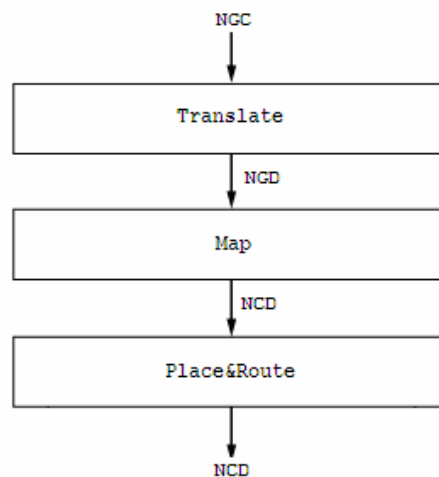


Imagen 6 - Detalle de los subprocesos del proceso de implementación

- *Translate*, que fusiona las *netlists* de entrada con las *constraints* en un mismo fichero de diseño de Xilinx.
- *Map*, que adecúa el diseño en los recursos disponibles del dispositivo destinatario.
- *Place and Route*, que ubica y conecta el diseño para conseguir los objetivos de temporización.
- *Programming file generation*, que genera una cadena de bits que puede ser descargada en el dispositivo *fpga*.

El proceso *Translate* (traducción) fusiona las *netlist* de entrada y las *constraints* del diseño en un mismo fichero de base de datos genérico nativo de Xilinx (NGD), que describe el diseño lógico reducido a primitivas de Xilinx. La herramienta ejecutada en el proceso es *NGDBuild* y los ficheros implicados son:

Extensión	Tipo de fichero	I/O	Descripción
.edf .edif	Electronic Data Interchange Format	I	Especifica la <i>netlist</i> del diseño en un formato estándar industrial
.ngc	Netlist Generic Constraints	I	Contiene los datos del diseño lógico y las <i>constraints</i> en una misma <i>netlist</i>
.ucf	User Constraints File	I	

.bmm	Block Ram Memory Map	I	Usado en sistemas con procesador embebido para describir la organización de los recursos Block Ram
.bld	Translate Report	O	
.ngd	Native Generic Database	O	Describe el diseño lógico reducido a primitivas de Xilinx

El proceso *Map* mapea la lógica definida por un fichero NGD en recursos de la FPGA, como CLBs y IOBs. El fichero resultante es una descripción del diseño nativo que representa físicamente el diseño mapeado en los recursos del dispositivo destino. La herramienta ejecutada en el proceso es *Map* y los ficheros implicados son:

Extensión	Tipo de fichero	I/O	Descripción
.ngd	Native Generic Database	I	Describe el diseño lógico reducido a primitivas de Xilinx
.nmc		I	Contiene un <i>hard macro</i> física
.ncd	Native Circuit Description	O	Representa la descripción física del circuito de un diseño de entrada de la forma aplicada en el dispositivo destino tras el <i>Mapping</i>
.pcf	Physical Constraints File	O	Contiene las <i>constraints</i> físicas que son derivadas de las condiciones lógicas tras el mapeado
.ngm		O	Contiene información sobre el diseño lógico y sobre su correspondencia con el diseño físico
.mrp	Map Report		

El proceso *Place and Route* toma el fichero NCD mapeado, realiza el emplazado y el rutado del diseño, y produce un fichero NCD definitivo. La herramienta ejecutada en el proceso es *Par* y los ficheros implicados son:




Extensión	Tipo de fichero	I/O	Descripción
.ncd	Native Circuit Description	I	Representa la descripción física del circuito de un diseño de entrada de la forma aplicada en el dispositivo destino tras el <i>Mapping</i>
.pcf	Physical Constraints File	I	Contiene las <i>constraints</i> físicas que son derivadas de las condiciones lógicas tras el mapeado
.dly		O	Contiene el <i>report</i> de los retardos asíncronos
.pad		O	Define la conexión externa física del circuito integrado.
.ncd	Native Circuit Description	O	Representa la descripción física del circuito de un diseño de entrada de la forma aplicada en el dispositivo destino tras el P&R
.par	P&R Report	O	

2.4.2 ModelSim SE 6.0d

ModelSim SE (*Special Edition*) es un entorno de simulación y depurado desarrollado por *Mentor Graphics Corporation*. Trabaja en entornos basados en *Unix*, *Linux* y *Windows*, y combina un elevado rendimiento con un poderoso e intuitivo *GUI* (interfaz gráfica de usuario) de la industria. Permite simular diseños escritos en *verilog*, *vhdl* y *SystemC* (o incluso *netlist*). Los límites entre lenguajes están impuestos al nivel

de cada unidad de diseño. Por lo tanto, cualquier instancia en la jerarquía del diseño puede corresponder a una unidad de diseño escrita en otro lenguaje sin ninguna restricción. A partir de los ficheros fuente de los cores o de los diferentes modelos de simulación obtenidos durante el proceso de síntesis e implementación con el entorno *ISE*, la herramienta *ModelSim* permite depurar el diseño mediante simulación funcional y simulación *post-layout* entre otras.

En la siguiente tabla se muestra una referencia para las tareas de compilado, descarga y simulación de un diseño en *ModelSim*:

Tarea	Comando	Menu	Icono
Mapear Librerías	vlib <library_name> vmap work <library_name>	File→New→Project Introducir <library_name> Añadir ficheros de diseño al proyecto	N/A
Compilar diseño	vlog <files> (verilog/vhdl) sscom <top> (SystemC)	Compile→Compile (All)	
Descargar diseño	vsim <top>	Simulate→Start Simulation Seleccionar top_file	
Simular	run, step	Simulate→Run	
Depurar	add wave, bp, describe, drivers, examine, force, log, checkpoint, restore, show...	N/A	N/A

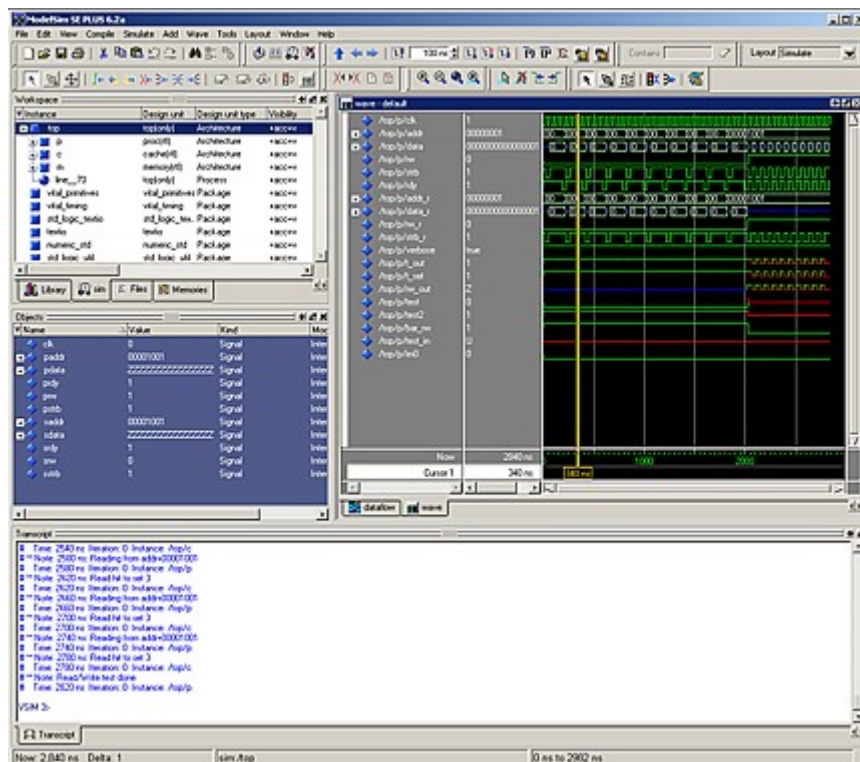


Imagen 7 - Ventana principal del programa *ModelSim* de *Mentor Graphics*

2.4.3 GNU Toolchain

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. Los proyectos incluidos en la cadena de desarrollo son:

- GNU Compiler Collection (GCC): es una colección de compilación que incluye el ANSI C Compiler, C++, Java, Fortran y otros compiladores. Todos los productos se utilizan a menudo en productos comerciales. GCC es software libre con el código fuente completo disponible.
- GNU Make: proyecto dedicado a la automatización de la estructura y de la compilación.
- GNU Binutils: es una colección de herramientas binarias. Las principales son el enlazador (GNU linker) y el ensamblador (GNU assembler). Dispone de otras herramientas como:
 - addr2line: convierte direcciones en *filenames* y números de línea.
 - ar: utilidad para crear, modificar y extraer de archivos.
 - cfilt: filtro decodificar símbolos de C++
 - gprof: muestra información de perfil
 - nlmconv: convierte código de objeto en un nml
 - nm: crea listas de símbolos de ficheros de objetos
 - objcopy: copia y traduce ficheros de objetos
 - objdump: muestra información de ficheros de objetos
 - ranlib: genera un índice de los contenidos de un fichero
 - readelf: muestra información de cualquier fichero de objetos de formato ELF
 - size: crea una lista de tamaños de sección de un objeto o fichero de objetos
 - strings: crea una lista de cadenas imprimibles de ficheros
 - strip: descarta símbolos
 - windres: compilador de ficheros de recursos de Windows
- GNU Debugger (GDB): depurador interactivo, es decir, una herramienta destinada a la localización y al tratamiento de *bugs* en el programa.
- Newlib: es una librería destinada al uso en sistemas embebidos. Es una conglomeración de diferentes partes de librerías, todas bajo licencias de software libre.

2.4.4 Cygwin

El equipo con el que se ha desarrollado el proyecto trabaja con *Windows XP* como sistema operativo nativo y realiza una vinculación con *Unix* mediante el entorno *Cygwin*. Se trata de una colección de herramientas desarrollada por *Cygnus Solutions* (*Red Hat*) para proporcionar un comportamiento similar al de los sistemas *Unix* en

sistemas *Windows*. Su objetivo es portar software que ejecuta en sistemas *POSIX* a *Windows* mediante una recompilación a partir de sus fuentes. El sistema *Cygwin* tiene varias partes diferenciadas:

- La biblioteca 'cygwin1.dll' que implementa la *API POSIX* usando para ello llamadas a la *API* nativa de *Windows* *Win32*.
- Una cadena de desarrollo *GNU (GNU Toolchain)* para facilitar las tareas básicas de desarrollo.
- Aplicaciones equivalentes (clones) a los programas más comunes de los sistemas *UNIX*.

3 Memoria Descriptiva

3.1 Bus Wishbone

Se trata de un bus para computadora basado en hardware de código abierto pensado para permitir la comunicación entre las partes de un circuito integrado. La idea es permitir la conexión de *cores* diferentes dentro de un mismo chip de una forma estándar. Utilizado en muchos de los diseños del proyecto *OpenCores*. Un gran número de diseños de código abierto para CPU's y periféricos auxiliares han sido publicados con interfaces de Wishbone.

Wishbone está destinado a ser un "bus lógico". Su especificación no trata información eléctrica o la topología de bus. En lugar de eso, la especificación está escrita en términos de señales, ciclos de reloj y niveles altos y bajos. Además, se trata de una especificación de carácter abierto que permite al usuario customizar el sistema basándose en una serie de permisiones y prohibiciones definidas en el texto.

Esta ambigüedad es intencionada. Wishbone está pensado para permitir a los diseñadores combinar muchos diseños escritos en Verilog, VHDL u otros lenguajes de descripción lógica para EDA. El bus proporciona una forma estándar de combinar estos diseños lógicos hardware, llamados *cores*.

Wishbone se adapta a las topologías de bus comunes como *point-to-point* (punto a punto), *many-to-many* (bus), *hierarchical* (jerárquico) o *switched fabrics*. En las topologías más "exóticas", el bus requiere el uso de un controlador o árbitro, pero los dispositivos siguen manteniendo el mismo interface.

El bus está definido para permitir estructuras de 8, 16, 32 y 64 bits. Todas las señales están sincronizadas a un único reloj, aunque las respuestas de los *slave* deben ser generadas de forma combinacional para un rendimiento máximo. Además permite la adición de un "tag bus" para describir los datos.

3.1.1 Objetivos

Los objetivos principales de la especificación son:

- crear una interconexión flexible para *IP cores* para formar sistemas *SoC*
- asegurar compatibilidad entre *IP cores* (mejora la reutilización de diseños)
- crear un estándar robusto que no perjudique la creatividad del diseñador

- facilitar el entendimiento para el programador y el usuario
- facilitar las metodologías de diseño estructurado usadas en equipos de proyecto grandes
- crear un interface portable independiente de la tecnología del semiconductor
- independizar los interfaces Wishbone de los niveles lógicos de las señales
- ser independiente del lenguaje *hdl* utilizado
- limitar la extensión de la documentación
- permitir a los usuarios la creación de componentes *SoC* sin infringir los derechos sobre patentes de terceros
- crear una arquitectura que permita la adaptación a nuevas tecnologías (*smooth transition path*)
- minimizar el uso de *glue logic*
- crear una arquitectura que permita una anchura de buses variable, que soporte *big/little endian* y soporten el uso de *tags*
- crear una arquitectura que soporte topologías basadas en *master/slave* e interconexiones *point-to-point*, *shared bus*, *crossbar switches* y *switched fabrics*.
- crear un protocolo síncrono que soporte múltiples frecuencias de reloj y que contenga una especificación simple de la temporización

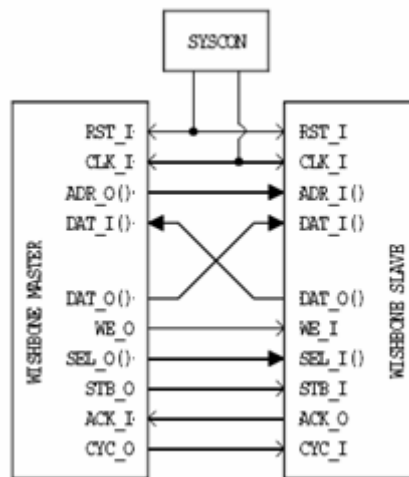


Imagen 8 - Conexión point-to-point

3.1.2 Principios de funcionamiento

En este apartado se describen brevemente las señales y los modos de operación del bus utilizados en el presente proyecto.

3.1.2.1 Descripción de señales

Señal	#bits	Descripción
rst_i	1	Realiza el <i>reset</i> del interface
clk_i	1	El reloj interno coordina todas las actividades para la lógica interna
adr_o	32	Array de datos usado para pasar direcciones binarias
dat_i	32	Array de datos usado para pasar datos binarios en un ciclo de escritura
dat_o	32	Array de datos usado para pasar datos binarios en un ciclo de lectura
we_o	1	Write enable. Indica si el ciclo de bus actual se trata de un ciclo de lectura (0) o escritura (1)

sel_o	4	Especifica en que byte se espera información válida en <i>dat_i</i> y <i>dat_o</i>
stb_o	1	Indica un ciclo de transferencia de datos válido. Valida la señal <i>sel_o</i>
ack_i	1	Indica la terminación normal del ciclo de bus
cyc_o	1	Indica un ciclo de bus válido.

3.1.2.2 Ciclos básicos de bus

a) Ciclo de lectura

Flanco de reloj 0:

- *master* pone dirección válida en *adr_o*
- *master* niega *we_o*
- *master* establece señal *sel_o*
- *master* aserta *cyc_o* y *stb_o*

Setup en el flanco 1:

- *slave* decodifica las entradas
- *slave* presenta datos válidos en *dat_i*
- *slave* introduce cualquier número de *wait states* para establecer la velocidad de transferencia adecuada
- *slave* aserta *ack_i* en respuesta a *stb_o*

Flanco de reloj 1:

- *master* almacena los datos de *dat_i*
- *master* niega *cyc_o* y *stb_o*
- *slave* niega *ack_i* en respuesta a *stb_o*

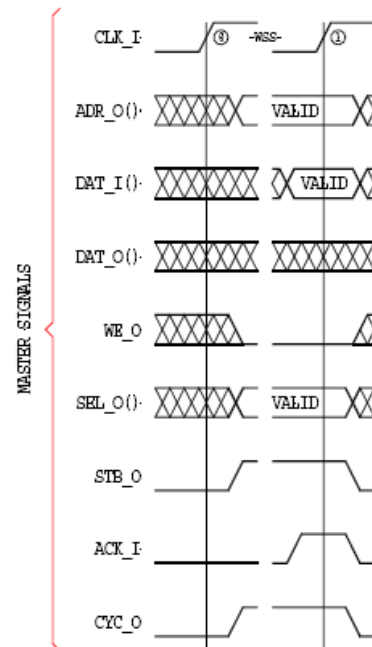


Imagen 9 - Ciclo de lectura

b) Ciclo de escritura

Flanco de reloj 0:

- *master* pone dirección válida en *adr_o*
- *master* pone datos válidos en *dat_o*
- *master* aserta *we_o*
- *master* establece señal *sel_o*
- *master* aserta *cyc_o* y *stb_o*

Setup en el flanco 1:

- *slave* decodifica las entradas
- *slave* introduce cualquier número de *wait states* para establecer la velocidad de transferencia adecuada
- *slave* aserta *ack_i* en respuesta a *stb_o*

Flanco de reloj 1:

- *slave* almacena los datos de *dat_o*
- *master* niega *cyc_o* y *stb_o*
- *slave* niega *ack_i* en respuesta a *stb_o*

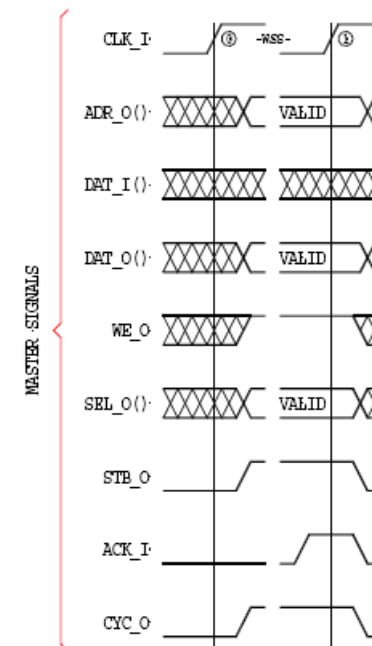


Imagen 10 - Ciclo de escritura

3.2 Proyecto ORP

OpenRISC Reference Platform es la definición de un sistema basado en el microprocesador *OpenRISC*. Define la organización del espacio de direcciones (mapa de memoria) y la asignación de las interrupciones de los diferentes dispositivos periféricos. Se trata de un proyecto en vías de desarrollo por lo que, por el momento, una definición bastante limitada está disponible en la página de *opencores*.

En la tabla siguiente se muestra el mapa de memoria definido en el proyecto. Se describe el espacio de memoria asignado a cada posible componente, así como los espacios de memoria reservados. Los nueve periféricos destacados con la etiqueta *target* se corresponden con los *slaves* conectados al árbitro del bus mediante el fichero de *traffic cop* añadido al proyecto.

0xf0000000-0xffffffff	256MB		ROM
0xc0000000-0xefffffffff	768MB		Reserved
0xb8000000-0xbfffffffff	128MB		Reserved for custom devices
0xa6000000-0xb7ffffffff	288MB		Reserved
0xa5000000-0xa5ffffffff	16MB		Debug 0-15
0xa4000000-0xa4ffffffff	16MB		Digital Camera Controller 0-15
0xa3000000-0xa3ffffffff	16MB		I2C Controller 0-15
0xa2000000-0xa2ffffffff	16MB		TDM Controller 0-15
0xa1000000-0xa1ffffffff	16MB		HDLC Controller 0-15
0xa0000000-0xa0ffffffff	16MB		Real-Time Clock 0-15
0x9f000000-0x9fffffffff	16MB	Target8	FireWire Controller 0-15
0x9e000000-0x9effffffff	16MB	Target7	IDE Controller 0-15
0x9d000000-0x9dffffffff	16MB	Target4	Audio Controller 1-15
0x9c000000-0x9cffffffff	16MB		USB Host Controller 0-15
0x9b000000-0x9bffffffff	16MB		USB Func Controller 0-15
0x9a000000-0x9affffffff	16MB		General Purpose DMA 0-15
0x99000000-0x99ffffffff	16MB		PCI Controller 0-15
0x98000000-0x98ffffffff	16MB		IrDA Controller 0-15
0x97000000-0x97ffffffff	16MB	Target2	Graphics Controller 0-15
0x96000000-0x96ffffffff	16MB		PWM/Timer/Counter Controller 0-15
0x95000000-0x95ffffffff	16MB		Traffic Cop 0-15
0x94000000-0x94ffffffff	16MB	Target6	PS/2 Controller 0-15
0x93000000-0x93ffffffff	16MB		Memory Controller 0-15
0x92000000-0x92ffffffff	16MB	Target3	Ethernet Controller 0-15
0x91000000-0x91ffffffff	16MB		GP I/O 0-15
0x90000000-0x90ffffffff	16MB	Target5	UART16550 Controller 0-15
0x80000000-0x8fffffffff	256MB		PCI I/O
0x40000000-0x7fffffffff	1GB		Reserved
0x00000000-0x3fffffffff	1GB		RAM
04000000 04ffffffff	16MB	Target1	Flash Controller
00000000 00ffffffff	16MB	Target0	SRAM Controller

A continuación, se muestra una tabla que describe la asignación de las interrupciones generadas por los diferentes periféricos:

0	Reserved
1	Reserved
2	UART16550 Controller
3	General-Purpose I/O
4	Ethernet Controller
5	PS/2 Controller
6	Traffic Cop, Real-Time Clock
7	PWM/Timer/Counter Controller
8	Graphics Controller
9	IrDA Controller
10	PCI Controller
11	General-Purpose DMA
12	USB Func Controller
13	USB Host Controller
14	Audio Controller
15	IDE Controller
16	Firewire Controller
17	HDLC Controller
18	TDM Controller
19	I2C Controller Digital Camera Controller

El propósito general del *ORP* consiste en:

- obtener una plataforma de *openrisc* común para que el *software* pueda ser compartido
- permitir una sencilla creación de nuevos *system-on-chip* basados en *openrisc* con menor tiempo de verificación del sistema

3.2.1 *ORPSoC*

Se trata de un *system-on-chip* basado en las definiciones de espacio de memoria y mapeado de interrupciones de los periféricos de la plataforma de referencia *ORP*. Se puede obtener de la página web de *opencores* y está recomendado como punto de partida de diseño para los desarrolladores de *hardware* en sistemas *SoC*. El sistema *ORPSoc* utiliza los siguientes *IP Cores*:

- procesador OR1200
- Ethernet MAC 10/100
- UART 16550
- Interface SRAM/Flash
- Interface Audio
- Interface PS/2
- Interface VGA
- Interface JTAG debug

Actualmente, el sistema está preparado para trabajar a una frecuencia máxima de 10 MHz. Hasta el momento, los colaboradores de *opencores* han centrado sus esfuerzos en conseguir que el sistema presente un comportamiento funcional correcto. Todavía no se ha trabajado a fondo la optimización del sistema en cuanto a frecuencia u ocupación de recursos se refiere.

3.2.1.1 OpenRISC 1200

La arquitectura OpenRISC 1000 es el último avance en desarrollo de arquitecturas abiertas modernas y la base de una familia de procesadores RISC/DSP de 32 y 64 bits.

El *core* OpenRISC 1200 está formado por diferentes unidades modulares, desarrolladas en lenguaje *verilog*:

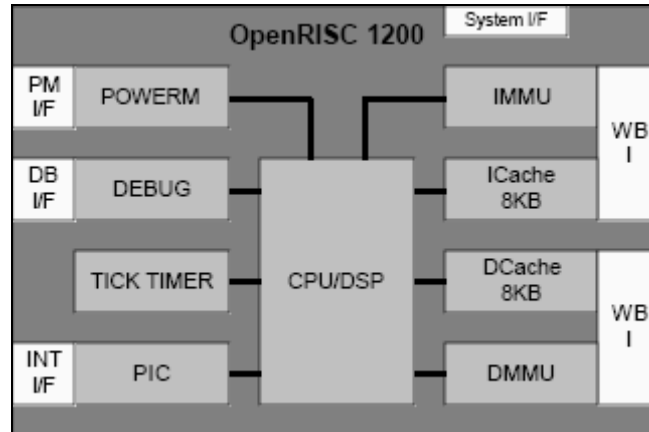


Imagen 11 - Esquema del *core* del microprocesador OpenRISC 1200

- CPU/DSP de 32 bits
 - Arquitectura de 32 bits con *set* de instrucciones *ORBIS32* implementado
 - Estructura *pipeline* de 5 etapas
 - Ejecución de instrucciones en un ciclo de reloj (en la mayoría de los casos)
 - 250 MIPS (millones de instrucciones por segundo)
 - Tasa de ejecución predecible para aplicaciones *hard real-time*
 - Respuesta a interrupción interna rápida y determinística
 - 32 registros de propósito general
 - DSP MAC 32x32
 - Instrucciones de usuario personalizables
- Caches L1
 - Modelo *harvard* con separación de cache de datos e instrucciones
 - Tamaño de cache escalable desde 1 a 64KB
 - Direccionado físicamente
 - Registros de propósito especial para gestión de cache
- Unidad de gestión de memoria
 - Modelo *harvard* con separación de MMU de datos e instrucciones
 - Tamaño de TLB escalable desde 16 a 256 entradas
 - TLB basado en *hash* con mapeo directo
 - Espacio de direcciones lineal con direcciones virtuales de 32 bits y direcciones físicas de 24 a 32 bits

- Tamaño de página de 8KB
- Unidad de gestión de potencia
 - Reducción de potencia de 2X a 100X
 - Frecuencia de reloj controlada por *software* en modos *slow* o *idle*
 - Interrupción *wake-up* en modos *doze* y *sleep*
- Unidad de depurado
 - Agente *target-debug* convencional con un manejador de excepciones de depurado
 - Seguimiento/depurado no intrusivo
 - Seguimiento en tiempo real
 - Acceso y control de la unidad de depurado desde el *RISC* o interfaz de desarrollo
 - Condiciones encadenadas complejas de *watchpoint* y *breakpoint*
- *Tick Timer* integrado
 - Programación de tareas y medición temporal precisa
 - Rango máximo de reloj de 2^{32} ciclos
 - Interrupción de reloj enmascarable
 - Modos *single-run*, *restartable* y *continuous*
- Controlador de interrupciones programable
 - 2 interrupciones no enmascarables
 - 30 interrupciones enmascarables
 - dos prioridades de interrupción
- Unidades customizables y opcionales
 - Unidades opcionales (p.e.: unidad de punto flotante, unidad de división)
 - adición opcional de 8 unidades customizables
- Herramientas de desarrollo
 - Compiladores GNU ANSI C, C++, Java y Fortran
 - Depurador, enlazador, ensamblador y utilidades GNU
 - Simulador de arquitecturas
- Sistemas Operativos soportados
 - Linux
 - uCLinux
 - OAR RTEMS de tiempo real
- Interfaz del sistema
 - Optimizado para aplicaciones *SoC*

- Doble interfaz *Wishbone*
- Variedad de *cores* de periféricos optimizados para una conexión transparente con *OpenRISC 1200*

3.2.1.2 UART16550

El *core* UART (*Universal Asynchronous Receiver/Transmitter*) provee de la capacidad de comunicación serie, que permite la comunicación con un módem u otros dispositivos externo, como otra computadora mediante el uso de un cable serie y el protocolo RS232. Diseñado para tener máxima compatibilidad con el dispositivo estándar de la industria 16550A de *National Semiconductors*.

Las características principales son:

- Interfaz *Wishbone* para los modos de bus de datos de 32 y 8 bits
- Operación FIFO
- Nivel de registros y funcionabilidad compatible con *NS16550A*
- Interfaz de depurado para el modo de bus de datos de 32 bits

3.2.1.3 Development Interface

Se trata de una interfaz de desarrollo entre el *OpenRISC*, *cores* de periféricos, y cualquier *GDB* libre o depurador/emulador comercial o dispositivo de verificación BS (*boundary scan*). El depurador externo o dispositivo BS se conecta al *core* mediante un puerto *JTAG* compatible con el estándar *IEEE 1149.1*.

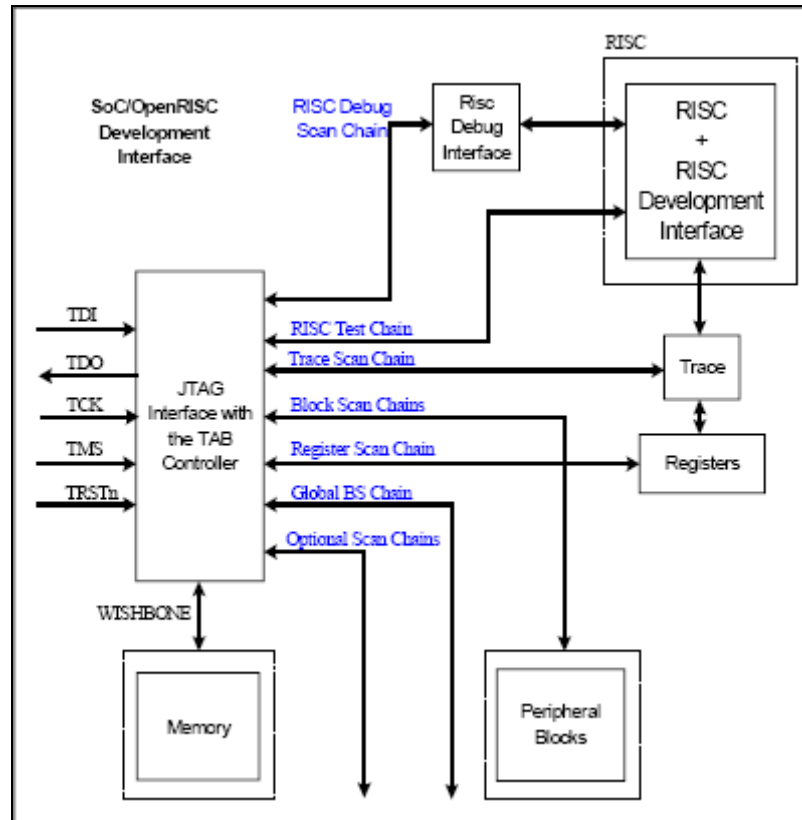


Imagen 12 - Esquema del *core* de la Interfaz de Desarrollo

Las características principales son:

- Monitorización directa de señales internas (no necesita *software* adicional)
- Controlador *TAP (Test Access Port)* compatible con el estándar *IEEE*. Se trata de la máquina de estados de control, parte de la arquitectura *boundary scan* del JTAG
- Testeo *Boundary Scan* (se necesita equipamiento adicional conectado a los pines del JTAG)
- *Trace* incorporado para monitorización del flujo del programa
- Interfaz de depurado *OpenRISC*
- Interfaz *Wishbone*

3.2.2 Topología de Bus

La topología de bus implementada en el sistema se identifica con la interconexión mediante bus compartido (*shared bus*). Esta configuración se utiliza para la conexión de múltiples *masters* con múltiples *slaves*. El bus implementado para el presente proyecto permite un número máximo de dispositivos de 8 *masters* y 9 *slaves*. En esta topología un *master* inicia un ciclo de bus para un *slave* objetivo y, entonces, el *slave* objetivo participa en uno o más ciclos de bus con el *master*. En la imagen siguiente se muestra un diagrama de bloques.

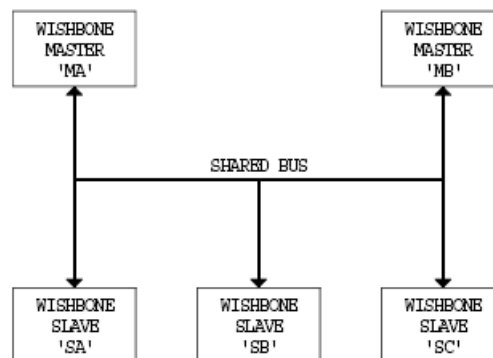


Imagen 13 - Diagrama de bloques de una topología shared bus

Un árbitro, no mostrado en la imagen anterior, determina cuando un *master* gana acceso al bus compartido. El árbitro actúa como un agente de tráfico (también llamado *traffic cop*) para determinar cuando y como cada *master* accede al recurso compartido. Además, el tipo de árbitro es definido completamente por el diseñador del sistema. Por lo tanto, el bus compartido puede estar basado en un sistema de prioridades o en un *round robin*.

La ventaja principal de esta técnica es que los sistemas con interconexión compartida son relativamente compactos. Generalmente, requieren menos puertas lógicas y recursos de rutado que otras configuraciones. Como desventaja destaca el tiempo que el *master* debe esperar antes de ganar acceso al bus compartido. Esto degrada la velocidad media de transmisión de datos de cada *master*.

3.2.2.1 Código fuente

Los ficheros implicados en la definición y parametrización de la topología de bus utilizada en el sistema son: *tc_top.v* (definición del árbitro del bus), *xsv_fpga_top.v*

(implementación del árbitro del bus) y *xsv_fpga_defines.v* (definición de algunos parámetros generales). A continuación se realiza una descripción funcional de los bloques de código de interés:

- Fichero *xsv_fpga_defines.v*:
 - Definición de las constantes que describen la parte del mapa de direcciones de la *fpga* dedicado a los periféricos:

APP_ADDR_DEC_W = 8	APP_ADDR_ETH = 0x92
APP_ADDR_SRAM = 0x00	APP_ADDR_AUDIO = 0x9d
APP_ADDR_FLASH = 0x04	APP_ADDR_UART = 0x90
APP_ADDR_DECP_W = 4	APP_ADDR_PS2 = 0x94
APP_ADDR_PERIP = 0x9	APP_ADDR_RES1 = 0x9e
APP_ADDR_VGA = 0x97	APP_ADDR_RES2 = 0x9f
- Fichero *xsv_fpga_top.v*:
 - Definición del módulo *xsv_fpga_top* {*clk, rstn, sw, uart_stx, uart_srx, sram_Ncs, flash_Ncs, jtag_tck, jtag_tms, jtag_tdi, jtag_tdo, jtag_trst, jtag_tvref, jtag_tgnd*}
 - Definición de la *glue logic* de la arquitectura
 - Instancia de los bloques principales: *debug unit, uart, or1200 y sram controller*
 - Instancia del árbitro del bus (*traffic cop*). Se pasan las constantes definidas en el fichero *xsv_fpga_defines.v* como parámetros de inicialización del bloque *tc_top*. En este trozo de código se realizan las conexiones de los componentes del sistema a los puertos de *master* o *slave* del bus compartido.
- Fichero *tc_top.v*:
 - Definición de parámetros generales del fichero:

TC_AW = 32	TC_TIN_W = 34
TC_DW = 32	TC_IIN_W = 72
TC_BSW = 4	
 - Definición del módulo *tc_top* {*clk, rst, i0..7, t0..8*}
 - Parámetros (inicializados en instancia):

t0_addr_w = 8	t28i_addr_w = 8
t0_addr = 0x00	t2_addr = 0x97
t1_addr_w = 8	t3_addr = 0x92
t1_addr = 0x04	t4_addr = 0x9d
t28c_addr_w = 4	t5_addr = 0x90
t28_addr = 0x9	t6_addr = 0x94
	t7_addr = 0x9e
	t8_addr = 0x9f
 - Declaración de *wires* internos:


```
xi0..7 (dat_o, ack, err)
yi0..7 (dat_o, ack, err)
z (cyc, stb, cab, adr, sel, we, dat_i)
```
 - Lógica OR de las señales (*xi0..7* con *yi0..7*)

- Instancia de $t0_ch$ ($tc_mi_to_st$) $\{clk, rst, i_{0..7}(xi), t_o \rightarrow t_0\}$. Se pasan parámetros de inicialización.
- Instancia de $t18_ch_upper$ ($tc_mi_to_st$) $\{clk, rst, i_{0..7}(yi), t_o \rightarrow z\}$. Se pasan parámetros de inicialización.
- Instancia de $t18_ch_lower$ ($tc_si_to_mt$) $\{i_o(yi), t_{0..7} \rightarrow t_{1..8}\}$. Se pasan parámetros de inicialización.

- Definición del módulo $tc_mi_to_st$

- Parámetros (inicializados en instancia):

	t0_ch	t18_ch upper
t0_addr_w	8	8
t0_addr	0x00	0x04
multitarg	0	1
t17_addr_w	8	4
t17_addr	0x00	0x9

- $i_{0..7_in} = \{i_{0..7}(\text{entradas})\}$
 $\{i_{0..7}(\text{salidas})\} = i_{0..7_out}$
 $\{t_0(\text{salidas})\} = t_0_out$
 $t_0_in = \{t_0(\text{entradas})\}$
- $i_{0..7_out} = t_0_in$ (en función del valor req_won , asigna uno y los demás se ponen a cero)
- $t_0_out = i_{0..7_in}$ (en función del valor req_won , sinó se ponen a cero)
- $req_i[\alpha] = cyc_\alpha \& ((adr_\alpha[31:24] = t_0_addr) \text{ or } (multitarg \& adr_\alpha[31:28] = t17_addr))$
- gestión de req_won :
 si *master* actual no finaliza (req_cont) \rightarrow mantener req_won
 sino \rightarrow chequea $req_i(0..7)$ con prioridad de 0 a 7
 si ningún req_i activado $\rightarrow master0$ por defecto
- gestión de req_cont

- Definición del módulo $tc_si_to_mt$

- Parámetros (inicializados en instancia):

$t0_addr_w = 8$
 $t0_addr = 0x04$
 $t17_addr = 8$
 $t1_addr = 0x97$
 $t2_addr = 0x92$
 $t3_addr = 0x9d$
 $t4_addr = 0x90$
 $t5_addr = 0x94$
 $t6_addr = 0x9e$
 $t7_addr = 0x9f$

- $i_0_in = \{i_0(\text{entradas})\}$

```

{i0(salidas)} = i0_out
{t0..7(salidas)} = t0..7_out
t0..7_in = {t0..7(entradas)}

```

- t_{0..7_out} = i_{0_in} (en función del valor req_t[0..7], asigna uno y los demás se ponen a cero)
- i_{0_out} = t_{0..7_in} (en función del valor req_t[0..7], sinó se ponen a cero)
- req_t[α] = cyc_α & (adr_α[31:24]=t_{α_addr})

3.2.2.2 Diseño

La *Imagen 14* muestra el diagrama esquemático del árbitro del bus. Cuando un *master* desea iniciar un ciclo de bus aserta la señal *wb_cyc_o* e introduce la dirección a la que desea acceder en el bus *wb_adr_o* (además de asertar otras señales necesarias del interface de wishbone). En el primer bloque (*tc_mi_to_st*), el árbitro del bus gestiona en función del valor de las señales *wb_cyc_i* recibidas el acceso al bus con un sistema de prioridades que otorga el nivel máximo de prioridad al *master0* y el mínimo nivel al *master7*. Las señales del interface wishbone del *master* que ha ganado acceso al bus pasan al segundo bloque del módulo *tc_top*. Las señales son introducidas en el interface wishbone del *slave* adecuado en función del valor almacenado en la señal el bus de direcciones *wb_adr_i*. De esta manera se conecta el *master* adecuado con el *slave* adecuado.

Sin embargo, cuando cualquiera de los *masters* pretende acceder al *slave0* el flujo de los datos no sigue el mismo camino. El esquema del árbitro muestra un segundo bloque *tc_mi_to_st* llamado *to_ch*. Este canal posee una cantidad de lógica menor, por lo que el ciclo de bus se realizará a una velocidad superior. Esto permite una velocidad de acceso a memoria mejorada, siempre y cuando el diseñador del sistema conecte el controlador de memoria al interface wishbone del *slave0*.

El proyecto *ORP_SoC* propone la siguiente tabla de conexión *master/slave* al bus Wishbone:

M0	VGA	S0	SRAM Controller
M1	Ethernet	S1	Flash Controller
M2	Audio	S2	VGA
M3	Debug	S3	Ethernet
M4	Risc Data	S4	Audio
M5	Risc Instruction	S5	UART16550
M6	nc	S6	PS/2 Keyboard
M7	nc	S7	nc
		S8	nc

3.2.2.2.1 Bloque *tc_mi_to_st*

Se trata del bloque lógico implementado para resolver los múltiples accesos de los *masters* del sistema a un único canal del bus. El nombre completo del bloque es *traffic cop multiple initiator to single target*. Como se muestra en el esquema anterior, el bus está formado por dos canales de comunicación por lo que se precisa un bloque para cada uno de los canales implementados.

Los puertos de entrada/salida de este bloque lógico son los interfaces *wishbone* de los ocho *initiators* de ciclo de bus del sistema y del único *target* de salida. Los parámetros utilizados para la inicialización del bloque son:

- *t0_addr_w* define la anchura del parámetro *t0_addr*
- *t0_addr* define el mapa de direcciones ocupado por el *slave* en un canal *monotarget*
- *multitarg* especifica la naturaleza *multitarget* del canal
- *t17_addr_w* define la anchura del parámetro *t17_addr*
- *t17_addr* define el mapa de direcciones ocupado por los *slaves* en un canal *multitarget*

A partir de las señales obtenidas en los interfaces *wishbone* de los *initiators* y del *target* en cada instante concreto, se asignan los valores de los subconjuntos de señales de entrada o salida internas al bloque. Se consigue mediante las asignaciones concurrentes de las señales de cada *initiator* (*i0...i7*) y del *target* a los subconjuntos de señales previamente definidos (*io_in*, *io_out*, ..., *i7_in*, *i7_out*, *to_in*, *to_out*):

```
assign io_in = {i0_wb_cyc_i, i0_wb_stb_i, i0_wb_cab_i, i0_wb_adr_i,
               i0_wb_sel_i, i0_wb_we_i, i0_wb_dat_i};
assign {i0_wb_dat_o, i0_wb_ack_o, i0_wb_err_o} = io_out;
...
assign i7_in = {i7_wb_cyc_i, i7_wb_stb_i, i7_wb_cab_i, i7_wb_adr_i,
               i7_wb_sel_i, i7_wb_we_i, i7_wb_dat_i};
assign {i7_wb_dat_o, i7_wb_ack_o, i7_wb_err_o} = i7_out;

assign {t0_wb_cyc_o, t0_wb_stb_o, t0_wb_cab_o, t0_wb_adr_o, t0_wb_sel_o,
        t0_wb_we_o, t0_wb_dat_o} = t0_out;
assign t0_in = {t0_wb_dat_i, t0_wb_ack_i, t0_wb_err_i};
```

La lógica utilizada para determinar qué *masters* pretenden realizar el uso de un canal determinado del bus se muestra en el cuadro de texto siguiente. Sin embargo, la lógica implementada en el bloque para los dos canales del bus es ligeramente diferente.

```
assign req_i[0] = i0_wb_cyc_i & ((i0_wb_adr_i[`TC_AW-1:`TC_AW-t0_addr_w]
== t0_addr) | multitarg & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
t17_addr));
...
assign req_i[7] = i7_wb_cyc_i & ((i7_wb_adr_i[`TC_AW-1:`TC_AW-t0_addr_w]
== t0_addr) | multitarg & (i7_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
t17_addr));
```

En el caso del bloque *t0_ch*, la condición de petición de acceso del bus requiere que la señal de ciclo de bus (*wb_cyc_i*) se encuentre activada y que la señal de direcciones (*wb_addr_i*) especifique una posición dentro del mapa de direcciones ocupado por el *slave 0* del canal.

En el caso del bloque *t17_ch_upper*, la señal *wb_cyc_i* y el parámetro *multitarg* deben encontrarse a nivel alto y la señal *wb_addr_i* debe especificar una posición dentro del mapa de direcciones ocupado por los múltiples *slaves* del canal.

La lógica que determina cuál es el *master* que obtiene acceso al canal del bus se implementa mediante un decodificador de prioridad. Si el *master* con actual control del bus mantiene asertada su señal de petición (*req_cont*), mantiene el control (*req_r*). Sino, se comprueba la señal de petición (*req_i*) de cada *master* empezando por el *initiator 0* (máxima prioridad).

```
assign req_won = req_cont ? req_r :
    req_i[0] ? 3'd0 :
    req_i[1] ? 3'd1 :
    req_i[2] ? 3'd2 :
    req_i[3] ? 3'd3 :
    req_i[4] ? 3'd4 :
    req_i[5] ? 3'd5 :
    req_i[6] ? 3'd6 :
    req_i[7] ? 3'd7 : 3'd0;
```

A partir del valor de la señal interna *req_won*, se procede a la asignación de las señales de entrada/salida del interface *wishbone* del *master* que posee acceso al bus. Para las señales de entrada a los *masters*, se realizan asignaciones concurrentes simples que, en función del valor de *req_won*, actualizan un determinado subconjunto *ix_out* con ceros lógicos o con los datos provenientes del *target* único del bloque. En el caso de las señales de salida de los *masters*, se resuelve mediante un decodificador de prioridad que asigna los datos provenientes de un *master* determinado por la señal *req_won* a las señales de entrada al *slave* único.

```
assign i0_out = (req_won == 3'd0) ? t0_in : {`TC_TIN_W{1'b0}};
...
assign i7_out = (req_won == 3'd7) ? t0_in : {`TC_TIN_W{1'b0}};

assign t0_out = (req_won == 3'd0) ? i0_in :
    (req_won == 3'd1) ? i1_in :
    (req_won == 3'd2) ? i2_in :
    (req_won == 3'd3) ? i3_in :
    (req_won == 3'd4) ? i4_in :
    (req_won == 3'd5) ? i5_in :
    (req_won == 3'd6) ? i6_in :
    (req_won == 3'd7) ? i7_in : {`TC_IIN_W{1'b0}};
```

La lógica implementada para la gestión de las señales internas *req_cont* y *req_r* se resuelve con un decodificador sin prioridad para el caso de *req_cont* y con un *latch* en el caso de *req_r*.

```
always @(req_r or req_i)
    case (req_r) //synopsys parallel_case
        3'd0: req_cont = req_i[0];
        3'd1: req_cont = req_i[1];
        3'd2: req_cont = req_i[2];
        3'd3: req_cont = req_i[3];
        3'd4: req_cont = req_i[4];
        3'd5: req_cont = req_i[5];
        3'd6: req_cont = req_i[6];
        3'd7: req_cont = req_i[7];
    endcase

always @(posedge wb_clk_i or posedge wb_rst_i)
    if (wb_rst_i)
        req_r <= #1 3'd0;
    else
        req_r <= #1 req_won;
    endif
```

La señal *req_cont* especifica si el *master* con actual acceso al canal pretende mantener el control del bus y *req_r* registra qué *master* posee acceso actualmente al bus.

3.2.2.2.2 Bloque *tc_si_to_mt*

Se trata del bloque lógico implementado para resolver el acceso del *master* con control del bus a los múltiples *targets* del sistema por un único canal *multitarget*. El nombre completo del bloque es *traffic cop single initiator to multiple target*.

Los puertos de entrada/salida de este bloque lógico son los interfaces *wishbone* del *initiator* (se trata de un elemento de apoyo interno al *traffic cop*) y de los ocho *targets* conectados al canal *multitarget* del bus.. Los parámetros utilizados para la inicialización del bloque son:

- *t0_addr_w* define la anchura del parámetro *t0_addr*
- *t0_addr* define el mapa de direcciones ocupado por el *slave* 0 en un canal *monotarget*
- *t17_addr_w* define la anchura de los parámetros *t1_addr*, ..., *t7_addr*
- *t1_addr* define el mapa de direcciones ocupado por el *slave* 1 en un canal *multitarget*
- ...
- *t7_addr* define el mapa de direcciones ocupado por el *slave* 7 en un canal *multitarget*

A partir de las señales obtenidas en los interfaces *wishbone* de los *initiators* y del *target* en cada instante concreto, se asignan los valores de los subconjuntos de señales de entrada o salida internas al bloque. Se consigue mediante las asignaciones concurrentes de las señales del *initiator* (*i0*) y de cada *target* (*t0...t7*) a los subconjuntos de señales previamente definidos (*io_in*, *io_out*, *t0_in*, *t0_out*, ..., *t7_in*, *t7_out*).

```
assign i0_in = {i0_wb_cyc_i, i0_wb_stb_i, i0_wb_cab_i, i0_wb_adr_i,
               i0_wb_sel_i, i0_wb_we_i, i0_wb_dat_i};
assign {i0_wb_dat_o, i0_wb_ack_o, i0_wb_err_o} = i0_out;

assign {t0_wb_cyc_o, t0_wb_stb_o, t0_wb_cab_o, t0_wb_adr_o,
        t0_wb_sel_o, t0_wb_we_o, t0_wb_dat_o} = t0_out;
assign t0_in = {t0_wb_dat_i, t0_wb_ack_i, t0_wb_err_i};
...
assign {t7_wb_cyc_o, t7_wb_stb_o, t7_wb_cab_o, t7_wb_adr_o,
        t7_wb_sel_o, t7_wb_we_o, t7_wb_dat_o} = t7_out;
assign t7_in = {t7_wb_dat_i, t7_wb_ack_i, t7_wb_err_i};
```

La lógica que determina qué *target* está siendo accedido en el canal *multitarget* se resuelve a partir de asignaciones concurrentes. La condición de acceso para un *target* específico requiere que la señal de ciclo de bus (*wb_cyc_i*) del *initiator* se encuentre asertada y que la señal de direcciones (*wb_addr_i*) apunte a una posición de memoria que se encuentre dentro del mapa de direcciones del *target* determinado. De esta forma se actualizan los ocho bits de la señal *req_t* que se corresponden con la condición de acceso a los ocho *targets* conectados al bloque *t17_ch_lower*.

```
assign req_t[0] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t0_addr_w]
== t0_addr);
...
assign req_t[7] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w]
== t7_addr);
```

A partir del valor de la señal *req_t*, se procede a la asignación de las señales de entrada/salida al interface *wishbone* del *target* especificado. Para las señales de entrada a los *targets*, se realizan asignaciones concurrentes simples que, en función del valor de *req_t*, actualizan un determinado subconjunto *tx_out* con zeros lógicos o con los datos provenientes del *master* único del bloque. En el caso de las señales de salida de los *targets*, se resuelve mediante un decodificador de prioridad que asigna los datos provenientes de un *target* determinado por la señal *req_t* a las señales de entrada al *master* único.

```
assign t0_out = req_t[0] ? i0_in : {'TC_IIN_W{1'b0}};
...
assign t7_out = req_t[7] ? i0_in : {'TC_IIN_W{1'b0}};

assign i0_out = req_t[0] ? t0_in :
    req_t[1] ? t1_in :
    req_t[2] ? t2_in :
    req_t[3] ? t3_in :
    req_t[4] ? t4_in :
    req_t[5] ? t5_in :
    req_t[6] ? t6_in :
    req_t[7] ? t7_in : {'TC_TIN_W{1'b0}};
```

3.3 Sistema Básico

En la presente sección del proyecto, se pretende implementar un sistema básico (mostrado en la *Imagen 15*) a partir del código fuente del proyecto *ORPSoC* de *OpenCores*. Los objetivos principales de este apartado son:

- modificación de las fuentes del proyecto *ORPSoC* para obtener el sistema básico descrito
- obtención de una cadena de herramientas de *GNU* y generación de los ficheros de formato *elf* del programa *hello-uart*
- comprobación del correcto funcionamiento del sistema mediante la simulación funcional del diseño con la herramienta de simulación *ModelSim*
- comprobación de la característica de sintetizabilidad de los diferentes *cores* que componen el diseño

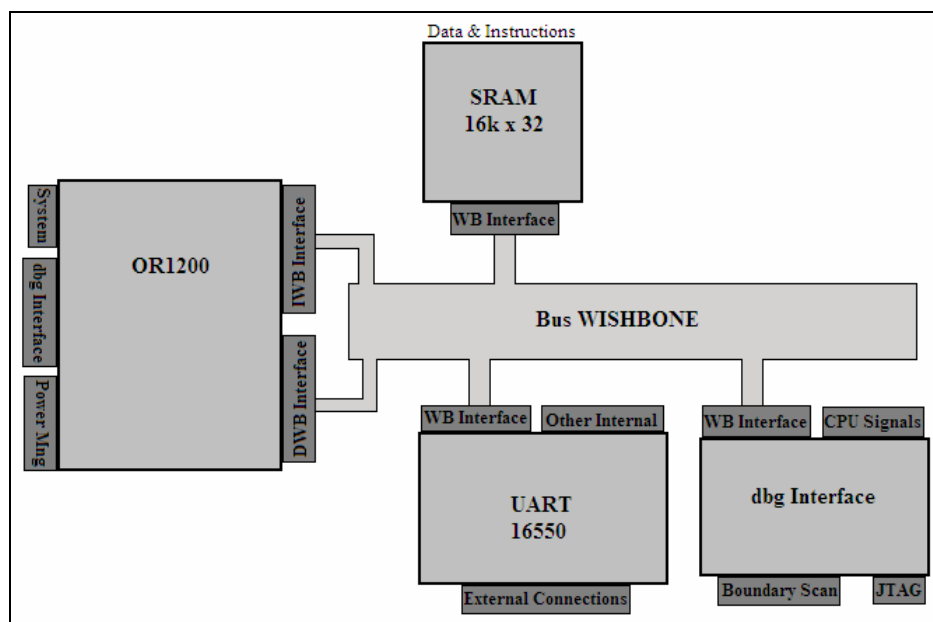


Imagen 15 - Esquema del sistema básico

La configuración de bus *master/slave* que se propone para obtener el sistema básico mostrado en la imagen anterior es:

M0	nc	S0	SRAM Controller Slave
M1	nc	S1	nc
M2	nc	S2	nc
M3	Debug core Master	S3	nc
M4	RISC Data Master	S4	nc
M5	RISC Instruction Master	S5	UART core Slave
M6	nc	S6	nc
M7	nc	S7	nc
		S8	nc

En esta sección se realizan las siguientes tareas:

- **Hardware:** realizar los ajustes necesarios en el código fuente del proyecto *ORP_SoC* para obtener un sistema básico formado por el microcontrolador *openrisc*, la interfaz de comunicación serie *uart16550* y la interfaz de depurado, todos ellos conectados por una implementación del bus *wishbone*.
- **Software:** instalar el sistema operativo *Cygwin* y las herramientas de desarrollo de la *GNU Toolchain*. Compilar y enlazar el programa *hello-uart*.
- **Simulación funcional:** añadir un bloque de memoria interna inicializado con los datos del programa *hello-uart*, simular el sistema y depurar mediante procedimientos de simulación
- **Síntesis:** obtener una memoria interna sintetizable que utilice recursos específicos del dispositivo y comprobar que el sistema obtenido sintetiza correctamente

3.3.1 Hardware

En este apartado se pretenden describir los ajustes de *hardware* realizados para conseguir el sistema básico mostrado en la introducción. El código fuente del sistema del que se parte (proyecto *ORP_SoC*) se encuentra en la página web de la comunidad *OpenCores*. Se accede a la sección *‘or1k/orp/orp_soc’* y se descarga mediante el servidor *CVS*. Se trata de una serie de ficheros y directorios de código libre escritos en lenguaje *verilog*.

3.3.1.1 Suprimir los ficheros innecesarios

En el directorio *‘orp_soc/rtl/verilog/’* se eliminan:

- los siguientes directorios: *audio*, *ethernet*, *ethernet.old*, *or1200.old*, *ps2*, *ps2.old*, *svga* y *uart16550.old*.
- el fichero *‘tdm_slave_if.v’*

De esta forma se eliminan los ficheros *hdl* fuente de los componentes no implementados en el sistema básico. Entre otros, se encuentran las fuentes obsoletas de los cores del microcontrolador *openrisc* y de la *uart*. Los elementos que permanecen en el directorio *‘orp_soc/rtl/verilog/’* son:

- los directorios: *or1200*, *uart16550* y *dbg_interface*.

- los ficheros: *xsv_fpga_top*, *xsv_fpga_defines* y *tc_top*.

Se trata de las fuentes *hdl* de los *cores* del microcontrolador *openrisc*, el controlador de la *uart* para la comunicación serie con el computador, la interfaz de depurado, además del fichero de mayor jerarquía, el fichero de definiciones del sistema y el árbitro del bus.

3.3.1.2 Ajuste del fichero '*xsv_fpga_top.v*'

a) Módulo '*xsv_fpga_top*':

Existen cuatro grupos de señales necesarias entre los puertos de entrada/salida que definimos en el chip de la *fpga*:

- dos señales globales (*clk*, *rst*)
- dos señales de comunicación de la *uart* (*uart_stx*, *uart_srx*)
- siete señales del *jtag* (*jtag_tvref*, *jtag_tgnd*, *jtag_tck*, *jtag_tms*, *jtag_trst*, *jtag_tdi*, *jtag_tdo*)
- señales de chip enable para deshabilitar cualquier circuito integrado no utilizado de la placa *fpga* (*sram_Ncs*, *flash_Ncs*)

b) Lista de entradas/salidas:

Para cada señal declarada en el módulo anterior, se define el sentido de los datos:

- entradas: *clk*, *rstn*, *jtag_tck*, *jtag_tms*, *jtag_tdi*, *jtag_trst*, *uart_srx*
- salidas: *jtag_tvref*, *jtag_tgnd*, *jtag_tdo*, *uart_stx*

c) Lista de cableado interno:

Todo el cableado que no aparezca en la siguiente lista es eliminado. Se trata del cableado que conectaban los *cores* no implementados en el sistema y previamente eliminados :

- *debug core wires*, *debug<->risc wires*
- *RISC instruction & data master wires*
- *UART16550 wires*, *UART external wires*
- *JTAG wires*
- *chip enable wires* (para poder asignar un valor a las señales *chip enable* de la placa)

d) Bloque de asignaciones:

Se suprimen los bloques de código dedicados a las asignaciones de los *cores* no implementados:

- *SRAM tri-state data*, *Ethernet tri-state*, *PS/2 Keyboard tri-state*
- *Risc Instruction address for Flash*
- *Unused Interrupts*
- *Unused Wishbone signals* (excepto por la línea `assign wb_us_err_o = 1'b0;`)

Las líneas de código siguientes se añaden al final del bloque de asignaciones:

```
// External SRAM & FLASH
assign sram_Ncs = 1'b1;
assign flash_Ncs = 1'b1;
// JTAG
assign jtag_tvref = 1'b1;
```

```
assign jtag_tgnd = 1'b0;
```

e) Parte de instancias:

Las instancias de los *cores* no implementados se suprimen del código del fichero:

- *VGA CRT controller, Audio controller*
- *Flash controller, sram_top*
- *Ethernet 10/100 MAC*
- *PS/2 Keyboard controller*
- *CPLD TDM*

El código de las instancias conservadas es ajustado de la siguiente forma:

- *or1200_top*: se sustituye la señal conectada al puerto *clk_i* por la señal *wb_clk*
- *or1200_top*: la señal conectada al puerto *pic_ints* debe ser de la forma *20'b0* (un bus de anchura 20 bits conectados al valor lógico cero)
- *uart_top*: el puerto *int_o* se deja sin conexión (sin interrupciones)
- *tc_top*: se deben desconectar del bus los *cores* no implementados
 - o MASTERS: los *initiators* de wishbone 0, 1 y 2 son reemplazados por desconexiones (de la misma forma en que se encuentran los *initiators* 6 y 7)
 - o SLAVES: los *targets* de wishbone 1, 2, 3, 4 y 6 son reemplazados por desconexiones (de la misma forma en que se encuentran los *targets* 7 y 8)
 - o *.i4_wb_ack_o (wb_rdm_ack)* debe ser *.i4_wb_ack_o (wb_rdm_ack_i)*
 - o *.i5_wb_adr_i (wb_rif_adr)* debe ser *.i5_wb_adr_i (wb_rim_adr_o)*

3.3.1.3 Ajuste del fichero '*or1200_defines.v*'

Algunas definiciones son habilitadas para que el microprocesador trabaje directamente con el *core* de memoria. Existen dos bloques principales de código en el fichero: el primero contenido dentro de la misiva *ifdef OR1200_ASIC* y el segundo contenido en una misiva del tipo *else* (p.e. un dispositivo *fpga*). En el último bloque, las definiciones a habilitar son:

- *OR1200_NO_DC* y *OR1200_NO_IC* (deshabilita la memoria cache de datos e instrucciones)
- *OR1200_NO_DMMU* y *OR1200_NO_IMMU* (deshabilita las unidades de control de memoria de datos e instrucciones)

De la misma manera se deshabilita la definición siguiente (utilizado en un módulo de división de frecuencia de reloj DCM opcional):

- ``define OR1200_CLKDIV_2_SUPPORTED`

3.3.1.4 Ajuste del fichero '*or1200_cpu.v*'

Susituir la línea de código *.genpc_stop_prefetch (genpc_stop_prefetch)* del componente *or1200_genpc* por la línea *.genpc_stop_prefetch(1'b0)*.

3.3.1.5 Ajuste del fichero '*or1200_sprs.v*'

En la definición de comportamiento del *Supervision Register* se sustituye la línea de código

```
sr<=#1{1'b1,`OR1200_SR_EPH_DEF,{`OR1200_SR_WIDTH-3{1'b0}},1'b1};
```

por la línea

```
sr<=#1{1'b1,{`OR1200_SR_WIDTH-2{1'b0}},1'b1};
```

para mapear los vectores de excepción (p.e. reset) del sistema en las primeras posiciones del mapa de memoria.

3.3.2 Software

3.3.2.1 Entorno Cygwin:

Para proceder a la instalación del entorno *Cygwin* para un sistema operativo nativo *Windows* se deben seguir los pasos descritos a continuación:

1. Descargar el *Cygwin Net Release Setup Program* (setup.exe) de la página www.cygwin.com
2. Ejecutar el programa (setup.exe)
3. Seleccionar la opción de instalar desde internet

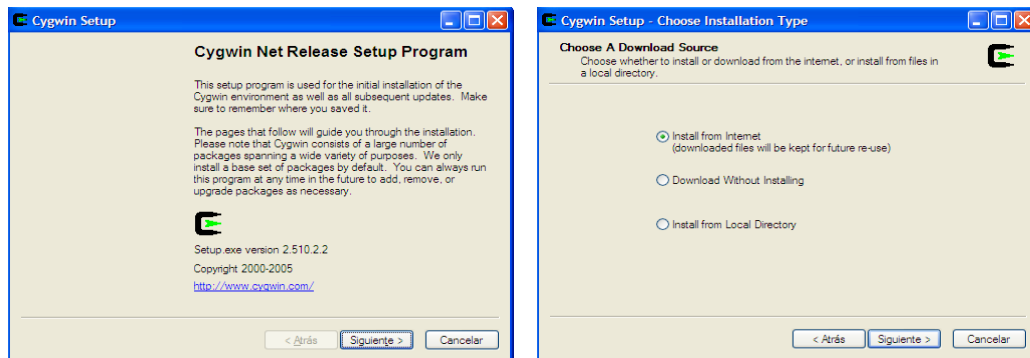


Imagen 16 - Pantallas de la aplicación de instalación de Cygwin

4. Especificar la carpeta de instalación y diversas opciones (instalar para todos los usuarios y Tipo de fichero de texto por defecto: Unix/binario)
5. Seleccionar el directorio donde la aplicación almacenará los ficheros descargados para su futura utilización

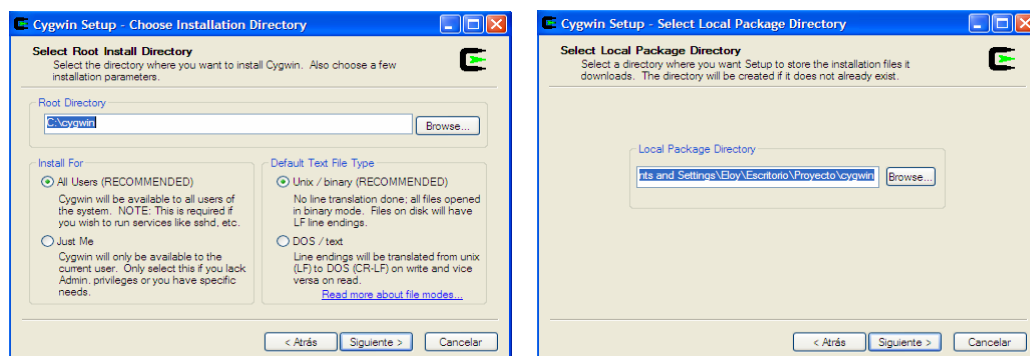


Imagen 17 - Pantallas de la aplicación de instalación de Cygwin

6. Seleccionar el tipo de conexión a internet
7. Seleccionar el *mirror* de una lista (conexión a la ftp)

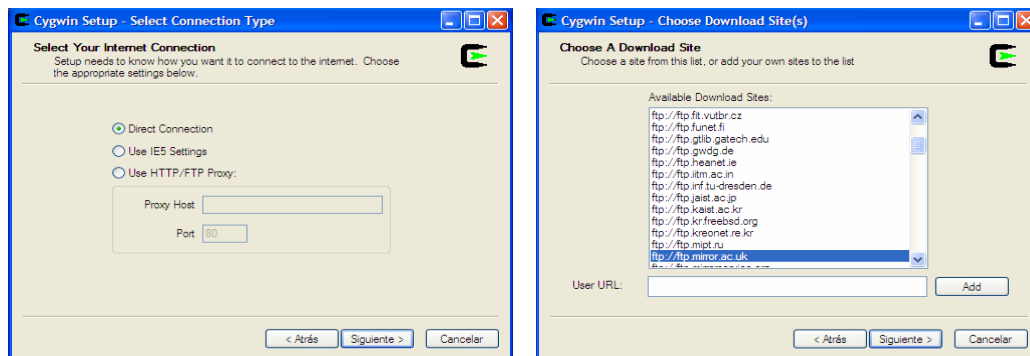


Imagen 18 - Pantallas de la aplicación de instalación de Cygwin

Se descarga una lista de todos los paquetes relacionados con cygwin disponibles.

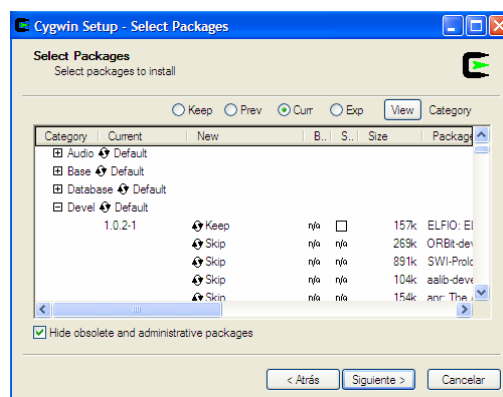


Imagen 19 - Pantalla de la aplicación de instalación de Cygwin

8. Seleccionar la opción *Default* para instalar una colección básica de los paquetes necesarios para el correcto funcionamiento de cygwin. Otros paquetes necesarios son: *make*, *gcc* y *binutils* (herramientas de desarrollo precompiladas para el sistema operativo nativo), *w32api*, *ELFIO* (editor de texto de ficheros *elf*)...

9. Si en algún momento se desea instalar nuevos paquetes o actualizar los previamente instalados, repetir los pasos anteriores y seleccionar los paquetes y/o las versiones de la lista de paquetes.

3.3.2.2 GNU Toolchain

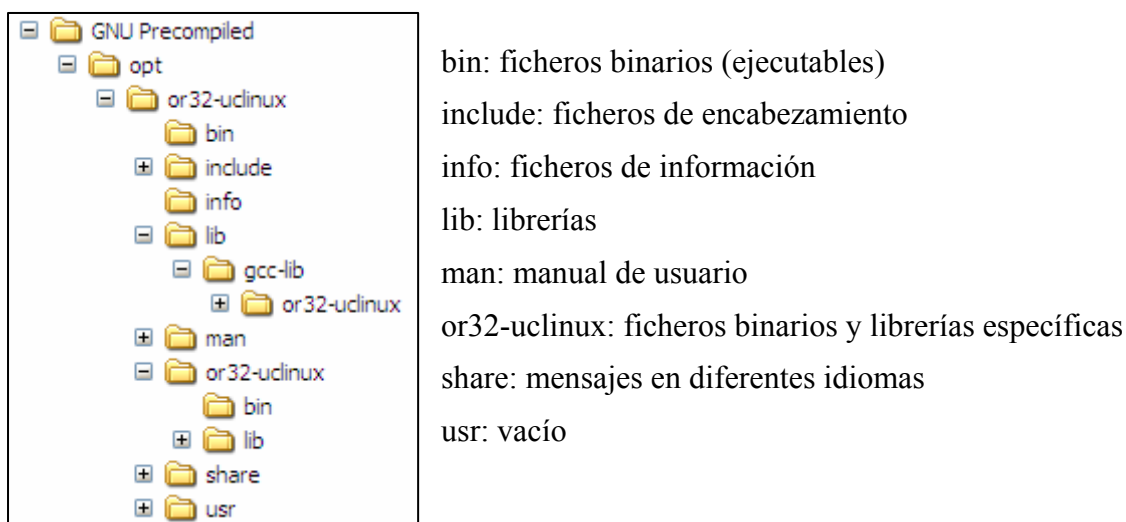
Para obtener nuestra cadena de desarrollo GNU podemos seguir dos procedimientos:

- Descargar e instalar una GNU Toolchain precompilada. Se trata del proceso de instalación más sencillo, aunque no es posible realizar una configuración personalizada de las herramientas. Sin embargo, para procesar los ficheros fuente del programa de prueba *hello-uart* la cadena de herramientas precompiladas es suficiente. Por lo tanto y para estudiar el funcionamiento de las herramientas, se trata del proceso seguido en esta sección del proyecto
- Compilar nuestra propia GNU Toolchain. Se trata de un proceso complicado que implica la configuración y compilación manual de las herramientas de desarrollo. Se encuentra una descripción del proceso en el apartado 3.4.2 del proyecto.

La cadena de desarrollo precompilada se puede descargar de la página www.opencores.org, en el proyecto OR1000, en el apartado de GNU Tools. Se trata de una serie de herramientas cruzadas configuradas para soportar ficheros escritos en lenguaje C y generar ficheros ejecutables para el microprocesador embebido *OpenRISC*. Existen tres opciones de configuración del entorno de destino:

- *uclinux* la aplicación se ejecuta bajo un sistema operativo embebido *uclinux* (*linux* específico para microprocesadores)
- *elf* la aplicación se ejecuta sin sistema operativo embebido (nuestro caso) y ejecuta código nativo
- *rterms* la aplicación se ejecuta bajo un sistema operativo embebido de tiempo real

De todos modos, cualquiera de las opciones descritas generan ficheros ejecutables válidos para diseños sin sistema operativo embebido. Por lo tanto, la cadena de herramientas precompilada es válida para nuestro diseño. A continuación se describe la estructura de directorios de la cadena:



En una consola de *Cygwin*, se accede a la ruta adecuada y se introduce la línea de comandos `tar -xzv filename.tar.gz` para descomprimir el fichero descargado. El resultado se mueve a la ruta '`c:/cygwin/`', quedando la entrada *opt* visible en el directorio. A continuación se edita el fichero '`/etc/bash_rc.bash_rc`' introduciendo la línea siguiente para que cambios sean permanentes:

```
PATH=$PATH:/opt/or32-uclinux/bin
```

De esta forma se añade la nueva ruta al contenido de la variable de sistema *PATH*. Esta variable indica los directorios que el sistema comprueba para encontrar ficheros ejecutables.

3.3.2.3 Programa *hello-uart*

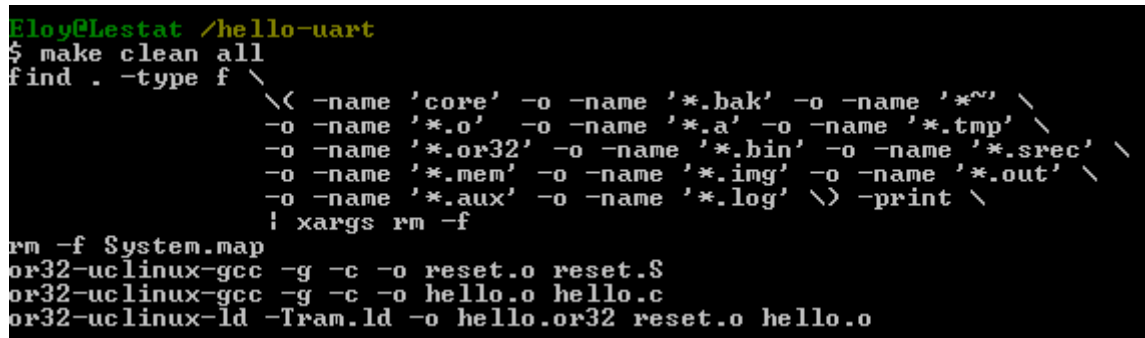
Se trata de un programa simple escrito en lenguaje C y utilizado para testear el funcionamiento del sistema básico. Los ficheros que componen el programa son:

- *board.h* fichero de encabezamiento (definición de parámetros de la placa)

- hello.c fichero fuente del programa principal
- makefile scripts del comando make
- mc.h fichero de encabezamiento (no incluido)
- ram.ld fichero de scripts de enlazado
- reset.S fichero fuente que define la operación de reset
- sim.cfg script de configuración del simulador orlksim
- uart.h fichero de encabezamiento (definición de parámetros de la uart)

Para compilar el proyecto se abre una consola de Cygwin y se ejecutan las siguientes líneas de instrucción:

```
cd c:
cd cygwin
cd hello-uart
make clean all
```



```
Floy@lestat /hello-uart
$ make clean all
find . -type f \
    \< -name 'core' -o -name '*.bak' -o -name '*~' \
    -o -name '*.o' -o -name '*.a' -o -name '*.tmp' \
    -o -name '*.or32' -o -name '*.bin' -o -name '*.srec' \
    -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
    -o -name '*.aux' -o -name '*.log' \> -print \
    ! xargs rm -f
rm -f System.map
or32-uclinux-gcc -g -c -o reset.o reset.S
or32-uclinux-gcc -g -c -o hello.o hello.c
or32-uclinux-ld -Tram.ld -o hello.or32 reset.o hello.o
```

El compilador cruzado (or32-uclinux-gcc) genera los ficheros reset.o y hello.o. Entonces el linkador cruzado (or32-uclinux-ld) genera hello.or32 a partir de los anteriores. Todos ellos son ficheros en formato ELF32 (big endian).

3.3.2.3.1 Fichero Makefile

- inicio fichero -

```
1  ifndef CROSS_COMPILE
2  CROSS_COMPILE = or32-uclinux-
3  CC = $(CROSS_COMPILE)gcc
4  LD = $(CROSS_COMPILE)ld
5  NM = $(CROSS_COMPILE)nm
6  endif
7
8  export      CROSS_COMPILE
9
10 all: hello.or32
11 reset.o: reset.S Makefile
12     $(CC) -g -c -o $@ $< $(CFLAGS)
13
14 hello.o: hello.c Makefile
15     $(CC) -g -c -o $@ $< $(CFLAGS)
16
17 hello.or32: reset.o hello.o Makefile
18     $(LD) -Tram.ld -o $@ reset.o hello.o $(LIBS)
19
20 System.map: hello.or32
21     @$ (NM) $< | \
22     grep -v '\(compiled\)\|\(\.o$$\)\|\( [aUw]
        \)\|\(\.\.ng$$\)\|\(LASH[RL]DI\) ' | \
```

```

23             sort > System.map
24 #####
25 clean:
26 find . -type f \
27     \( -name 'core' -o -name '*.bak' -o -name '*~' \
28     -o -name '*.o' -o -name '*.a' -o -name '*.tmp' \
29     -o -name '*.or32' -o -name '*.bin' -o -name '*.srec' \
30     -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
31     -o -name '*.aux' -o -name '*.log' \) -print \
32     | xargs rm -f
33 rm -f System.map
34
35 distclean: clean
36 find . -type f \
37     \( -name '.depend' -o -name '*.srec' -o -name '*.bin' \
38     -o -name '*.pdf' \) \
39     -print | xargs rm -f
40 rm -f $(OBJJS) *.bak tags TAGS
41 rm -fr *.*~

```

- fin fichero -

Descripción del código (fichero Makefile)

1 a 6: En el caso que la variable de entorno `CROSS_COMPILE` no haya sido definida previamente, se define como `'or32-uclinux-'`. A su vez, se definen las llamadas a los programas como `CC='or32-uclinux-gcc'`, `LD='or32-uclinux-ld'` y `NM='or32-uclinux-nm'`.

Previamente se configura la cadena de desarrollo para trabajar con herramientas cruzadas, es decir, funcionan sobre el sistema operativo nativo (formato *COFF* para *Windows*) pero generan ficheros objeto con el formato para otro sistema operativo destino (formato *ELF*).

8: Permite el uso de la variable `CROSS_COMPILE` a otros ficheros.

10 a 23: Script vinculado al comando `'make all'` con fichero de salida `'hello.or32'`.

11 a 15: `or32-uclinux-gcc -g -c -o reset.o reset.S`
`or32-uclinux-gcc -g -c -o hello.o hello.c`

Ejecuta el compilador cruzado. El argumento final (`$@`) de cada línea de instrucción apunta al nombre del fichero fuente que se pretende compilar.

`-c` Compila o ensambla los ficheros fuente sin linkarlos. La salida tiene la forma de un fichero objeto para cada fichero fuente.

`-g` Produce la información de depurado (*debugging*) en el formato del S.O. nativo (*stabs*, *coff*, *xcoff* o *dwarf*). GDB puede trabajar con esta información.

`-o file` Llama *file* al fichero de salida.

`<` Redirecciona la entrada leyendo de los archivos indicados (en este caso no añade nada)

17 a 18: `or32-uclinux-ld reset.o hello.o -Tram.ld -o hello.or32`

Ejecuta el enlazador cruzado.

- T[file.ld] Incluye un fichero ('ram.ld') de script de linkado
 - o file Llama *file* al fichero de salida
- 20 a 22: Script vinculado al comando 'make System.map'
- ```
or32-uclinux-nm hello.or32 | grep -v '\(expression\)' |
 sort > system.map
```
- Busca los símbolos en el fichero hello.or32 (*nm*). Utiliza los símbolos para buscar un patrón en todos los ficheros indicados por *expression* (*grep*), omitiendo todo lo coincidente. Guarda en system.map una concatenación ordenada de todos los datos de entrada.
- 25 a 33: El comando *find* busca en el directorio actual.
- .Añade los subdirectorios a la búsqueda.
  - type f Especifica un formato cualquiera de los ficheros buscados.
  - \(...\) La expresión indica que el nombre del archivo debe ser de uno de los formatos indicados
  - print Escribe la ruta en la pantalla en el caso de encontrar alguna coincidencia.
- 32 | conecta la salida de find con la entrada de xargs
- Lee argumentos de la entrada estándar (del commando find) y ejecuta el comando siguiente una o tantas veces como sea necesario para no hacer un *overflow* del buffer de líneas de comandos.
- 33 Elimina los ficheros especificados.
- 35 a 41: Líneas de poco interés.

### 3.3.2.3.2 Fichero ram.ld

#### - inicio fichero -

```
MEMORY
{
 vectors : ORIGIN = 0x00000000, LENGTH = 0x00002000
 ram : ORIGIN = 0x00002000, LENGTH = 0x00002000
}

SECTIONS
{
 .vectors :
 {*(.vectors)} > vectors
 .text :
 {*(.text)} > ram
 .data :
 {*(.data)} > ram
 .rodata :
 {*(.rodata)} > ram
 .bss :
 {*(.bss)} > ram
 .stack :
 {*(.stack)}
 _src_addr = .;
} > ram
```

```
}
- fin fichero -
```

#### **Descripción del fichero (ram.ld)**

En el módulo *MEMORY* se generan dos bloques de memoria: vectors y ram.

En el módulo *SECTIONS* se especifica el orden y el bloque de memoria donde se introduce cada sección del fichero ELF ‘hello.or32’.

### **3.3.2.3.3 Código fuente**

#### **3.3.2.3.3.1 Fichero reset.S**

**- inicio fichero -**

```
1 #include "board.h"
2 #include "mc.h"
3
4 .global __main
5 .section .stack, "aw", @nobits
6 .space STACK_SIZE
7 _stack:
8
9 .section .vectors, "ax"
10 .org 0x100
11 _reset:
12 l.movhi r1,hi(_stack-4)
13 l.ori r1,r1,lo(_stack-4)
14 l.addi r2,r0,-3
15 l.and r1,r1,r2
16
17 l.movhi r2,hi(_main)
18 l.ori r2,r2,lo(_main)
19 l.jr r2
20 l.addi r2,r0,0
21
22 __main:
23 l.jr r9
24 l.nop
```

**- fin fichero -**

#### **Descripción del fichero (reset.S)**

Las instrucciones con formato ‘l.xxx’ trabajan con operandos de 32 bits.

- 4      Declara la rutina \_\_main como global para poder ser accedida desde el exterior.
- 5 a 7   Define la sección stack como AW (alloc/write) y ¿?(@nobits)  
         Busca dinámicamente un lugar para la stack con un tamaño de STACK\_SIZE  
         (0x1000 según board.h).  
         En ese lugar coloca la etiqueta \_stack (o punto de llamada)
- 9 a 20   Define la sección vectors como AX (alloc/execute)  
         Sitúa el origen en 0x100 (vector de reset)
- 11      Etiqueta \_reset (rutina reset)
 

|               |                                               |
|---------------|-----------------------------------------------|
| @1 → r1       | @1 = hi(_stack-4)                             |
|               | Almacena los 4 MSB del registro SR (contexto) |
| r1 or @2 → r1 | @2 = lo(_stack-4)                             |

|                                                                                                                |                                                                                        |
|----------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| $r0 - 3 \rightarrow r2$                                                                                        | Almacena los 4 LSB del registro SR (contexto)<br>(máscara: $r2 = 0 - 3 = 0xffffffff$ ) |
| $r1 \text{ and } r2 \rightarrow r1$                                                                            | (aplica máscara a $r1 \rightarrow$ 2 bits LSB son ceros)                               |
| Ha recuperado el último SR (contexto) antes de la excepción y pasa a modo usuario y deshabilita interrupciones |                                                                                        |
| $@_1 \rightarrow r2$                                                                                           | $((@_1 = hi\_main))$<br>Almacena los 4 MSB de la dirección de salto a la rutina main   |
| $r2 \text{ or } @_2 \rightarrow r2$                                                                            | $((@_2 = lo\_main))$<br>Almacena los 4 LSB de la dirección de salto a la rutina main   |
| $r2 \rightarrow PC$                                                                                            | Salta a la dirección de salto de la rutina main                                        |
| $r0 + 0 \rightarrow r2$                                                                                        | Pone a cero el registro ( $r2 = 0 - 0$ )                                               |
| 22 a 24 Etiqueta <code>__main</code> (salto a la rutina <i>main</i> )                                          |                                                                                        |
| $r9 \rightarrow PC$                                                                                            | Salta a la dirección de inicio del programa principal                                  |
| <code>wait</code>                                                                                              | No realiza operación                                                                   |

### 3.3.2.3.3.2 Fichero *hello.c*

#### - inicio fichero -

```
#define BOTH_EMPTY ...
#define WAIT_FOR_XMITR ...
#define WAIT_FOR_THRE ...
#define CHECK_FOR_CHAR ...
#define WAIT_FOR_CHAR ...

void uart_init(void)
{...}
void uart_putc(char c)
{...}
char uart_getc(void)
{...}

char *str = "Hello world!!!\n";
int main (void)
{
 char *s;
 uart_init ();
 for (s = str; *s; s++)
 uart_putc (*s);
 while (1)
 uart_putc (uart_getc () + 1);
 return 0;
}
```

#### - fin fichero -

#### Descripción del fichero (*hello.c*)

El fichero define diversas macros de comportamiento de la uart. Además describe las funciones `'uart_init'`, `'uart_putc'` y `'uart_getc'`. Se trata de las funciones dedicadas a la inicialización del dispositivo de comunicación serie *uart* y a enviar o recibir un carácter por los puertos de comunicación del dispositivo. En la función `main` el programa inicializa la uart, envía todos los caracteres de la cadena `"Hello world!!!\n"` a la uart y

entra en un bucle infinito que espera a recibir un carácter de la uart, incrementa su valor (código ascii) y reenvía el carácter obtenido.

### 3.3.2.3.3.3 Ficheros de encabezamiento

#### a) **board.h**

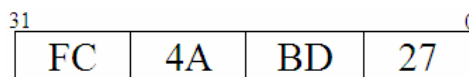
Definición de: - diversos parámetros generales de la placa  
- macros de acceso a registros internos

#### b) **uart.h**

Definición de: - el registro de control del FIFO  
- el registro de control de línea  
- el registro de estado de línea  
- el registro de identificación de interrupciones  
- el registro de permisión de interrupciones  
- el registro de control de módem  
- el registro de estado de módem  
- el registro de parámetros extendidos

### 3.3.2.3.4 Ficheros ELF

En subapartados anteriores se comenta que el compilador cruzado ha generado los ficheros `hello.o` y `reset.o`. El fichero `hello.or32` es el resultado de la operación del enlazador cruzado. Los tres archivos de salida son ficheros *elf* con formato *big-endian*. Esta característica afecta a la distribución de los *bytes* (8 bits) dentro de la *word* (unidad básica de información en el sistema) compuesta por 32 bits.



**Imagen 20** – Ejemplo de word en formato big-endian

El fichero con extensión *or32* contiene los datos que serán volcados sobre la fpga. En el apartado 3.5.7 se trata este tema en profundidad.

Existen diferentes herramientas para visualizar en un formato determinado el contenido de un fichero *elf* (`hello.or32`). Se trata del editor de ficheros de formato *elf* (*readelf*) y del desensamblador de ficheros objeto (*objdump*). Las líneas de comando siguientes deben ser introducidas en una consola de *cygwin*:

```
a) or32-uclinux-readelf -a hello.or32 > program.txt
-a visualiza toda la información disponible
equivalente a: -h visualiza la cabecera del fichero elf
 -l visualiza la cabecera del programa
 -S visualiza la cabecera de las secciones
 -s visualiza la tabla de símbolos
 -r visualiza las relocations
 -d visualiza la sección dinámica
 -V visualiza las secciones de versión
 -A visualiza la información específica de arquitectura
 -I visualiza el histograma de longitudes de lista
> vuelca los datos visualizados en el fichero de texto especificado
```



# ELF Header:

```

Magic: 7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00
Class: ELF32
Data: 2's complement, big endian
Version: 1 (current)
OS/ABI: UNIX - System V
ABI Version: 0
Type: EXEC (Executable file)
Machine: OpenRISC
Version: 0x1
Entry point address: 0x2000
Start of program headers: 52 (bytes into file)
Start of section headers: 11148 (bytes into file)
Flags: 0x0
Size of this header: 52 (bytes)
Size of program headers: 32 (bytes)
Number of program headers: 2
Size of section headers: 40 (bytes)
Number of section headers: 17
Section header string table index: 14

```

## Section Headers:

| [Nr] | Name         | Type     | Addr     | Off    | Size   | ES | Flg | Lk | Inf | Al |
|------|--------------|----------|----------|--------|--------|----|-----|----|-----|----|
| [ 0] |              | NULL     | 00000000 | 000000 | 000000 | 00 |     | 0  | 0   | 0  |
| [ 1] | .vectors     | PROGBITS | 00000000 | 001000 | 000128 | 00 | AX  | 0  | 0   | 1  |
| [ 2] | .rel.vectors | REL      | 00000000 | 0030c0 | 000000 | 08 |     | 15 | 1   | 4  |
| [ 3] | .text        | PROGBITS | 00002000 | 002000 | 00029c | 00 | AX  | 0  | 0   | 4  |
| [ 4] | .rel.text    | REL      | 00000000 | 0030c0 | 000000 | 08 |     | 15 | 3   | 4  |
| [ 5] | .data        | PROGBITS | 0000229c | 00229c | 000004 | 00 | WA  | 0  | 0   | 4  |
| [ 6] | .rel.data    | REL      | 00000000 | 0030c0 | 000000 | 08 |     | 15 | 5   | 4  |
| [ 7] | .rodata      | PROGBITS | 000022a0 | 0022a0 | 000010 | 00 | A   | 0  | 0   | 1  |
| [ 8] | .bss         | NOBITS   | 000022b0 | 0022b0 | 000000 | 00 | WA  | 0  | 0   | 1  |
| [ 9] | .stack       | NOBITS   | 000022b0 | 0022b0 | 001000 | 00 | WA  | 0  | 0   | 1  |
| [10] | .stab        | PROGBITS | 00000000 | 0022b0 | 0003f0 | 0c |     | 12 | 0   | 4  |
| [11] | .rel.stab    | REL      | 00000000 | 0030c0 | 000000 | 08 |     | 15 | a   | 4  |
| [12] | .stabstr     | STRTAB   | 000003f0 | 0026a0 | 000457 | 00 |     | 0  | 0   | 1  |
| [13] | .comment     | PROGBITS | 00000847 | 002af7 | 000028 | 00 |     | 0  | 0   | 1  |
| [14] | .shstrtab    | STRTAB   | 00000000 | 002b1f | 00006c | 00 |     | 0  | 0   | 1  |
| [15] | .symtab      | SYMTAB   | 00000000 | 002e34 | 000200 | 10 |     | 16 | 19  | 4  |
| [16] | .strtab      | STRTAB   | 00000000 | 003034 | 00008b | 00 |     | 0  | 0   | 1  |

## Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)  
 I (info), L (link order), G (group), x (unknown)  
 0 (extra OS processing required) o (OS specific), p (processor specific)

## Program Headers:

| Type | Offset   | VirtAddr   | PhysAddr   | FileSiz | MemSiz  | Flg | Align  |
|------|----------|------------|------------|---------|---------|-----|--------|
| LOAD | 0x001000 | 0x00000000 | 0x00000000 | 0x00128 | 0x00128 | R E | 0x1000 |
| LOAD | 0x002000 | 0x00002000 | 0x00002000 | 0x002b0 | 0x012b0 | RWE | 0x1000 |

## Section to Segment mapping:

| Segment | Sections...                |
|---------|----------------------------|
| 00      | .vectors                   |
| 01      | .text .data .rodata .stack |

There is no dynamic segment in this file.

There are no relocations in this file.

There are no unwind sections in this file.

## Symbol table '.symtab' contains 32 entries:

| Num: | Value    | Size | Type   | Bind  | Vis     | Ndx | Name    |
|------|----------|------|--------|-------|---------|-----|---------|
| 0:   | 00000000 | 0    | NOTYPE | LOCAL | DEFAULT | UND |         |
| 1:   | 00000000 | 0    | FILE   | LOCAL | DEFAULT | ABS | reset.S |
| 2:   | 00000000 | 0    | FILE   | LOCAL | DEFAULT | ABS | mc.h    |
| 3:   | 00000000 | 0    | FILE   | LOCAL | DEFAULT | ABS | reset.S |

```

4: 00000000 0 FILE LOCAL DEFAULT ABS board.h
5: 00000000 0 FILE LOCAL DEFAULT ABS reset.S
6: 00000000 0 FILE LOCAL DEFAULT ABS <command line>
7: 00000000 0 FILE LOCAL DEFAULT ABS <built-in>
8: 00000000 0 FILE LOCAL DEFAULT ABS reset.S
9: 00002000 0 SECTION LOCAL DEFAULT 3
10: 0000229c 0 SECTION LOCAL DEFAULT 5
11: 000022b0 0 SECTION LOCAL DEFAULT 8
12: 000022b0 0 SECTION LOCAL DEFAULT 9
13: 000032b0 0 NOTYPE LOCAL DEFAULT 9 _stack
14: 00000000 0 SECTION LOCAL DEFAULT 1
15: 00000100 0 NOTYPE LOCAL DEFAULT 1 _reset
16: 00000000 0 FILE LOCAL DEFAULT ABS hello.c
17: 00002000 0 SECTION LOCAL DEFAULT 3
18: 0000229c 0 SECTION LOCAL DEFAULT 5
19: 000022b0 0 SECTION LOCAL DEFAULT 8
20: 000022a0 0 SECTION LOCAL DEFAULT 7
21: 0000229c 0 NOTYPE LOCAL DEFAULT 3 letext
22: 00000000 0 SECTION LOCAL DEFAULT 10
23: 000003f0 0 SECTION LOCAL DEFAULT 12
24: 00000847 0 SECTION LOCAL DEFAULT 13
25: 000032b0 0 NOTYPE GLOBAL DEFAULT 9 _src_addr
26: 000020bc 216 FUNC GLOBAL DEFAULT 3 _uart_putc
27: 00000120 0 NOTYPE GLOBAL DEFAULT 1 __main
28: 0000229c 4 OBJECT GLOBAL DEFAULT 5 _str
29: 00002000 188 FUNC GLOBAL DEFAULT 3 _uart_init
30: 00002194 96 FUNC GLOBAL DEFAULT 3 _uart_getc
31: 000021f4 168 FUNC GLOBAL DEFAULT 3 _main

```

No version information found in this file.

- b) `or32-uclinux-objdump -d -S hello.or32 > disassemble.txt`
- d visualiza los datos desensamblados de las secciones ejecutables
  - S combina la visualización de código fuente con desensamblado
  - > vuelca los datos visualizados en el fichero de texto especificado

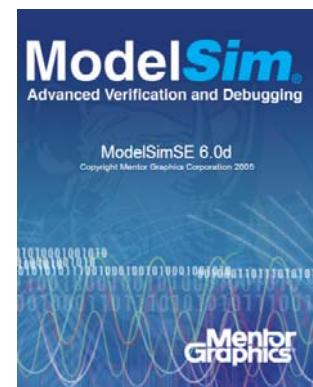
El fichero de texto de salida está dividido en secciones, y cada sección subdividida en funciones. Las diferentes informaciones de los datos se distribuyen en columnas, que contienen:

- Dirección en formato hexadecimal donde está mapeada la instrucción de código máquina
- Codificación hexadecimal de los 32 bits que forman la instrucción
- Instrucción en código máquina desensamblada

### 3.3.3 Simulación Funcional

#### 3.3.3.1 Software de simulación

Para simular el comportamiento del sistema en los diferentes puntos del proceso de diseño, se utiliza el programa ModelSim SE PLUS 6.0d de MentorGraphics. Sin embargo, para que el simulador sea capaz de compilar los ficheros verilog del bloque de memoria interno (apartado 3.3.3.3) generados con la herramienta CORE Generator de Xilinx, se precisa la instalación de la librería XilinxCoreLib\_ver. Con este objetivo se introduce la siguiente línea de código en la ventana *Transcript* del simulador: `"compplib -s mti_se -f all:u:s:c:m -l all -o xilinx_lib"`.



`complib`      compilar librería de xilinx  
`-s mti_se`      para el simulador ModelSim SE  
`-fall:u:s:c:m` para todas las arquitecturas (familia Virtex, familia Spartan...) y selecciona las librerías *unisim*, *simprim*, *xilinxcorelib* y *smartmodel*  
`-l all`          para los lenguajes *vhdl* y *verilog*  
`-o xilinx_lib`    especifica el directorio de salida

### 3.3.3.2 Creación del proyecto de simulación

Se crea un nuevo proyecto accediendo a la opción *File* → *New* → *Project* del menú principal del programa. La imagen lateral muestra los valores especificados para cada campo. Se requiere introducir el nombre y la ruta del proyecto y la librería de trabajo utilizada por defecto. Para añadir los ficheros fuente del diseño HDL al proyecto, se selecciona en el menú flotante de la ventana *Workspace* la opción *Add to Project* → *Existing File...* El programa muestra una ventana de diálogo donde se especifica el nombre y la ruta del fichero fuente, además de activar la opción *reference from current location*. Una vez añadidos todas las referencias a los ficheros fuente del proyecto, la ventana *workspace* queda de la siguiente manera:

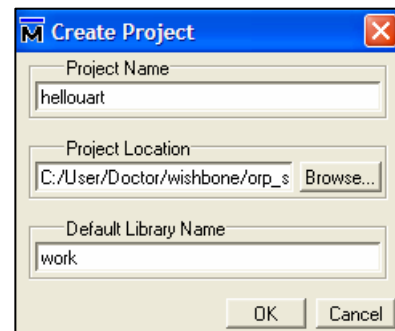


Imagen 21 - Creación del proyecto

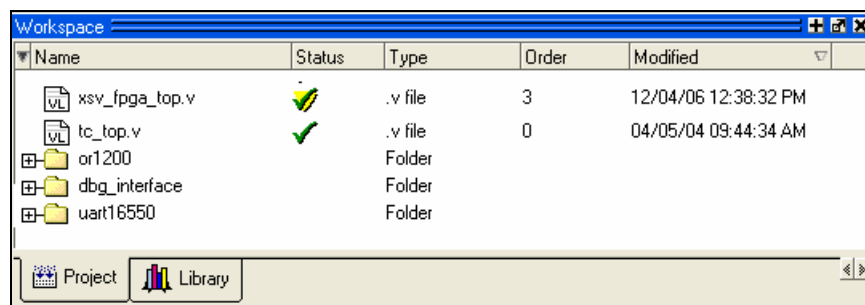


Imagen 22 - Ventana de workspace

La Imagen 22 muestra los ficheros de nivel superior del proyecto (*xsv\_fpga\_top.v* y *tc\_top.v*) que corresponden a la definición del módulo básico de la fpga e instancia de sus módulos principales, y a la definición del módulo *TrafficCop* (árbitro del bus). Los tres directorios contienen los ficheros fuente de los bloques principales: el microprocesador (*or1200*), la interfaz del depurador (*dbg\_interface*) y la interfaz de la *uart* del PC (*uart16550*).

El siguiente paso es modificar las propiedades para añadir algunas rutas de acceso a los ficheros incluye necesarias para la simulación. Para ello se agrega la ruta adecuada en el menú de propiedades de los ficheros correspondientes para que el compilador encuentre el fichero 'timescale.v' y los ficheros de definición. Además, en el fichero 'xsv\_fpga\_top.v' se editan las propiedades añadiendo las siguientes instrucciones:

```

-y C:/Modeltech_6.0d/examples/xilinx_lib
+incdir+C:/User/Doctor/wishbone/orp_soc_tutorial/bench/verilog
+incdir+C:/User/Doctor/wishbone/orp_soc_tutorial/rtl/verilog

```

### 3.3.3.3 Memoria ROM interna

Con el objetivo de simplificar el diseño y posterior inicialización de la memoria, se decide utilizar una memoria ROM de 32 bits específica para la simulación. Para ello se utiliza la herramienta *Coregen* de *Xilinx*.

#### 3.3.3.3.1 Creación del proyecto

Para crear un nuevo proyecto de *Coregen* se selecciona la opción *New Project* y se especifica el nombre y la ruta principal del proyecto en el diálogo. Entonces la herramienta muestra el control de pestañas de la imagen siguiente:

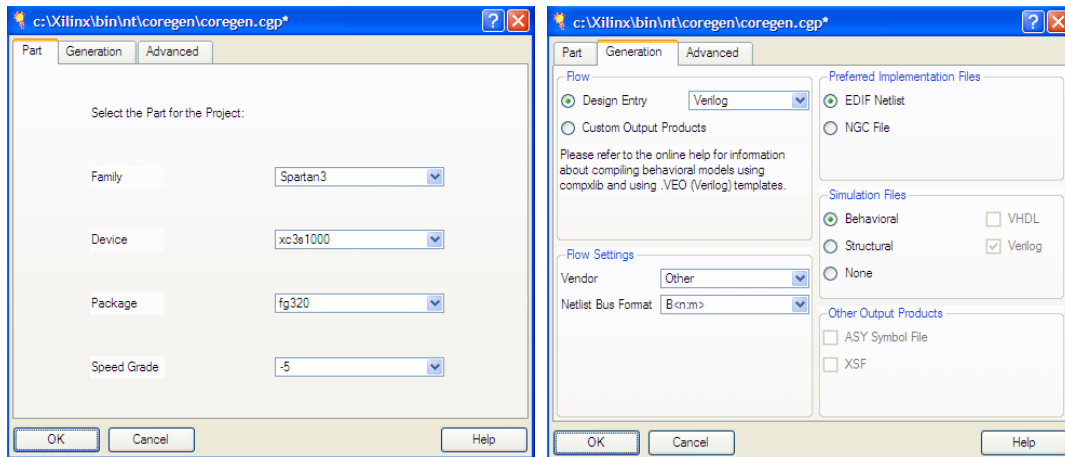
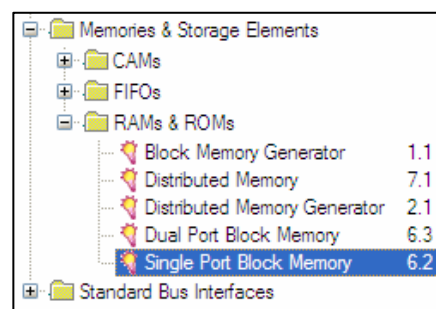


Imagen 23 - Controles de creación de proyecto con la herramienta *CoreGen*

En la pestaña *Part* se especifica la arquitectura de destino del diseño (familia, dispositivo, encapsulado y grado de velocidad). La pestaña *Generation* permite especificar los parámetros utilizados para generar los ficheros fuente del *core* (en este caso los importantes son: diseño en lenguaje *verilog* y descripción de comportamiento). La tercera pestaña no tiene opciones relevantes para nuestro objetivo.

#### 3.3.3.3.2 Generación del core

Para generar un bloque de memoria ROM se acciona el elemento mostrado en la imagen. El programa muestra un control de cuatro pestañas llamado *Single Port Block Memory*, que permite especificar las características de nuestro bloque de memoria. En este caso, la única pestaña de interés para el proyecto es la llamada *Parameters*, que permite navegar por cuatro ventanas para caracterizar el *core*.



Los parámetros destacables por su interés son:

- Configuración de los puertos: solo lectura
- Tamaño de la memoria:
  - Anchura = 32
  - Profundidad = 16384 (16Kword = 16 x 4KB)
- Activo en el flanco de reloj ascendente
- Contenido inicial: cargar fichero de inicialización

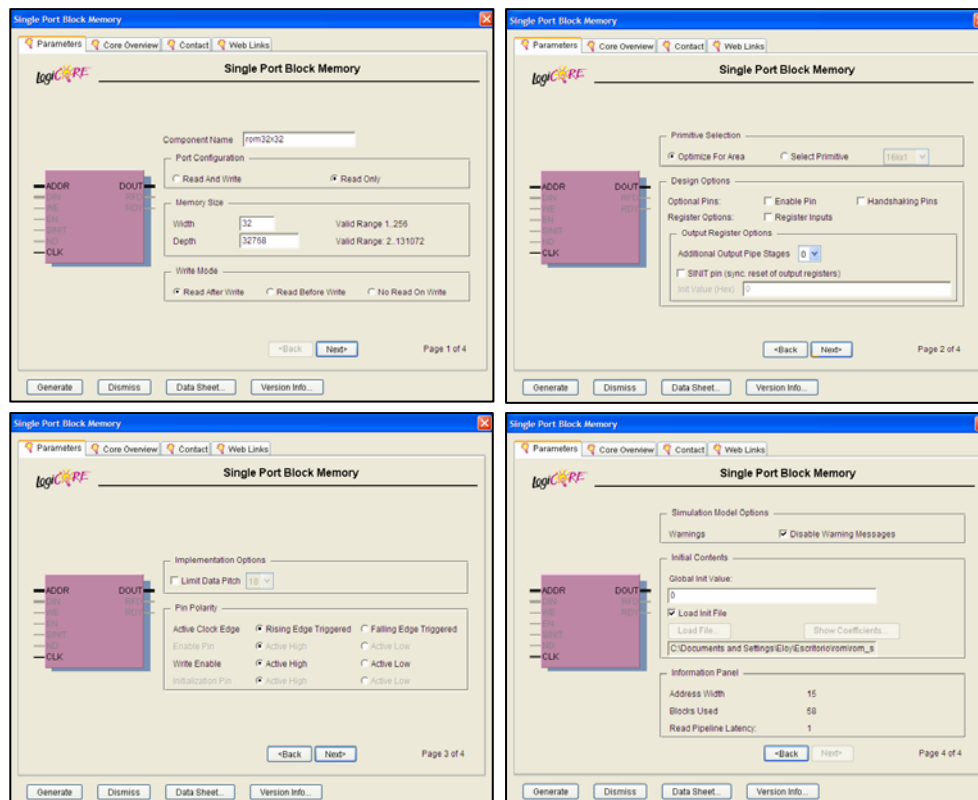


Imagen 24 - Controles de generación de *cores* personalizables con la herramienta *CoreGen*

### 3.3.3.3.3 Fichero COE

Se trata del fichero utilizado para la inicialización de la memoria del sistema. El programa *CORE Generator* de *Xilinx* trabaja con este tipo de ficheros. Para obtener un archivo con formato *coe* se precisa de una serie de conversiones del archivo de formato *elf* (*hello.or32*) obtenido tras el compilado y enlazado del programa *hello-uart*. La conversión se describe en la fórmula:

or32 → bin → hex → coe

Para lograr la conversión se ejecutan las siguientes líneas de comandos en una consola de *Cygwin* en el directorio que contiene los ficheros fuente del programa *hello*:

```
or32-uclinux-objcopy -O binary hello.or32 hello.bin
bin2hex hello.bin > hello.hex
hex2coe hello.hex hello.coe
```

De esta forma se obtienen los ficheros de formato binario, hexadecimal y *coe* correctos (en este orden). Sin embargo, se deben compilar previamente las aplicaciones *bin2hex* y *hex2coe*. Se trata de programas escritos en lenguaje C.

#### 3.3.3.3.3.1 Aplicaciones de conversión

Para compilar las aplicaciones de conversión se introducen las siguientes líneas de comandos en una consola de *Cygwin* (y posteriormente se copian los ficheros *.exe* en el directorio de ejecutables de *Cygwin* “*c:/cygwin/bin*”):

```
gcc bin2hex.c -o bin2hex.exe
gcc hex2coe.c -o hex2coe.exe
```

#### a) Fichero *bin2hex.c*

### **- inicio fichero -**

```
#include <stdio.h>
#include <stdlib.h>
#define BREAK 4 //Number of bytes before line is broken
int main(int argc, char **argv)
{
 FILE *fd;
 int c;
 int i = 0;
 if(argc < 2)
 {
 fprintf(stderr,"no input file specified\n");
 exit(1);
 }
 if(argc > 2)
 {
 fprintf(stderr,"too many input files (more than one)
specified\n");
 exit(1);
 }
 fd = fopen(argv[1], "r");
 if (fd == NULL)
 {
 fprintf(stderr,"failed to open input file: %s\n",argv[1]);
 exit(1);
 }
 while ((c = fgetc(fd)) != EOF)
 {
 printf("%.2x", (unsigned int) c);
 if (++i == BREAK)
 {
 printf("\n");
 i = 0;
 }
 }
 return 0;
}
```

### **- fin fichero -**

#### **Descripción del fichero**

Se trata del programa utilizado para la conversión de los datos en formato binario de un fichero determinado al formato hexadecimal. En el código, se define la constante `BREAK = 4` ya que se pretende distribuir los datos en palabras hexadecimales de 32 bits. La rutina *main* abre el fichero de entrada para su lectura y entra en un bucle que en cada iteración convierte un carácter del fichero de entrada a un entero sin signo en formato de dos dígitos hexadecimales. Cuando completa una línea de 32 bits, añade un retorno de carro. El resultado de la conversión lo muestra en cada iteración por la salida *standard output*.

#### **b) Fichero hex2coe.c**

### **- inicio fichero -**

```
#include <stdio.h>
#define NHEX 8

int main(int argc, char *argv[])
{
 FILE *fdin, *fdout;
```

```

char temp,tempaux;
int readed;
if((fdin=fopen(argv[1],"r"))==NULL)
{
 printf("Error opening input file\n");
 return(-1);
}
if((fdout=fopen(argv[2],"w"))==NULL)
{
 printf("Error opening output file\n");
 return(0);
}
fprintf(fdout,"memory_initialization_radix=16;\n");
fprintf(fdout,"memory_initialization_vector=\n");
readed=0;
temp=fgetc(fdin);
while(1)
{
 if(temp!='\n' && temp!=EOF)
 {
 fputc(temp,fdout);
 }
 if(temp=='\n')
 {
 readed++;
 }
 if(readed==NHEX)
 {
 temp=fgetc(fdin);
 if(temp=='\n')
 {
 tempaux=fgetc(fdin);
 if(tempaux==EOF)
 {
 fputc(';',fdout);
 fputc('\n',fdout);
 fclose(fdin);
 fclose(fdout);
 return 0;
 }
 else
 {
 fputc(',',fdout);
 temp=tempaux;
 }
 }
 fputc('\n',fdout);
 readed=0;
 }
 else
 temp=fgetc(fdin);
}
}

```

**- fin fichero -**

### ***Descripción del fichero***

Se trata del programa utilizado para la conversión de los datos en formato hexadecimal de un fichero determinado al formato *coe* utilizado por la herramienta *CoreGen* para inicializar bloques de memoria. En el código, se define la constante `NHEX` = 8 (número de cifras hexadecimales en cada línea). La rutina *main* primero abre los

ficheros de entrada y salida para lectura y escritura respectivamente. Luego escribe dos líneas en el inicio del fichero de salida para darle el formato *coe* deseado. Entra en un bucle que en cada iteración toma un carácter del fichero de entrada y lo copia en el fichero de salida, añadiendo una coma cada  $N_{HEX}$  caracteres y añade un carácter ‘;’ y un retorno de carro al final del fichero.

### 3.3.3.3.2 Formato del fichero COE

Para describir de forma simplificada el formato de un fichero *coe* se muestra el código genérico siguiente (se trata de un fichero de inicialización de una memoria con una anchura de 32 bits):

```
memory_initialization_radix=16;
memory_initialization_vector=
XXXXXXXXXX,
XXXXXXXXXX,
XXXXXXXXXX,
XXXXXXXXXX,
...
...;
```

### 3.3.3.3.3 Comprobación del contenido de la rom

Se comprueba la información mapeada en la memoria *rom* editando el fichero ‘*rom32x32.mif*’ (uno de los ficheros generados por la herramienta *Coregen*). El formato de la información contenida en el archivo es una matriz binaria formada por 2221 palabras de 32 bits de anchura. La información está distribuida de la forma siguiente:

| @mem<br>(byte) | @rom<br>(word) | Datos   |
|----------------|----------------|---------|
| 0x0            | 0x0            | -       |
| ...            | ...            | -       |
| 0x100          | 0x40           | reset.S |
| ...            | ...            |         |
| 0x109          | 0x49           |         |
| ...            | ...            | -       |
| 0x2000         | 0x800          | hello.c |
| ...            | ...            |         |
| 0x2221         | 0x1021         |         |

Los datos desensamblados del bloque *reset.S* (0x100 a 0x109) son:

| @                                               | 32bits                              | inst. teórica            | inst. ensamblada       |
|-------------------------------------------------|-------------------------------------|--------------------------|------------------------|
| Descripción bits                                |                                     |                          | resultado              |
|                                                 |                                     |                          |                        |
| 0x40                                            | 00011000 00100000 00000000 00000000 | l.movhi r1,hi(_stack-4)  | l.movhi r1,0           |
| 31..26: inst / 25..21: D / 15..0: k             |                                     |                          | $r1 \leftarrow 0$      |
| 0x41                                            | 10101000 00100001 00110010 10101100 | l.ori r1,r1,lo(_stack-4) | l.ori r1,r1,12972      |
| 31..26: inst / 25..21: D / 21..16: A / 15..0: k |                                     |                          | $r1 \leftarrow r1   k$ |



|                                                 |                                     |                       |                  |
|-------------------------------------------------|-------------------------------------|-----------------------|------------------|
| <b>0x42</b>                                     | 10011100 01000000 11111111 11111101 | l.addi r2,r0,-3       | l.addi r2,r0,-3  |
| 31..26: inst / 25..21: D / 21..16: A / 15..0: k |                                     |                       | r2 ← r0-3        |
| <b>0x43</b>                                     | 11100000 00100001 00010000 00000011 | l.and r1,r1,r2        | l.and r1,r1,r2   |
| 31..26: inst / 25..21: D / 15..0: k             |                                     |                       | r1 ← r1&&r2      |
| <b>0x44</b>                                     | 00011000 01000000 00000000 00000000 | l.movhi r2,hi(_main)  | l.movhi r2,0     |
| 31..26: inst / 25..21: D / 15..0: k             |                                     |                       | r2 ← 0           |
| <b>0x45</b>                                     | 10101000 01000010 00100001 11110100 | l.ori r2,r2,lo(_main) | l.ori r2,r2,8692 |
| 31..26: inst / 25..21: D / 21..16: A / 15..0: k |                                     |                       | r2 ← r2   k      |
| <b>0x46</b>                                     | 01000100 00000000 00010000 00000000 | l.jr r2               | l.jr r2          |
| 31..26: inst / 15..11: D                        |                                     |                       | PC ← r2          |
| <b>0x47</b>                                     | 10011100 01000000 00000000 00000000 | l.addi r2,r0,0        | l.addi r2,r0,0   |
| 31..26: inst / 25..21: D / 21..16: A / 15..0: k |                                     |                       | r2 ← r0+0        |
| <b>0x48</b>                                     | 01000100 00000000 01001000 00000000 | l.jr r9               | l.jr r9          |
| 31..26: inst / 15..11: D                        |                                     |                       | PC ← r9          |
| <b>0x49</b>                                     | 00010101 00000000 00000000 00000000 | nop                   | nop              |
| 31..24: inst                                    |                                     |                       | wait             |

Los datos desensamblados son correctos. Coinciden con las instrucciones contenidas en el fichero fuente *reset.S* del programa *hello-uart*.

### 3.3.3.3.4 Controlador de memoria

Se requiere la integración de un controlador de memoria para conectar el bloque de memoria *rom* generado por la herramienta *Coregen* al bus *Wishbone*. Este módulo debe ser capaz de adaptar las señales provenientes del bus al formato de las señales que necesita el bloque de memoria (en este caso: un bus de datos de 32 bits y un bus de direcciones de 15 bits). Además, el controlador debe gestionar las señales de control que pertenecen al bus *Wishbone* y operar de forma adecuada.

Se puede descargar un controlador de memoria para bloques *rom* generados con *Coregen* de la página web de *OpenCores*. Se trata de un fichero escrito en lenguaje *verilog* que describe el comportamiento de dicho controlador. La autoría se atribuye a Javier Castillo (miembro perteneciente a la comunidad *OpenCores*) y se distribuye como software de código abierto.

#### - inicio fichero -

```

////////////////////////////////////
//// ROM Controller for Coregen ROM block ////
//// ////
//// Description ////
//// Manages access from Wishbone to internal ROM ////
//// Author(s): ////
//// - Javier Castillo, javier.castillo@urjc.es ////
////////////////////////////////////
//// Copyright (C) 2004 OpenCores ////
//// ... ////
////////////////////////////////////
// synopsys translate_off
`include "timescale.v"
// synopsys translate_on

```

```

module wb_rom_controller(clk,reset,
 wb_stb_i,wb_dat_o,wb_dat_i,
 wb_ack_o,wb_adr_i,wb_we_i,
 wb_cyc_i,wb_sel_i,
 address,data);

input clk;
input reset;
input wb_stb_i;
output [31:0] wb_dat_o;
input [31:0] wb_dat_i;
output wb_ack_o;
input [31:0] wb_adr_i;
input wb_we_i;
input wb_cyc_i;
input [3:0] wb_sel_i;
output [14:0] address;
input [31:0] data;

reg [31:0] wb_dat_o;
reg wb_ack_o;
reg [14:0] address;
reg next_reading;
reg reading;

//read_data:
always @(posedge clk or posedge reset)
begin
 if(reset==1)
 begin
 wb_ack_o<=#1 1'b0;
 wb_ack_o<=#1 1'b0;
 wb_dat_o<=#1 1'b0;
 end
 else
 begin
 wb_dat_o <= #1 1'b0;
 wb_ack_o <= #1 1'b0;
 if (reading)
 begin
 wb_ack_o <= #1 1'b1;
 wb_dat_o <= #1 data;
 end
 end
end

reg [31:0] addr_var;
always @(wb_adr_i or wb_stb_i or wb_we_i or wb_cyc_i
 or reading or wb_ack_o)
begin
 next_reading = 1'b0;
 addr_var = wb_adr_i >> 2;
 address = addr_var[14:0];

 if(~reading && ~wb_ack_o)
 begin
 if (wb_cyc_i && wb_stb_i && !wb_we_i)
 begin
 addr_var = wb_adr_i >> 2;
 address = addr_var[14:0];
 next_reading = 1'b1;
 end
 end
end

```

```

 end
 end
 if(reading)
 next_reading=1'b0;
 end

//register_proc:
always @(posedge clk or posedge reset)
begin
 if (reset)
 begin
 reading <= #1 1'b0;
 end
 else
 begin
 reading <= #1 next_reading;
 end
 end
end
endmodule

```

**- fin fichero -**

### **Descripción del fichero**

En primer lugar, se declara el módulo *wb\_rom\_controller* y se define el formato de sus puertos. A continuación, se describe la operación de lectura de datos, la actualización de las señales y el procesamiento de los registros internos.

#### **3.3.3.3.5 Conexión del controlador al bus Wishbone**

La conexión del controlador de memoria *rom* al bus *Wishbone* se consigue con la edición del fichero *xsv\_fpga\_top.v* añadiendo los siguientes bloques de código:

- Cableado para la conexión del controlador al *slave 0* del árbitro del bus (*traffic cop*).

```

// simulation ROM controller slave i/f wires
wire [31:0] wb_srom_dat_i;
wire [31:0] wb_srom_dat_o;
wire [31:0] wb_srom_adr_i;
wire [3:0] wb_srom_sel_i;
wire wb_srom_we_i;
wire wb_srom_cyc_i;
wire wb_srom_stb_i;
wire wb_srom_ack_o;

```

- Instancia del controlador de memoria rom

```

// Instantiation of the simulation ROM controller
wb_rom_controller wb_rom_controller(
 // WISHBONE common
 .clk (wb_clk),
 .reset (wb_rst),

 // WISHBONE slave
 .wb_dat_i (wb_srom_dat_i),
 .wb_dat_o (wb_srom_dat_o),
 .wb_adr_i (wb_srom_adr_i),
 .wb_sel_i (wb_srom_sel_i),
 .wb_we_i (wb_srom_we_i),
 .wb_cyc_i (wb_srom_cyc_i),
 .wb_stb_i (wb_srom_stb_i),
 .wb_ack_o (wb_srom_ack_o));

```

- Conexión del controlador al *traffic cop* mediante los cables definidos en el primer punto.

```
// Instantation of the Traffic Cop
// WISHBONE Initiators 0..7
...
// WISHBONE Target 0 (Simulation ROM Controller)
.t0_wb_cyc_o (wb_srom_cyc_i),
.t0_wb_stb_o (wb_srom_stb_i),
.t0_wb_cab_o (wb_srom_cab_i),
.t0_wb_adr_o (wb_srom_adr_i),
.t0_wb_sel_o (wb_srom_sel_i),
.t0_wb_we_o (wb_srom_we_i),
.t0_wb_dat_o (wb_srom_dat_i),
.t0_wb_dat_i (wb_srom_dat_o),
.t0_wb_ack_i (wb_srom_ack_o),
.t0_wb_err_i (1'b0),
// WISHBONE Targets 1..7
...
```

#### 3.3.3.3.6 Añadir el bloque de memoria al proyecto de simulación

En la ventana de proyecto de *ModelSim* se selecciona la opción *Add to project* → *Existing file* del menú emergente del *mouse*. Se añaden al proyecto las fuentes ‘rom32x32.v’ y ‘rc203\_romcontroller.v’. Posteriormente, se editan las propiedades del fichero ‘rc203\_romcontroller’ en *ProjectCompilerSettings* → *Verilog* → *IncludeDirectory* y se le añade la línea de código “+incdir+../bench/verilog”.

Además, se añade el fichero *rom32x32.mif* al directorio del proyecto de simulación: “/modelsim\_hellouart”. De esta forma el programa de simulación podrá acceder al fichero de inicialización de la memoria *rom*.

#### 3.3.3.4 Fichero testbench

Se define *test bench* como un entorno virtual utilizado para verificar la corrección y la validez de un diseño. Toda simulación precisa de un fichero de este tipo para estimular las entradas del sistema y posibilitar el acceso a los puntos de medición adecuados. Este fichero se añade al proyecto de simulación de la forma habitual. Se ha tratado de maximizar la sencillez del fichero simplificando la operación del *test bench*:

- Instancia del módulo de nivel superior
- Generación de la señal de reloj
- Reset durante los 100 ns iniciales

```
module sim_test1;
reg clk, Nrst;
reg[2:1] sw;
wire sram_Ncs, flash_Ncs;
xsv_fpga_top chip_fpga (
 .clk(clk),
 .rstn(Nrst),
 .sw(sw),
 .sram_Ncs(sram_Ncs),
 .flash_Ncs(flash_Ncs));
initial begin
 clk=1'b0;
 sw=2'b00;
 Nrst=1'b0;
 #100 Nrst=1'b1;
end
always begin
 #10 clk=!clk;
```

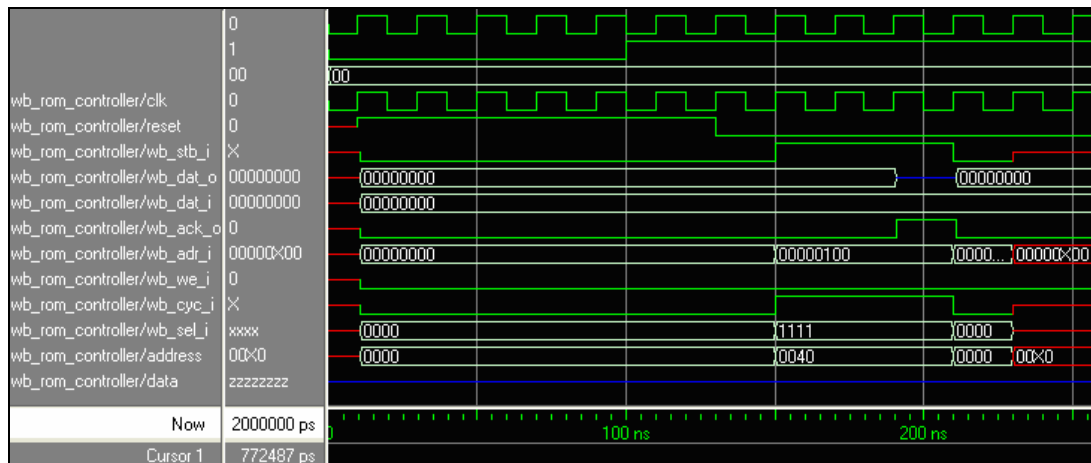
#### 3.3.3.5 Compilación del proyecto

Para compilar el proyecto de simulación, simplemente se selecciona la opción *Compile* → *Compile All* del menú emergente del *mouse*. En este caso particular, el proyecto está formado íntegramente por ficheros en lenguaje *verilog*, por lo que no se debe tener en cuenta el orden de compilación. Sin embargo, en apartados posteriores se

sustituyen algunos ficheros por ficheros fuente en lenguaje *vhdl*. El orden de compilación debe ser respetado entonces.

### 3.3.3.6 Simulación funcional con memoria ROM

En el menú principal de ModelSim se selecciona la opción *Simulate* → *Start Simulation*. En la pestaña *Libraries* se añade la ruta '*C:/Modeltech\_6.0d/examples/xilinx\_lib/XilinxCoreLib\_ver*' para especificar la utilización de la librería de *CoreGen*. Entonces, en la pestaña *Design* se selecciona el fichero *sim\_test1* de la librería *work*. El resultado de la simulación en la ventana *Waveform* es:



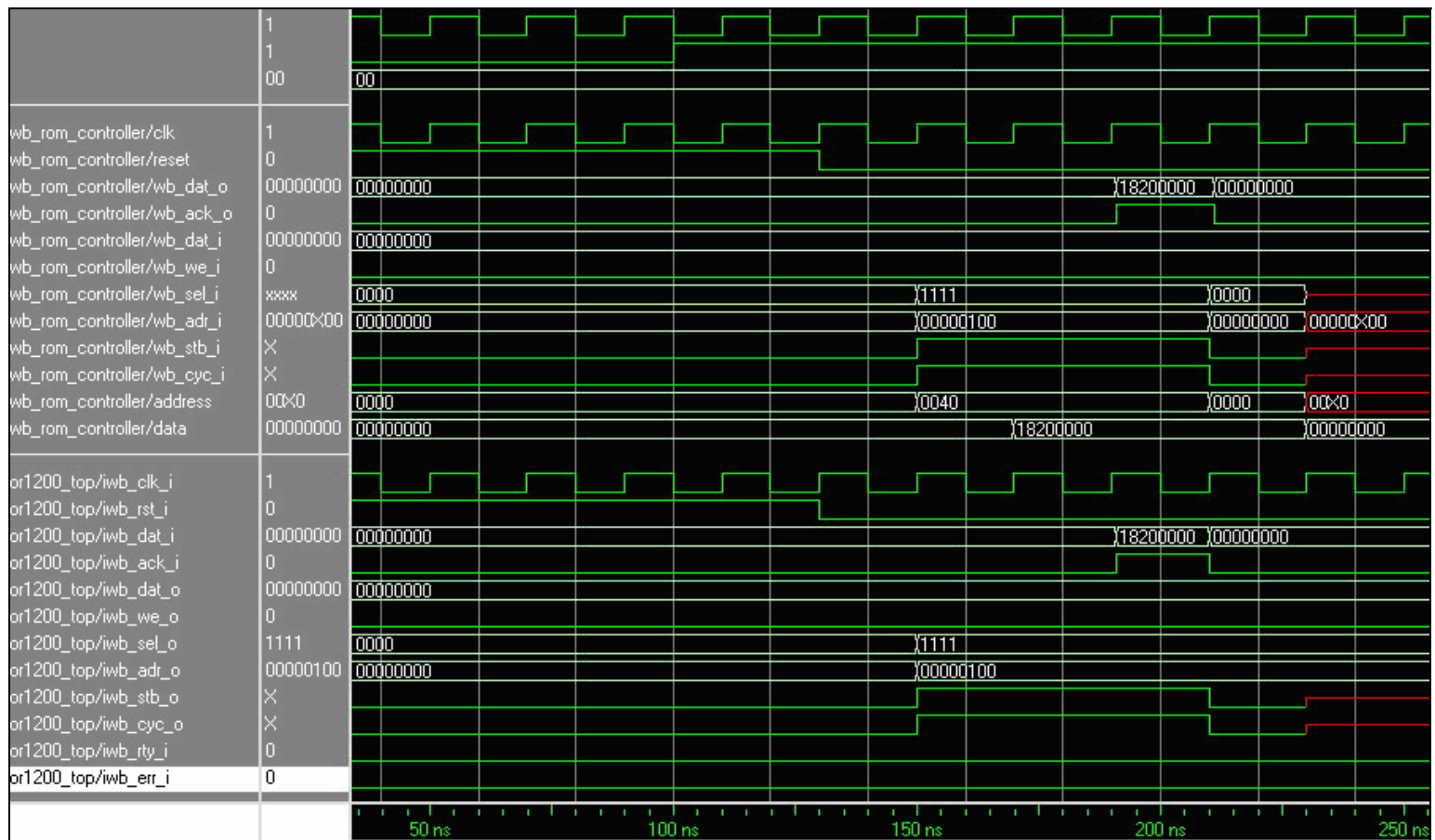
**Imagen 25** - Simulación. El bus data permanece en alta impedancia

Se observa en la simulación anterior que el bus de datos de salida de la memoria rom (*data*) mantiene su valor en '*zzzzzzzz*' (alta impedancia). Esto puede significar que el bloque de memoria no está conectado de forma correcta al controlador. Sin embargo se aprecia que el bus de direcciones de 15 bits de entrada al bloque de memoria (*address*) se actualiza correctamente con la dirección de memoria del bus de direcciones del Wishbone (*wb\_adr\_i*) cuando la señal *wb\_stb\_i* es asertada. Este hecho no descarta la posibilidad de una conexión incorrecta ya que el bus *address* es una señal actualizada por el controlador de memoria, no la memoria rom.

**Solución:** Realizar la instancia del bloque de memoria. Se añaden las líneas de código mostradas en el cuadro lateral al final del fichero *rc203\_romcontroller.v*, justo antes de la línea *end module*.

```
rom32x32 simu_rom (
 .addr(address),
 .clk(clk),
 .dout(data));
```

El resultado de la simulación tras recompilar el proyecto es el mostrado en la página siguiente:

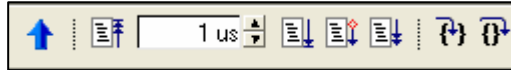


**Imagen 26 - Simulación. Aparecen señales con valor desconocido**

Tras el reset, el uP hace el *fetch* de la primera instrucción ubicada en la dirección 0x100 (word 0x40), el bus se queda en espera y en el tiempo de simulación 230ns el sistema falla. Se observa un comportamiento incorrecto ya que el bus de direcciones *address* adquiere el valor 0x00X0 (en lugar de acceder a la dirección de la instrucción siguiente, es decir, @word = 0x41).

Ante la dificultad de depuración visual del error mediante simple inspección del código o simulación, se trata de encontrar la fuente del error mediante diversos métodos de depurado:

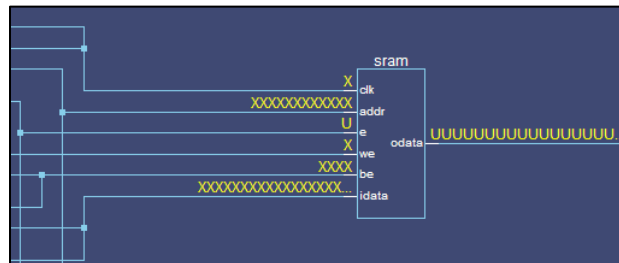
- Simulación paso a paso: se activan puntos de ruptura (*breakpoints*) en líneas clave del código, y se simula paso a paso (*step*) en las zonas de programa críticas. No se ha obtenido información relevante sobre la raíz del problema.



### Imagen 27 - Controles de simulación

- Tracing de un unknown: la ventana de depurado *Dataflow* permite rastrear un valor desconocido X (*unknown*) y observar su propagación por el diseño. Esta ventana está enlazada a la ventana *Waveform*, así que se puede simular el diseño y aislar el problema. Las herramientas principales para el *rastreo* son: *TraceX* (da un paso en el rastreo hacia la fuente) y *ChaseX* (realiza el *rastreo* completo y muestra la fuente del problema). Los pasos seguidos para el *rastreo* son:

- Abrir ventana *Waveform* y simular 1us
- Situar una *marca* en el instante de tiempo en que la señal *address* adquiere un valor desconocido
- Seleccionar en el menú *Sim* del *workspace* el bloque *wb\_romcontroller*
- Abrir la ventana *Dataflow* y seleccionar la opción *Navigate* → *Add Region* para cargar el esquema lógico del módulo seleccionado (*wb\_romcontroller*)



**Imagen 28 - Zoom ampliado de esquema lógico**

- Localizar el bloque y las señales con valor desconocido (*always 107*)
- Seleccionar la señal *address* y accionar el botón *TraceX*
- Repetir último paso hasta encontrar la señal que origina el problema

El camino seguido durante el *rastreo* es:

|                                                                                                                                           |                                                                                          |                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|-------------------------------------------------------|
| <b>memoria rom</b><br>address $\leftarrow$                                                                                                | <b>controlador rom</b><br>wb_addr_i $\leftarrow$ {wb_cyc_i,wb_stb_i} $\leftarrow$        | <b>traffic cop</b><br>target0 $\leftarrow$ initiator5 |
| <b>risc insn master</b><br>iwb_biu $\leftarrow$ icbiu_cyc_o $\leftarrow$                                                                  | <b>cpu</b><br>icpu_cycstb_i $\leftarrow$ genpc_freeze $\leftarrow$ du_stall $\leftarrow$ |                                                       |
| <b>debugger</b><br>dbg_stall_i $\leftarrow$ risc_stall_o $\leftarrow$ sync_out $\leftarrow$ set1_q2 $\leftarrow$ reset2 $\leftarrow$ trst |                                                                                          |                                                       |

La fuente del valor desconocido nace en la señal *trst*. Se trata de una señal de entrada al módulo *debugger* que proviene del exterior mediante el cable *jtag*. Es decir, el fichero *test bench* del proyecto de simulación está incompleto. Ahora se añaden las entradas externas de los módulos *uart* y *debugger* y se inicializan a un valor válido. Resulta lógico que si queda alguna señal externa sin conexión tome el valor de alta impedancia y provoque un funcionamiento no deseado del sistema.

**Solución:** editar el fichero de *test bench* de manera que se tengan en cuenta todas las entradas externas del sistema en la inicialización. El fichero *sim\_test1.v* resultante es:

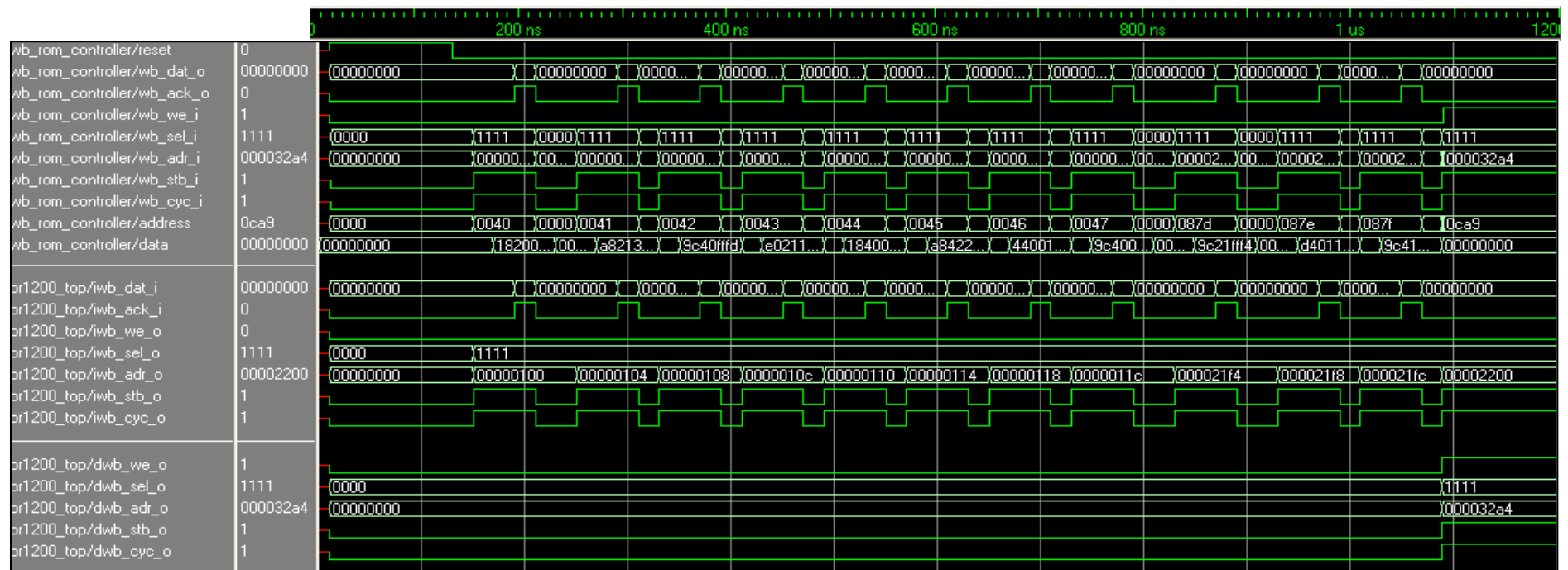
```
module sim_test1;
reg clk, Nrst;
reg[2:1] sw;
wire uart_srx;
wire uart_stx;
wire sram_Ncs, flash_Ncs;
wire jtag_tck, jtag_tms, jtag_tdi, jtag_trst;
wire jtag_tvref, jtag_tgnd, jtag_tdo;

assign uart_stx=1'b0;
assign {jtag_tck,jtag_tms,jtag_tdi,jtag_trst}='b0;
xsv_fpga_top chip_fpga (
 .clk(clk),
 .rstn(Nrst),
 .sw(sw),
 .uart_srx(uart_srx),
 .uart_stx(uart_stx),
 .sram_Ncs(sram_Ncs),
 .flash_Ncs(flash_Ncs),
 .jtag_tck(jtag_tck),
 .jtag_tms(jtag_tms),
 .jtag_tdi(jtag_tdi),
 .jtag_trst(jtag_trst),
 .jtag_tvref(jtag_tvref),
 .jtag_tgnd(jtag_tgnd),
 .jtag_tdo(jtag_tdo));
initial begin
 clk=1'b0;
 sw=2'b00;
 Nrst=1'b0;
 #100 Nrst=1'b1;
end
always
begin
 #10 clk=!clk;
end
endmodule
```

Se ha añadido cableado para las señales externas de los módulos *uart* (*uart\_srx* y *uart\_stx*) y *debugger* (*jtag\_tck*, *jtag\_tms*, *jtag\_tdi*, *jtag\_trst*, *jtag\_tvref*, *jtag\_tgnd* y *jtag\_tdo*). Además se han inicializado las entradas y se han añadido todas las señales externas a la instancia del módulo *xsv\_fpga\_top*.

El resultado de la simulación tras recompilar el proyecto es el mostrado en la página siguiente:





**Imagen 29** - El bus de datos Wishbone trata de acceder a la memoria rom para escritura de datos. Operación no posible

\*En la página siguiente se muestra un detalle ampliado de la simulación. Ésta permite comprobar que los accesos a memoria son correctos (consultar apartado de operación de Wishbone)

Tras el reset general, el *uP* accede a la dirección de memoria 0x100 por el bus de instrucciones *Wishbone*. El puntero de programa (*PC*) se incrementa para hacer un fetch de las instrucción siguiente en cada ciclo de bus. A los 830ns, el *PC* hace un salto al programa principal y el bus de instrucciones accede a la posición de memoria 0x21F4. La simulación continua correctamente. Sin embargo, a los 1090ns, el bus de datos de *Wishbone* inicia una operación de escritura en memoria. La simulación se bloquea debido a que nuestra memoria es una rom (*read only memory*).

**Solución:** con la memoria rom prediseñada por *Xilinx* se ha conseguido que el sistema opere correctamente. El siguiente paso es diseñar e integrar una memoria *ram* (de lectura y escritura) al sistema.

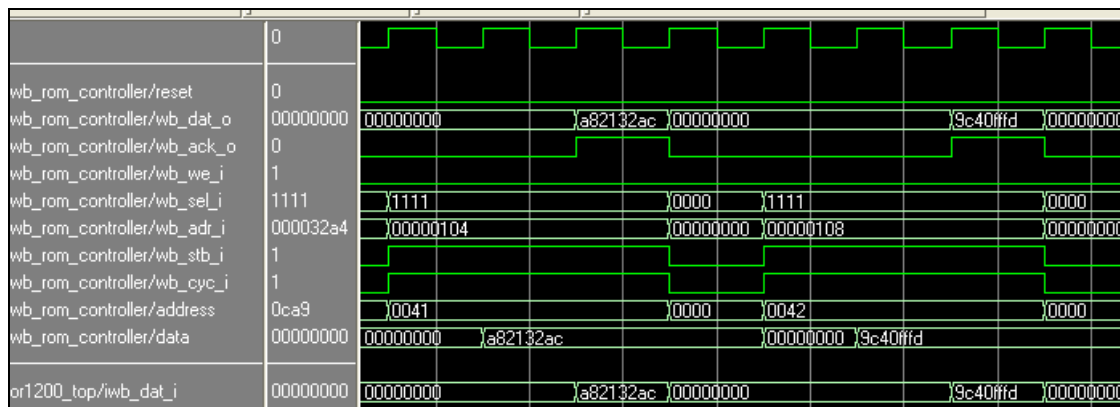


Imagen 30- Detalle ampliado de la simulación (\*)

### 3.3.3.7 Memoria RAM interna

Uno de los objetivos principales es conseguir un sistema de código abierto. Por esta razón, se pretende prescindir de las librerías de código de *Xilinx* y de la herramienta *Coregen* para la generación del *core*. Por lo tanto, la idea es diseñar un bloque de memoria *ram*, es decir, escribir el código en un lenguaje *HDL*. Debido a la formación académica recibida, los ficheros fuente de el presente *core* han sido escritos en código *vhdl*.

#### 3.3.3.7.1 Concepto

La idea es crear un bloque de memoria de lectura y escritura compatible con el bus *Wishbone* tratando de utilizar un código simple y rápido de ejecutar. El esquema lógico es, aproximadamente, el mostrado en la figura siguiente. El *core* consta de cuatro bloques de memoria con una anchura de 8 bits y profundidad programable (según la anchura del bus de direcciones), conectados a un controlador de memoria compatible con el bus *Wishbone*.

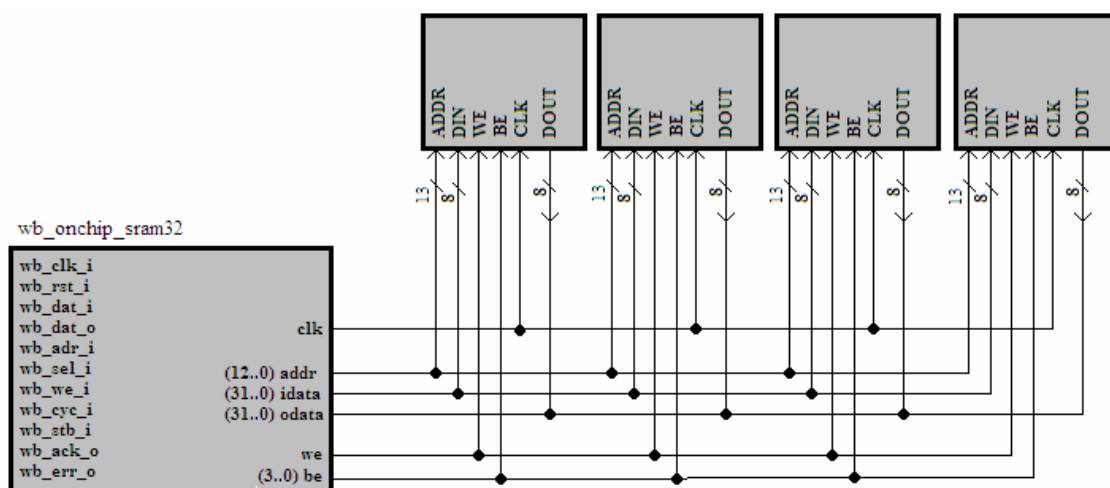


Imagen 31 - Esquema lógico del bloque de memoria ram

Para definir el comportamiento del *core* se han escrito tres ficheros en código *vhdl*:

- *pack\_bin\_hellouart* paquete que contiene la declaración e inicialización de la constante (*array*) que almacena los datos iniciales de la memoria.
- *pack\_sram32* paquete que contiene la definición de los tipos de datos relacionados con la memoria y la definición de la función de inicialización de la *ram*.
- *wb\_onchip\_sram32* fichero principal del *core*. Define las entidades *sram\_N32* (bloque de memoria) y *wb\_onchip\_sram* (controlador).

### 3.3.3.7.2 Código fuente

A continuación se muestran las líneas de código de los ficheros mencionados y se hace una breve descripción del comportamiento.

#### a) Fichero *pack\_sram32.vhd*

##### - inicio fichero -

```
library ieee;
use ieee.std_logic_1164.all;

package pack_sram32 is
 constant BIG_ENDIAN: integer:=1;
 constant ADDR_WIDTH: integer:=15;
 constant NWORDS: integer:=(2**(ADDR_WIDTH-2));
 type T_SRAM_Nx8 is array(0 to NWORDS-1) of std_logic_vector(7 downto 0);
 type T_SRAM_Nx32 is array(0 to 3) of T_SRAM_Nx8;
 constant BINIMG_U: T_SRAM_Nx32:=(others=>(others=>(others=>'U')));
 constant BINIMG_0: T_SRAM_Nx32:=(others=>(others=>(others=>'0')));
 constant BINIMG_1: T_SRAM_Nx32:=(others=>(others=>(others=>'1')));
 type T_RAWDATA is array(natural range<>) of std_logic_vector(31 downto 0);
 function F_BINIMG(rawdata: T_RAWDATA) return T_SRAM_Nx32;
end package;
```

```
package body pack_sram32 is
 function F_BINIMG(rawdata: T_RAWDATA) return T_SRAM_Nx32 is
 variable k: integer;
 variable word32: std_logic_vector(31 downto 0);
 variable binimg: T_SRAM_Nx32:=BINIMG_U;
 begin
 for k in rawdata'range loop
 word32:=rawdata(k);
 binimg(0)(k):=word32(7 downto 0);
 binimg(1)(k):=word32(15 downto 8);
 binimg(2)(k):=word32(23 downto 16);
 binimg(3)(k):=word32(31 downto 24);
 end loop;
 return binimg;
 end function;
 end package body;
```

##### - fin fichero -

##### - Descripción de fichero -

En el paquete *pack\_sram32* se definen las constantes que determinan la distribución de los bytes de información dentro de la *word* (*big endian*), la anchura del bus de direcciones (15 bits) y la capacidad máxima de *words* en la memoria ( $2^{15-2}$ ). Además se definen los tipos de datos que representan un bloque de memoria ram de 8

bits de anchura ( $T\_SRAM\_Nx8$ ) y un bloque de memoria ram de 32 bits de anchura ( $T\_SRAM\_Nx32$ ) formado por cuatro de los sub-bloques anteriores. Con esto, se definen tres constantes que permitirán al usuario inicializar la memoria con una matriz uniforme de valores 'U', 'I' o '0'. El último tipo de datos definido es  $T\_RAWDATA$ , utilizado para almacenar la información binaria en formato hexadecimal antes de convertirla a un tipo de dato de  $T\_SRAM\_Nx32$ .

En el cuerpo del paquete se define la función de inicialización  $F\_BINIMG$ . El programa entra en un bucle que recorre todas las posiciones de la variable *rawdata*, guarda la palabra apuntada en *word32* y almacena los datos de forma ordenada en el formato de la variable de salida *binimg*.

## b) Fichero `wb_onchip_sram32`

### - inicio fichero -

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_sram32.all;
use work.pack_bin_hello_uart.all;

entity sram_Nx32 is
 port(
 clk: in std_logic;
 e: in std_logic;
 addr: in std_logic_vector(ADDR_WIDTH-1-2 downto 0);
 we: in std_logic;
 be: in std_logic_vector(3 downto 0);
 idata: in std_logic_vector(31 downto 0);
 odata: out std_logic_vector(31 downto 0));
end entity;

architecture beh1 of sram_Nx32 is
 constant BINIMG_SRAM: T_SRAM_Nx32:=F_BINIMG(BIN_HELLO_UART);
begin
 sram: process
 variable idx: integer range 0 to NWORDS-1;
 variable sram: T_SRAM_Nx32:=BINIMG_SRAM;
 begin
 ...
 end process;
end architecture;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_sram32.all;

entity wb_onchip_sram32 is
 port(
 wb_clk_i : in std_logic;
 wb_rst_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_sel_i : in std_logic_vector(3 downto 0);
 wb_we_i : in std_logic;
 wb_cyc_i : in std_logic;
 wb_stb_i : in std_logic;
 wb_ack_o : out std_logic;
```

```

 wb_err_o : out std_logic);
end entity;

architecture beh1 of wb_onchip_sram32 is
...
begin
 onchip_sram32: entity work.sram_Nx32(beh1) port map(
 clk=> wb_clk_i,
 e=> enabled,
 addr=> wb_adr_i(ADDR_WIDTH-1 downto 2),
 we=> wb_we_i,
 be=> wb_sel_i,
 idata=> wb_dat_i,
 odata=> wb_dat_o);

 strobe<=(not wb_rst_i) and wb_cyc_i and wb_stb_i;
 wb_ack_o<=ack_re or ack_we;
 wb_err_o<=strobe and or_bits(wb_adr_i(23 downto ADDR_WIDTH));
 enabled<=strobe and not or_bits(wb_adr_i(23 downto ADDR_WIDTH));

 ack: process (enabled,wb_we_i,wb_clk_i) begin
 ack_we<=enabled and wb_we_i;
 if enabled='0' then
 ack_re<='0';
 elsif rising_edge(wb_clk_i) then
 ack_re<=not wb_we_i;
 end if;
 end process;
end architecture;

```

**- fin fichero -**

### **Descripción del fichero**

En este fichero se definen las dos entidades del *core*: la memoria *ram* y el controlador de memoria. En primer lugar, se define la entidad *sram\_Nx32* con los puertos siguientes:

- *clk* (reloj, se trata de una memoria *ram* síncrona)
- *e* (enable: habilitación de la memoria)
- *addr* (bus de direcciones de 13 bits)
- *we* (bit de control de lectura/escritura)
- *be* (cuatro bits de control de selección de byte)
- *idata* (bus de datos de entrada, para las operaciones de escritura a memoria)
- *odata* (bus de datos de salida, para las operaciones de lectura de memoria)

En la descripción de comportamiento de la entidad se realiza una llamada a la función de inicialización *F\_BINIMG* descrita en el paquete anterior. Se entra en el proceso llamado *sram* que, en el flanco positivo de reloj, realiza las operaciones de lectura o escritura de datos con la memoria *ram* de la forma indicada por las señales de control *e*, *we*, *be* y la constante de control de distribución de datos *BIG\_ENDIAN*.

En segundo lugar, se define la entidad *wb\_onchip\_sram32* con los puertos provenientes del bus *Wishbone*. En la descripción de comportamiento se define la función *orbits* utilizada para comprobar la existencia del error de tipo *overflow* en el bus de direcciones *addr*. Tras definir varias señales de control, se realizan las conexiones pertinentes en el mapa de puertos y las asignaciones de las señales definidas (todo de forma concurrente, es decir, en paralelo). A su vez, se entra en el proceso llamado *ack*

(ejecutado de forma secuencial) que realiza la siguiente operación cuando aparece un cambio en las señales *enabled*, *wb\_we\_i* o *wb\_clk\_i*:

- aserta *ack\_we* en caso que la memoria esté habilitada y se trate de una operación de escritura. En caso contrario la deja a nivel bajo.
- hace *ack\_re* = 0 en caso que la memoria esté deshabilitada
- actualiza *ack\_re* con el valor opuesto a *ack\_we* en caso que la memoria esté habilitada y se trate de un flanco positivo de reloj.

### 3.3.3.7.3 Fichero de inicialización

#### Generación del fichero

De la misma manera que se utilizaban programas escritos en lenguaje C para realizar las conversiones de formato y obtener el fichero de formato *coe* a partir del fichero binario en formato *elf*, en este caso se debe crear una aplicación C para convertir el fichero *elf* al formato adecuado para la inicialización de la memoria *ram* del *core* generado (tipo de dato *T\_RAWDATA*). El formato de salida debe ser el mostrado en el cuadro de texto siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use work.pack_sram32.all;

package pack_bin_hello_uart is
 constant BIN_HELLO_UART: T_RAWDATA := (
 X"01234567", X"89abcdef", X"01234567", X"89abcdef",
 X"01234567", X"89abcdef", X"01234567", X"89abcdef",
 . . .
 X"01234567", X"89abcdef", X"01234567", X"89abcdef");
end package;
```

A partir del fichero *or32* obtenido tras la compilación y el enlazado de los ficheros fuente del programa (*hello-uart* en este caso), se introduce la línea de código “*or32-uclinux-objcopy -O binary hello.or32 hello.bin*” y se llama al ejecutable del programa mostrado a continuación mediante la línea de código “*bin2vhdl hello.bin hello.vhd*”:

#### - inicio fichero -

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
 FILE *fdin, *fdout;
 int c, i=0, a=0;
 if ((fdin=fopen(argv[1], "r"))==NULL)
 {
 printf("Error opening input file\n");
 return(-1);
 }
 if ((fdout=fopen(argv[2], "w"))==NULL)
 {
 printf("Error opening output file\n");
 return(0);
 }
 fprintf(fdout, "library ieee;\n");
 fprintf(fdout, "use ieee.std_logic_1164.all;\n");
 fprintf(fdout, "use work.pack_sram32.all;\n\n");
```

```

fprintf(fdout, "package pack_bin_hello_uart is\n");
fprintf(fdout, "\tconstant BIN_HELLO_UART:
 T_RAWDATA:=(\n\t\tX\"");
c = fgetc(fdin);
while (1)
{
 if (c != EOF)
 fprintf(fdout, "%.2x", (unsigned int) c);
 if (++i == 4)
 {
 c = fgetc(fdin);
 if (c == EOF)
 {
 fprintf(fdout, "\");\n");
 fprintf(fdout, "end package;\n");
 fclose(fdin);
 fclose(fdout);
 return 0;
 }
 else if (++a == 4)
 {
 fprintf(fdout, "\", \n\t\tX\"");
 a = 0;
 }
 else fprintf(fdout, "\", X\"");
 i = 0;
 }
 else c = fgetc(fdin);
}
}

```

**- fin fichero -**

### **Descripción del fichero**

El programa copia la información contenida en el fichero binario de entrada al fichero de salida modificando la estructura para obtener los datos en el formato descrito.

#### **3.3.3.7.4 Conexión del controlador al bus Wishbone**

La conexión del controlador de memoria *ram* al bus *Wishbone* se consigue editando el fichero *xsv\_fpga\_top.v* añadiendo los siguientes bloques de código en la sección adecuada (además de suprimir los bloques de código añadidos en el apartado 3.3.3.3.5):

- Cableado para la conexión del controlador al *slave 0* del árbitro del bus (*traffic cop*).

```

// onchip-SRAM controller slave i/f wires
wire [31:0] wb_ss_dat_i;
wire [31:0] wb_ss_dat_o;
wire [31:0] wb_ss_adr_i;
wire [3:0] wb_ss_sel_i;
wire wb_ss_we_i;
wire wb_ss_cyc_i;
wire wb_ss_stb_i;
wire wb_ss_ack_o;
wire wb_ss_err_o;

```

- Instancia del controlador de memoria *ram*

```

wb_onchip_sram32 onchip_ram_top(
 // WISHBONE common
 .wb_clk_i (wb_clk),
 .wb_rst_i (wb_rst),

 // WISHBONE slave
 .wb_dat_i (wb_ss_dat_i),
 .wb_dat_o (wb_ss_dat_o),
 .wb_adr_i (wb_ss_adr_i),
 .wb_sel_i (wb_ss_sel_i),
 .wb_we_i (wb_ss_we_i),
 .wb_cyc_i (wb_ss_cyc_i),
 .wb_stb_i (wb_ss_stb_i),
 .wb_ack_o (wb_ss_ack_o),
 .wb_err_o (wb_ss_err_o)
);

```

- Conexión del controlador al *traffic cop* mediante los cables definidos en el primer punto.

```

// Instantation of the Traffic Cop
// WISHBONE Initiators 0..7
...
// WISHBONE Target 0
.t0_wb_cyc_o (wb_ss_cyc_i),
.t0_wb_stb_o (wb_ss_stb_i),
.t0_wb_cab_o (wb_ss_cab_i),
.t0_wb_adr_o (wb_ss_adr_i),
.t0_wb_sel_o (wb_ss_sel_i),
.t0_wb_we_o (wb_ss_we_i),
.t0_wb_dat_o (wb_ss_dat_i),
.t0_wb_dat_i (wb_ss_dat_o),
.t0_wb_ack_i (wb_ss_ack_o),
.t0_wb_err_i (wb_ss_err_o),
// WISHBONE Targets 1..7
...

```

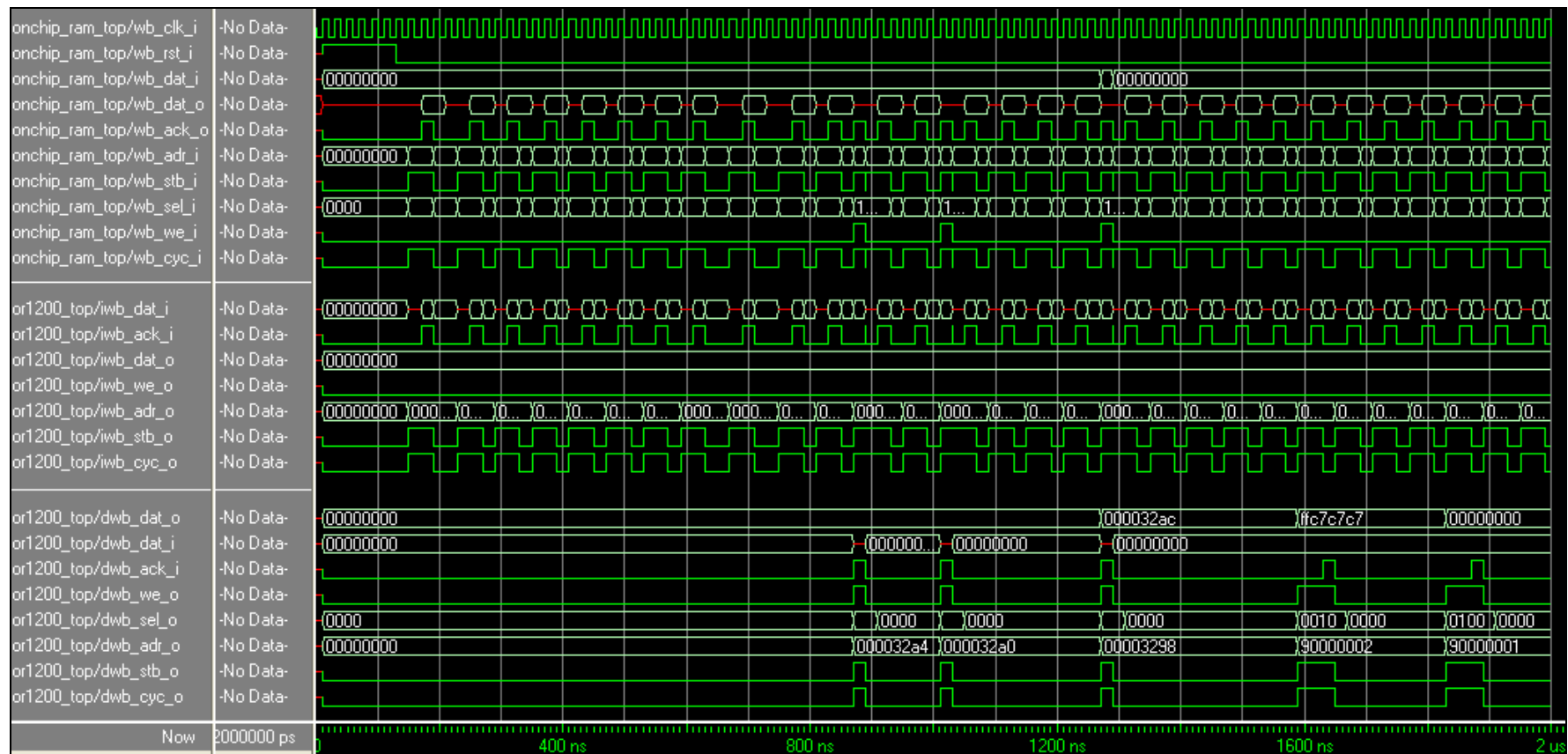
### 3.3.3.7.5 Añadir el bloque de memoria al proyecto de simulación

En la ventana de proyecto de *ModelSim* se selecciona *Add to project* → *Existing file* en el menú emergente del *mouse*. Se añaden al proyecto las fuentes ‘*pack\_sram32.vhd*’, ‘*pack\_bin\_hellouart.vhd*’ y ‘*wb\_onchip\_sram32.vhd*’. Al tratarse de código *vhdl*, se debe tener en cuenta el orden en el proceso de compilación.

### 3.3.3.8 Simulación funcional con memoria RAM

En el menú principal de *ModelSim* se selecciona la opción *Simulate* → *Start Simulation*. Sin embargo, en este caso no se requiere la adición de la librería de *xilinx*, ya que se utiliza el *core* de memoria creado en los apartados anteriores en lugar del generado por la herramienta *Coregen* de *Xilinx*. Entonces, en la pestaña *Design* se selecciona el fichero *sim\_test1* de la librería *work*. El resultado de la simulación en la ventana *Waveform* es:





**Imagen 32-** Simulación. Vista general de los 2us iniciales de operación del bus Wishbone y la memoria RAM (para más detalle ver las ventanas de zoom ampliado)

Tras el reset, el uP accede a memoria para realizar el fetch de la primera instrucción en la dirección 0x100 (programa *reset.S*). Tras esto accede a las instrucciones siguientes hasta que el PC (*program counter*) salta a la dirección 0x21F4. El bus de instrucciones *iwb* realiza el fetch de las instrucciones, operación que se combina con los accesos de escritura y lectura a memoria del bus de datos *dwb*.

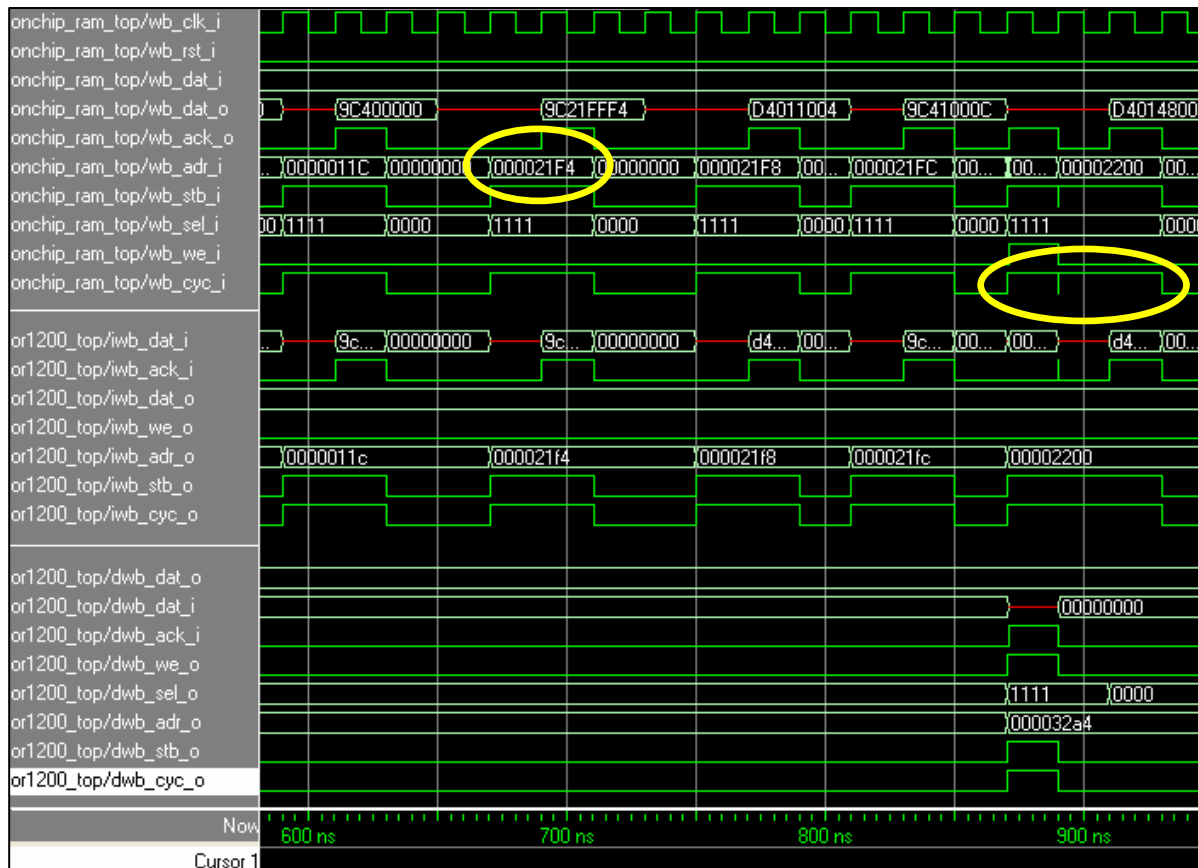


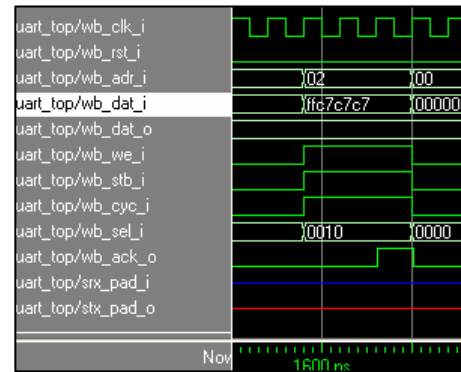
Imagen 33 - Detalle de simulación ampliado. 1: Salto del PC – 2: Acceso de lectura y escritura

El programa accede a la dirección 0x100 y ejecuta las instrucciones contenidas en el fichero fuente *reset.S*. A continuación el *PC* salta a la dirección 0x21F4, ejecuta un trozo de código y salta a la dirección 0x2000 (inicio del fichero fuente *hello.c*). Entonces inicializa la *uart* e inicia la transferencia de datos por el puerto serie. En el tiempo de simulación 6700 ns aproximadamente, el programa entra en un bucle infinito (contemplado en la función *main* del fichero fuente *hello.c*).

**1 – Salto del PC:** se observa que la señal *iwb\_adr\_o* (bus de instrucciones) pasa de 0x011C a 0x21F4. Esto significa que el *uP* realiza el fetch (acceso de lectura) de la primera instrucción del programa *hello.c* mapeado en la memoria.

**2 – Acceso de lectura y escritura:** se observa que el bus de datos *dwb* pretende hacer un acceso de escritura a memoria (*cyc\_o* = 1 y *we\_o* = 1) al mismo tiempo que el bus de instrucciones *iwb* pretende hacer el fetch de la instrucción siguiente (*cyc\_o* = 1 y *we\_o* = 0). El árbitro del bus (*traffic cop*) resuelve el algoritmo de prioridades y permite en primer lugar el acceso del *dwb* y seguidamente el acceso del *iwb*.

El código fuente del programa *hello-uart* mapeado en la memoria realiza una serie de accesos a los registros internos de la *uart*. La imagen lateral muestra un acceso de escritura del dato 0xC7 al registro interno de 8 bits mapeado en la dirección 0x02. En la tabla siguiente se presenta una descripción de los accesos a los registros de la *uart* realizados:



**Imagen 34** - Detalle de simulación ampliado. Acceso a registro de la *uart*.

| Registro | w/r | Dato | Descripción                                                                                                                                                                                                                                                                                                                                                    |
|----------|-----|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 02       | w   | 0xC7 | Despeja los FIFO del receptor y el transmisor y hace un reset de su lógica                                                                                                                                                                                                                                                                                     |
| 01       | w   | 0x00 | Deshabilita las interrupciones                                                                                                                                                                                                                                                                                                                                 |
| 03       | w   | 0x03 | Especifica el tamaño del caracter (8 bits), un bit de stop y deshabilita la paridad                                                                                                                                                                                                                                                                            |
| 03       | r   | 0x03 | Lectura del registro de control de línea                                                                                                                                                                                                                                                                                                                       |
| 03       | w   | 0x83 | Permite el acceso a los <i>divisor latches</i> (para especificar el <i>baud rate</i> , es decir, la frecuencia de transmisión serie)                                                                                                                                                                                                                           |
| 00       | w   | 0x20 | Escribe el dato en el LSB del <i>divisor latch</i>                                                                                                                                                                                                                                                                                                             |
| 01       | w   | 0x00 | Escribe el dato en el MSB del <i>divisor latch</i>                                                                                                                                                                                                                                                                                                             |
| 03       | r   | 0x83 | Lectura del registro de control de línea                                                                                                                                                                                                                                                                                                                       |
| 03       | w   | 0x03 | No permite el acceso a los <i>divisor latches</i>                                                                                                                                                                                                                                                                                                              |
| 05       | r   | 0x00 | Lectura del registro de estado de línea (para el WAIT_FOR_THRE)                                                                                                                                                                                                                                                                                                |
| 00       | w   | 0x48 | Escribe el dato en el registro de entrada al FIFO de salida                                                                                                                                                                                                                                                                                                    |
| 05       | r   | 0x00 | Lectura del registro de estado de línea (para el WAIT_FOR_THRE)                                                                                                                                                                                                                                                                                                |
| ...      | ... | ...  | Se repite el último acceso esperando a que la <i>uart</i> avise que el carácter ya ha sido enviado. Entonces envía el carácter siguiente al FIFO de salida<br>La cadena enviada es:<br>48-H 65-e 6c-l 6c-l 6f-o 20-<br>77-w 6f-o 72-r 6c-l 64-d 21-! 21-! 21-!<br>0a-\n 0d-\r<br>La cadena termina de transmitirse a los 1600us de simulación aproximadamente. |
| 05       | r   | 0x60 | Lee que los FIFO de entrada y salida están vacíos. Repite el acceso. Ha entrado en el bucle infinito.                                                                                                                                                                                                                                                          |

El valor guardado en el *divisor latch* equivale a:

$$\text{SystemClockSpeed} / (16 \times \text{DesiredBaudRate}) = 32$$

### 3.3.4 Síntesis

En este apartado se pretende conseguir que los *cores* de la memoria *ram* interna y el controlador diseñados sean sintetizables. Además, se comprueba la misma

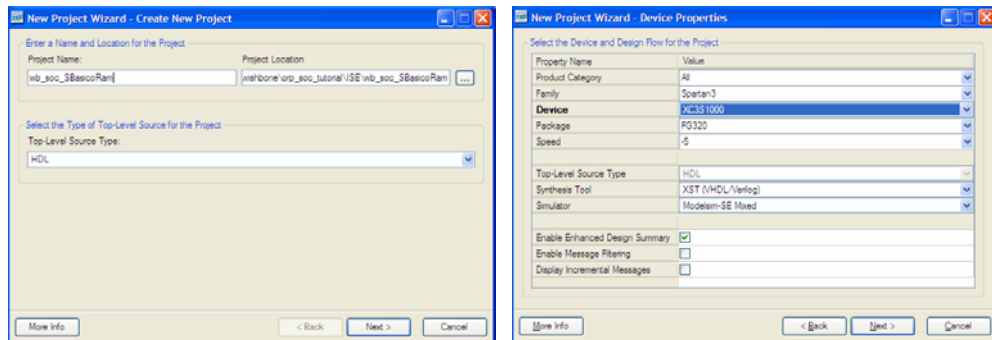
característica en los *cores* prediseñados por *OpenCores* que definen el sistema completo. Para ello se utiliza el programa *Project Navigator* del entorno *ISE* de *Xilinx*.

### 3.3.4.1 Memoria ram interna

#### 3.3.4.1.1 Memoria *wb\_onchip\_sram32*

Sintetizar el sistema completo supone un proceso largo y pesado. Por ello, la propuesta es tratar de sintetizar la memoria *ram* generada y simulada en apartados anteriores. Los pasos seguidos para el proceso son:

- Crear un nuevo proyecto llamado *wb\_onchip\_sram32* sin dar demasiada importancia a las características de la placa destino (no son relevantes en este paso).
- Añadir las fuentes que describen el comportamiento de la memoria *ram*: *pack\_bin\_hellouart.vhd*, *pack\_sram32.vhd* y *wb\_onchip\_sram32.vhd*.
- Seleccionar la opción *Synthesize* en la ventana de procesos disponibles para la instancia de jerarquía más alta (*wb\_onchip\_sram32*).



**Imagen 35** - Controles de creación de un proyecto en *Project Navigator*

El resultado del proceso es un informe de síntesis (*synthesis report*) que presenta abundante información sobre las diferentes tareas dentro del proceso:

- 1) Synthesis Options Summary
- 2) HDL Compilation
- 3) Design Hierarchy Analysis
- 4) HDL Analysis
- 5) HDL Synthesis
  - 5.1) HDL Synthesis Report
- 6) Advanced HDL Synthesis
  - 6.1) Advanced HDL Synthesis Report
- 7) Low Level Synthesis
- 8) Partition Report
- 9) Final Report
  - 9.1) Device utilization summary
  - 9.2) TIMING REPORT

A continuación, se presenta un resumen del informe obtenido que contiene los datos relevantes para nuestro objetivo:

```

 HDL Synthesis
Synthesizing Unit <sram_Nx32>.
Found 32-bit register for signal <odata>.
Found 8-bit 8192-to-1 multiplexer for signal <$mux0000> created at line 33.
Found 8-bit 8192-to-1 multiplexer for signal <$mux0001> created at line 34.
Found 8-bit 8192-to-1 multiplexer for signal <$mux0002> created at line 35.
Found 8-bit 8192-to-1 multiplexer for signal <$mux0003> created at line 36.
Found 65536-bit register for signal <sram<0>>.
Found 65536-bit register for signal <sram<1>>.
Found 65536-bit register for signal <sram<2>>.
Found 65536-bit register for signal <sram<3>>.
INFO:Xst:738 - HDL ADVISOR - 65536 flip-flops were inferred for signal
<sram<0>>. You may be trying to describe a RAM in a way that is incompatible
with block and distributed RAM resources available on Xilinx devices, or
with a specific template that is not supported. Please review the Xilinx
resources documentation and the XST user manual for coding guidelines.
Taking advantage of RAM resources will lead to improved device usage and
reduced synthesis time.
...
Summary: inferred 262176 D-type flip-flop(s).
 inferred 32 Multiplexer(s).
Unit <sram_Nx32> synthesized.

```

En la sección de *HDL Synthesis* del informe se presenta un mensaje de información que indica que la memoria *ram* definida no es compatible con los bloques de memoria embebidos como recursos de la placa *fpga*. A su vez, aconseja consultar la documentación distribuida por *Xilinx* para adaptar el código.

La memoria consume un total de 32 multiplexores y 262.176 flip-flops. Sin duda una cantidad de recursos demasiado elevada para conseguir un sistema de bajo coste. Por lo tanto, el siguiente paso es conseguir una memoria *ram* sintetizable, compatible con los bloques de memoria de la *fpga* y con un buen comportamiento en simulación.

#### 3.3.4.1.2 Memoria *wb\_onchip\_4sram8*

Antes del diseño de la memoria *ram wb\_onchip\_4sram8* se pretende identificar los errores cometidos en la definición de la memoria *wb\_onchip\_sram32*. Existen varios problemas posibles:

- Es posible que el uso de la función de inicialización de la memoria definida en el *package body* del paquete *pack\_sram32* sea problemático en el proceso de síntesis. En el nuevo diseño se inicializará (en el caso que sea necesario) mediante constantes definidas en el paquete *pack\_sam32*.
- Los bloques de memoria *ram* de la placa son incompatibles con la definición del tipo de dato *T\_SRAM\_Nx32* en el paquete *pack\_sram32*. Se define como un array de cuatro objetos *T\_SRAM\_Nx8*.

##### 3.3.4.1.2.1 Código fuente

###### a) Fichero *pack\_sram8.vhd*

```

library ieee;
use ieee.std_logic_1164.all;
package pack_sram8 is
 constant BIG_ENDIAN: integer:=1;
 constant ADDR_WIDTH: integer:=14;
 constant NBYTES: integer:=(2**ADDR_WIDTH);
 type T_SRAM_Nx8 is array(0 to NBYTES-1) of std_logic_vector(7
downto 0);
 constant BINIMG_U: T_SRAM_Nx8:=(others=>(others=>'U'));
 constant BINIMG_0: T_SRAM_Nx8:=(others=>(others=>'0'));
 constant BINIMG_1: T_SRAM_Nx8:=(others=>(others=>'1'));
end package;

```

El paquete contiene la definición del tipo de dato *T\_SRAM\_Nx8* (utilizado para almacenar los datos en la memoria *ram*) y la definición de las constantes utilizadas para la customización y la inicialización de la memoria.

#### b) Fichero *wb\_onchip\_4sram8.vhd*

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_sram8.all;

entity sram_Nx8 is
 port(
 clk: in std_logic;
 e: in std_logic;
 addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
 we: in std_logic;
 idata: in std_logic_vector(7 downto 0);
 odata: out std_logic_vector(7 downto 0));
end entity;

architecture beh1 of sram_Nx8 is
 signal ram: T_SRAM_Nx8;
begin
 process begin
 wait until rising_edge(clk);
 odata<=(others=>'X');
 if e='1' then
 if we='1' then
 ram(conv_integer(unsigned(addr)))<=idata;
 else
 odata<=ram(conv_integer(unsigned(addr)));
 end if;
 end if;
 end process;
end architecture;

```

Se trata de la primera parte del fichero. En este módulo se definen los puertos de la entidad *sram\_Nx8* y se describe su comportamiento en la arquitectura *beh1*. El proceso escribe los ocho bits contenidos en *idata* en la posición indicada de la memoria en caso que el bloque esté habilitado ( $e=1$ ) y se trate de un acceso de escritura ( $we=1$ ). En caso de que se trate de un acceso de lectura ( $e=1$  y  $we=0$ ), el proceso introduce los bits señalados por *addr* en el bus de salida *odata*. Toda la operación es síncrona debido a la línea de código *wait until rising\_edge(clk);*.

```

library ieee;use ieee.std_logic_1164.all;use ieee.std_logic_arith.all;
use work.pack_sram8.all;
entity wb_onchip_4sram8 is
 port(
 wb_clk_i : in std_logic;
 wb_rst_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_sel_i : in std_logic_vector(3 downto 0);
 wb_we_i : in std_logic;
 wb_cyc_i : in std_logic;
 wb_stb_i : in std_logic;
 wb_ack_o : out std_logic;
 wb_err_o : out std_logic);
end entity;

architecture beh1 of wb_onchip_4sram8 is
 ...
begin
 strobe<=(not wb_rst_i) and wb_cyc_i and wb_stb_i;
 wb_ack_o<=ack_re or ack_we;
 enabled<=strobe and not select_mem(wb_adr_i(23 downto ADDR_WIDTH));
 ack: process (enabled,wb_we_i,wb_clk_i) begin
 ack_we<=enabled and wb_we_i;
 if enabled='0' then
 ack_re<='0';
 elsif rising_edge(wb_clk_i) then
 ack_re<=not wb_we_i;
 end if;
 end process;
 sram_chip3: entity work.sram_Nx8(beh1) port map(
 ...);
 sram_chip2: entity work.sram_Nx8(beh1) port map(
 ...);
 sram_chip1: entity work.sram_Nx8(beh1) port map(
 ...);
 sram_chip0: entity work.sram_Nx8(beh1) port map(
 ...);
end architecture;

```

En este módulo se definen los puertos de la entidad *wb\_onchip\_4sram8*. Este bloque es el puente que conecta el bus *Wishbone* a los bloques de memoria *ram*. A continuación, se describe el comportamiento en la arquitectura *beh1*:

- se define la función utilizada para comprobar que el bus de direcciones accede a una posición válida
- se describe la gestión de las señales de control
- se realiza la instancia de los cuatro bloques *sram\_Nx8* y se define la conexión de sus puertos

#### 3.3.4.1.2.2 Simulación funcional

Para comprobar el correcto funcionamiento de la memoria *wb\_onchip\_4sram8* se realiza una simulación del *core* aisladamente. Para ello, se crea un nuevo proyecto de *ModelSim* y se añaden las fuentes de la memoria: *pack\_sram8.vhd* y *wb\_onchip\_4sram8.vhd*. A continuación se crea un fichero de *testbench* que estimula las entradas del controlador de memoria como si se tratara de las señales enviadas por el procesador al bus *Wishbone*. El código del fichero *testbench* es el siguiente:

```

library IEEE;
use IEEE.std_logic_1164.all;

entity test_sram is
end entity;

architecture test of test_sram is
signal clk: std_logic:='1';
signal rst: std_logic:='0';
signal dat_i: std_logic_vector(31 downto 0):=X"00000000";
signal dat_o: std_logic_vector(31 downto 0):=X"00000000";
signal adr_i: std_logic_vector(31 downto 0):=X"00000000";
signal sel_i: std_logic_vector(3 downto 0):="0000";
signal we_i: std_logic:='0';
signal cyc_i: std_logic:='0';
signal stb_i: std_logic:='0';
signal ack_o: std_logic:='0';
signal err_o: std_logic:='0';

begin
 clk<=not clk after 10 ns;
 rst<='1', '0' after 20 ns;
 dat_i<=X"00000000", X"12348765" after 40 ns, X"00000000" after
 100 ns, X"abcdfedc" after 120 ns, X"00000000" after 180 ns;
 adr_i<=X"00000000", X"00000ffd" after 40 ns, X"00000000" after
 100 ns, X"00000ffe" after 120 ns, X"00000000" after 180 ns,
 X"00000ffe" after 200 ns, X"00000000" after 260 ns,
 X"00000ffd" after 280 ns, X"00000000" after 340 ns,
 X"00000ffc" after 360 ns, X"00000000" after 420 ns;
 sel_i<="0000", "1111" after 40 ns, "0000" after 100 ns, "1111"
 after 120 ns, "0000" after 180 ns, "1111" after 200 ns,
 "0000" after 260 ns, "1111" after 280 ns, "0000" after 340
 ns, "1111" after 360 ns, "0000" after 420 ns;
 we_i<='0', '1' after 40 ns, '0' after 100 ns, '1' after 120
 ns, '0' after 180 ns;
 cyc_i<='0', '1' after 40 ns, '0' after 100 ns, '1' after 120
 ns, '0' after 180 ns, '1' after 200 ns, '0' after 260 ns,
 '1' after 280 ns, '0' after 340 ns, '1' after 360 ns, '0'
 after 420 ns;
 stb_i<='0', '1' after 40 ns, '0' after 100 ns, '1' after 120
 ns, '0' after 180 ns, '1' after 200 ns, '0' after 260 ns,
 '1' after 280 ns, '0' after 340 ns, '1' after 360 ns, '0'
 after 420 ns;

 sram: entity work.wb_onchip_4sram8 port map(
 wb_clk_i=>clk,
 wb_rst_i=>rst,
 wb_dat_i=>dat_i,
 wb_dat_o=>dat_o,
 wb_adr_i=>adr_i,
 wb_sel_i=>sel_i,
 wb_we_i=>we_i,
 wb_cyc_i=>cyc_i,
 wb_stb_i=>stb_i,
 wb_ack_o=>ack_o,
 wb_err_o=>err_o
);
end architecture;

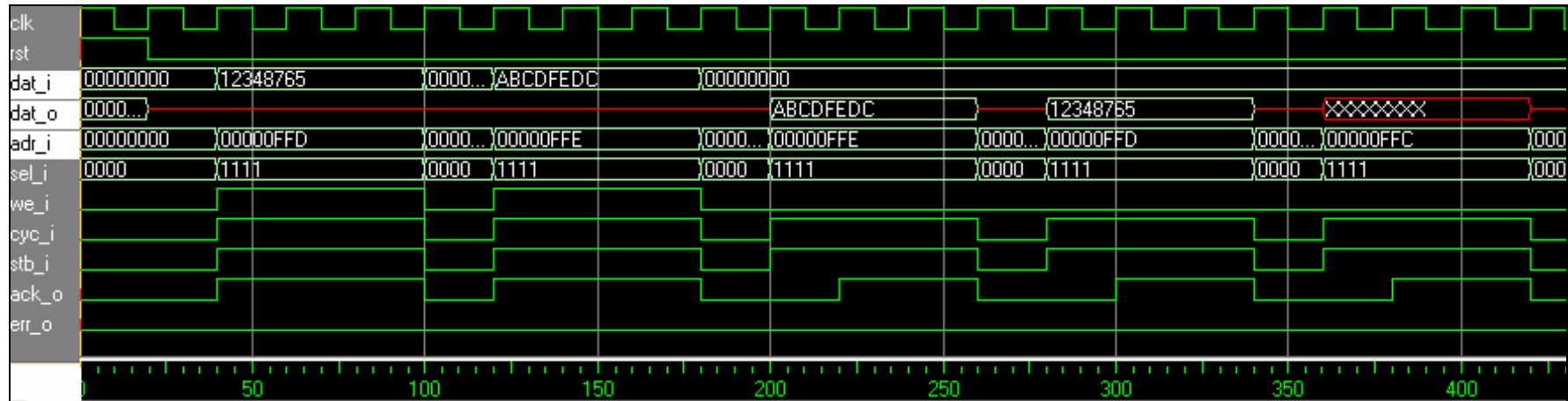
```

En el código se realizan tres tareas muy simples:

- Creación del módulo *test\_sram* y declaración/inicialización de las señales.
- Estimulación de las señales para simular dos accesos de escritura y dos accesos de lectura a la memoria.
- Instanciación del controlador de memoria y conexión de las señales a sus puertos I/O.



El resultado de la simulación se muestra en la imagen siguiente:



**Imagen 36** - Detalle de simulación del sistema básico con memoria interna *block ram*

Tras el reset del sistema, se comprueba que el controlador de memoria interna realiza los siguientes accesos a memoria:

- Acceso de escritura del dato 0x12348765 a la dirección de memoria 0xFFD
- Acceso de escritura del dato 0xABCD FEDC a la dirección de memoria 0xFFE
- Acceso de lectura a la dirección 0xFFD. Se obtiene por el bus de datos 0x12348765
- Acceso de lectura a la dirección 0xFFE. Se obtiene por el bus de datos 0xABCD FEDC

El sistema accede de forma correcta a la memoria interna para escritura y lectura de datos.

### 3.3.4.1.2.3 Síntesis

Se crea un proyecto nuevo llamado *wb\_onchip\_4sram8* con un procedimiento parecido al descrito en el apartado 3.3.4.1.1. El proyecto está compuesto por los archivos fuente del modelo de memoria escrito en el apartado x.1.2: *pack\_sram8.vhd* y *wb\_onchip\_4sram8.vhd*. Se selecciona la opción sintetizar de la ventana de procesos vinculada a la instancia de jerarquía superior (*wb\_onchip\_4sram8*). Se obtiene un nuevo informe de síntesis del cual se destaca la siguiente información:

| HDL Synthesis                                                                                                                                                                                                                                                                       |                             |              |          |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|--------------|----------|
| Synthesizing Unit <sram_Nx8>.                                                                                                                                                                                                                                                       |                             |              |          |
| Related                                                                                                                                                                                                                                                                             | source                      | file         | is       |
| "C:/User/Doctor/wishbone/orp_soc_tutorial/ISE/onchip_ram_syn/wb_onchip_4sram8.vhd".                                                                                                                                                                                                 |                             |              |          |
| Found 16384x8-bit single-port distributed RAM for signal <ram>.                                                                                                                                                                                                                     |                             |              |          |
| ram_style                                                                                                                                                                                                                                                                           | Auto                        |              |          |
| -----                                                                                                                                                                                                                                                                               |                             |              |          |
| Port A                                                                                                                                                                                                                                                                              |                             |              |          |
| aspect ratio                                                                                                                                                                                                                                                                        | 16384-word x 8-bit          |              |          |
| clkA                                                                                                                                                                                                                                                                                | connected to signal <clk>   | rise         |          |
| weA                                                                                                                                                                                                                                                                                 | connected to internal node  | high         |          |
| addrA                                                                                                                                                                                                                                                                               | connected to signal <addr>  |              |          |
| diA                                                                                                                                                                                                                                                                                 | connected to signal <idata> |              |          |
| doA                                                                                                                                                                                                                                                                                 | connected to internal node  |              |          |
| -----                                                                                                                                                                                                                                                                               |                             |              |          |
| INFO:Xst:1442 - HDL ADVISOR - The RAM contents appears to be read asynchronously. A synchronous read would allow you to take advantage of available block RAM resources, for optimized device usage and improved timings. Please refer to your documentation for coding guidelines. |                             |              |          |
| Found 8-bit register for signal <odata>.                                                                                                                                                                                                                                            |                             |              |          |
| Summary:                                                                                                                                                                                                                                                                            |                             |              |          |
| inferred 1 RAM(s).                                                                                                                                                                                                                                                                  |                             |              |          |
| inferred 8 D-type flip-flop(s).                                                                                                                                                                                                                                                     |                             |              |          |
| Unit <sram_Nx8> synthesized.                                                                                                                                                                                                                                                        |                             |              |          |
| Advanced HDL Synthesis                                                                                                                                                                                                                                                              |                             |              |          |
| INFO:Xst:2131 - Distributed RAM <Mram_ram> in block <sram_Nx8> is combined with register <odata> in block <sram_Nx8>. Implementation is forced on block RAM resources.                                                                                                              |                             |              |          |
| Final Report                                                                                                                                                                                                                                                                        |                             |              |          |
| Device utilization summary:                                                                                                                                                                                                                                                         |                             |              |          |
| -----                                                                                                                                                                                                                                                                               |                             |              |          |
| Selected Device : 3s1000fg320-5                                                                                                                                                                                                                                                     |                             |              |          |
| Number of Slices:                                                                                                                                                                                                                                                                   | 6                           | out of 7680  | 0%       |
| Number of Slice Flip Flops:                                                                                                                                                                                                                                                         | 1                           | out of 15360 | 0%       |
| Number of 4 input LUTs:                                                                                                                                                                                                                                                             | 11                          | out of 15360 | 0%       |
| Number of IOs:                                                                                                                                                                                                                                                                      | 107                         |              |          |
| Number of bonded IOBs:                                                                                                                                                                                                                                                              | 98                          | out of 221   | 44%      |
| Number of BRAMs:                                                                                                                                                                                                                                                                    | 32                          | out of 24    | 133% (*) |
| Number of GCLKs:                                                                                                                                                                                                                                                                    | 1                           | out of 8     | 12%      |
| WARNING:Xst:1336 - (*) More than 100% of Device resources are used                                                                                                                                                                                                                  |                             |              |          |

En la fase de síntesis el programa identifica la señal *ram* (definida en el código) como un modelo de *distributed ram*. Este tipo de memoria forma parte de los recursos embebidos de una placa *fpga*. Por otra parte, el modelo de memoria *block ram* (también parte de los recursos de la placa) ofrece mejores prestaciones a nuestro objetivo debido a la localización, distribución y velocidad de sus bloques de memoria. El programa informa que la señal no se ha identificado como un modelo de *block ram* (*BRAM*) porque se realiza algún tipo de lectura asíncrona.

Sin embargo, en la fase de síntesis avanzada el programa *impone* la implementación de la memoria en recursos de *block ram*. Cada *block ram* equivale a 16kb, es decir, 2kB. El sistema precisa de una memoria de 32 bits (4 bytes) de anchura: 2kB x 4 = 2kword. Si definimos el bus de direcciones de 14 bits, tenemos una profundidad de 16kword: (2kword) x 8 = 16kword. Por lo tanto, el sistema consume un total de 32 *BRAMs*.

#### 3.3.4.1.2.4 Inicialización de la memoria

Se debe posibilitar la inicialización de la memoria creada para poder simular sistemas avanzados en la sección posterior del proyecto. Sin embargo, se destaca que las modificaciones específicas para la inicialización de la memoria interna realizadas en este apartado no son sintetizables, por lo tanto solamente serán utilizadas para simulación. Para el proceso de síntesis y posterior descarga del sistema en la *fpga* se utilizará el modelo de memoria creado en el apartado 3.3.4.1.2.

El primer paso es añadir el fichero *pack\_bin\_hellouart.vhd* creado anteriormente para reutilizarlo en este apartado. A continuación, en el cuerpo del paquete *pack\_sram8* se escriben cuatro funciones de inicialización paralelas de la forma siguiente (solamente varían las partes subrayadas):

```
function F_BINIMG1(rawdata: T_RAWDATA) return T_SRAM_Nx8 is
variable k: integer;
variable word32: std_logic_vector(31 downto 0);
variable binimg: T_SRAM_Nx8:=BINIMG_U;
begin
 for k in rawdata'range loop
 word32:=rawdata(k);
 binimg(k):=word32(31 downto 24);
 end loop;
 return binimg;
end function;
```

El siguiente paso es la definición de cuatro arquitecturas (*beh1*, *beh2*, *beh3* y *beh4*) que realizan la misma operación, difiriendo solamente en la función de inicialización llamada. Finalmente, en la entidad *wb\_onchip\_4sram8* se modifican ligeramente las cuatro instancias a los bloques *sram Nx8* estableciendo una arquitectura diferente de las previamente definidas para cada bloque.

#### 3.3.4.2 Sistema Básico

En primer lugar se sustituye la memoria creada recientemente para ser sintetizada como *block ram* (*wb\_onchip\_4sram8*) por la memoria utilizada hasta ahora pero no sintetizable (*wb\_onchip\_sram32*). Se trata de un cambio muy simple debido al paralelismo de la interfaz Wishbone de ambos *cores*. Se cambia el nombre de la entidad utilizada en la instancia de la memoria *ram* en el fichero del *Traffic Cop* (*tc\_top.v*).

Seguidamente se crea un nuevo proyecto de síntesis llamado *wb\_onchip\_SB* y se especifica el dispositivo *fpga* destino (*xc3s1500-5-fg320*). El procedimiento para añadir las fuentes del sistema al proyecto es el siguiente:

- Añadir las fuentes de jerarquía superior.
- Sintetizar. El proceso termina con error: no encuentra una lista de módulos.
- Añadir las fuentes que contienen los módulos requeridos.
- Sintetizar y repetir el proceso hasta que la síntesis progrese sin errores.

A continuación se presenta la lista de fuentes añadidas al proyecto:

|                  |                    |                 |                      |
|------------------|--------------------|-----------------|----------------------|
| xsv_fpga_top     | dbg_sync_clk1_clk2 | orl200_iwb_biu  | orl200_rf            |
| tc_top           | dbg_registers      | orl200_wb_biu   | orl200_operandmuxes  |
| xsv_fpga_defines | dbg_crc8_d1        | orl200_immu_top | orl200_alu           |
|                  | dgb_register       | orl200_ic_top   | orl200_mult_mac      |
| wb_onchip_4sram8 | uart_wb            | orl200_cpu      | orl200_sprs          |
| pack_sram8       | uart_regs          | orl200_dmmu_top | orl200_lsu           |
| (ambos vhdl)     | uart_debug_if      | orl200_dc_top   | orl200_wbmux         |
|                  | uart_transmitter   | orl200_qmem_top | orl200_freeze        |
| dgb_top          | uart_sync_flops    | orl200_sb       | orl200_except        |
| orl200_top       | uart_receiver      | orl200_du       | orl200_cfgr          |
| uart_top         | uart_tfifo         | orl200_pic      | orl200_dc_fsm        |
| dgb_defines      | uart_rfifo         | orl200_tt       | orl200_dc_ram        |
| uart_defines     | raminfr            | orl200_pm       | orl200_dc_tag        |
| orl200_defines   |                    | orl200_ic_fsm   | orl200_spram_2048x32 |
| bench_defines    |                    | orl200_ic_ram   | orl200_rfram_generic |
|                  |                    | orl200_ic_tag   | orl200_gmultp2_32x32 |
|                  |                    | orl200_genpc    | orl200_mem2reg       |
|                  |                    | orl200_if       | orl200_reg2mem       |
|                  |                    | orl200_ctrl     |                      |

A continuación se presenta un resumen del informe de síntesis:

|                                 |       |        |       |      |
|---------------------------------|-------|--------|-------|------|
| Device utilization summary:     |       |        |       |      |
| Selected Device : 3s1500fg320-5 |       |        |       |      |
| Number of Slices:               | 5498  | out of | 13312 | 41%  |
| Number of Slice Flip Flops:     | 2940  | out of | 26624 | 11%  |
| Number of 4 input LUTs:         | 18736 | out of | 26624 | 70%  |
| Number used as logic:           | 10512 |        |       |      |
| Number used as RAMs:            | 8224  |        |       |      |
| Number of IOs:                  | 15    |        |       |      |
| Number of bonded IOBs:          | 13    | out of | 221   | 5%   |
| Number of BRAMs:                | 32    | out of | 32    | 100% |
| Number of MULT18X18s:           | 4     | out of | 32    | 12%  |
| Number of GCLKs:                | 2     | out of | 8     | 25%  |

En resumen, en esta sección del proyecto se ha conseguido implementar un sistema básico (formado por el microcontrolador *openrisc*, una memoria *ram* interna, la interfaz de comunicación serie *uart16550* y la *interfaz* de depuración) que simula y sintetiza de forma correcta. Temas como la optimización de frecuencia del sistema o de utilización de recursos del dispositivo destino no han sido tenidas en cuenta en esta sección, aunque no es factible obtener buenos resultados en esa dirección puesto que el proyecto de *OpenCores* se encuentra en proceso de desarrollo. De todos modos, en apartados posteriores se pretende optimizar el tamaño de la memoria interna según la necesidad de espacio para mapear el software ejecutado por el microprocesador. Para este procedimiento es necesaria la edición de las constantes y/o variables definidas en los paquetes *vhdl* pertenecientes al *core* de la memoria interna.

### 3.4 Sistema Avanzado con Controlador de Memoria Externa

En este apartado se pretende añadir al sistema básico un controlador de memoria conectado a un chip de memoria ram externa a la *fpga*. Los objetivos principales de este apartado son los siguientes:

- el estudio de la topología del bus para la conexión de nuevos módulos al sistema. Se describe la metodología para la conexión de un elemento relativamente sencillo

(controlador de memoria) para, en apartados posteriores, facilitar la conexión de un módulo de complejidad superior (coprocesador).

- la familiarización con las técnicas utilizadas para la adaptación de las señales provenientes de un chip de memoria ram y las utilizadas para la inicialización o mapeado de los datos en un sistema compuesto por múltiples bloques de memoria.
- la comprobación del correcto funcionamiento del sistema conectado a diferentes bloques de memoria. Este proceso se realiza mediante simulación funcional.

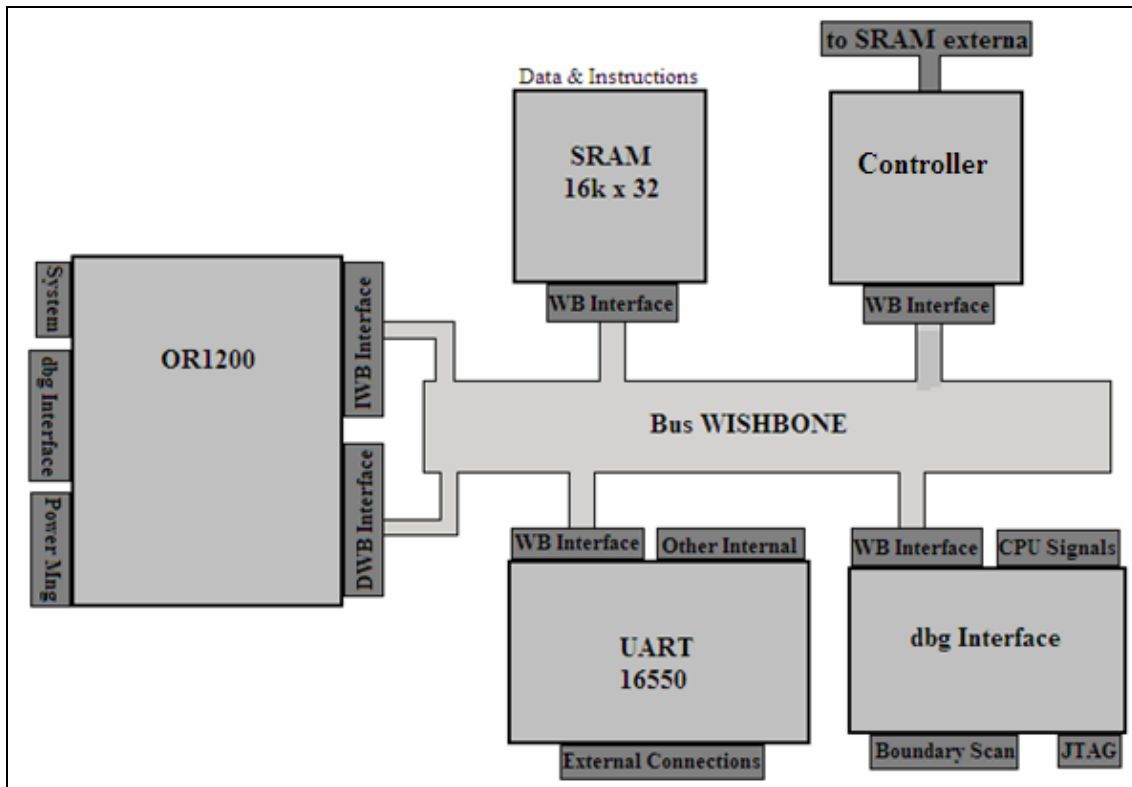


Imagen 37 - Diagrama esquemático del Sistema Avanzado con controlador de memoria

Para conseguir el diseño mostrado se debe conectar el controlador de memoria al *Slave1* del bus Wishbone. Por lo tanto, la configuración de bus que se obtiene es la siguiente:

| #Master |                         | #Slave |                       |
|---------|-------------------------|--------|-----------------------|
| 0       | nc                      | 0      | SRAM Controller Slave |
| 1       | nc                      | 1      | SRAM Controller Slave |
| 2       | nc                      | 2      | nc                    |
| 3       | Debug core Master       | 3      | nc                    |
| 4       | RISC Data Master        | 4      | nc                    |
| 5       | RISC Instruction Master | 5      | UART core Slave       |
| 6       | nc                      | 6      | nc                    |
| 7       | nc                      | 7      | nc                    |

En esta sección se realizan las siguientes tareas para el sistema propuesto:

- **Hardware:** añadir al sistema básico de la sección anterior un controlador de memoria externa que se comunica con una memoria externa de 32 bits de anchura

- **Software:** generar las herramientas de desarrollo de una *GNU Toolchain* compatible con el lenguaje de programación *C++*. Desarrollo del software de prueba del controlador de memoria externa. Compilar y enlazar el programa generado.
- **Simulación funcional:** comprobar el correcto funcionamiento del sistema mediante simulación funcional

### 3.4.1 Hardware

En este apartado se pretenden describir los ajustes de *hardware* realizados para obtener el sistema mostrado en la introducción. Se parte del código fuente relativo al sistema básico obtenido en la sección del proyecto anterior. Para añadir correctamente la memoria externa al sistema, se conecta un controlador de memoria al bus *wishbone* que, a su vez, se conecta a una memoria ram externa.

#### 3.4.1.1 Controlador de memoria

El módulo del controlador de memoria debe conectar la memoria al bus *wishbone* correctamente. Para ello debe adaptar y gestionar las señales que provienen del chip de memoria externo mediante los puertos I/O de la *fpga*. Sin embargo, el código fuente del controlador ha sido escrito para ser operativo con independencia del bus al que se conecte. Por lo tanto, se escribe un fichero “puente” que adapte las señales del bus *wishbone* a las señales del módulo del controlador genérico.

El código fuente del controlador se agrupa en dos ficheros escritos en lenguaje *vhdl*: *sramcontrol\_entity.vhd* y *sramcontrol\_beh3.vhd*.

##### a) Fichero *sramcontrol\_entity.vhd*

```
library ieee; use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;
use work.pack_utils.all;

entity sramcontrol is
generic(
 ADDRUP_WIDTH: integer:=16;
 DATAUP_WIDTH: integer:=32;
 DATARAM_WIDTH: integer:=8;
 START_STATE: integer:=0;
 END_STATE: integer:=0;
 WAIT_STATES: integer:=2);
port(
 clk: in std_logic;
 initrst: in std_logic;
 strt: in std_logic;
 rdy: out std_logic;
 size: in std_logic_vector(1 downto 0);
 rNw: in std_logic;
 addr: in std_logic_vector(ADDRUP_WIDTH-1 downto 0);
 din: in std_logic_vector(DATAUP_WIDTH-1 downto 0);
 dout: out std_logic_vector(DATAUP_WIDTH-1 downto 0);
 doutack: out std_logic;
 sram_Ncs,sram_Nwe,sram_Noe: out std_logic;
 sram_Nbe: out std_logic_vector((DATARAM_WIDTH/8)-1 downto 0);
 sram_addr: out std_logic_vector(ADDRUP_WIDTH-1 downto
 num_bits(DATARAM_WIDTH/8));
 sram_data_I: in std_logic_vector(DATARAM_WIDTH-1 downto 0);
 sram_data_O: out std_logic_vector(DATARAM_WIDTH-1 downto 0);
 sram_data_T: out std_logic);
end entity;
```

Este fichero define el mapa de puertos de entrada/salida al módulo del controlador. A su vez define una lista de genéricos utilizados para la adaptación del controlador al sistema conectado. La descripción esquemática del módulo es la siguiente:

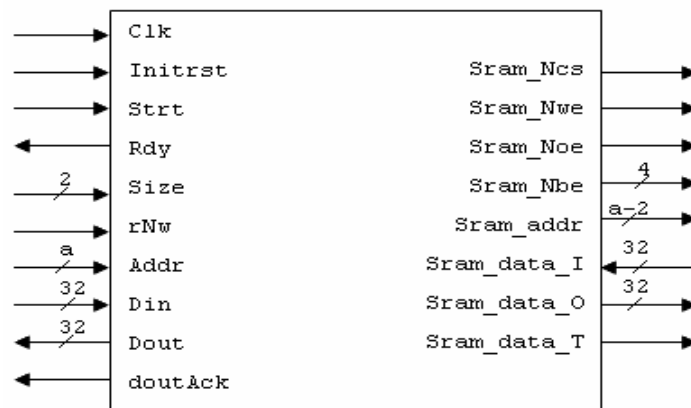


Imagen 38 - Esquema del módulo del controlador

#### b) Fichero sramcontrol\_beh3.vhd

El fichero describe el comportamiento de la entidad definida basado en el funcionamiento de una máquina de estado compuesta por cuatro estados: *START*, *RUN*, *END* y *STOP*. El esquema simplificado es:

El código fuente contiene una serie de procesos que corresponden con las diferentes fases de un ciclo de lectura o escritura del bus:

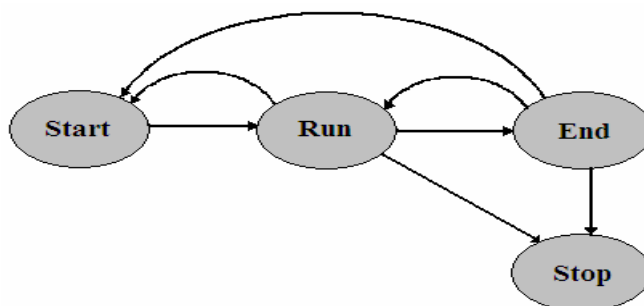
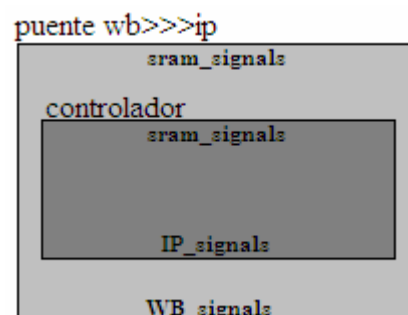


Imagen 39 - Máquina de estados

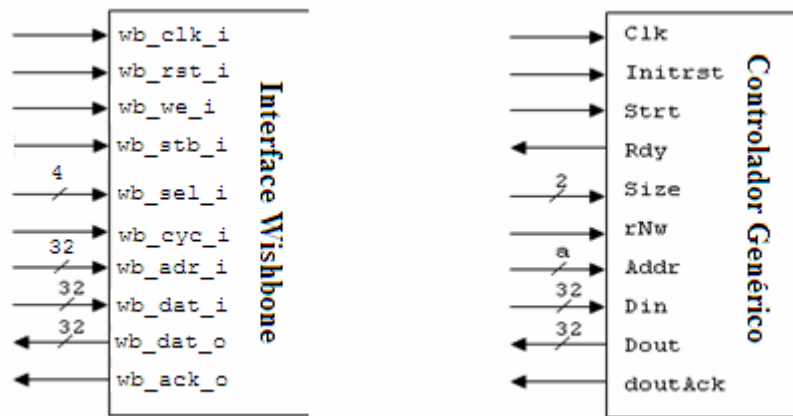
- UC0: gestiona la variable *nextstate*. Describe las condiciones de cambio de estado de la máquina de estados
- UC: gestiona las variables de temporización en función de los ciclos de espera especificados en los parámetros genéricos
- UP1: gestiona la dirección de memoria accedida y las señales de control de la memoria externa
- UP2: gestiona la transmisión de datos en un ciclo de escritura
- UP3: gestiona la recepción de datos en un ciclo de lectura
- UP4: gestiona la señal *data acknowledge*

#### 3.4.1.2 Fichero puente

Este fichero adapta las señales conectadas al interface del bus *wishbone* con los puertos del módulo del controlador genérico. Puesto que el fichero es algo extenso, se muestran los trozos de código importantes y se explica la causa de las conversiones.



Los grupos de señales que se deben adaptar son los siguientes:



**Imagen 40** - Esquema de puertos de entrada/salida

En varios puntos del código se utiliza la señal *cs* (*chip select*) para la gestión de otras señales. Esta señal indica si el chip de memoria externa está siendo accedido. El valor se obtiene a partir de las señales *cyc* y *stb* del interface *wishbone* y comprobando, además, que la dirección accedida se encuentra dentro del mapa de direcciones correspondiente a la memoria. El código relativo a la obtención de la señal *cs* es:

```
cycstb<=wb_cyc_i and wb_stb_i;
process(wb_adr_i,cycstb) begin
 cs<='0';
 if cycstb='1' then
 if unsigned(wb_adr_i)>=unsigned(C_BASEADDR) and
 unsigned(wb_adr_i)<=unsigned(C_HIGHADDR) then
 cs<='1';
 end if;
 end if;
end process;
```

|         |                       |                                                                                                                                                                                                                                                                                                             |
|---------|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Clk     | señal de reloj        | conexión directa                                                                                                                                                                                                                                                                                            |
| Initrst | reset inicial         | conexión directa                                                                                                                                                                                                                                                                                            |
| Strt    | start                 | señales de gestión de la máquina de estados definida en la entidad <i>sramcontrol_entity</i> del controlador de memoria                                                                                                                                                                                     |
| Rdy     | ready                 |                                                                                                                                                                                                                                                                                                             |
| Size    | tamaño de datos       | en función del valor de <i>wb_sel_i</i><br>"0001","0010","0100","1000" → size=00(8 bits)<br>"0011","1100" → size=01(16 bits)<br>"1111" → size=11(32 bits)                                                                                                                                                   |
| rNw     | lectura/escritura     | negación de la señal<br>- <i>wb_we_i</i> (0→lectura, 1→escritura)<br>- <i>rNw</i> (0→escritura, 1→lectura)                                                                                                                                                                                                  |
| Addr    | bus direcciones       | adaptación del tamaño del bus<br>- <i>wb_adr_i</i> es de 32 bits<br>- <i>Addr</i> depende del tamaño de la memoria externa ( <i>a</i> bits → 2 <sup>a</sup> = tamaño memoria)                                                                                                                               |
| Din     | bus datos (escritura) | conexión directa                                                                                                                                                                                                                                                                                            |
| Dout    | bus datos (lectura)   | en función del valor de <i>wb_sel_i</i><br>"0001","0010","0100","1000" → <i>wb_dat_o</i> =<br>= <i>Dout</i> (7:0) & <i>Dout</i> (7:0) & <i>Dout</i> (7:0) & <i>Dout</i> (7:0)<br>"0011","1100" → <i>wb_dat_o</i> = <i>Dout</i> (15:0) & <i>Dout</i> (15:0)<br>"1111" → <i>wb_dat_o</i> = <i>Dout</i> (31:0) |
| doutAck | data acknowledge      | cuando es lectura → conexión directa<br>cuando es escritura → indica si la memoria                                                                                                                                                                                                                          |



|  |  |                                                       |
|--|--|-------------------------------------------------------|
|  |  | externa está preparada (acceso de escritura especial) |
|--|--|-------------------------------------------------------|

El siguiente cuadro de texto muestra el código dedicado a la gestión de las señales *Strt* y *Rdy*, dedicadas a la gestión de la máquina de estados:

```
process(wb_rst_i,wb_clk_i,cs,ip_rdy)
 variable started: std_logic;
 begin
 ip_initrst<='0';
 ip_strt<='0';
 waiting<='0';
 if wb_rst_i='1' then
 ip_initrst<='1';
 started:='0';
 elsif rising_edge(wb_clk_i) then
 if started='0' and ip_strt='1' then
 started:='1';
 elsif started='1' and cs='0' then
 started:='0';
 end if;
 end if;
 if cs='1' and started='0' then
 if ip_rdy='1' then
 ip_strt<='1';
 else
 waiting<='1';
 end if;
 end if;
 end process;
```

Un reset del controlador mediante la señal *wb\_rst\_i* transmite la señal de reset a los puertos del controlador genérico además de inicializar la variable *started*, que indica si el controlador ha empezado un ciclo de bus de lectura o escritura. En caso que no se haya forzado el reset del dispositivo, se realiza la gestión de la variable *started* en cada flanco positivo del reloj, es decir, se aserta en el caso que la señal *Strt* lo solicite y el dispositivo se encuentre en estado inicializado (*started* = 0) o se niega en caso que no se acceda al mapa de direcciones de la memoria externa (*cs* = 0). Sin embargo, en el caso que se acceda a dicho mapa y el dispositivo se encuentre inicializado, se aserta la señal *Strt* (iniciar ciclo de bus) si el dispositivo está preparado (*Rdy* = 1) o se entra en espera (*waiting* = 1) hasta que el dispositivo esté preparado.

### 3.4.1.3 Modelo de memoria

Para poder simular el sistema con *ModelSim*, se debe crear un modelo de memoria externa, es decir, un módulo escrito en lenguaje *hdl* que imite el comportamiento de un *chip* de memoria ram. De esta forma, se podrán simular los accesos a memoria externa sin conectar realmente el sistema al exterior. El código fuente del modelo *vhdl* se encuentra en el fichero *sram\_sim.vhd* y se explica a continuación:

```

entity sram_sim is
 generic(
 init: t_sram:=(others=>sram_word_U);
 delay: time:=0 ns);
 port(
 Ncs,Noe: in std_logic;
 Nwe: in std_logic;
 addr: in std_logic_vector(sram_addr_width-1 downto 0);
 data: inout std_logic_vector(sram_data_width-1 downto 0));
end sram_sim;

```

La entidad *sram\_sim* define el mapa de genéricos y de puertos del módulo representado en la imagen lateral. Se destaca la naturaleza negada de lógica utilizada para las señales de control, además del carácter bidireccional del bus de datos, utilizado tanto para la lectura como para la escritura. La anchura del bus de direcciones depende del tamaño de la memoria externa conectada.

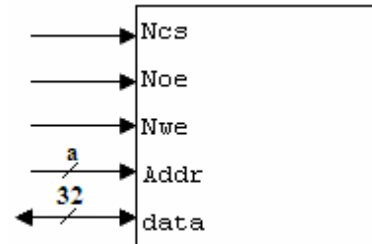


Imagen 41 - Entidad *sram\_sim*

El siguiente código define el comportamiento de la entidad *sram\_sim*:

```

architecture beh1 of sram_sim is
begin
 process
 variable ram: t_sram:=init_sram(ADDRESS0,FINGERPRINT_EXAMPLE);
 variable index: integer;
 begin
 wait on Ncs,Noe,Nwe,addr,data;
 data<=(others=>'X');
 if Ncs='0' then
 index:=conv_integer(unsigned(addr));
 if Nwe='0' then
 data<=(others=>'Z');
 wait for delay;
 ram(index):=data;
 else
 if Noe='0' then
 data<=ram(index) after delay;
 else
 data<=(others=>'Z');
 end if;
 end if;
 else
 data<=(others=>'Z');
 end if;
 end process;
end beh1;

```

Se trata de un comportamiento secuencial definido dentro de un proceso. Se definen la variable *ram* (del tipo *t\_ram*), obtenida por la función de inicialización que se describirá en el subapartado siguiente, y la variable entera *index*, utilizada para seleccionar la posición accedida de la memoria, es decir, de la variable *ram*. Se inicia el proceso cuando cualquiera de los puertos del módulo modifique su valor y, entonces, todos los bits del bus bidireccional de datos toman el valor 'X' (indefinido). El comportamiento es el siguiente:

- Si no se accede a la memoria externa (*Ncs=1*) el bus de datos se pone a alta impedancia ('Z').

- Si se accede a la memoria externa para escritura ( $Ncs=0$  y  $Nwe=0$ ), tras esperar un retardo (*delay*), se escriben los datos en la dirección de memoria indicada por *addr* y el bus de datos pasa a alta impedancia.
- Si se accede a la memoria externa para lectura ( $Ncs=0$  y  $Nwe=1$ ), tras esperar un retardo y si la salida está habilitada ( $Noe=0$ ), se leen los datos señalados por la dirección de memoria indicada por *addr* y se guardan en el bus de datos.

#### 3.4.1.4 Inicialización del modelo de memoria

En la siguiente sección del proyecto se pretende obtener un sistema basado en el procesador *or1200* que ejecuta un programa mapeado en memoria interna y un coprocesador que obtiene los datos de huellas dactilares de memoria externa. Por ello, es interesante inicializar el modelo de memoria externa con los datos de al menos una huella dactilar. La inicialización se realiza mediante una función llamada *init\_sram* definida en el fichero *sram\_sim.vhd* y un fichero (*fingerprint.vhd*) que contiene funciones y los datos de la huella en el formato adecuado. La función de inicialización es la siguiente:

```
function init_sram(ADDR_IMG0:std_logic_vector; fingerprint:T_FINGERPRINT)
return t_sram is
 variable i: integer;
 variable val_ram: t_sram;
 variable ADDR0_IMG: unsigned(sram_addr_width-1 downto 0):=
 unsigned(ADDR_IMG0(sram_addr_width-1 downto 0));
 variable ADDR1_IMG: unsigned(sram_addr_width-1 downto 0):=
 ADDR0_IMG+X"FFFF"; --64kB/fingerprint
begin
 for i in conv_integer(ADDR0_IMG) to conv_integer(ADDR1_IMG) loop
 val_ram(i):=fingerprint(i-conv_integer(ADDR0_IMG));
 end loop;
 return val_ram;
end function;
```

Los argumentos de la función *init\_sram* son:

- ADDR\_IMG0: se trata de un vector lógico que codifica una dirección de memoria que apunta donde se deben escribir los datos de inicialización
- fingerprint: se trata de un array de 65536 vectores lógicos (256x256) que contiene los datos de la huella dactilar inicial

Se definen las variables de la función como:

- i: entero
- val\_ram: array de palabras de 8 bits de  $2^a$  posiciones (a: anchura del bus de direcciones)
- ADDR0\_IMG: codifica la dirección inicial en *a* bits
- ADDR1\_IMG: codifica la dirección final en *a* bits

La función inicializa las FFFF posiciones de la variable *val\_ram* a partir del array de enteros pasados a la función y devuelve la variable inicializada.

El fichero *fingerprint.vhd* contiene el paquete *fingerprint* que:

- Define los tipos de datos T\_FINGERPRINT y T\_INT\_FINGERPRINT (array de vectores lógicos y array de enteros)
- Define la función CONV\_FINGERPRINT que realiza la conversión para obtener un array de vectores de 8 bits a partir de un array de enteros:

```

function CONV_FINGERPRINT(a:T_INT_FINGERPRINT) return
T_FINGERPRINT is
 variable i: integer;
 variable s: T_FINGERPRINT;
begin
 for i in 0 to 256*256-1 loop
 s(i):=conv_std_logic_vector(a(i),8);
 end loop;
 return s;
end function;

```

- Define la constante `INT_FINGERPRINT` del tipo `T_INT_FINGERPRINT`. Se trata de un array de 65536 vectores lógicos que codifica los datos de una huella dactilar.

### 3.4.1.5 Conexión al bus

El dispositivo que se pretende conectar al bus *wishbone* es un controlador de memoria conectado a un chip de memoria externo a la *fpga*. El controlador de memoria se conecta al *slave1* del árbitro del bus mediante las señales del interface *wishbone*. Las señales del controlador destinadas a la comunicación con el *chip* externo se mapean a los puertos de entrada/salida de la *fpga*. Para la simulación se debe conectar el modelo de memoria externa a los puertos I/O de la *fpga*. El proceso de conexión se describe en los siguientes pasos:

- Añadir los puertos I/O de la *fpga* (`Ncs`, `Noe`, `Nwe`, `Nbe`, `addr` y `data`) en la definición del módulo *xsv\_fpga\_top* del fichero con el mismo nombre.

```

module xsv_fpga_top (
 clk, rstn,
 sw,
 uart_stx, uart_srx,
 sram_Ncs, sram_Noel, sram_Nwe, sram_Nbe, sram_addr,
 sram_data,
 flash_Ncs,
 jtag_tck, jtag_tms, jtag_tdi, jtag_trst,
 jtag_tvref, jtag_tgnd, jtag_tdo);

```

- Definir el sentido de los puertos añadidos.

```

output sram_Ncs, sram_Noel, sram_Nwe;
output [3:0] sram_Nbe;
output [`SRAM_EXT_AW+1:2] sram_addr;
inout [31:0] sram_data;

```

El bus de datos del chip de memoria es bidireccional.

La anchura del bus de direcciones depende del tamaño de la memoria utilizada. El número de palabras de 32 bits contenidas en la memoria es  $2^{\text{SRAM\_EXT\_AW}}$ .

- Eliminar la línea dedicada a inhabilitar el *chip* de memoria externa en la placa:  
`assign sram_Ncs = 1'b1;`
- Añadir el cableado dedicado a conectar el interface *wishbone* del controlador de memoria al *slave1* del *Traffic Cop* (árbitro del bus)

```

wire [31:0] wb_res_dat_i;
wire [31:0] wb_res_dat_o;
wire [31:0] wb_res_adr_i;
wire [3:0] wb_res_sel_i;
wire wb_res_we_i;
wire wb_res_cyc_i;
wire wb_res_stb_i;
wire wb_res_ack_o;

```

- Instanciar el controlador de memoria externa.

```

wb_sramcontrol ramext_controller(
 .wb_clk_i (wb_clk),
 .wb_rst_i (wb_rst),
 .wb_dat_i (wb_res_dat_i),
 .wb_dat_o (wb_res_dat_o),
 .wb_adr_i (wb_res_adr_i),
 .wb_sel_i (wb_res_sel_i),
 .wb_we_i (wb_res_we_i),
 .wb_cyc_i (wb_res_cyc_i),
 .wb_stb_i (wb_res_stb_i),
 .wb_ack_o (wb_res_ack_o),
 .wb_err_o (wb_res_err_o),
 .sram_Ncs (sram_Ncs),
 .sram_Nwe (sram_Nwe),
 .sram_No (sram_No),
 .sram_Nbe (sram_Nbe),
 .sram_addr (sram_addr),
 .sram_data_I (sram_data_I),
 .sram_data_O (sram_data_O),
 .sram_data_T (sram_data_T));

```

El bus de datos no es bidireccional. Se trata de un bus de entrada, un bus de salida y una señal de control. Por lo tanto, se define un *tri-state buffer* para adaptar los tipos de bus.

- Añadir el cableado interno destinado a conectar los puertos I/O de la *fpga*.

```

wire sram_Ncs, sram_No, sram_Nwe;
wire [3:0] sram_Nbe;
wire [`SRAM_EXT_AW+1:2] sram_addr;
wire [31:0] sram_data;
wire [31:0] sram_data_I;
wire [31:0] sram_data_O;
wire sram_data_T;

```

- Conectar el controlador al *slave1* del *Traffic Cop* en la instancia del árbitro del bus.

```

.tl_wb_cyc_o (wb_res_cyc_i),
.tl_wb_stb_o (wb_res_stb_i),
.tl_wb_cab_o (wb_res_cab_i),
.tl_wb_adr_o (wb_res_adr_i),
.tl_wb_sel_o (wb_res_sel_i),
.tl_wb_we_o (wb_res_we_i),
.tl_wb_dat_o (wb_res_dat_i),
.tl_wb_dat_i (wb_res_dat_o),
.tl_wb_ack_i (wb_res_ack_o),
.tl_wb_err_i (wb_res_err_o),

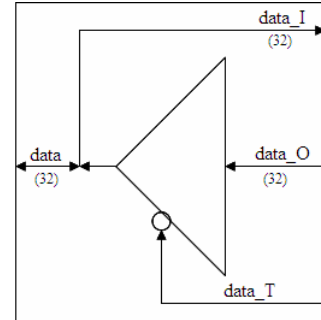
```

- Diseñar el *tri-state buffer* (*tristate.v*) dedicado a adaptar el carácter bidireccional del bus de datos que viene del chip de memoria a los dos buses de datos y la señal de control pertenecientes al controlador de memoria.

```

`define DATA_W 32
module bidirec (data_I, data_O, data_T, data);
output [`DATA_W-1:0] data_I;
input [`DATA_W-1:0] data_O;
input data_T;
inout [`DATA_W-1:0] data;
assign data = data_T ? `DATA_W'bZ : data_O;
assign data_I = data;
endmodule

```



- Instanciar el *tri-state buffer*.

```

bidirec bidirec (
 .data_T (sram_data_T),
 .data_I (sram_data_I),
 .data_O (sram_data_O),
 .data (sram_data));

```

- Actualizar la instancia del chip de la *fpga* en el fichero testbench (*sim\_test1.v*) según las modificaciones de los puertos en su definición.
- Añadir el cableado dedicado a la interconexión del chip de la *fpga* con el modelo del bloque de memoria externa en el fichero *testbench*.

```

wire sram_Ncs, sram_No, sram_Nwe;
wire [3:0] sram_Nbe;
wire [`SRAM_EXT_AW+1:2] sram_addr;
wire [31:0] sram_data;

```

- Instanciar el modelo de los bloques de memoria en el fichero *testbench* definido previamente en el fichero *sram\_sim.vhd*.

```

sram_sim ramext_chip3(
 .Ncs(bank_Ncs[3]),
 .Noe(sram_No),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[31:24])
);
sram_sim ramext_chip2(
 .Ncs(bank_Ncs[2]),
 .Noe(sram_No),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[23:16])
);

```

```

sram_sim ramext_chip1(
 .Ncs(bank_Ncs[1]),
 .Noe(sram_No),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[15:8])
);
sram_sim ramext_chip0(
 .Ncs(bank_Ncs[0]),
 .Noe(sram_No),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[7:0])
);

```

El chip de memoria externa tiene 8 bits de anchura. Por lo tanto, para obtener una memoria de 1kword (32 bits), se generan cuatro bloques de 1kB (8 bits) gestionados por cuatro señales del tipo *chip select* (*bank\_Ncs*). La lógica dedicada a la obtención de estas cuatro señales es:

```

assign bank_Ncs[3]= sram_Ncs | sram_Nbe[3];
assign bank_Ncs[2]= sram_Ncs | sram_Nbe[2];
assign bank_Ncs[1]= sram_Ncs | sram_Nbe[1];
assign bank_Ncs[0]= sram_Ncs | sram_Nbe[0];

```

- Definir los límites del mapa de memoria de la memoria externa en el fichero *wb\_bridge\_sramcontrol.vhd*.

```
generic (
 C_BASEADDR : std_logic_vector(31 downto 0) := X"01000000";
 C_HIGHADDR : std_logic_vector(31 downto 0) := X"010FFFFF";
 ...);
```

Se define, en este caso, un chip de memoria externa de 1MB. El mapa de memoria es: 0x01000000 – 0x010FFFFF

- Añadir la siguiente línea al fichero *xsv\_fpga\_defines.v* para actualizar el mapa de direcciones del *traffic\_cop*.

```
`define APP_ADDR_RAMEXT `APP_ADDR_DEC_W'h01
```

Sustituir la nueva constante por *APP\_ADDR\_FLASH* en los parámetros genéricos pasados en la instancia del *traffic\_cop* en el fichero *xsv\_fpga\_top.v*. El mapa de direcciones del *slave1* del *traffic\_cop* resulta de 16MB mapeados en las direcciones: 0x01000000 – 0x01FFFFFF

### 3.4.2 Software

En este apartado se pretenden preparar las herramientas de software necesarias para la compilación y el enlazado del programa. El programa será mapeado en la memoria *sram* interna y ejecutado por el microprocesador *or1200*. De la misma manera que en el sistema básico de la sección anterior del proyecto, se necesitan diferentes formatos de ficheros para la simulación mediante *ModelSim* o para el mapeado del sistema en la *fpga*. En la sección anterior se utilizaron herramientas cruzadas precompiladas para la compilación y enlazado de los ficheros escritos en lenguaje C. En este caso la situación no es la misma.

El sistema propuesto en la presente sección es un paso intermedio entre el sistema básico de la sección anterior y el sistema avanzado con coprocesador de la sección posterior. Por lo tanto, en lo relativo al *software* se pretende realizar una aproximación simplificada del comportamiento del *openrisc* definido por el programa dedicado al procesado de huellas dactilares. Éste es un programa de relativa complejidad escrito en lenguaje C++ que, entre otras operaciones, accede a registros y datos en direcciones mapeadas en memoria externa.

Por lo tanto, el código ejecutado por el microprocesador en el sistema avanzado con controlador de memoria externa es un programa simple escrito en lenguaje C++ que realiza accesos a datos con formatos y tamaños diferentes en direcciones mapeadas en memoria externa.

#### 3.4.2.1 Tutorial de generación de la GNU Toolchain

A continuación se detalla el proceso para generar una cadena de herramientas cruzadas *GNU* compatibles con los lenguajes de programación C y C++. Es importante seguir el orden de instalación estrictamente. Además, para generar un compilador *GNU* cruzado se recomienda tener un compilador *GNU* nativo (versión 2.95 o posterior) instalado en el sistema. El proceso seguido para obtener un entorno adecuado para la instalación de la *GNU Toolchain* está descrito en el apartado 3.3.2.1 del presente documento.

Para empezar se abre una consola del entorno *Cygwin*. Se crea y cambia al directorio de fuentes de las herramientas de desarrollo llamado *or1k-tools*.

```
$ cd ~
```

```
$ mkdir orlk-tools; cd orlk-tools
```

Se obtienen los ficheros fuente de las herramientas de desarrollo. En este caso no se trata de los ficheros binarios ejecutables. Los ficheros se descargan de la página web de Asisi '<http://www.asisi.co.uk/openrisc/>'.

```
or32-cvs-20051215-1645-binutils-2.16.1.tar.bz2
or32-cvs-20060130-0900-gdb-5.3.tar.bz2
or32-cvs-20051201-1252-gcc-3.4.4-newlib-1.13.0.tar.bz2
```

Se extraen las fuentes con el compresor *tar* del sistema. Las opciones especificadas son:

|          |                                     |
|----------|-------------------------------------|
| <i>x</i> | extraer ficheros de un archivo      |
| <i>j</i> | filtrar el archivo con <i>bzip2</i> |
| <i>f</i> | usar el archivo especificado        |

```
$ tar xjf or32-cvs-20051215-1645-binutils-2.16.1.tar.bz2
$ tar xjf or32-cvs-20060130-0900-gdb-5.3.tar.bz2
$ tar xjf or32-cvs-20051201-1252-gcc-3.4.4-newlib-1.13.0.tar.bz2
```

Se accede al modo *superuser* para realizar toda la generación e instalación (no requerido para Cygwin)

```
$ su
```

### *Generación de Binutils 2.16.1*

Se crea y cambia al directorio de instalación de la colección de utilidades binarias de GNU llamado *b-b*.

```
$ mkdir b-b; cd b-b
```

Se ejecuta el *script* de configuración. Las opciones especificadas son:

|                                |                                                                                        |
|--------------------------------|----------------------------------------------------------------------------------------|
| <code>--target</code>          | especifica la arquitectura destino (requerido para compiladores cruzados)              |
| <code>--prefix</code>          | especifica el directorio de instalación (por defecto <i>/usr/local</i> )               |
| <code>-v</code>                | lista los ficheros procesados                                                          |
| <code>2&gt;&amp;1   tee</code> | muestra la información de salida del proceso y la redirecciona al fichero especificado |

```
$../binutils-2.16.1/configure --target=or32-elf --prefix=/opt/or32-elf \
-v 2>&1 | tee configure.out
```

Se ejecuta el *script* de generación e instalación. Las opciones especificadas son:

|                      |                                       |
|----------------------|---------------------------------------|
| <code>-w</code>      | visualiza el directorio actual        |
| <code>all</code>     | llama al <i>script</i> de generación  |
| <code>install</code> | llama al <i>script</i> de instalación |

```
$ make -w all install 2>&1 | tee make.out
```

Se añaden los binarios generados a la ruta de ejecutables del sistema.

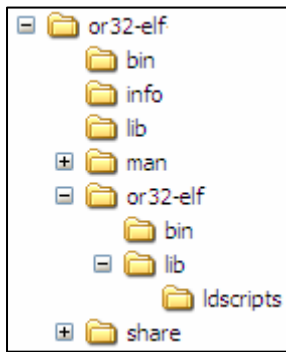
```
$ export PATH=/opt/or32-elf/bin:$PATH
```

Se retrocede al directorio *ork-tools*.

```
$ cd ..
```

Finalizado el proceso de instalación de las utilidades, la jerarquía de directorios en la ruta */opt/or32-elf* es la siguiente:





Aplicaciones instaladas en la carpeta *bin*:

*addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings y strip*

Librerías instaladas en la carpeta *lib*:

*libiberty, libbfd y libopcodes*

### Generación de GCC 3.4.4

Se crea y cambia al directorio de instalación del compilador de la GNU llamado *b-gcc*.

```
$ mkdir b-gcc; cd b-gcc
```

Se ejecuta el *script* de configuración. Las opciones especificadas son:

```
--target especifica la arquitectura destino
--prefix especifica el directorio de instalación
--enable-languages especifica que solamente un subconjunto de librerías
 runtime y compiladores debe ser instalado. Lenguajes soportados:
 all, ada, c, c++, fortran, java, objc, obj-c++, treelang
--with-gnu-as especifica GNU assembler como ensamblador utilizado
--with-gnu-ld especifica GNU linker como enlazador utilizado
--with-newlib especifica newlib como librería de C utilizada
--with-gxx-include-dir especifica el directorio de instalación de los ficheros
 de encabezamiento para G++
```

```
$../gcc-3.4.4/configure --target=or32-elf --prefix=/opt/or32-elf \
--enable-languages=c,c++ --with-gnu-as --with-gnu-ld --with-newlib \
--with-gxx-include-dir=/opt/or32-elf/include \
-v 2>&1 | tee configure.out
```

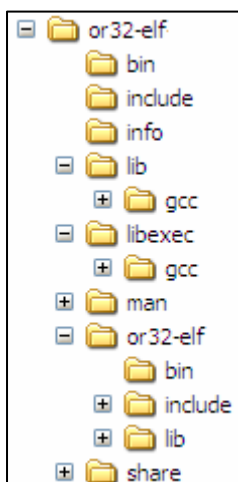
Se ejecuta el *script* de generación e instalación.

```
$ make -w all install 2>&1 | tee make.out
```

Se retrocede al directorio *or1k-tools*.

```
$ cd ..
```

Finalizado el proceso de instalación del compilador, la jerarquía de directorios en la ruta */opt/or32-elf* es la siguiente:



Aplicaciones instaladas en la carpeta *bin*:

*c++, cpp, g++, gcc-3.4.4, gcc, gccbug, gcov,*

Directorio de ficheros de encabezamiento (*includes*) para la referencia a funciones contenidas en las librerías instaladas.

### Generación de GDB 5.3

Se crea y cambia al directorio de instalación de la herramienta de depuración de GNU llamado *b-gdb*.

```
$ mkdir b-gdb; cd b-gdb
```

Se ejecuta el *script* de configuración. Existe un *bug* en la configuración de la actual versión del depurador que no soporta el uso de la opción *-prefix*. Por lo tanto, se debe copiar y renombrar manualmente el fichero ejecutable tras la instalación.

```
$../gdb-5.3/configure --target=or32-elf
```

Se ejecuta el *script* de generación.

```
$ make all
```

Se renombra y copia el ejecutable a la ruta de ejecutables de la *GNU Toolchain* cruzada.

```
$ cp gdb/gdb.exe /opt/or32-elf/bin/or32-elf-gdb.exe
```

Se retrocede al directorio *or1k-tools*.

```
$ cd ..
```

Se sale del modo *superuser* (no requerido para *Cygwin*)

```
exit
```

Se añade la siguiente línea en el fichero */etc/bash.bashrc* para que los cambios de entorno sean permanentes

```
PATH=$PATH:/opt/or32-elf/bin
```

#### 3.4.2.2 Programa simplificado

El programa utilizado para comprobar el correcto funcionamiento del sistema propuesto se compone de dos partes diferentes. El fichero *reset.S* describe la rutina de servicio a la excepción del reset del sistema escrita en código ensamblador. El código gestiona la pila y fuerza el *program counter (PC)* a la rutina *\_\_main*, desde donde se hace un salto a la primera instrucción del programa principal. El fichero *checkmem.cpp* describe el programa principal escrito en lenguaje C++. El programa realiza una serie de accesos de lectura y escritura a la memoria externa con tamaños de datos diferentes (32, 16 y 8 bits). El código fuente del fichero *checkmem.cpp* es el siguiente:

```
#define CHECK_I 0x01020000
#define CHECK_F 0x0102000F
typedef unsigned short int uword;
typedef unsigned int uint;
typedef unsigned char uchar;

int check_memoria(void *direccion_i, void *direccion_f)
{
 uint *p32;
 uword *p16;
 uchar *p8;

 for(p32=(uint*)direccion_i; p32<=((uint*)direccion_f); p32++)
 *p32=(uint)p32;
 for(p32=(uint*)direccion_i; p32<=((uint*)direccion_f); p32++)
 if(*p32!=(uint)p32)
 return -1;

 for(p16=(uword*)direccion_i; p16<=((uword*)direccion_f); p16++)
```

```

 *p16=(uword)p16;
 for(p16=((uword*)direccion_i); p16<=((uword*)direccion_f); p16++)
 if(*p16!=(uword)p16)
 return -1;

 for(p8=((uchar*)direccion_i); p8<=((uchar*)direccion_f); p8++)
 *p8=(unsigned char)p8;
 for(p8=((uchar*)direccion_i); p8<=((uchar*)direccion_f); p8++)

 if(*p8!=(uchar)p8)
 return -1;
 return 0;
 }
int main()
{
 if (check_memoria((void*)CHECK_I,(void*)CHECK_F)) return -1;
 return 0;
}

```

La rutina principal *main* realiza una llamada a la rutina de comprobación *check\_memoria* introduciendo como argumentos la dirección inicial y la dirección final del bloque de memoria que debe ser comprobado. La rutina *check\_memoria* declara tres punteros:

- *p32* apunta a registros de 32 bits (*unsigned int*)
- *p16* apunta a registros de 16 bits (*unsigned short int*)
- *p8* apunta a registros de 8 bits (*unsigned char*)

A continuación se realizan las comprobaciones:

- Se inicializa el puntero *p32* a la dirección de memoria inicial (0x01020000), se escribe la dirección en el registro de 32 bits, se incrementa el puntero y se repite la operación hasta sobrepasar la dirección de memoria final (0x0102000f). A continuación se realizan las lecturas de los registros de 32 bits y se compara el contenido almacenado con la dirección. El contenido del bloque de memoria queda:

|                 | +3       | +2 | +1 | +0 |
|-----------------|----------|----|----|----|
| <b>0102000c</b> | 0102000c |    |    |    |
| <b>01020008</b> | 01020008 |    |    |    |
| <b>01020004</b> | 01020004 |    |    |    |
| <b>01020000</b> | 01020000 |    |    |    |

- Se inicializa el puntero *p16* a la dirección de memoria inicial (0x01020000), se escribe la dirección “adaptada” en el registro de 16 bits, se incrementa el puntero y se repite la operación hasta sobrepasar la dirección de memoria final (0x0102000f). A continuación se realizan las lecturas de los registros de 16 bits y se compara el contenido almacenado con la dirección. El contenido del bloque de memoria queda:

|                 | +3   | +2 | +1   | +0 |
|-----------------|------|----|------|----|
| <b>0102000c</b> | 000e |    | 000c |    |
| <b>01020008</b> | 000a |    | 0008 |    |
| <b>01020004</b> | 0006 |    | 0004 |    |
| <b>01020000</b> | 0002 |    | 0000 |    |

- Se inicializa el puntero *p8* a la dirección de memoria inicial (0x01020000), se escribe la dirección “adaptada” en el registro de 8 bits, se incrementa el puntero

y se repite la operación hasta sobrepasar la dirección de memoria final (0x0102000f). A continuación se realizan las lecturas de los registros de 8 bits y se compara el contenido almacenado con la dirección. El contenido del bloque de memoria queda:

|                 | +3 | +2 | +1 | +0 |
|-----------------|----|----|----|----|
| <b>0102000c</b> | 0f | 0e | 0d | 0c |
| <b>01020008</b> | 0b | 0a | 09 | 08 |
| <b>01020004</b> | 07 | 06 | 05 | 04 |
| <b>01020000</b> | 03 | 02 | 01 | 00 |

### 3.4.2.3 Ficheros *ELF*

Para la obtención de los ficheros de salida en fomato ELF se utilizan las herramientas de la *GNU* compatibles con C++ generadas en apartados anteriores. Para ello se utiliza el fichero *script* llamado *Makefile* mostrado en el cuadro siguiente:

```
ifndef CROSS_COMPILE
CROSS_COMPILE = or32-elf-
CC = $(CROSS_COMPILE)gcc
LD = $(CROSS_COMPILE)ld
NM = $(CROSS_COMPILE)nm
endif
export CROSS_COMPILE

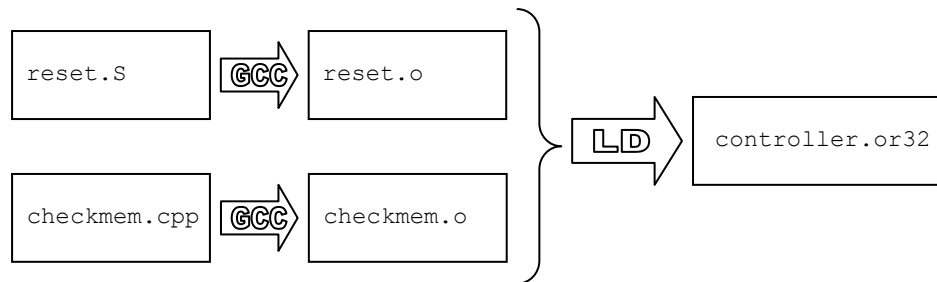
all: controller.or32
checkmem.o: checkmem.cpp Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
reset.o: reset.S Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
controller.or32: checkmem.o reset.o Makefile
 $(LD) -Tram.ld -o $@ checkmem.o reset.o $(LIBS)
System.map: controller.or32
 @$ (NM) $< | \
 grep -v '\(compiled\)\|\(\.o$$\)\|\([aUw]
 \)\|\(\.ng$$\)\|\(LASH[RL]DI\)' | \
 sort > System.map

clean:
 find . -type f \
 \(-name 'core' -o -name '*.bak' -o -name '*~' \
 -o -name '*.o' -o -name '*.a' -o -name '*.tmp' \
 -o -name '*.or32' -o -name '*.bin' -o -name '*.srec' \
 -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
 -o -name '*.aux' -o -name '*.log' \) -print \
 | xargs rm -f
 rm -f System.map
distclean: clean
 find . -type f \
 \(-name .depend -o -name '*.srec' -o -name '*.bin' \
 -o -name '*.pdf' \) \
 -print | xargs rm -f
 rm -f $(OBJJS) *.bak tags TAGS
 rm -fr *.*~
```

En realidad, se trata del mismo fichero utilizado en la sección anterior al que se le han realizado algunas adaptaciones. Se ha modificado la constante `CROSS_COMPILE` para que el sistema utilice las herramientas cruzadas recientemente generadas (compilador compatible con lenguaje C++) `or32-elf-`. Además, se ha adaptado el código para que el *script* realice la compilación y el enlazado de los ficheros fuente adecuados (*checkmem.cpp* y *reset.S*). Para el enlazado se utiliza el fichero *ram.ld*,

idéntico al utilizado en la sección anterior. Las líneas ejecutadas en la consola de *Cygwin* son:

```
cd c:/cygwin/Proyectos/program_SA_controller
make clean all
```



**Imagen 42** - Diagrama de flujo de archivos

El fichero de salida del *script* es *controller.or32*. Se trata del fichero en formato ELF utilizado para la programación de la *fpga* y para la obtención de los ficheros necesarios para la simulación del sistema con *ModelSim*. Como se ha explicado en el apartado *Software* de la sección anterior, se puede visualizar el contenido del fichero ELF en distintos formatos utilizando las líneas de comandos siguientes en una consola de *Cygwin*:

```
$ or32-elf-readelf -a controller.or32 > program.txt
$ or32-elf-objdump -d -S controller.or32 > disassemble.txt
```

La información volcada en el fichero *program.txt* se muestra en el cuadro de texto siguiente:

```

ELF Header:
 Magic: 7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00
 Class: ELF32
 Data: 2's complement, big endian
 Version: 1 (current)
 OS/ABI: UNIX - System V
 ABI Version: 0
 Type: EXEC (Executable file)
 Machine: OpenRISC
 Version: 0x1
 Entry point address: 0x2000
 Start of program headers: 52 (bytes into file)
 Start of section headers: 19632 (bytes into file)
 Flags: 0x0
 Size of this header: 52 (bytes)
 Size of program headers: 32 (bytes)
 Number of program headers: 1
 Size of section headers: 40 (bytes)
 Number of section headers: 14
 Section header string table index: 11

Section Headers:
[Nr] Name Type Addr Off Size ES Flg Lk Inf Al
[0] NULL 00000000 000000 000000 00 0 0 0 0
[1] .vectors PROGBITS 00000000 002000 000128 00 AX 0 0 1
[2] .rel.vectors REL 00000000 00508c 000000 08 12 1 4
[3] .text PROGBITS 00002000 004000 0003e4 00 AX 0 0 4
[4] .data PROGBITS 000023e4 0043e4 000000 00 WA 0 0 1
[5] .bss NOBITS 000023e4 0043e4 000000 00 WA 0 0 1
[6] .stack NOBITS 000023e4 0043e4 001000 00 WA 0 0 1
[7] .stab PROGBITS 00000000 0043e4 000360 0c 9 0 4
[8] .rel.stab REL 00000000 00508c 000000 08 12 7 4
[9] .stabstr STRTAB 00000000 004744 0004fc 00 0 0 1

```

|      |           |          |          |        |        |    |    |    |   |
|------|-----------|----------|----------|--------|--------|----|----|----|---|
| [10] | .comment  | PROGBITS | 00000000 | 004c40 | 000012 | 00 | 0  | 0  | 1 |
| [11] | .shstrtab | STRTAB   | 00000000 | 004c52 | 00005c | 00 | 0  | 0  | 1 |
| [12] | .symtab   | SYMTAB   | 00000000 | 004ee0 | 000150 | 10 | 13 | 17 | 4 |
| [13] | .strtab   | STRTAB   | 00000000 | 005030 | 00005a | 00 | 0  | 0  | 1 |

Key to Flags:  
W (write), A (alloc), X (execute), M (merge), S (strings)  
I (info), L (link order), G (group), x (unknown)  
O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

Program Headers:

| Type | Offset   | VirtAddr   | PhysAddr   | FileSiz | MemSiz  | Flg | Align  |
|------|----------|------------|------------|---------|---------|-----|--------|
| LOAD | 0x002000 | 0x00000000 | 0x00000000 | 0x023e4 | 0x033e4 | RWE | 0x2000 |

Section to Segment mapping:  
Segment Sections...  
00 .vectors .text .stack

There is no dynamic section in this file.  
There are no relocations in this file.  
There are no unwind sections in this file.

Symbol table '.symtab' contains 21 entries:

| Num: | Value    | Size | Type    | Bind   | Vis     | Ndx | Name                    |
|------|----------|------|---------|--------|---------|-----|-------------------------|
| 0:   | 00000000 | 0    | NOTYPE  | LOCAL  | DEFAULT | UND |                         |
| 1:   | 00000000 | 0    | FILE    | LOCAL  | DEFAULT | ABS | checkmem.cpp            |
| 2:   | 00002000 | 0    | SECTION | LOCAL  | DEFAULT | 3   |                         |
| 3:   | 000023e4 | 0    | SECTION | LOCAL  | DEFAULT | 4   |                         |
| 4:   | 000023e4 | 0    | SECTION | LOCAL  | DEFAULT | 5   |                         |
| 5:   | 000023e4 | 0    | NOTYPE  | LOCAL  | DEFAULT | 3   | Letext                  |
| 6:   | 00000000 | 0    | SECTION | LOCAL  | DEFAULT | 7   |                         |
| 7:   | 00000000 | 0    | SECTION | LOCAL  | DEFAULT | 9   |                         |
| 8:   | 00000000 | 0    | SECTION | LOCAL  | DEFAULT | 10  |                         |
| 9:   | 00000000 | 0    | FILE    | LOCAL  | DEFAULT | ABS | reset.S                 |
| 10:  | 000023e4 | 0    | SECTION | LOCAL  | DEFAULT | 3   |                         |
| 11:  | 000023e4 | 0    | SECTION | LOCAL  | DEFAULT | 4   |                         |
| 12:  | 000023e4 | 0    | SECTION | LOCAL  | DEFAULT | 5   |                         |
| 13:  | 000023e4 | 0    | SECTION | LOCAL  | DEFAULT | 6   |                         |
| 14:  | 000033e4 | 0    | NOTYPE  | LOCAL  | DEFAULT | 6   | _stack                  |
| 15:  | 00000000 | 0    | SECTION | LOCAL  | DEFAULT | 1   |                         |
| 16:  | 00000100 | 0    | NOTYPE  | LOCAL  | DEFAULT | 1   | _reset                  |
| 17:  | 000033e4 | 0    | NOTYPE  | GLOBAL | DEFAULT | 6   | _src_addr               |
| 18:  | 00002000 | 888  | FUNC    | GLOBAL | DEFAULT | 3   | __Z13check_memoriaPvS__ |
| 19:  | 00000120 | 0    | NOTYPE  | GLOBAL | DEFAULT | 1   | __main                  |
| 20:  | 00002378 | 108  | FUNC    | GLOBAL | DEFAULT | 3   | main                    |

### 3.4.3 Simulación Funcional

En este apartado se pretende comprobar el correcto funcionamiento del sistema diseñado mediante la simulación funcional. Con este objetivo, se debe conectar el modelo de memoria ram externa diseñado en el apartado de *hardware*, mapear la memoria interna con el programa escrito en el apartado de *software* mediante una función de inicialización, simular el fichero de *testbench* sobre el sistema con el programa *ModelSim* y comprobar la correcta ejecución del programa en el microprocesador, es decir, que los accesos a memoria externa son correctos.

### 3.4.3.1 Creación del proyecto de simulación

Con el propósito de aprovechar el trabajo realizado en la sección anterior con la simulación funcional del sistema básico, se realiza una copia del proyecto de simulación *modelsim\_SB* y se renombra como *modelsim\_controlador*. Se edita el fichero de proyecto *modelsim.mpf* y se modifican las rutas de los ficheros fuente para que *ModelSim* utilice los ficheros fuente del sistema avanzado con controlador, es decir, se reemplaza *rtl\_SB* por *rtl\_SA\_controlador*. En este punto se añaden al proyecto los ficheros del controlador de memoria ram externa (todos escritos en lenguaje *vhdl*):

- *sramcontrol\_beh2.vhd*
- *sramcontrol\_beh3.vhd*
- *sramcontrol\_entity.vhd*
- *wb\_bridge\_sramcontrol.vhd*

A su vez, el fichero que modela la memoria ram externa (*sram\_sim.vhd*) y la inicializa (*fingerprint.vhd*), el fichero que define el comportamiento del *buffer tri-state* (*tristate.v*) y el fichero de *testbench* son añadidos a la jerarquía de nivel superior del proyecto de simulación.



Imagen 43 - Ventana *workspace* del proyecto de simulación de *ModelSim*

La imagen muestra el nivel superior en la jerarquía del proyecto de simulación. Los ficheros y carpetas describen:

- *xsv\_fpga\_top.v* definición del módulos de la *fpga* e instancia de sus módulos principales
- *tc\_top.v* definición del módulo del árbitro del bus
- *sram\_sim.vhd* definición del modelo de memoria externa usado para simulación
- *sim\_test1.v* fichero de *testbench*
- *fingerprint.vhd* funciones y datos para la inicialización del modelo de memoria externa
- *tristate.v* definición del *buffer tri-state*
- *or1200* carpeta que contiene los ficheros referentes al módulo del microprocesador *or1200*
- *dbg\_interface* carpeta que contiene los ficheros referentes al módulo del interface del depurador

- *uart16550* carpeta que contiene los ficheros referentes al módulo del interface de la uart del PC
- *4sram8* carpeta que contiene los ficheros referentes al módulo del controlador de memoria interna y los ficheros dedicados a su simulación
- *ramext\_control* carpeta que contiene los ficheros referentes al módulo del controlador de memoria externa y los ficheros dedicados a su simulación

### 3.4.3.2 Fichero *testbench*

El fichero de simulación del sistema es editado para realizar la estimulación de las entradas y la conexión de los diferentes bloques correctamente. Las modificaciones realizadas en el fichero son las listadas a continuación:

- Añadir al chip de la *fpga* los puertos de entrada y/o salida utilizados para la comunicación con el chip de memoria externa
- Crear el cableado necesario para la conexión de los chips y la lógica de adaptación.
- Instanciar los cuatro bloques del modelo de memoria ram externa
- Añadir la lógica de gestión de las señales de *chip select* de los bloques

El código resultante en el fichero de *testbench* es:

```
`define SRAM_EXT_AW 18
module sim_test1;

reg clk, Nrst;
reg[2:1] sw;
wire uart_srx;
wire uart_stx;
wire sram_Ncs, sram_No, sram_Nwe;
wire [3:0] sram_Nbe;
wire [`SRAM_EXT_AW+1:2] sram_addr;
wire [31:0] sram_data;
wire flash_Ncs;
wire jtag_tck, jtag_tms, jtag_tdi, jtag_trst;
wire jtag_tvref, jtag_tgnd, jtag_tdo;
wire [3:0] bank_Ncs;
assign uart_stx=1'b0;
assign {jtag_tck,jtag_tms,jtag_tdi,jtag_trst}=1'b0;

assign bank_Ncs[3]= sram_Ncs | sram_Nbe[3];
assign bank_Ncs[2]= sram_Ncs | sram_Nbe[2];
assign bank_Ncs[1]= sram_Ncs | sram_Nbe[1];
assign bank_Ncs[0]= sram_Ncs | sram_Nbe[0];

xsv_fpga_top chip_fpga (
 .clk(clk),
 .rstn(Nrst),
 .sw(sw),
 .uart_srx(uart_srx),
 .uart_stx(uart_stx),
 .sram_Ncs(sram_Ncs),
 .sram_No(sram_No),
 .sram_Nwe(sram_Nwe),
 .sram_Nbe(sram_Nbe),
 .sram_addr(sram_addr),
 .sram_data(sram_data),
 .flash_Ncs(flash_Ncs),
```



```

 .jtag_tck(jtag_tck),
 .jtag_tms(jtag_tms),
 .jtag_tdi(jtag_tdi),
 .jtag_trst(jtag_trst),
 .jtag_tvref(jtag_tvref),
 .jtag_tgnd(jtag_tgnd),
 .jtag_tdo(jtag_tdo)
);

 sram_sim ramext_chip3(
 .Ncs(bank_Ncs[3]),
 .Noe(sram_Noel),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[31:24])
);
 sram_sim ramext_chip2(
 .Ncs(bank_Ncs[2]),
 .Noe(sram_Noel),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[23:16])
);

 sram_sim ramext_chip1(
 .Ncs(bank_Ncs[1]),
 .Noe(sram_Noel),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[15:8])
);
 sram_sim ramext_chip0(
 .Ncs(bank_Ncs[0]),
 .Noe(sram_Noel),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[7:0])
);

 initial begin
 clk=1'b0;
 sw=2'b00;
 Nrst=1'b0;
 #100 Nrst=1'b1;
 end

 always begin
 #10 clk=!clk;
 end
endmodule

```

### 3.4.3.3 Inicialización de la memoria interna

En este apartado se pretende generar el fichero escrito en lenguaje *vhdl* con el formato adecuado para poder ser utilizado para inicializar la memoria interna del sistema. A partir del fichero ELF (*controller.or32*) obtenido tras el compilado y el enlazado de los ficheros fuente se realizan dos conversiones de formato de los datos. En este fin, se puede reciclar el trabajo realizado en la sección anterior y utilizar la aplicación escrita en lenguaje C. Se introduce la línea de código “*or32-elf-objcopy -O binary controller.or32 controller.bin*” en la consola de *Cygwin* para convertir el formato del fichero ELF en formato binario. A continuación se llama al ejecutable de la aplicación creada en la sección anterior mediante la línea de código

“bin2vhd controller.bin controller.vhd” para convertir los datos binarios al formato de carácter hexadecimal deseado.

### 3.4.3.4 Comprobación del funcionamiento

En este punto se pretende comprobar que el microprocesador realiza los accesos de lectura y de escritura a memoria externa de forma correcta. El sistema responde de la misma manera a la reinicialización que el sistema básico de la sección anterior. Tras el reset del sistema, el microprocesador realiza un acceso de lectura al vector de excepción de reset (dirección 0x100) mapeado en la memoria interna del sistema. El interface de instrucciones (IWB) del *or1200* realiza el *fetch* de las instrucciones mapeadas tras el vector (definidas en el fichero fuente *reset.S*) hasta que el programa fuerza un *fetch* de la primera instrucción del programa principal (definido en el fichero fuente *checkmem.cpp*).

En la rutina principal se realiza el *fetch* de las primeras instrucciones del programa (mapeadas en memoria interna) alternando con accesos de lectura y escritura a datos (también mapeados en memoria interna) cuando las instrucciones lo determinan. A los 1450ns de simulación el *or1200* realiza un *fetch* de la primera instrucción de la función *check\_memoria* (dirección 0x2000) y es en el tiempo de simulación de 2100ns cuando el procesador entra en el primer bucle de accesos a memoria externa de la función. A los 3140ns se realiza el primer acceso a memoria externa:

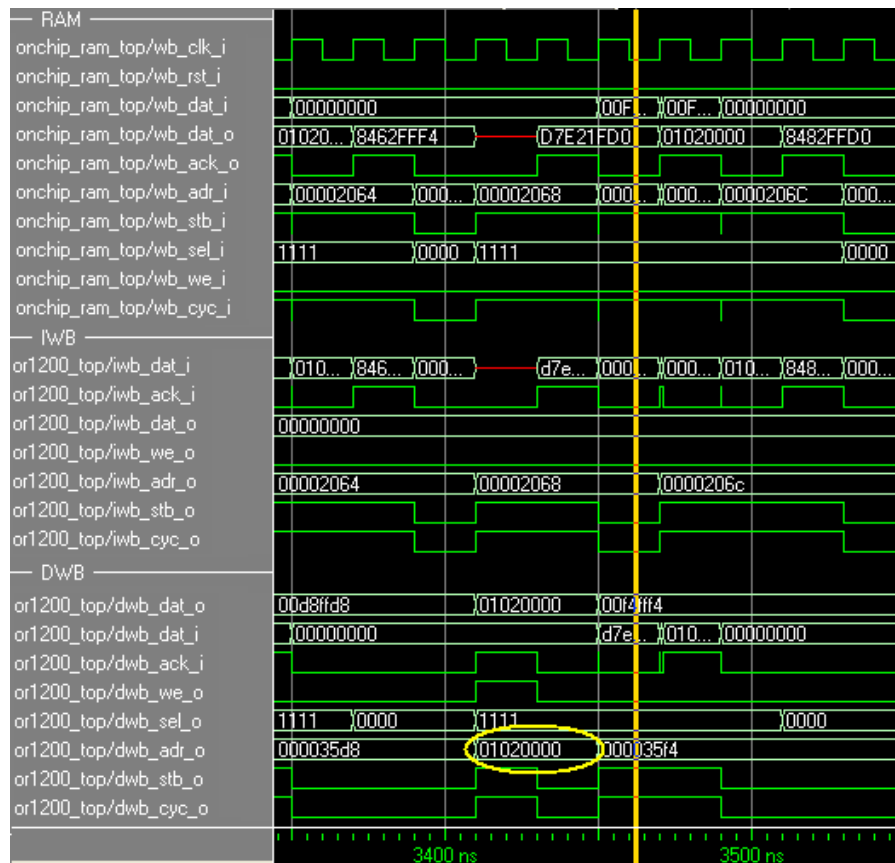
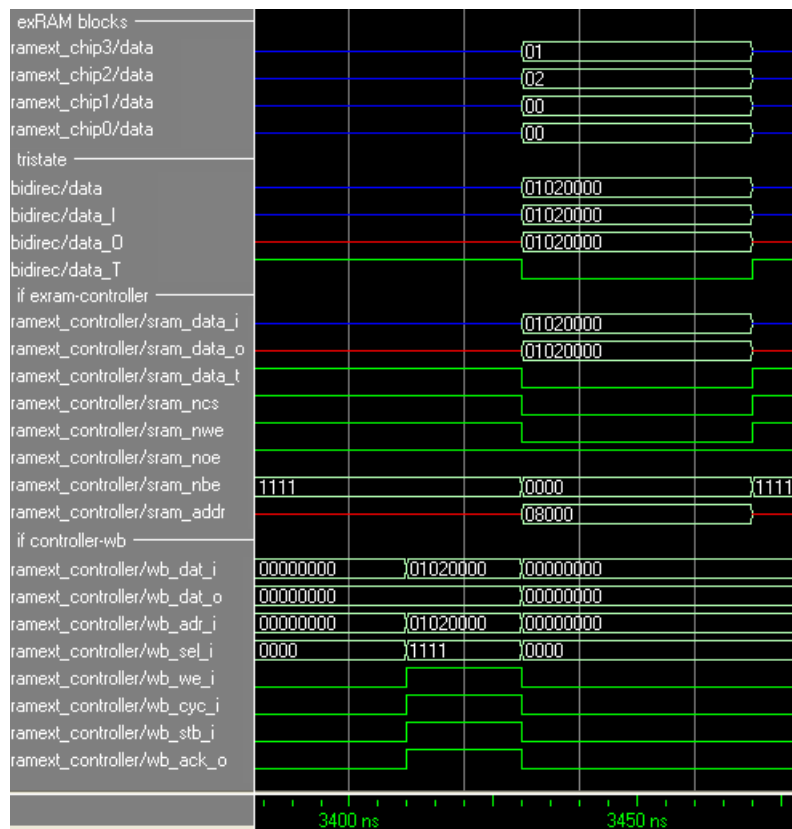


Imagen 44 - Detalle de acceso simultáneo a instrucciones y datos

En la imagen se observa que en el tiempo de simulación de 3410 ns el procesador realiza el *fetch* de la instrucción mapeada en la dirección 0x00002068 además de, al mismo tiempo, realizar un acceso de escritura de datos a la dirección 0x01020000 (mapeada en memoria externa). Como se ha explicado en apartados anteriores, el

*or1200* posee interfaces separados para los accesos a datos (*dwb*) y los accesos a instrucciones (*iwb*). Además, la topología de bus definida contempla la posibilidad de trabajar con instrucciones y datos de forma simultánea en situaciones específicas. Estas situaciones tienen lugar cuando el *or1200* pretende realizar un *fetch* de la siguiente instrucción de su programa (almacenado en memoria interna) y cualquier otro *master* del bus (excepto *iwb* del *or1200*) pretende realizar un acceso de lectura o escritura de datos a una posición de memoria no perteneciente al mapa de direcciones de la memoria interna del sistema.

En el caso específico de la simulación anterior, a los 3410ns el *iwb* inicia un ciclo de lectura en la dirección 0x00002068 de la memoria interna y realiza el *fetch* de la instrucción 0xd7e21fd0 en el siguiente ciclo de reloj. Al mismo tiempo, el *dwb* realiza un acceso de escritura en la dirección 0x01020000 de la memoria externa del dato 0x01020000.



**Imagen 45 - Detalle de acceso de escritura de 32 bits a memoria**

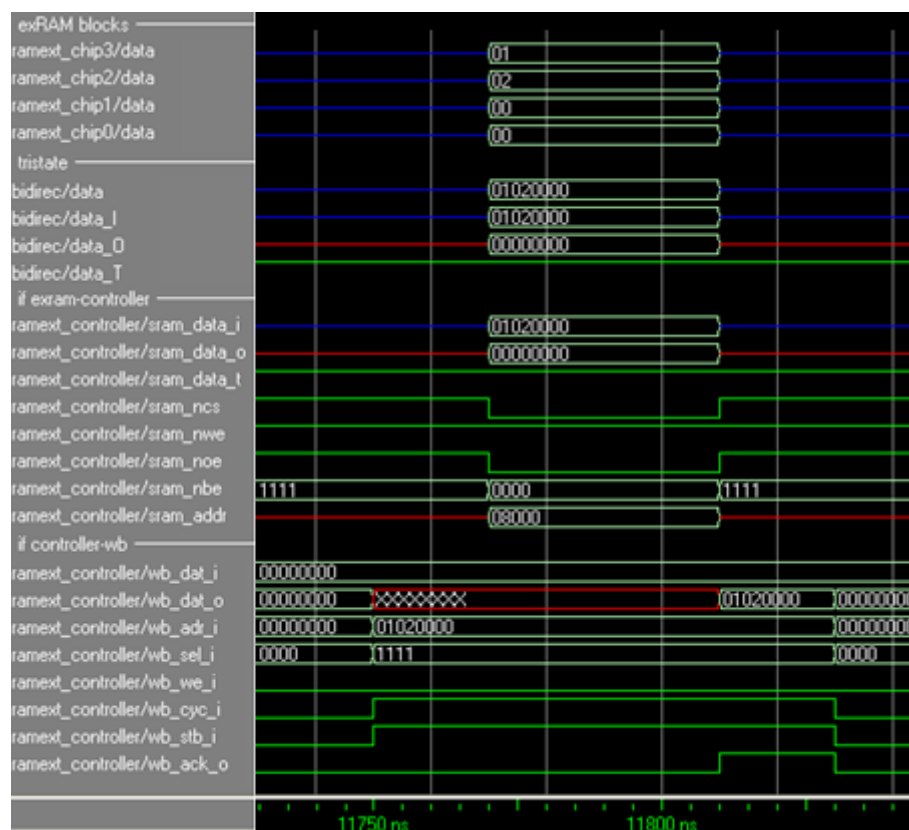
La imagen muestra el detalle ampliado de la simulación de un acceso de escritura de un dato de 32 bits a una dirección mapeada en memoria externa. Se pueden distinguir cuatro grupos de señales:

- *if controller-wb* interface entre el controlador y el bus wishbone
- *if exram-contr* interface entre el controlador y los puertos I/O de la *fpga*
- *tristate* puertos del *buffer tri-state* que adapta el bus de datos
- *exram blocks* buses de datos de los cuatro bloques de memoria externa

El controlador de memoria externa recibe las señales del bus por el interface cuando el árbitro del bus lo permite ( $0x01000000 < @ < 0x01FFFFFF$ ). En el siguiente flanco positivo del reloj, el controlador introduce las señales que salen por los puertos I/O de la *fpga*, convirtiendo en el *buffer* los buses de datos de entrada y salida a un

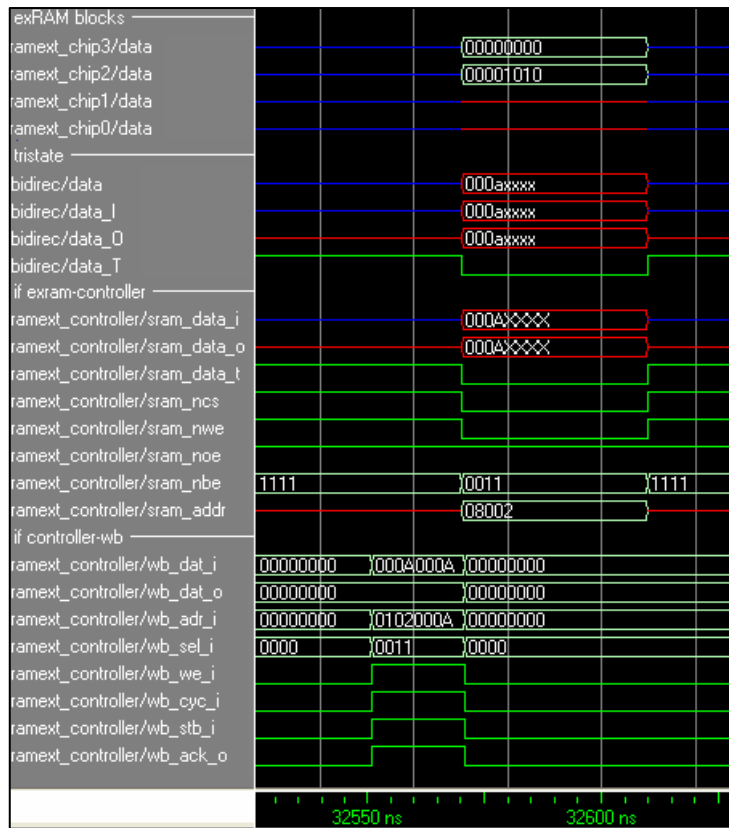
único bus bidireccional (*data*). El contenido del bus de datos se distribuye correctamente en los cuatro bloques de memoria externa.

La imagen muestra el detalle ampliado de la simulación de un acceso de lectura de un dato de 32 bits a una dirección mapeada en memoria externa. El controlador “despierta” cuando recibe la asertación de la señal *wb\_cyc\_i* del bus. Entonces comprueba las demás entradas y determina que se trata de un acceso de lectura (*wb\_we\_i* = 0) de 32 bits (*wb\_sel\_i* = 1111) a la dirección 0x01020000 (*wb\_adr\_i* con *wb\_stb\_i* = 1). En el siguiente flanco positivo de reloj el controlador “traduce” la dirección de memoria e introduce las señales de gestión de la memoria externa. El controlador recibe los 32 bits de datos del *buffer bidireccional*, una vez obtenidos los cuatro *bytes* de datos de los bloques de memoria. El controlador presenta los datos en el bus *wishbone* en el siguiente flanco positivo del reloj.

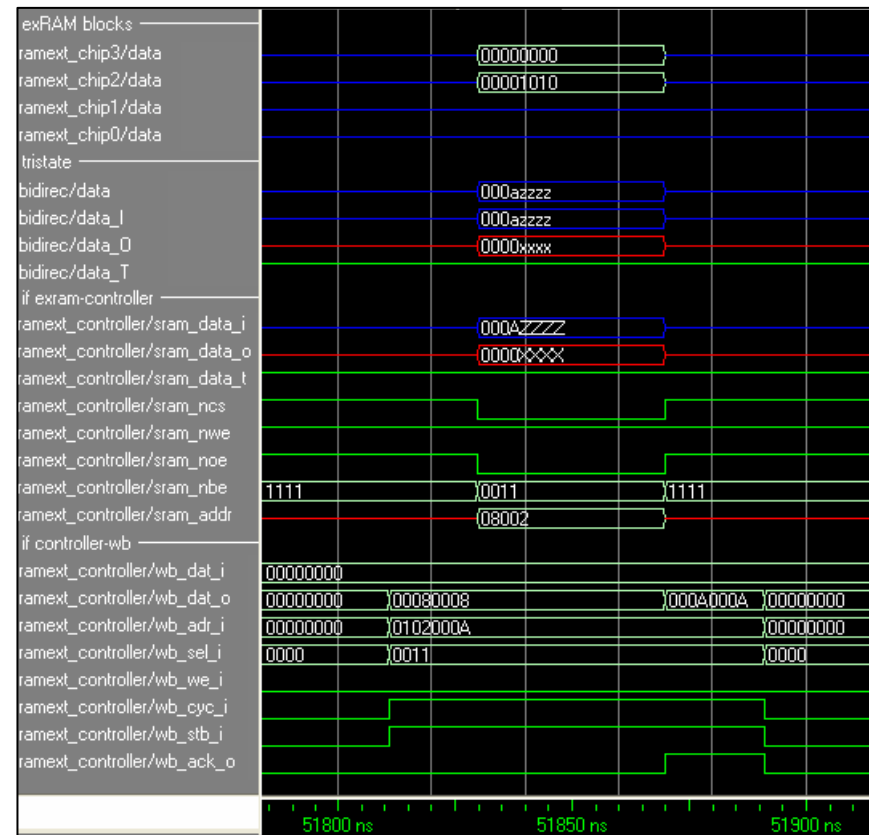


**Imagen 46 - Detalle de acceso de lectura de 32 bits en memoria externa**

Las imágenes siguientes muestran el detalle ampliado de la simulación de un acceso de escritura de 16 bits, un acceso de lectura de 16 bits, un acceso de escritura de 8 bits y un acceso de lectura de 8 bits respectivamente a una dirección mapeada en memoria externa. El procedimiento es similar con valores diferentes de *wb\_sel\_i* y una distribución de los datos diferente.



**Imagen 48** - Detalle de acceso de escritura de 16 bits a memoria externa



**Imagen 47** - Detalle de acceso de lectura de 16 bits en memoria externa

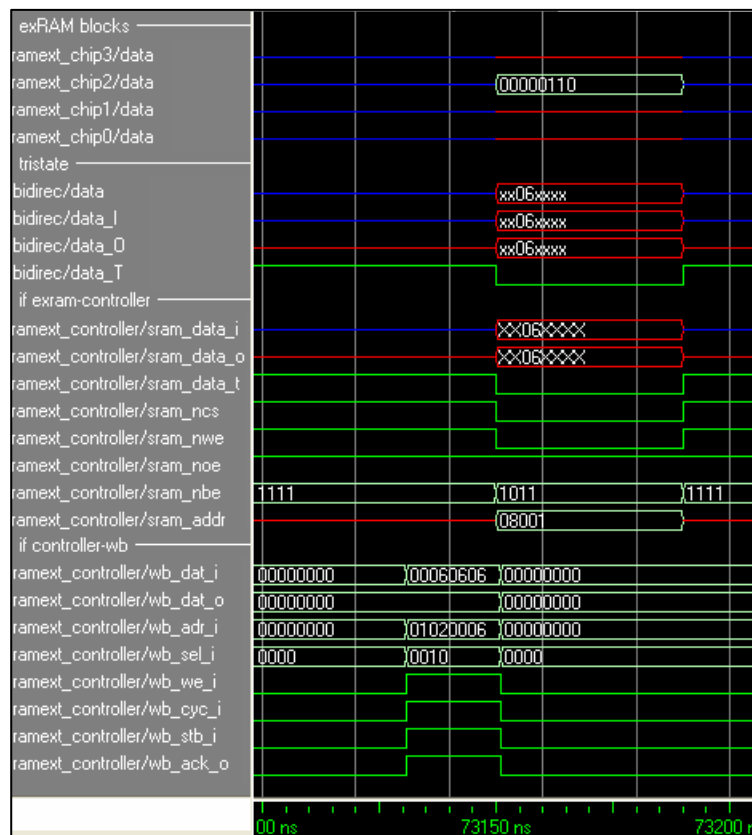


Imagen 49 - Detalle de acceso de escritura de 8 bits a memoria externa

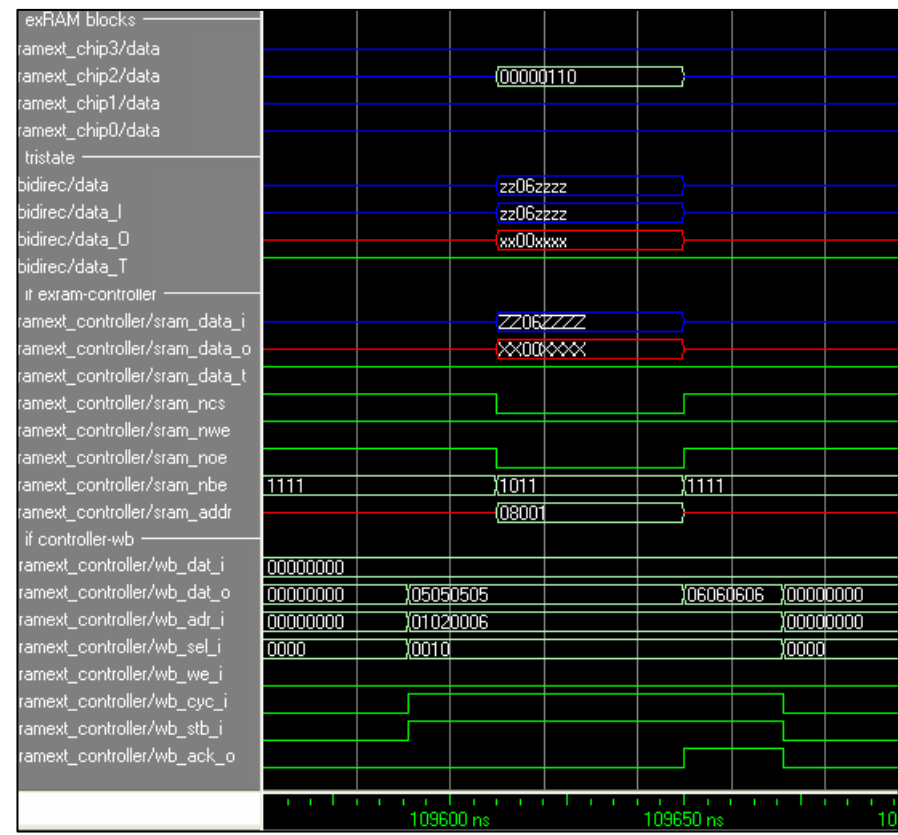


Imagen 50 - Detalle de acceso de lectura de 8 bits en memoria externa

Durante la ejecución de todo el programa, el *or1200* realiza (en memoria externa):

- 4 accesos de escritura de 32 bits
- 4 accesos de lectura de 32 bits
- 8 accesos de escritura de 16 bits
- 8 accesos de lectura de 16 bits
- 16 accesos de escritura de 8 bits
- 16 accesos de lectura de 8 bits

Como se esperaba, la zona de memoria en la que se realizan las comprobaciones queda de la siguiente forma tras cada una de las iteraciones de accesos de escritura:

| 32 bits         | +3       | +2 | +1 | +0 |
|-----------------|----------|----|----|----|
| <b>0102000c</b> | 0102000c |    |    |    |
| <b>01020008</b> | 01020008 |    |    |    |
| <b>01020004</b> | 01020004 |    |    |    |
| <b>01020000</b> | 01020000 |    |    |    |

| 16 bits         | +3   | +2 | +1   | +0 |
|-----------------|------|----|------|----|
| <b>0102000c</b> | 000e |    | 000c |    |
| <b>01020008</b> | 000a |    | 0008 |    |
| <b>01020004</b> | 0006 |    | 0004 |    |
| <b>01020000</b> | 0002 |    | 0000 |    |

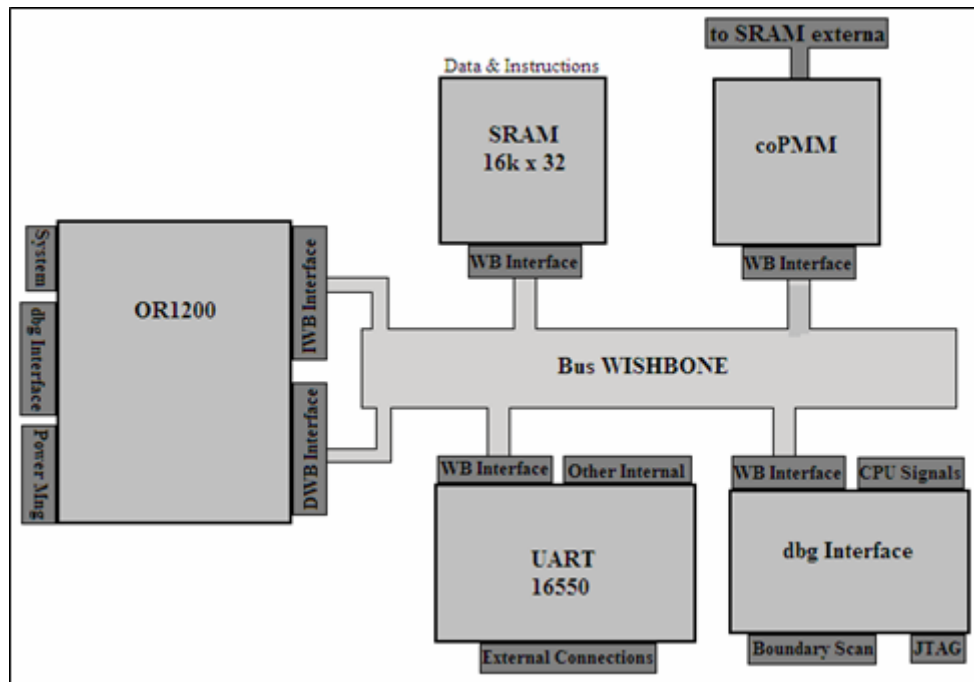
| 8 bits          | +3 | +2 | +1 | +0 |
|-----------------|----|----|----|----|
| <b>0102000c</b> | 0f | 0e | 0d | 0c |
| <b>01020008</b> | 0b | 0a | 09 | 08 |
| <b>01020004</b> | 07 | 06 | 05 | 04 |
| <b>01020000</b> | 03 | 02 | 01 | 00 |

### 3.5 Sistema Avanzado con Coprocesador

En esta sección se pretende añadir un coprocesador de extracción de minutía al sistema básico diseñado en la primera sección del proyecto. El módulo del coprocesador realiza las funciones de controlador de memoria externa en condiciones normales además de, cuando se realiza un acceso concreto, realizar las operaciones de *hardware* específicas del coprocesador. Los objetivos principales de este apartado son los siguientes:

- el estudio breve de la arquitectura del coprocesador *coPMM*
- la adaptación del *core* del coprocesador a una interfaz *Wishbone* y su posterior conexión al sistema básico mediante un fichero puente
- la creación de modelos de memoria y que imiten el comportamiento de los *chips* de memoria *ram* integrados en las placas de los dispositivos *fpga* de las familias *Spartan2E* (8 bits) y *Spartan3E* (32 bits)
- la creación y procesado de programas de comprobación del funcionamiento del coprocesador en sus diferentes modos de funcionamiento
- depurado y procesado del programa específico del coprocesador para la ejecución del algoritmo de *Maio-Maltoni*

- la comprobación mediante simulación funcional del correcto funcionamiento del coprocesador en el entorno original (con una memoria externa de 8 bits)
- la adaptación del sistema y la comprobación mediante simulación funcional del correcto funcionamiento del coprocesador en el entorno real (con una memoria externa de 32 bits)
- la síntesis e implementación del sistema obtenido sobre un dispositivo *fpga* de la familia *Spartan3E*
- la verificación y depurado del diseño implementado mediante la simulación *post-layout*
- la configuración del dispositivo destino con el diseño implementado
- la verificación del diseño implementado y la comprobación de la correcta ejecución del algoritmo mediante depuración *on board*



**Imagen 49** - Esquema del sistema avanzado con coprocesador

Para conseguir el diseño mostrado se debe conectar el coprocesador *coPMM* al *Slave1* del bus Wishbone. Por lo tanto, la configuración de bus que se obtiene es la siguiente:

| #Master |                         | #Slave |                       |
|---------|-------------------------|--------|-----------------------|
| 0       | nc                      | 0      | SRAM Controller Slave |
| 1       | nc                      | 1      | coPMM                 |
| 2       | nc                      | 2      | nc                    |
| 3       | Debug core Master       | 3      | nc                    |
| 4       | RISC Data Master        | 4      | nc                    |
| 5       | RISC Instruction Master | 5      | UART core Slave       |
| 6       | nc                      | 6      | nc                    |
| 7       | nc                      | 7      | nc                    |

En esta sección se realizan las siguientes tareas para el sistema propuesto:

- **Hardware:** añadir al sistema básico un coprocesador de extracción de minutía que se comunica con una memoria externa de 8 bits de anchura. Crear un modelo de memoria externa de 8 bits.



- **Software:** desarrollar el software para la verificación del funcionamiento del coprocesador en sus dos modos de funcionamiento (controlador de memoria externa y coprocesador). Compilar y enlazar software de prueba. Depurar el software para la ejecución del algoritmo de *Maio-Maltoni* sobre el dispositivo destino. Compilar y enlazar software.
- **Simulación funcional:** verificar el funcionamiento del sistema con el coprocesador en el contexto para el que fue desarrollado (memoria externa de 8 bits) mediante simulación funcional. Adaptar la lógica de adaptación al contexto real (memoria de 32 bits). Verificar el funcionamiento del sistema con el coprocesador en el contexto real mediante simulación funcional.
- **Sintetización:** realizar los procesos de síntesis e implementación del diseño para un dispositivo destino de la familia *Spartan3E*
- **Simulación post-layout:** generar un modelo del diseño para simulación temporal. Verificar el funcionamiento del sistema tras la implementación del diseño en el dispositivo destino. Depurar mediante simulación.
- **Configuración FPGA:** preparar el diseño y las herramientas para la configuración del dispositivo destino en el laboratorio. Realizar y detallar el proceso de configuración.
- **Depurado:** preparar las herramientas para el depurado del diseño y del programa en el laboratorio. Realizar y detallar el proceso de depurado.

### 3.5.1 Coprocesador coPMM

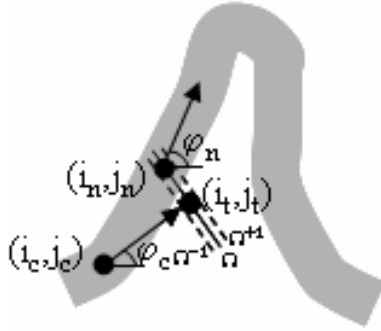
#### 3.5.1.1 Introducción

Los sistemas automáticos de identificación biométrica están en fase de creciente implantación en un elevado rango de aplicaciones. La identificación biométrica por huella dactilar es muy extendida por sus buenas características: invariabilidad, prestaciones, comodidad, sensores de bajo coste y tamaño. Una división entre los diferentes tipos de identificación por huella dactilar es: correlación de imágenes, los basados en minutia y los basados en topología de las colinas. La minutia es el conjunto de puntos en los cuales las colinas de las huellas terminan o bifurcan, y es la clase de identificación por huella dactilar más extendida debido a que esta información permite un alto grado de distinción entre individuos, además de baja cantidad de memoria necesaria para su almacenamiento. Sin embargo, su principal inconveniente es la complejidad de los algoritmos de extracción de la minutia, por lo que estos algoritmos suelen ser implementados en software y ejecutados sobre microprocesadores de altas prestaciones. La identificación biométrica consta de tres fases básicas: adquisición, extracción y comparación. El método utilizado en la extracción de la minutia es el algoritmo de Maio-Maltoni por permitir la extracción directa a partir de la imagen de la huella (menos costoso computacionalmente).

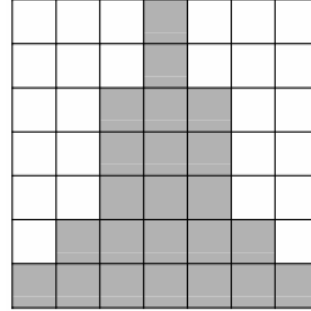
#### 3.5.1.2 Algoritmo Maio-Maltoni

El algoritmo se basa en la idea de navegar sobre las colinas de la imagen de la huella dactilar. Dado un punto inicial  $(i,j)_c$  y dirección  $\varphi_c$  se calcula un nuevo punto  $(i,j)_f$  distante  $\mu$  píxeles en la dirección  $\varphi_c$ . Se calcula entonces la sección  $\Omega$  como el conjunto de  $2\sigma+1$  puntos ortogonales a  $\varphi_c$  y como punto central  $(i,j)_t$ . El tercer paso es el cálculo

promediado de la sección, expresado como  $\text{avg}(\Omega)$ , como la media local de los puntos de  $\Omega$  con los puntos de las secciones de los dos planos paralelos distantes  $\pm 1$  píxel, expresados como  $\Omega^{+1} \Omega^{-1}$ . Se calcula entonces la correlación de la sección promediada  $\text{avg}(\Omega)$  con la máscara gaussiana, y se devuelve el índice  $k$  del máximo local débil como el punto más cercano al centro de  $\Omega$  que cumple la condición  $\text{corr}(\Omega_{k-1}) \leq \text{corr}(\Omega_k) \leq \text{corr}(\Omega_{k+1})$ . Se define a partir de  $k$  el nuevo punto  $(i_n, j_n)$  de  $\Omega$  y se calcula la nueva dirección  $\phi_n$  de la colina centrada en este punto, que serán el nuevo punto y dirección para la próxima iteración. Se actualiza una imagen binaria auxiliar  $T$  de los puntos trazados y sus adyacentes con el fin de evitar que el algoritmo examine de nuevo la misma colina. Estos pasos se repiten hasta que se cumple un criterio de finalización.



**Imagen 51** - Algoritmo Maio-Maltoni



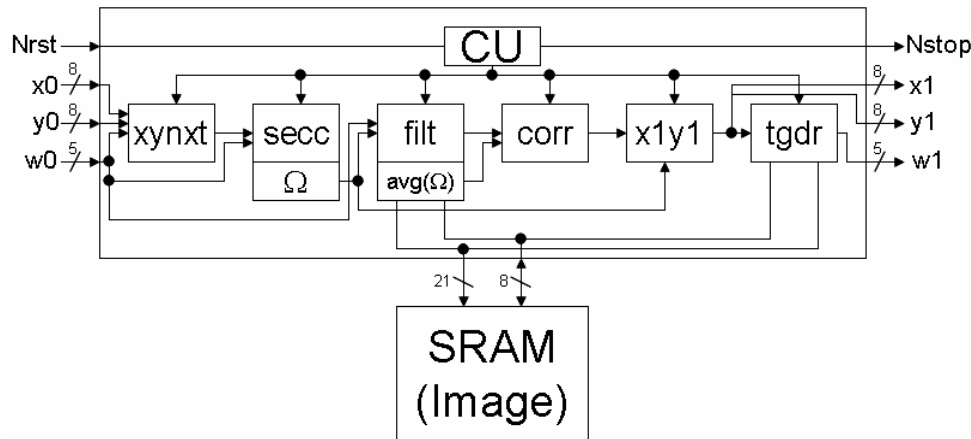
**Imagen 50** - Máscara gaussiana

```
while (continue) {
 next_point(angle_c, 3, &delta_i, &delta_j);
 i_t=i_c+delta_i; j_t=j_c+delta_j;
 createsecc(i_t, j_t, angle_c, sigma, seccX, seccY);
 filtersecc(seccX, seccY, angle_c, avg);
 index=3+corr_weak_max(avg);
 i_n=seccX[index]; j_n=seccY[index];
 angle_n=tg_dir(j_n, i_n, angle_c);
 ...
 updateT(j_c, i_c, j_n, i_n);
 ...}
```

### 3.5.1.3 Coprocesador coPMM

El coprocesador ha sido dividido en seis etapas, que se corresponden con las siete primeras líneas de código, además de un controlador de memoria SRAM externa y la unidad de control. Las dos primeras líneas de código se corresponden con la primera etapa del coprocesador, mientras que las siguientes cinco líneas del código se corresponden una a una con las siguientes cinco etapas del coprocesador. El coprocesador está diseñado para trabajar con imágenes de  $256 \times 256$  píxeles, en  $2^8$  niveles de grises por píxel, por lo que la memoria externa que almacena la imagen debe de ser de al menos 64Kbytes. Además usa dos bancos de memoria interna, la primera para el almacenamiento de las coordenadas de la sección  $\Omega$  de 256 bits, y la segunda para el almacenamiento de la sección promediada  $\text{avg}(\Omega)$  de 150 bits. A partir del punto y ángulo inicial, dado en los puertos  $x_0$ ,  $y_0$  y  $w_0$ , se aserta la señal  $Nrst$  para que el coprocesador comience el cálculo, negando la unidad de control la señal  $Nstop$  mientras está en ejecución, y devolviendo al finalizar el punto y ángulo para la próxima iteración en los puertos  $x_1$ ,  $y_1$  y  $w_1$ . El número de ciclos de reloj requeridos por el coprocesador para completar el cálculo es de 720. El controlador de memoria externa es usado por el coprocesador para acceder a la imagen de la huella dactilar, pero también se puede usar

externamente al coprocesador para que el microprocesador pueda acceder a la imagen, además de poder usarla para almacenar el programa, pila u otros datos. Todas las etapas del coprocesador han sido diseñadas con una topología de pipeline, con el objeto de incrementar la frecuencia de reloj y el throughput, ya que el uso de registros entre fases permite reducir el path crítico de cada fase y realizar cálculos en paralelo de diferentes iteraciones en cada fase. Este coprocesador ha sido diseñado en *vhdl* genérico, de forma que puede ser sintetizado para cualquier *fpga* o *asic* en cuya síntesis se tenga posibilidad de inferir memoria interna.



**Imagen 52** - Esquema de la arquitectura interna del coprocesador

### 3.5.1.4 Controlador de memoria

Aunque el coprocesador es adaptable a cualquier microprocesador, el coPMM se desarrolló para trabajar conectado al microprocesador embebido *Microblaze* mapeado sobre una *fpga* de bajo coste de la familia *Spartan2E*. Las placas de esta familia de dispositivos programables contienen un *chip* de memoria externa con una anchura de 8 bits. Por lo tanto, el controlador de memoria externa realiza cuatro accesos consecutivos de 8 bits para leer o escribir una palabra completa de 32 bits en memoria externa. Además, algunas de las etapas del coprocesador (*filt* y *tgdr*) realizan accesos de lectura a memoria externa, por lo que su descripción *hardware* está adaptada a la forma de trabajo del controlador.

En el presente proyecto se muestra la adaptación del coprocesador *coPMM* al bus *Wishbone* del microprocesador embebido *OpenRisc* mapeado sobre una *fpga* de la familia *Spartan3E*. Por lo tanto el contexto ha cambiado, es decir, los bloques de memoria de 8 bits han quedado obsoletos. Las placas de la familia *Spartan3E* contienen dos *chips* de 16bits o bien un *chip* de 32 bits, resultando en ambos casos en una memoria con una anchura de 32 bits. Sin embargo, la actualización de la descripción *hardware* del coprocesador se encuentra fuera de los límites del presente proyecto. El sistema debe trabajar con el coprocesador original y su código fuente debe permanecer inalterado. De esta manera, la adaptación de las señales del controlador obsoleto se realiza en una etapa intermedia entre el coprocesador y la memoria externa.

### 3.5.2 Hardware

En este apartado, se pretende realizar los ajustes del código fuente necesarios para conectar el coprocesador al sistema básico obtenido en la primera sección del proyecto. Por lo tanto, las tareas a realizar son:

- Estudio de la arquitectura del coprocesador.
- Desarrollo de un fichero puente para la adaptación de la lógica del coprocesador al interfaz *wishbone*.
- Desarrollo de un modelo de memoria externa de 8 bits, compatible con el contexto original para el que fue desarrollado el coprocesador.
- Conexión del coprocesador al sistema y conexión de la memoria externa al coprocesador.

### 3.5.2.1 Coprocesador

La descripción hardware del coprocesador fue desarrollada en lenguaje *vhdl* genérico para poder ser sintetizado para cualquier *fpga* o *asic*. El código fuente describe un módulo principal que contiene las diferentes etapas del coprocesador, además del controlador de memoria y la unidad de control. El proyecto se compone de los siguientes ficheros de extensión '*.vhd*':

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>follow</i>           | <ul style="list-style-type: none"> <li>- define el <i>core</i> principal del coprocesador</li> <li>- contiene la instancia de las etapas del coprocesador y del controlador de memoria</li> <li>- describe la quinta etapa del coprocesador: se define a partir de <math>k</math> el nuevo punto <math>(i_n, y_n)</math> de <math>\Omega</math></li> <li>- gestiona los procesos <i>muxsecc</i>, <i>muxmemc</i>, <i>muxmode</i> y <i>x1y1</i></li> <li>- gestiona la máquina de estados de la UC (unidad de control)</li> </ul> |
| <i>pack_maioHw</i>      | - define las constantes y los tipos de datos globales                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>sramcontrol_8_8</i>  | - no utilizado (microprocesadores de 8 bits)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>sramcontrol_8_32</i> | - define el controlador de memoria externa con accesos consecutivos de 8 bits a memoria para microprocesadores de 32 bits                                                                                                                                                                                                                                                                                                                                                                                                       |
| <i>xynext</i>           | <ul style="list-style-type: none"> <li>- describe la primera etapa del coprocesador: a partir de un punto <math>(x_0, y_0)</math> inicial y una dirección <math>w_0</math> calcula el siguiente punto <math>(x_1, y_1)</math> a una distancia <math>\mu</math> en la dirección <math>w_0</math></li> <li>- contiene la instancia de la entidad <i>nextpoint</i></li> </ul>                                                                                                                                                      |
| <i>section</i>          | <ul style="list-style-type: none"> <li>- describe la segunda etapa del coprocesador: se calcula la sección <math>\Omega</math> como un conjunto de <math>2\sigma+1</math> puntos ortogonales a la dirección <math>w_0</math> y como punto central <math>(x_l, y_l)</math></li> <li>- contiene la instancia y definición de las entidades <i>section_ram</i> y <i>section_core</i> (que contiene la instancia a la entidad <i>nextpoint</i>)</li> </ul>                                                                          |
| <i>filtsec</i>          | <ul style="list-style-type: none"> <li>- describe la tercera etapa del coprocesador: se calcula el promediado de la sección como la media local de los puntos de <math>\Omega</math> con los puntos de las secciones de los planos paralelos distantes <math>\pm 1</math> pixel</li> <li>- contiene la instancia y definición de la entidad <i>filtsec_core</i></li> <li>- realiza accesos de lectura a memoria externa</li> </ul>                                                                                              |
| <i>corrmax</i>          | - describe la cuarta etapa del coprocesador: se calcula la correlación de la sección promediada con una máscara gaussiana predeterminada y se devuelve el índice $k$ del máximo local débil como el punto más cercano al centro de $\Omega$ que cumple la condición $corr(\Omega_{k-1}) \leq corr(\Omega_k) \leq corr(\Omega_{k+1})$                                                                                                                                                                                            |
| <i>tangdir</i>          | - describe la sexta etapa del coprocesador: se calcula la nueva dirección $w_n$ de la colina centrada en el punto $(x_n, y_n)$ , que serán el nuevo punto y dirección para la próxima iteración                                                                                                                                                                                                                                                                                                                                 |

|                  |                                                         |
|------------------|---------------------------------------------------------|
|                  | - realiza accesos de lectura a memoria externa          |
| <i>nextpoint</i> | - contiene la definición de la entidad <i>nextpoint</i> |

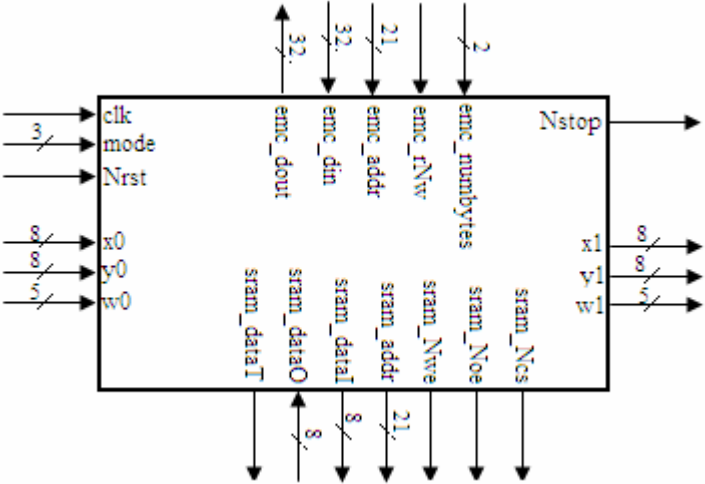
### 3.5.2.1.1 Entidad principal

El fichero *follow.vhd* define el módulo principal del coprocesador. Este módulo contiene la instancia de las diferentes etapas del coprocesador, además del controlador de memoria y la unidad de control.

```

entity follow is
 port(
 clk: in std_logic;
 mode: in std_logic_vector(2 downto 0);
 Nrst: in std_logic;
 Nstop: out std_logic;
 w0: in signed(4 downto 0);
 x0,y0: in std_logic_vector(7 downto 0);
 w1: out signed(4 downto 0);
 x1,y1: out std_logic_vector(7 downto 0);
 emc_numbytes: in unsigned(1 downto 0);
 emc_rNw: in std_logic;
 emc_addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
 emc_din: in std_logic_vector(MICRO_DATA_WIDTH-1 downto 0);
 emc_dout: out std_logic_vector(MICRO_DATA_WIDTH-1 downto 0);
 sram_Ncs,sram_No,sram_Nwe: out std_logic;
 sram_addr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
 sram_data_I: in std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram_data_O: out std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram_data_T: out std_logic
);
end entity;

```



**Imagen 53** - Esquema del módulo principal del coprocesador

En la tabla siguiente se describe la funcionalidad de los puertos de entrada y/o salida de la entidad.

|      |   |                                                                                                                                                  |
|------|---|--------------------------------------------------------------------------------------------------------------------------------------------------|
| clk  | I | reloj del sistema                                                                                                                                |
| mode | I | modo de trabajo del coprocesador<br>XX0 - controlador emc<br>X01 - copro iteración n<br>011 - copro iteración 0<br>111 - copro cálculo de ángulo |
| Nrst | I | reset negado<br>0 - copro inicia nueva operación                                                                                                 |

|              |    |   |                                                                                                       |
|--------------|----|---|-------------------------------------------------------------------------------------------------------|
| Nstop        |    | O | stop negado<br>0 - copro finaliza operación                                                           |
| x0           | y0 | I | punto inicial (8 bits: 0..255)                                                                        |
| w0           |    | I | dirección inicial (5 bits con signo: -12..+12)                                                        |
| x1           | y1 | O | punto final (8 bits: 0..255)                                                                          |
| w1           |    | O | dirección final (5 bits con signo: -12..+12)                                                          |
| emc_numbytes |    | I | numero de bytes accedidos<br>11 - 4 bytes (32 bits)<br>01 - 2 bytes (16 bits)<br>00 - 1 byte (8 bits) |
| emc_rNw      |    | I | read not write<br>0 - escritura<br>1 - lectura                                                        |
| emc_addr     |    | I | bus de direcciones (21 bits → 2MB)                                                                    |
| emc_din      |    | I | bus de datos de escritura (32 bits)                                                                   |
| emc_dout     |    | O | bus de datos de lectura (32 bits)                                                                     |
| sram_Ncs     |    | O | chip select negado<br>0 - se accede a memoria externa                                                 |
| sram_Noce    |    | O | output enable negado<br>0 - valor válido en bus dataO                                                 |
| sram_Nwe     |    | O | write enable negado<br>0 - escritura<br>1 - lectura                                                   |
| sram_addr    |    | O | bus de direcciones (21 bits → 2MB)                                                                    |
| sram_dataI   |    | I | bus de datos de lectura (8 bits)                                                                      |
| sram_dataO   |    | O | bus de datos de escritura (8 bits)                                                                    |
| sram_dataT   |    | O | control del buffer tri-state para bus de datos bidireccional                                          |

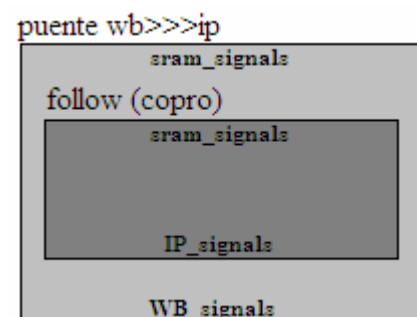
Las etiquetas precedidas por ‘*emc\_*’ se corresponden con señales específicas de comunicación entre el bus del sistema y el controlador de memoria

Las etiquetas precedidas por ‘*sram\_*’ se corresponden con señales específicas de comunicación entre el controlador de memoria y la memoria externa

### 3.5.2.2 Fichero puente

Puesto que el código fuente del coprocesador no debe ser modificado, se ha creado un fichero puente que adapta los puertos de entrada/salida del coprocesador al sistema propuesto para el presente proyecto. Por un lado, el fichero puente adapta las señales conectadas al interface del bus *wishbone* con los puertos de comunicación entre el bus y el coprocesador. Estas señales se corresponden con los puertos de control y de acceso al controlador de memoria, es decir, todos los puertos excepto los precedidos por la etiqueta ‘*sram\_*’. Por otro lado, el fichero debe adaptar el formato de las señales de comunicación entre el coprocesador y la memoria externa, además de reestructurar los accesos a memoria.

En este apartado se pretende comprobar el correcto funcionamiento del coprocesador, por lo que el fichero puente se limita a realizar las adaptaciones que corresponden a la primera tarea. Se pretende realizar la conexión del coprocesador al bus *wishbone* y comprobar mediante simulación el correcto funcionamiento del



dispositivo en su contexto original. Es decir, el modelo de memoria utilizado para la primera simulación se comporta como un *chip* de memoria *sram* externa con una anchura de 8 bits.

Puesto que el fichero *wb\_bridge\_maioHw.vhd* es algo extenso, se muestran los trozos de código relevantes y se explica la causa de las conversiones. Los grupos de señales que se deben adaptar son los siguientes:

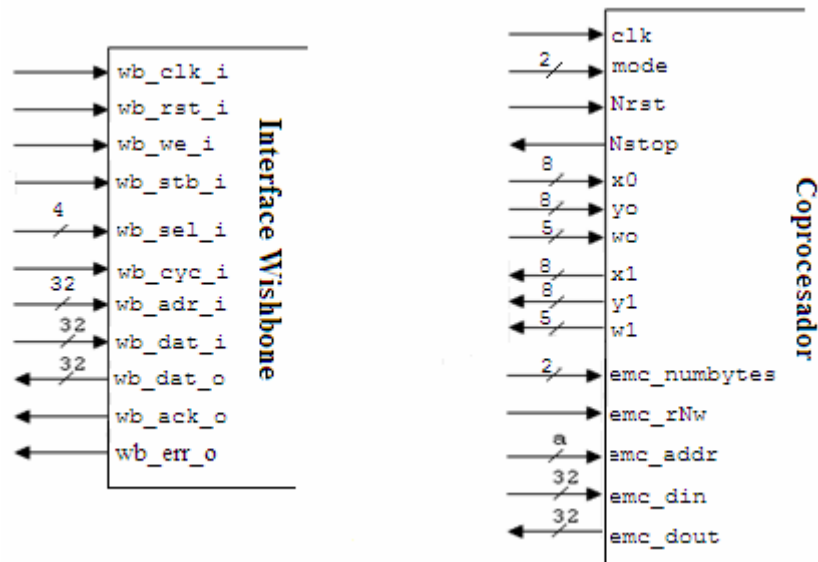


Imagen 54 - Interfaz Wishbone e interfaz del coprocesador

En varios puntos del código se utiliza la señal *cs* (*chip select*) para la gestión de otras señales. Esta señal indica si el chip de memoria externa está siendo accedido. El valor se obtiene a partir de las señales *cyc* y *stb* del interface *wishbone* y comprobando, además, que la dirección accedida se encuentra dentro del mapa de direcciones correspondiente a la memoria.

En varios puntos del código se utilizan señales auxiliares para la gestión de otras señales. Se trata de señales como *CS*, *running\_maioHw* o *access\_register* entre otras menos importantes. La señal *CS* es un valor lógico que se obtiene a partir de las señales *cyc* y *stb* del interface *wishbone*. Las dos restantes son valores *booleanos* que se obtienen a partir de las condiciones definidas, es decir, tienen un valor '1' si la condición se cumple o un valor '0' en caso contrario.

```
CS<=wb_cyc_i and wb_stb_i;
running_maioHw<=Nstop='1' and maioHw_Nrst='1';
access_register<=wb_adr_i=C_ADDR_REGISTER and wb_sel_i="1111";
```

La variable *booleana* *running\_maioHw* indica que el coprocesador está trabajando en modo copro cuando el coprocesador ha empezado una operación (*maioHw\_Nrst*) y todavía no se ha recibido la señal de operación finalizada (*Nstop*). Por otro lado, la variable *booleana* *access\_register* indica que se pretende realizar un acceso a memoria al registro específico de control del coprocesador cuando se trata de un acceso de 32 bits (*wb\_sel\_i*) a la dirección donde está mapeado el registro especial.

|      |                                                                                                                                                                                                                                                          |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Clk  | conexión directa ( <i>wb_clk_i</i> )                                                                                                                                                                                                                     |
| mode | proceso → en cada flanco positivo de reloj: <ul style="list-style-type: none"> <li>- si <i>wishbone</i> fuerza reset → "xx0"</li> <li>- sino si se escribe en el registro especial → "MM1" (*)</li> <li>- sino si se accede a memoria → "xx0"</li> </ul> |

|              |                                                                                                                                                                                       |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | (*)“MM” corresponde con los bits 25 y 24 de wb_dat_i                                                                                                                                  |
| Nrstst       | Nrst<=emc Nrst or maioHw Nrst;                                                                                                                                                        |
| Nstop        | gestionada en entidad <i>follow</i>                                                                                                                                                   |
| x0,y0,w0     | w0<=signed(reg_command(20 downto 16));<br>x0<=reg_command(7 downto 0);<br>y0<=reg_command(15 downto 8);<br><br>reg_command gestionada en proceso                                      |
| x1,y1,w1     | result(31 downto 24)<=(others=>'0');<br>result(23 downto 16)<=sxt(std_logic_vector(w1),8);<br>result(7 downto 0)<=x1;<br>result(15 downto 8)<=y1;<br><br>result gestionada en proceso |
| emc_numbytes | gestionada en proceso pBE                                                                                                                                                             |
| emc_rNw      | conexión directa negada (not wb_we_i)                                                                                                                                                 |
| emc_addr     | gestionada en proceso pBE                                                                                                                                                             |
| emc_din      | conexión directa (wb_dat_i)                                                                                                                                                           |
| emc_dout     | gestionada en proceso pBE                                                                                                                                                             |
| wb_dat_o     | gestionada en proceso                                                                                                                                                                 |
| wb_ack_o     | wb_ack_o<=emc_ack or maioHw_ack;                                                                                                                                                      |
| wb_err_o     | wb_err_o<='0';                                                                                                                                                                        |

Proceso pBE: - lista de sensibilidad (wb\_sel\_i,wb\_adr\_i,emc\_dout)  
- en función de wb\_sel\_i:

| wb_sel_i            | acceso  | emc_numbytes | emc_addr[1:0] | emc_dout0   |
|---------------------|---------|--------------|---------------|-------------|
| 1111                | 32 bits | 11           | 00            | D32         |
| 1100,0011           | 16 bits | 01           | X0            | D16&D16     |
| 1110,1101,1011,0111 | 8 bits  | 00           | XX            | D8&D8&D8&D8 |

- conexión directa de emc\_addr con wb\_adr\_i excepto los dos LSB

Proceso pMaioHw:

- en cada flanco positivo de reloj
- realiza la gestión del registro de propósito especial del coprocesador, además de la gestión de las señales de reset
  - o si se pretende escribir en el registro → prepara el coprocesador  
→ reg\_command = bus de datos de escritura
  - o mientras el copro trabaja → reg\_result = FFFF (= -1)
  - o cuando el copro acaba → reg\_result = result(x1,y1,w1)

Proceso pRead:

- lista de sensibilidad(access\_register,reg\_result,emc\_dout0)
- multiplexa las salidas del coprocesador al bus de datos de lectura del bus *wishbone*:
  - o cuando se accede al registro → reg\_result (x1,y1,w1)
  - o cuando se accede a otra posición → salida del controlador de memoria

### 3.5.3 Modelo de memoria

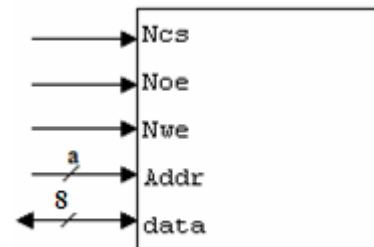
Para poder simular el sistema con *ModelSim*, se crea un modelo de memoria externa, es decir, un módulo escrito en lenguaje *hdl* que imite el comportamiento de un *chip* de memoria ram. De esta forma, se podrán simular los accesos a memoria externa



sin conectar realmente el sistema al exterior. El código fuente del modelo *vhdl* se encuentra en el fichero *sram\_sim.vhd*. Como ya se ha dicho, el coprocesador se desarrolló para ser mapeado en un placa *fpga* de la familia *Spartan2E*, que contienen un *chip* de memoria externa de 2MB con una anchura de 8 bits. Para comprobar el correcto funcionamiento del coprocesador se ha decidido simular el sistema conectado a un modelo de memoria con el comportamiento planteado originalmente. Una vez simulado y testeado el coprocesador, se tratará de adaptar el sistema al contexto adecuado. El modelo de memoria utilizado se muestra a continuación:

```
entity sram_sim is
 generic(
 init: t_sram:=(others=>sram_word_U);
 delay: time:=0 ns);
 port(
 Ncs,Noe: in std_logic;
 Nwe: in std_logic;
 addr: in std_logic_vector(sram_addr_width-1 downto 0);
 data: inout std_logic_vector(sram_data_width-1 downto 0));
end sram_sim;
```

La entidad *sram\_sim* define el mapa de genéricos y de puertos del módulo representado en la imagen lateral. Se destaca la naturaleza negada de lógica utilizada para las señales de control, además del carácter bidireccional del bus de datos, utilizado tanto para la lectura como para la escritura. La anchura del bus de direcciones depende del tamaño de la memoria externa conectada.



**Imagen 55** - Entidad *sram\_sim*

Se trata de un modelo de memoria con un comportamiento muy similar al utilizado en la sección anterior para comprobar el funcionamiento del controlador de memoria externa. Se trata de un comportamiento secuencial definido dentro de un proceso, que utiliza una función para inicializar algunas posiciones de la memoria con los datos de la huella dactilar. El sistema propuesto para este apartado presenta un único bloque de memoria de 2MB con una anchura de 8 bits, a diferencia de los cuatro bloques de 512KB de 8 bits de ancho utilizados en el sistema propuesto en la sección anterior. La razón de este cambio es el método usado por el controlador de memoria para acceder a una palabra de 32 bits en cada sistema. En la sección anterior el controlador realizaba un único acceso en paralelo a los cuatro bloques de memoria. En cambio, en este sistema el controlador embebido en el coprocesador realiza cuatro accesos consecutivos de 8 bits para conformar una palabra completa de 32 bits.

### 3.5.4 Conexión al bus

El dispositivo que se pretende conectar al bus *wishbone* es un coprocesador conectado a un chip de memoria externo a la *fpga*. El coprocesador se conecta al *slave1* del árbitro del bus mediante las señales del interface *wishbone*. Las señales del coprocesador destinadas a la comunicación con el *chip* externo se mapean a los puertos de entrada/salida de la *fpga*. Para la simulación se conecta el modelo de memoria externa a los puertos I/O de la *fpga*. El proceso de conexión se describe en los siguientes pasos:

- Añadir los puertos I/O de la *fpga* (*Ncs*, *Noe*, *Nwe*, *addr* y *data*) en la definición del módulo *xsv\_fpga\_top* del fichero con el mismo nombre.

```
module xsv_fpga_top (
 clk, rstn,
 sw,
 uart_stx, uart_srx,
 sram_Ncs, sram_Noel, sram_Nwe, sram_addr, sram_data,
 flash_Ncs,
 jtag_tck, jtag_tms, jtag_tdi, jtag_trst,
 jtag_tvref, jtag_tgnd, jtag_tdo);
```

- Definir el sentido de los puertos añadidos.

```
output sram_Ncs, sram_Noel, sram_Nwe;
output [`SRAM_EXT_AW-1:0] sram_addr;
inout [7:0] sram_data;
```

El bus de datos del chip de memoria es bidireccional.

La anchura del bus de direcciones depende del tamaño de la memoria utilizada. El número de palabras de 32 bits contenidas en la memoria es  $2^{\text{SRAM\_EXT\_AW}}$ .

- Eliminar la línea dedicada a inhabilitar el *chip* de memoria externa en la placa:  
`assign sram_Ncs = 1'b1;`
- Añadir el cableado dedicado a conectar el interface wishbone del coprocesador al *slave1* del *Traffic Cop* (árbitro del bus)

```
wire [31:0] wb_cop_dat_i;
wire [31:0] wb_cop_dat_o;
wire [31:0] wb_cop_adr_i;
wire [3:0] wb_cop_sel_i;
wire wb_cop_we_i;
wire wb_cop_cyc_i;
wire wb_cop_stb_i;
wire wb_cop_ack_o;
wire wb_res_err_o;
```

- Instanciar el coprocesador.

```
user_maioHw coprocesador (
 .wb_clk_i (wb_clk),
 .wb_rst_i (wb_rst),
 .wb_dat_i (wb_cop_dat_i),
 .wb_dat_o (wb_cop_dat_o),
 .wb_adr_i (wb_cop_adr_i),
 .wb_sel_i (wb_cop_sel_i),
 .wb_we_i (wb_cop_we_i),
 .wb_cyc_i (wb_cop_cyc_i),
 .wb_stb_i (wb_cop_stb_i),
 .wb_ack_o (wb_cop_ack_o),
 .wb_err_o (wb_cop_err_o),
 .sram_Ncs (sram_Ncs),
 .sram_Nwe (sram_Nwe),
 .sram_Noel (sram_Noel),
 .sram_addr (sram_addr),
 .sram_data_I(sram_data_I),
 .sram_data_O(sram_data_O),
 .sram_data_T(sram_data_T)
);
```

El bus de datos no es bidireccional. Se trata de un bus de entrada, un bus de salida y una señal de control. Por lo tanto, se define un *tri-state buffer* para adaptar los tipos de bus.

- Conectar el coprocesador al *slave1* del *Traffic Cop* en la instancia del árbitro del bus.

```
.tl_wb_cyc_o (wb_cop_cyc_i),
.tl_wb_stb_o (wb_cop_stb_i),
.tl_wb_cab_o (wb_cop_cab_i),
.tl_wb_adr_o (wb_cop_adr_i),
.tl_wb_sel_o (wb_cop_sel_i),
.tl_wb_we_o (wb_cop_we_i),
.tl_wb_dat_o (wb_cop_dat_i),
.tl_wb_dat_i (wb_cop_dat_o),
.tl_wb_ack_i (wb_cop_ack_o),
.tl_wb_err_i (wb_cop_err_o),
```

- Añadir el cableado interno destinado a conectar los puertos I/O de la *fpga*.

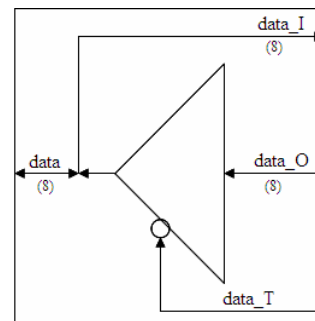
```
wire sram_Ncs, sram_Noe, sram_Nwe;
wire [`SRAM_EXT_AW-1:0] sram_addr;
wire [7:0] sram_data;
wire [7:0] sram_data_I;
wire [7:0] sram_data_O;
wire sram_data_T;
```

- Diseñar el *tri-state buffer* (*tristate.v*) dedicado a adaptar el carácter bidireccional del bus de datos que viene del chip de memoria a los dos buses de datos y la señal de control pertenecientes al coprocesador.

```
`define DATA_W 8
module bidirec (data_I, data_O, data_T, data);
 output [`DATA_W-1:0] data_I;
 input [`DATA_W-1:0] data_O;
 input data_T;
 inout [`DATA_W-1:0] data;
 assign data = data_T ? `DATA_W'bZ : data_O;
 assign data_I = data;
endmodule
```

- Instanciar el *tri-state buffer*.

```
bidirec bidirec (
 .data_T (sram_data_T),
 .data_I (sram_data_I),
 .data_O (sram_data_O),
 .data (sram_data));
```



- Actualizar la instancia del chip de la *fpga* en el fichero testbench (*sim\_test1.v*) según las modificaciones de los puertos en su definición.
- Añadir el cableado dedicado a la interconexión del chip de la *fpga* con el modelo del bloque de memoria externa en el fichero *testbench*.

```
wire sram_Ncs, sram_Noce, sram_Nwe;
wire [`SRAM_EXT_AW-1:0] sram_addr;
wire [7:0] sram_data;
```

- Instanciar el modelo del bloque de memoria en el fichero *testbench* definido previamente en el fichero *sram\_sim.vhd*.

```
sram_sim ramext_chip(
 .Ncs(sram_Ncs),
 .Noe(sram_Noce),
 .Nwe(sram_Nwe),
 .addr(sram_addr[`SRAM_EXT_AW-1:0]),
 .data(sram_data)
);
```

- Añadir la siguiente línea al fichero *xsv\_fpga\_defines.v* para actualizar el mapa de direcciones del *traffic\_cop*.

```
`define APP_ADDR_RAMEXT `APP_ADDR_DEC_W'h01
```

Sustituir la nueva constante por *APP\_ADDR\_FLASH* en los parámetros genéricos pasados en la instancia del *traffic\_cop* en el fichero *xsv\_fpga\_top.v*. El mapa de direcciones del *slave1* del *traffic\_cop* resulta de 16MB mapeados en las direcciones:

0x01000000 - 0x01FFFFFF

### 3.5.5 Software

El programa pretende ser mapeado en la memoria *sram* interna del sistema (embedida en la *fpga*) y ejecutado por el microprocesador *or1200* descrito. De la misma manera que el programa de la sección anterior, el software proporcionado con el coprocesador está escrito en lenguaje C++. Por lo tanto, los programas generados para la comprobación del funcionamiento del coprocesador en sus diferentes modos de trabajo serán escritos en el mismo lenguaje de programación. En este caso, las herramientas cruzadas necesarias para la compilación y el enlazado del programa compatibles con C++ ya han sido generadas en la sección anterior. Las tareas principales desarrolladas en el presente apartado son:

- Desarrollo en lenguaje C++ de los programas de comprobación del funcionamiento del coprocesador en modo controlador de memoria y en modo extracción de minutía.
- Compilado y enlazado de los programas de *test* desarrollados.
- Depurado del programa específico del coprocesador para la ejecución del algoritmo de *Maio-Maltoni*.
- Compilado y enlazado del programa depurado.

#### 3.5.5.1 Software de prueba

Como ya se ha mencionado, el coprocesador dispone de diferentes modos de trabajo que dependen de la señal *mode* conectada a un puerto de entrada de la entidad principal del coprocesador (*follow*). Cuando el fichero puente detecta un acceso general al mapa de direcciones de la memoria externa, activa el modo controlador de memoria del dispositivo. En cambio, cuando detecta un acceso de escritura al registro de propósito especial (mapeado en la dirección especificada por el usuario), activa el modo

coprocesador consultando el modo específico (*iteración inicial*, *iteración general* o *cálculo de ángulo*) en el bus de escritura de datos de *wishbone*.

Debido a la existencia de diferentes modos de trabajo, se han desarrollado dos rutinas diferentes para comprobar el comportamiento del sistema en las dos modalidades principales: controlador de memoria y coprocesador en iteración general. El fichero *reset.S* se incluye en ambos proyectos. Se trata del fichero que describe la rutina de servicio a la excepción del reset del sistema escrita en código ensamblador. El código gestiona la pila y fuerza el *program counter (PC)* a la rutina *\_\_main*, desde donde se hace un salto a la primera instrucción del programa principal.

#### a) Rutina *checkmem*

El fichero *checkmem.cpp* contiene la rutina encargada de la comprobación del funcionamiento del dispositivo cuando se comporta como un controlador de memoria. Se trata de la misma rutina utilizada para la comprobación del funcionamiento del controlador de memoria utilizado en la sección anterior del proyecto. Se destaca que el programa realiza una serie de accesos de lectura y escritura a la memoria externa con tamaños de datos diferentes (32, 16 y 8 bits).

#### b) Rutina *checkcopro*

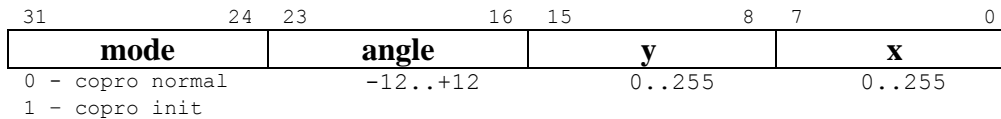
El fichero *checkcopro.cpp* describe la rutina encargada de comprobar el funcionamiento del dispositivo cuando trabaja como coprocesador. Principalmente, se trata de una función que realiza un acceso de escritura al registro de propósito especial del coprocesador (mapeado en memoria externa) y espera a que el coprocesador termine de operar y actualice el registro especial con los datos de salida.

```
#define MAIOHW_REG 0x01000000
typedef struct T_MAIOHW_FIELD
{
 unsigned char mode;
 signed char a;
 unsigned char y;
 unsigned char x;
};
typedef union T_MAIOHW_REG
{
 unsigned int data;
 T_MAIOHW_FIELD field;
};

T_MAIOHW_REG *p_copro_reg;
T_MAIOHW_REG check_copro(unsigned int data)
{
 p_copro_reg=(T_MAIOHW_REG*)MAIOHW_REG;
 p_copro_reg->data=data;
 while(p_copro_reg->data==--1);
 return *p_copro_reg;
}

int main ()
{
 T_MAIOHW_REG registro;
 registro.field.mode=0;
 registro.field.a=1;
 registro.field.y=100;
 registro.field.x=200;
 check_copro(registro.data);
 return 0;
}
```

El código del fichero define la constante `MAIOHW_REG` que contiene la dirección del registro de propósito especial del coprocesador. Se trata de la dirección global en el sistema (0x01000000) que se corresponde con la primera dirección local de la memoria externa. En las siguientes líneas define los tipos de datos específicos de este programa. La estructura de datos `T_MAIOHW_FIELD` representa el registro de propósito especial en 32 bits, dividido en cuatro campos de 8 bits que contienen los datos de modo, ángulo y punto (x,y) utilizados para la iteración del algoritmo.



La unión `T_MAIOHW_REG` permite al programador acceder a los datos ya sea con un acceso al registro de 32 bits o mediante un acceso al campo específico de la estructura del registro. A continuación se declara un puntero a la unión definida llamado `p_copro_reg`.

La rutina principal declara e inicializa con datos coherentes la variable `registro` del tipo unión definida al inicio del fichero. Antes de finalizar el programa se realiza la llamada a la función `check_copro` con el registro de 32 bits como argumento. La función inicializa el puntero `p_copro_reg` a la dirección `MAIOHW_REG` y realiza un acceso de escritura al registro de propósito especial mapeado en memoria externa con los datos pasados como argumento. Entonces el coprocesador *hardware* empieza a trabajar forzando, además, el registro especial a un valor '-1'. A su vez, el programa *software* entra en un bucle de espera mientras el registro contenga el valor '-1'. Cuando el coprocesador termina su ciclo de trabajo, éste actualiza el registro de propósito especial con los resultados de la iteración y el programa sale del bucle de espera.

#### 3.5.5.1.1 Ficheros ELF

Para la obtención de los ficheros de salida en fomato ELF se utilizan las herramientas cruzadas de la *GNU* compatibles con C++. En el caso del primer programa de prueba, que comprueba el funcionamiento del dispositivo cuando trabaja como controlador de memoria, se obtienen los ficheros de salida mostrados en la sección anterior con el mismo procedimiento puesto que se trata del código fuente idéntico.

Para el compilado y el enlazado del programa de prueba del dispositivo en modo coprocesador se adapta el *script* llamado *makefile* para que trabaje sobre los ficheros fuente adecuados (*checkcopro.cpp* y *reset.S*). Para el enlazado se utiliza el fichero *ram.ld*, idéntico al utilizado en la sección anterior. Las líneas ejecutadas en la consola de *Cygwin* son:

```
cd c:/cygwin/Proyectos/program_SA_copro_test2
make clean all
```

El fichero de salida del *script* es *copro.or32*. Se trata del fichero en formato *ELF* utilizado para la programación de la *fpga* y para la obtención de los ficheros necesarios para la simulación del sistema con *ModelSim*. Se puede visualizar el contenido del fichero *ELF* en distintos formatos utilizando las líneas de comandos siguientes en una consola de *Cygwin*:

```
$ or32-elf-readelf -a copro.or32 > program.txt
$ or32-elf-objdump -d -S copro.or32 > disassemble.txt
```

La información relevante volcada en el fichero *program.txt* se muestra en el cuadro de texto siguiente:

```

Section Headers:
[Nr] Name Type Addr Off Size ES Flg Lk Inf Al
[0] NULL 00000000 000000 000000 00 0 0 0
[1] .vectors PROGBITS 00000000 002000 000128 00 AX 0 0 1
[2] .rel.vectors REL 00000000 0052c0 000000 08 13 1 4
[3] .text PROGBITS 00002000 004000 0002d0 00 AX 0 0 4
[4] .rel.text REL 00000000 0052c0 000000 08 13 3 4
[5] .data PROGBITS 000022d0 0042d0 000000 00 WA 0 0 1
[6] .bss NOBITS 000022d0 0042d0 000008 00 WA 0 0 4
[7] .stack NOBITS 000022d8 0042d0 001000 00 WA 0 0 1
[8] .stab PROGBITS 00000000 0042d0 0002e8 0c 10 0 4
[9] .rel.stab REL 00000000 0052c0 000000 08 13 8 4
[10] .stabstr STRTAB 00000000 0045b8 00083a 00 0 0 1
[11] .comment PROGBITS 00000000 004df2 000012 00 0 0 1
[12] .shstrtab STRTAB 00000000 004e04 000060 00 0 0 1
[13] .symtab SYMTAB 00000000 0050bc 000180 10 14 17 4
[14] .strtab STRTAB 00000000 00523c 000084 00 0 0 1

Program Headers:
Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align
LOAD 0x002000 0x00000000 0x00000000 0x022d0 0x032d8 RWE 0x2000

Section to Segment mapping:
Segment Sections...
00 .vectors .text .bss .stack

Symbol table '.symtab' contains 24 entries:
Num: Value Size Type Bind Vis Ndx Name
0: 00000000 0 NOTYPE LOCAL DEFAULT UND
1: 00000000 0 FILE LOCAL DEFAULT ABS checkcopro.cpp
2: 00002000 0 SECTION LOCAL DEFAULT 3
3: 000022d0 0 SECTION LOCAL DEFAULT 5
4: 000022d0 0 SECTION LOCAL DEFAULT 6
5: 000022d0 0 NOTYPE LOCAL DEFAULT 3 Letext
6: 00000000 0 SECTION LOCAL DEFAULT 8
7: 00000000 0 SECTION LOCAL DEFAULT 10
8: 00000000 0 SECTION LOCAL DEFAULT 11
9: 00000000 0 FILE LOCAL DEFAULT ABS reset.S
10: 000022d0 0 SECTION LOCAL DEFAULT 3
11: 000022d0 0 SECTION LOCAL DEFAULT 5
12: 000022d8 0 SECTION LOCAL DEFAULT 6
13: 000022d8 0 SECTION LOCAL DEFAULT 7
14: 000032d8 0 NOTYPE LOCAL DEFAULT 7 _stack
15: 00000000 0 SECTION LOCAL DEFAULT 1
16: 00000100 0 NOTYPE LOCAL DEFAULT 1 _reset
17: 000020dc 336 FUNC GLOBAL DEFAULT 3 __z17check_fingerprintv
18: 000032d8 0 NOTYPE GLOBAL DEFAULT 7 _src_addr
19: 000022d0 4 OBJECT GLOBAL DEFAULT 6 _p_copro_reg
20: 00000120 0 NOTYPE GLOBAL DEFAULT 1 ___main
21: 00002000 220 FUNC GLOBAL DEFAULT 3 __z11check_coproj
22: 000022d4 4 OBJECT GLOBAL DEFAULT 6 _imatge
23: 0000222c 164 FUNC GLOBAL DEFAULT 3 _main

```

### 3.5.5.2 Programa del algoritmo de extracción de minutia

Se trata del programa proporcionado con el coprocesador *coPMM*. Su código ejecuta el algoritmo de *Maio-Maltoni* para la extracción de minutia de la imagen de una huella dactilar almacenada en memoria. Además, se permite elegir el uso opcional del coprocesador para la ejecución del algoritmo, facilitando así la comparativa de los resultados obtenidos en una ejecución *software* o *hardware*. El proyecto está formado por diferentes rutinas escritas en lenguaje C++ contenidas en la lista de ficheros *cpp* y ficheros de encabezamiento siguientes:

actualitza.cpp      followridge.cpp      weak\_max2.cpp

|                  |                |              |
|------------------|----------------|--------------|
| angle.cpp        | funcions.cpp   | weak_max.cpp |
| creaseccio.cpp   | next_point.cpp | maio.h       |
| estraccio.cpp    | tg_dir3.cpp    | maioHw.h     |
| filtraseccio.cpp | updateT.cpp    |              |

El programa original permite el estudio de los resultados experimentales de la ejecución del algoritmo con un coprocesador de extracción de minutia implementado sobre un dispositivo de la familia *Spartan2E*, además de su comparación con los resultados obtenidos mediante una ejecución íntegramente *software*. Sin embargo, para alcanzar los objetivos del presente proyecto se deben realizar algunas modificaciones en el código fuente del proyecto:

- Adaptar (o eliminar si es prescindible) el código relacionado con la gestión de los recursos específicos de la placa. El programa original utiliza algunos contadores, leds e interruptores de la placa para obtener resultados temporales o para controlar y visualizar algunas opciones de usuario del programa.
- Eliminar la ejecución *software* de la extracción de minutia. El programa original permite seleccionar la ejecución *software* o *hardware* de la parte de extracción de minutia del algoritmo para realizar una comparación de los resultados obtenidos con o sin el uso del coprocesador.
- Depurar el código fuente del proyecto.
- Añadir las rutinas de gestión de la *uart* y el código específico para la comunicación del microcontrolador por el puerto serie.

A continuación se muestra una descripción de los ficheros resultantes tras la simplificación del código fuente del proyecto original:

| Fichero         | Funciones                         | Descripción                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| reset.S         |                                   | Rutina de servicio a la excepción de reset                                                                                                                                                                                                                                                                                                                                                                                                                 |
| actualitza.cpp  | actualitza()                      | Si existe otra minutia cercana y no se trata de una bifurcación se filtra la minutia ya registrada en M<br>Sino se registra la minutia de forma ordenada en M                                                                                                                                                                                                                                                                                              |
| angle.cpp       | rect2polar()<br><br>angleerror()  | Se calcula el ángulo $w$ en función de las coordenadas $x$ e $y$<br>Se determina si el ángulo $w$ es erróneo en función de la dirección actual y la $w$ anterior                                                                                                                                                                                                                                                                                           |
| estraccio.cpp   | main()<br><br><br><br>Estraccio() | Contiene la imagen de una huella dactilar<br>Inicializa los punteros a la imagen de la huella dactilar y al registro de propósito especial del coprocesador.<br>Llama a las funciones <code>uart_init()</code> y <code>Estraccio()</code><br>Envía los resultados por la <i>uart</i><br>Llama a la función <code>InitVars()</code><br>Llama a la función <code>followridge()</code> en cada iteración del bucle para los parámetros iniciales establecidos |
| followridge.cpp | followridge()                     | Ejecuta el algoritmo de extracción de minutia para las coordenadas pasadas como argumentos.<br>En la primera iteración del coprocesador obtiene las coordenadas de inicio. En la segunda obtiene el ángulo inicial. Ejecuta iteraciones del coprocesador                                                                                                                                                                                                   |



|                |                                           |                                                                                                                                                                                                                                        |
|----------------|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                |                                           | hasta cumplir una de las condiciones de salida.                                                                                                                                                                                        |
| funcions.cpp   | InitVars()<br>checkT()<br><br>contrast()  | Inicializa los parámetros del algoritmo<br>Comprueba la posición especificada de la matriz T para reconocer los caminos ya seguidos.<br>Realiza operaciones matemáticas complejas para obtener el cálculos promediados y correlaciones |
| next_point.cpp | next_point()                              | Calcula las coordenadas del punto siguiente a partir del índice $k$ del máximo local débil                                                                                                                                             |
| uart.cpp       | uart_init()<br>uart_putc()<br>uart_getc() | Inicializa los parámetros de la <i>uart</i><br>Gestiona la transmisión de un carácter por la <i>uart</i><br>Gestiona la recepción de un carácter por la <i>uart</i>                                                                    |
| updateT.cpp    | UpdateT()                                 | Actualiza la matriz T que especifica los caminos ya seguidos                                                                                                                                                                           |
| board.h        |                                           | Define las constantes específicas de la placa                                                                                                                                                                                          |
| maio.h         |                                           | Define las constantes específicas del algoritmo y declara las variables globales y las rutinas del programa                                                                                                                            |
| maioHw.h       |                                           | Define los punteros y la estructura de datos del registro de propósito especial del coprocesador                                                                                                                                       |
| uart.h         |                                           | Define las constantes para las gestión de la <i>uart</i>                                                                                                                                                                               |

En este caso, se trata de un programa complejo formado por múltiples ficheros fuente y, por lo tanto, se recomienda el uso de un fichero *Makefile* de *scripts* de compilación y enlazado para simplificar el procedimiento. La estructura del código del fichero es similar a la de los ficheros utilizados en apartados anteriores. Se especifican las herramientas cruzadas de la *GNU* compatibles con *C++* (prefijo *or32-elf*), se define el comando de compilación para cada fichero fuente y el comando de enlazado del proyecto. Por último, se describen las *scripts* de limpieza del directorio de trabajo *clean* y *distclean*.

Para ejecutar los *scripts* definidos en el fichero *Makefile* se introduce el comando *make* seguido de la etiqueta de los *scripts* a ejecutar (por ejemplo *all* *System.map* *clean* *distclean* ...) en el directorio de fuentes del proyecto en una consola del entorno *Cygwin*. Se procede al compilado y enlazado del proyecto y se obtiene:

```
Eloy@Lestat /cygdrive/c/cygwin/Proyectos/program_SA_copro_fpga
$ make clean all
find . -type f \
 \< -name 'core' -o -name '*.bak' -o -name '*~' \
 -o -name '*.o' -o -name '*.a' -o -name '*.tmp' \
 -o -name '*.or32' -o -name '*.bin' -o -name '*.srec' \
 -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
 -o -name '*.aux' -o -name '*.log' \> -print \
 | xargs rm -f
rm -f System.map
or32-elf-gcc -g -c -o actualitza.o actualitza.cpp
actualitza.cpp:2:20: string.h: No such file or directory
actualitza.cpp: In function 'void actualitza(int, int, int, int)':
actualitza.cpp:23: error: 'memmove' undeclared (first use this function)
actualitza.cpp:23: error: (Each undeclared identifier is reported only once for
each function it appears in.)
make: *** [actualitza.o] Error 1
```

El compilador presenta un mensaje de error que informa que la función *memmove* llamada en el fichero *actualitza.cpp* no ha sido declarada. Es posible que la herramienta no sea capaz de encontrar los ficheros de encabezamiento o librerías en las rutas especificadas durante su configuración si la instalación no se ha realizado de la forma

correcta (no es nuestro caso). Para especificar una ruta alternativa se utilizan las opciones del compilador siguientes:

- I ruta      opción del compilador para añadir la ruta especificada para la búsqueda de ficheros de encabezamiento
- L ruta      opción del enlazador para añadir la ruta especificada para la búsqueda de librerías

Cabe mencionar que para que el enlazador cruzado ensamble las funciones específicas de las librerías, los ficheros objeto de entrada al proceso deben aparecer antes de las opciones `-L` en la línea de comandos de enlazado del fichero *makefile*. Se trata de un *bug* de las herramienta, pero el proceso se ejecuta correctamente si el orden es respetado.

Cuando se han resuelto los posibles problemas de rutas del compilador y el enlazador cruzado, el resultado de la ejecución del comando `make clean all` es:

```
or32-elf-gcc -g -c -o reset.o reset.S
or32-elf-gcc -g -c -o actualitza.o actualitza.cpp
or32-elf-gcc -g -c -o angle.o angle.cpp
or32-elf-gcc -I /opt/or32-uclinux/include -g -c -o estraccio.o estraccio.cpp
or32-elf-gcc -g -c -o followridge.o followridge.cpp
or32-elf-gcc -msoft-div -g -c -o funciones.o funciones.cpp
or32-elf-gcc -g -c -o next_point.o next_point.cpp
or32-elf-gcc -g -c -o UpdateT.o UpdateT.cpp
or32-elf-gcc -g -c -o uart.o uart.cpp
or32-elf-ld -Tram.ld reset.o actualitza.o angle.o estraccio.o followridge.o funciones.o
next_point.o UpdateT.o uart.o -L /opt/or32-elf/lib -L /opt/or32-elf/lib/gcc/or32-
elf/3.4.4 -lc -lgcc -o copro.or32
or32-elf-ld: region ram is full (copro.or32 section .Tmatrix)
or32-elf-ld: region ram is full (copro.or32 section .fingerprint)
or32-elf-ld: section .fingerprint [00001000 -> 00010fff] overlaps section .Tmatrix
[00001000 -> 00010fff]
or32-elf-ld: section .text [00001000 -> 000061fb] overlaps section .Tmatrix [00001000 ->
00010fff]
or32-elf-ld: section .data [000061fc -> 00006397] overlaps section .Tmatrix [00001000 ->
00010fff]
or32-elf-ld: section .rodata [00006398 -> 000065bf] overlaps section .Tmatrix [00001000
-> 00010fff]
or32-elf-ld: section .rodata.str1.1 [000065c0 -> 000065cb] overlaps section .Tmatrix
[00001000 -> 00010fff]
or32-elf-ld: section .bss [000065d0 -> 0000714f] overlaps section .Tmatrix [00001000 ->
00010fff]
or32-elf-ld: copro.or32: section .fingerprint lma 0x1000 overlaps previous sections
or32-elf-ld: copro.or32: section .data lma 0x61fc overlaps previous sections
or32-elf-ld: copro.or32: section .rodata lma 0x6398 overlaps previous sections
or32-elf-ld: copro.or32: section .rodata.str1.1 lma 0x65c0 overlaps previous sections
make: *** [copro.or32] Error 1
```

El enlazador `ld` presenta una serie de mensajes que informan que la región de memoria especificada (`ram`) no tiene espacio para almacenar las secciones `.Tmatrix` y `.fingerprint`, además de una lista de mensajes de superposición de secciones. En definitiva, las secciones mencionadas no caben en la memoria `ram` interna del sistema. El problema del tamaño de algunas estructuras de datos ya se había previsto, ya que las variables que contienen la imagen de la huella dactilar y la matriz `T` (ambas matrices da datos de 8 bits de  $256 \times 256$  posiciones) ocupan un espacio de memoria de 64KB cada una. Por lo tanto, el código de estas variables ha sido asignado a secciones separadas del programa principal creadas para este uso específico. Esto se consigue mediante la declaración de las matrices como variables globales la especificación de la sección destino mediante `__attribute__`:

```
unsigned char copy_imatge[256*256] __attribute__((section(".fingerprint")))
bool T[256][256] __attribute__((section(".Tmatrix")))
```

Las secciones recientemente creadas se asignan a zonas de memoria pertenecientes al espacio de la memoria externa del sistema. Esto se consigue mediante la edición del fichero *ram.ld* (*script* de la herramienta de enlazado). En este fichero se crean las regiones de memoria *Tmatrix* y *fingerprint* y se les asignan las secciones pertinentes:

```
MEMORY
{
 vectors : ORIGIN = 0x00000000, LENGTH = 0x00001000
 stack : ORIGIN = 0x00001000, LENGTH = 0x00001000
 ram : ORIGIN = 0x00002000, LENGTH = 0x00005FFF
 Tmatrix : ORIGIN = 0x010E0000, LENGTH = 0x00010000
 fingerprint : ORIGIN = 0x010F0000, LENGTH = 0x00010000
}

SECTIONS
{
 .Tmatrix :
 {
 *(.Tmatrix)
 } > Tmatrix

 .fingerprint :
 {
 *(.fingerprint)
 } > fingerprint
 ...
}
```

Por otro lado, para solventar los problemas de espacio en el almacenamiento del programa principal, se duplica el tamaño de la memoria interna del sistema. Para ello se incrementa la constante *ADDR\_WIDTH* definida en el paquete *pack\_sram8* del core de memoria interna *wb\_onchip\_4sram8* diseñado en el apartado 3.3.4.1.2 del trabajo. El fichero de mayor jerarquía (*xsv\_fpga\_top.v*) debe ser editado para adaptar el tamaño de las señales conectadas al *core* para que el sistema se comporte de forma coherente. Tras reimplementar la partición dedicada al *core* de la memoria embebida se obtiene una memoria interna de 32KB que utiliza el 50% de los recursos de *block ram* del dispositivo. En el caso que fuera necesario, es posible duplicar el tamaño de la memoria interna del sistema utilizando el 100% de los recursos de la *fpga*. Sin embargo, por el momento este paso no es necesario.

Otro procedimiento para minimizar el espacio de memoria ocupado por el programa principal es la sustitución de las llamadas a las funciones *memset()* y *memmove()* de la librería *libgcc.a* por bucles que tengan un comportamiento funcional similar. Este procedimiento reduce el rendimiento de nuestro sistema, puesto que los bucles no resultan una implementación tan óptima como las funciones específicas para el tratamiento de bloques de datos y consumen un mayor número de ciclos para ser ejecutadas. Lo trozos de código que sustituyen a las funciones son:

- `memset(T, false, sizeof T)`

```
for (int i=0; i<256; i++)
 for (int j=0; j<256; j++)
 T[i][j]=false;
```

- `memmove(&M[i][0], &M[i+1][0], sizeof(int)*4*(NumMinuties-i))`

```

for (int fila=i;fila<NumMinuties;fila++)
{
 M[fila][0]=M[fila+1][0];
 M[fila][1]=M[fila+1][1];
 M[fila][2]=M[fila+1][2];
 M[fila][3]=M[fila+1][3];
}

```

Tras realizar las modificaciones descritas para resolver el problema de espacio en memoria, se repite el proceso de compilado y enlazado del proyecto. En este caso, el proceso se ejecuta sin obtener mensajes de error.

```

Floy@lestat /cygdrive/c/cygwin/proyectos/programs/test_fpga_simpuart
$ make clean all
find . -type f \
 \< -name '*.bak' -o -name '*.a' -o -name '*.tmp' \
 -o -name '*.or32' -o -name '*.bin' -o -name '*.srec' \
 -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
 -o -name '*.aux' -o -name '*.log' \> -print \
 | xargs rm -f
rm -f System.map
or32-elf-gcc -g -c -o reset.o reset.S
or32-elf-gcc -g -c -o actualitza.o actualitza.cpp
or32-elf-gcc -g -c -o angle.o angle.cpp
or32-elf-gcc -I /opt/or32-uclinux/include -g -c -o estraccio.o estraccio.cpp
or32-elf-gcc -g -c -o followridge.o followridge.cpp
or32-elf-gcc -g -c -o functions.o functions.cpp
or32-elf-gcc -g -c -o next_point.o next_point.cpp
or32-elf-gcc -g -c -o UpdateI.o UpdateI.cpp
or32-elf-gcc -g -c -o uart.o uart.cpp
or32-elf-ld -Tram.ld reset.o actualitza.o angle.o estraccio.o followridge.o func
ions.o next_point.o UpdateI.o uart.o -L /opt/or32-elf/lib -L /opt/or32-elf/lib/g
cc/or32-elf/3.4.4 -lc -lgcc -o copro.or32

```

El fichero de salida del *script* es *copro.or32*. Se trata del fichero en formato *ELF* utilizado para la programación de la *fpga* y para la obtención de los ficheros necesarios para la simulación del sistema con *ModelSim*. Se puede visualizar el contenido del fichero *ELF* en distintos formatos utilizando las líneas de comandos siguientes en una consola de *Cygwin*:

```

$ or32-elf-readelf -a copro.or32 > program.txt
$ or32-elf-objdump -d -S copro.or32 > disassemble.txt

```

### 3.5.6 Simulación Funcional

En este apartado se pretende comprobar el correcto funcionamiento del sistema diseñado mediante la simulación funcional. Con este objetivo, se debe conectar el modelo de memoria ram externa diseñado en el apartado de *hardware*, mapear la memoria interna con el programa escrito en el apartado de *software* mediante una función de inicialización, simular el fichero de *testbench* sobre el sistema con el programa *ModelSim* y comprobar la correcta ejecución del programa en el microprocesador.

Inicialmente, se simulará el comportamiento del sistema mediante el software de prueba generado para el contexto original bajo el que el coprocesador fue diseñado. El *coPMM* fue desarrollado para ser mapeado sobre una *fpga* de bajo coste de la familia *Spartan2E*. Las placas contienen un único *chip* de memoria de 8 bits de ancho. Por lo tanto, el controlador de memoria no está preparado para interactuar con los bloques de memoria de 32 bits utilizados en el presente proyecto. Una vez verificado el funcionamiento del coprocesador, el objetivo es la comprobación del funcionamiento en el entorno de trabajo real, es decir, con un modelo de memoria de 32 bits mediante la adición de lógica de adaptación externa al coprocesador.

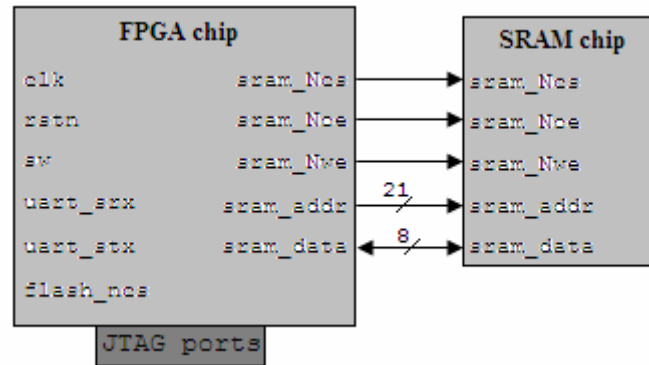


Imagen 56 - Detalle de la conexión entre FPGA y memoria externa de 8 bits

### 3.5.6.1 Creación del proyecto de simulación

Con el propósito de aprovechar el trabajo realizado en la primera sección del proyecto con la simulación funcional del sistema básico, se realiza una copia del proyecto de simulación *modelsim\_SB* y se renombra como *modelsim\_copro*. Se edita el fichero de proyecto *modelsim.mpf* y se modifican las rutas de los ficheros fuente para que *ModelSim* utilice los ficheros fuente del sistema avanzado con coprocesador, es decir, se reemplaza *rtl\_SB* por *rtl\_SA\_copro*. En este punto se añaden al proyecto los ficheros del coprocesador (todos escritos en lenguaje *vhdl*):

- *follow.vhd*
- *corrmax.vhd*
- *filtsec.vhd*
- *nextpoint.vhd*
- *pack\_maioHw.vhd*
- *section.vhd*
- *sramcontrol\_8\_8.vhd*
- *sramcontrol\_8\_32.vhd*
- *tangdir.vhd*
- *xynext.vhd*
- *wb\_bridge\_maioHw.vhd*

A su vez, el fichero que modela la memoria ram externa (*sram\_sim.vhd*) y la inicializa (*fingerprint.vhd*), el fichero que define el comportamiento del *buffer tri-state* (*tristate.v*) y el fichero de *testbench* son añadidos a la jerarquía de nivel superior del proyecto de simulación.

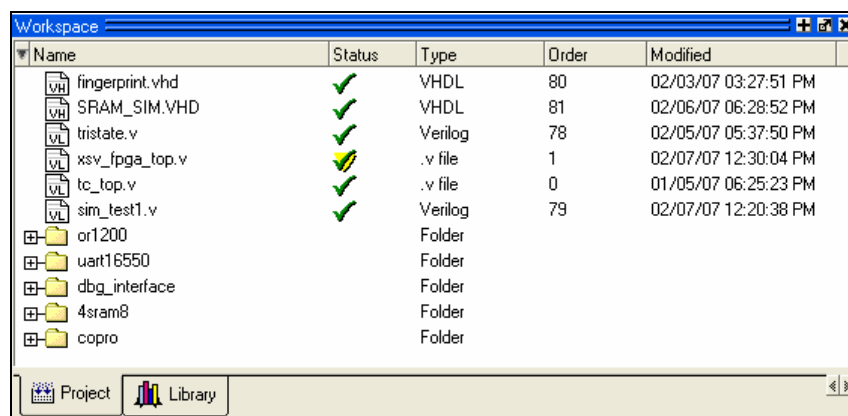


Imagen 57 - Ventana *workspace* del proyecto de simulación de *ModelSim*

La imagen muestra el nivel superior en la jerarquía del proyecto de simulación. Los ficheros y carpetas describen:

- *xsv\_fpga\_top.v* definición del módulos de la *fpga* e instancia de sus módulos principales
- *tc\_top.v* definición del módulo del árbitro del bus
- *sram\_sim.vhd* definición del modelo de memoria externa usado para simulación
- *sim\_test1.v* fichero de *testbench*
- *fingerprint.vhd* funciones y datos para la inicialización del modelo de memoria externa
- *tristate.v* definición del *buffer tri-state*
- *or1200* carpeta que contiene los ficheros referentes al módulo del microprocesador *or1200*
- *dbg\_interface* carpeta que contiene los ficheros referentes al módulo del interface del depurador
- *uart16550* carpeta que contiene los ficheros referentes al módulo del interface de la uart del PC
- *4sram8* carpeta que contiene los ficheros referentes al módulo del controlador de memoria interna y los ficheros dedicados a su simulación
- *copro* carpeta que contiene los ficheros referentes al módulo del coprocesador y los ficheros dedicados a su simulación

### 3.5.6.2 Fichero *testbench*

El fichero de simulación del sistema es editado para realizar la estimulación de las entradas y la conexión de los diferentes bloques correctamente. Las modificaciones realizadas en el fichero son las listadas a continuación:

- Añadir al chip de la *fpga* los puertos de entrada y/o salida utilizados para la comunicación con el chip de memoria externa
- Crear el cableado necesario para la conexión de los chips y la lógica de adaptación.
- Instanciar un único modelo de memoria ram externa con una anchura de 8 bits

El código resultante en el fichero de *testbench* es:

```
`define SRAM_EXT_AW 21
module sim_test1;

 reg clk, Nrst;
 reg[2:1] sw;
 wire uart_srx;
 wire uart_stx;
 wire sram_Ncs, sram_Noe, sram_Nwe;
 wire [`SRAM_EXT_AW-1:0] sram_addr;
 wire [7:0] sram_data;
 wire flash_Ncs;
 wire jtag_tck, jtag_tms, jtag_tdi, jtag_trst;
```

```

wire jtag_tvref, jtag_tgnd, jtag_tdo;
assign uart_stx=1'b0;
assign {jtag_tck,jtag_tms,jtag_tdi,jtag_trst}='b0;

xsv_fpga_top chip_fpga (
 .clk(clk),
 .rstn(Nrst),
 .sw(sw),
 .uart_srx(uart_srx),
 .uart_stx(uart_stx),
 .sram_Ncs(sram_Ncs),
 .sram_Noex(sram_Noex),
 .sram_Nwe(sram_Nwe),
 .sram_addr(sram_addr),
 .sram_data(sram_data),
 .flash_Ncs(flash_Ncs),
 .jtag_tck(jtag_tck),
 .jtag_tms(jtag_tms),
 .jtag_tdi(jtag_tdi),
 .jtag_trst(jtag_trst),
 .jtag_tvref(jtag_tvref),
 .jtag_tgnd(jtag_tgnd),
 .jtag_tdo(jtag_tdo)
);
sram_sim ramext_chip(
 .Ncs(sram_Ncs),
 .Noex(sram_Noex),
 .Nwe(sram_Nwe),
 .addr(sram_addr[`SRAM_EXT_AW-1:0]),
 .data(sram_data)
);

initial begin
 clk=1'b0;
 sw=2'b00;
 Nrst=1'b0;
 #100 Nrst=1'b1;
end
always
begin
 #10 clk=!clk;
end
endmodule

```

### 3.5.6.3 Inicialización de la memoria interna

En este apartado se pretende generar el fichero escrito en lenguaje *vhdl* con el formato adecuado para poder ser utilizado para inicializar la memoria interna del sistema. A partir del fichero ELF (*controller.or32*) obtenido tras el compilado y el enlazado de los ficheros fuente se realizan dos conversiones de formato de los datos. Con este fin, se puede reciclar el trabajo realizado en la sección anterior y utilizar la aplicación escrita en lenguaje C++. Se introduce la línea de código “*or32-elf-objcopy -O binary controller.or32 controller.bin*” en la consola de *Cygwin* para convertir el formato del fichero ELF en formato binario. A continuación se llama al ejecutable de la aplicación creada en la sección anterior mediante la línea de código “*bin2vhdl controller.bin controller.vhd*” para convertir los datos binarios al formato de carácter hexadecimal deseado.

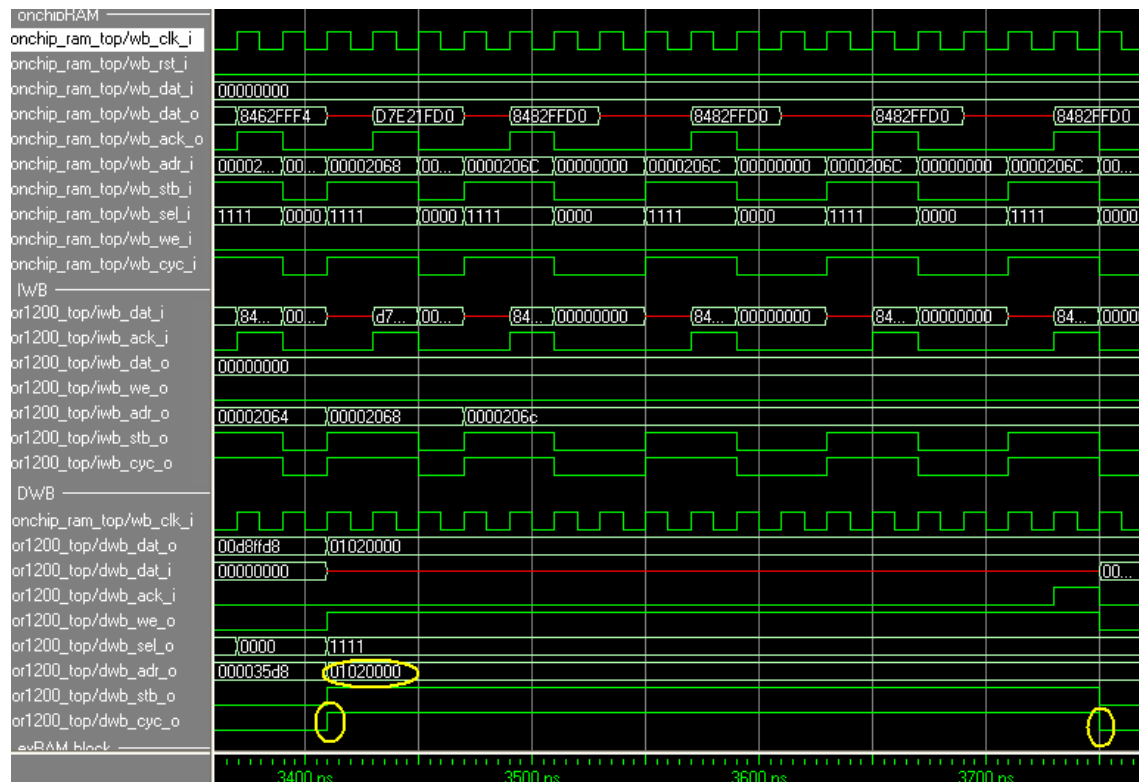
#### 3.5.6.4 Comprobación del funcionamiento en el contexto original

En este apartado se presenta una descripción de la simulación obtenida para los programas de prueba generados. Para cada programa se ha generado un paquete escrito en *vhd* que contiene las instrucciones en el formato adecuado para que una función específica de la simulación inicialice la memoria interna del sistema. Se trata de dos programas que comprueban el funcionamiento del coprocesador en sus dos modos principales: controlador de memoria e iteración del algoritmo.

### a) Modo controlador de memoria

En este punto se pretende comprobar que el microprocesador realiza los accesos de lectura y de escritura a memoria externa de forma correcta. El sistema responde de la misma manera a la reinicialización que el sistema básico de la sección anterior. Tras el reset del sistema, el microprocesador realiza un acceso de lectura al vector de excepción de reset (dirección 0x100) mapeado en la memoria interna del sistema. El interface de instrucciones (*IWB*) del *or1200* realiza el *fetch* de las instrucciones mapeadas tras el vector (definidas en el fichero fuente *reset.S*) hasta que el programa fuerza un *fetch* de la primera instrucción del programa principal (definido en el fichero fuente *checkmem.cpp*).

En la rutina principal se realiza el *fetch* de las primeras instrucciones del programa (mapeadas en memoria interna) alternando con accesos de lectura y escritura a datos (también mapeados en memoria interna) cuando las instrucciones lo determinan. A los 1450ns de simulación el *or1200* realiza un *fetch* de la primera instrucción de la función *check\_memoria* (dirección 0x2000) y es en el tiempo de simulación de 2100ns cuando el procesador entra en el primer bucle de accesos a memoria externa de la función. A los 3400ns se realiza el primer acceso a memoria externa:



**Imagen 58** - Detalle de un acceso de instrucciones y datos simultáneo

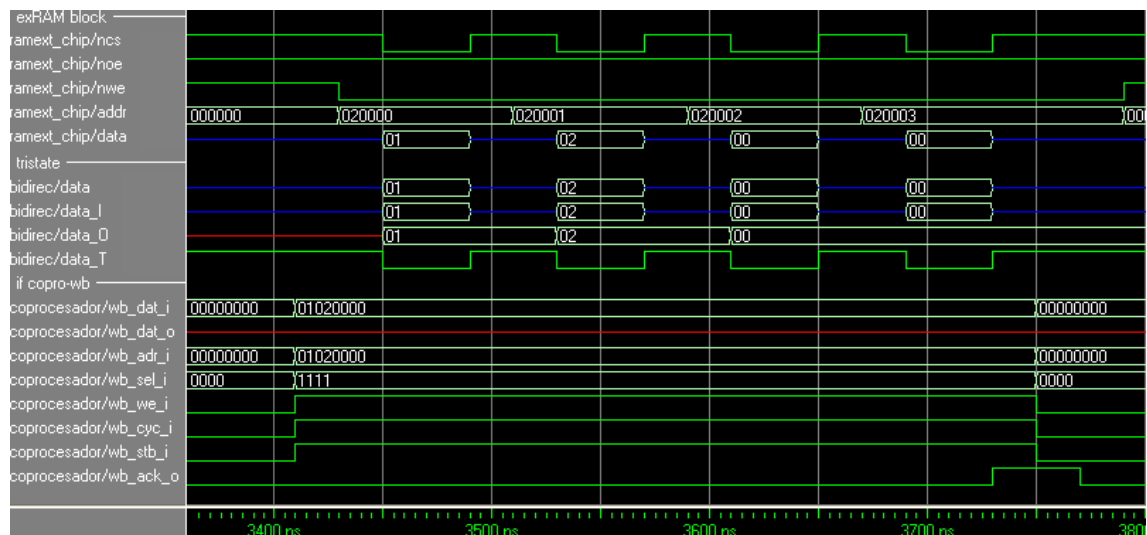


En la imagen se observa que en el tiempo de simulación de 3410ns el procesador realiza el *fetch* de la instrucción mapeada en la dirección 0x00002068 además de, al mismo tiempo, realizar un acceso de escritura de datos a la dirección 0x01020000 (mapeada en memoria externa). El *iwb* inicia un ciclo de lectura en la dirección 0x00002068 de la memoria interna y realiza el *fetch* de la instrucción 0xd7e21fd0 en el siguiente ciclo de reloj. Al mismo tiempo, el *dwb* realiza un acceso de escritura en la dirección 0x01020000 de la memoria externa del dato 0x01020000. La duración de un ciclo de escritura de una palabra de 32 bits es de 16 ciclos de reloj según la señal *dwb\_cyc\_o*. Esto se debe a que el controlador de memoria del coprocesador realiza cuatro accesos consecutivos de 8 bits para escribir cada *byte* de la palabra completa.

A continuación se muestran los detalles de los diferentes accesos de escritura y de lectura a memoria externa para los distintos tamaños de datos posibles: 8, 16 y 32 bits. Para la escritura de una palabra de 32 bits en memoria externa el *dwb* del *openrisc* transmite, a través del *traffic cop*, las siguientes señales al interface *wishbone* del coprocesador:

- *wb\_cyc\_i* = 1, *wb\_we\_i* = 1 → ciclo de escritura
- *wb\_adr\_i* = 0x01020000, *wb\_stb\_i* = 1 → a la dirección hexadecimal 01020000
- *wb\_dat\_i* = 0x01020000, *wb\_sel\_i* = 1111 → la palabra hexadecimal 01020000

El controlador de memoria embebido al coprocesador realiza cuatro accesos consecutivos de escritura de datos de 8 bits en el formato *big endian*, es decir, el *MSB* se escribe en la dirección inferior y el *LSB* en la superior. Esta codificación es la utilizada por el *openrisc* para los accesos a memoria. Cuando el coprocesador ha escrito los cuatros *bytes* de la palabra en memoria, éste aserta la señal *wb\_ack\_o* que indica al *openrisc* que el ciclo de escritura ha terminado con éxito.



**Imagen 59** - Detalle de un acceso de escritura de 32 bits a memoria externa de 8 bits

Tras varios accesos de escritura de 32 bits a la memoria externa, el *openrisc* realiza la operación opuesta, es decir, accede a memoria externa para la lectura de las palabras previamente escritas. Para ello, el *dwb* del microprocesador transmite:

- *wb\_cyc\_i* = 1, *wb\_we\_i* = 0 → ciclo de lectura
- *wb\_adr\_i* = 0x01020000, *wb\_stb\_i* = 1 → a la dirección hex. 01020000
- *wb\_sel\_i* = 1111 → acceso de 32 bits

El controlador realiza cuatro accesos consecutivos de lectura para los cuatro *bytes* de la palabra y transmite el dato de 32 bits al *openrisc* por la señal *wb\_dat\_o* asertando la señal de control *wb\_ack\_o*.

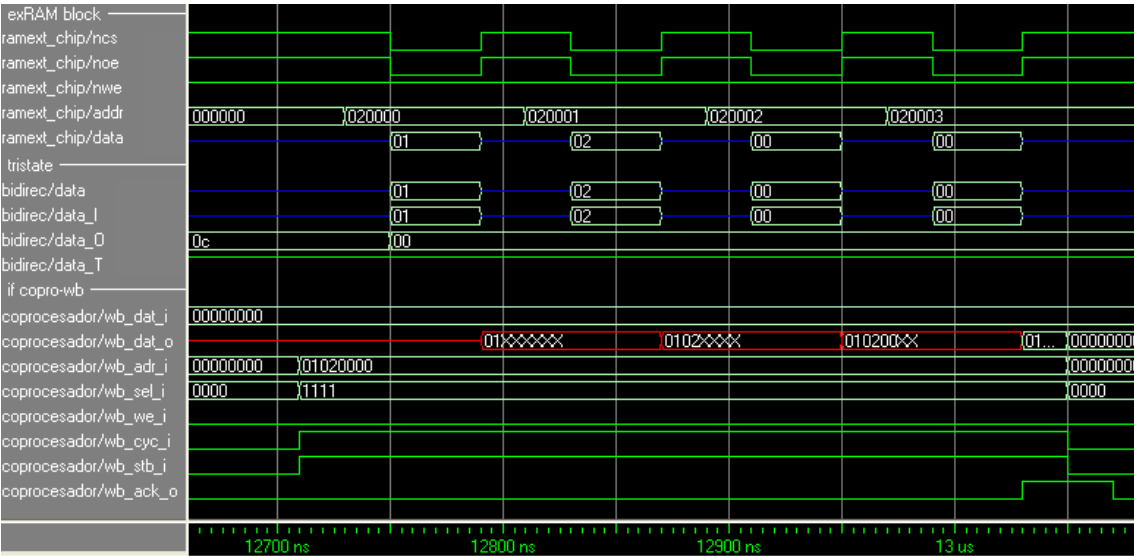


Imagen 60 - Detalle de un acceso de lectura de 32 bits a memoria externa de 8 bits

Tras cuatro accesos de escritura de 32 bits, se comprueba mediante los accesos de lectura que el bloque de memoria accedido queda de la siguiente manera:

|          | +3 | +2 | +1 | +0 |
|----------|----|----|----|----|
| 0102000c | 0c | 00 | 02 | 01 |
| 01020008 | 08 | 00 | 02 | 01 |
| 01020004 | 04 | 00 | 02 | 01 |
| 01020000 | 00 | 00 | 02 | 01 |

Para la escritura de una palabra de 16 bits en memoria externa el *dwb* del microprocesador transmite las mismas señales que en el caso anterior con la diferencia de que replica los 16 bits en el bus de datos *wb\_dat\_i* y solamente valida 16 de los 32 bits de dicho bus mediante la señal *wb\_sel\_i*. Por lo tanto, los accesos consecutivos realizados por el controlador en este caso son dos.

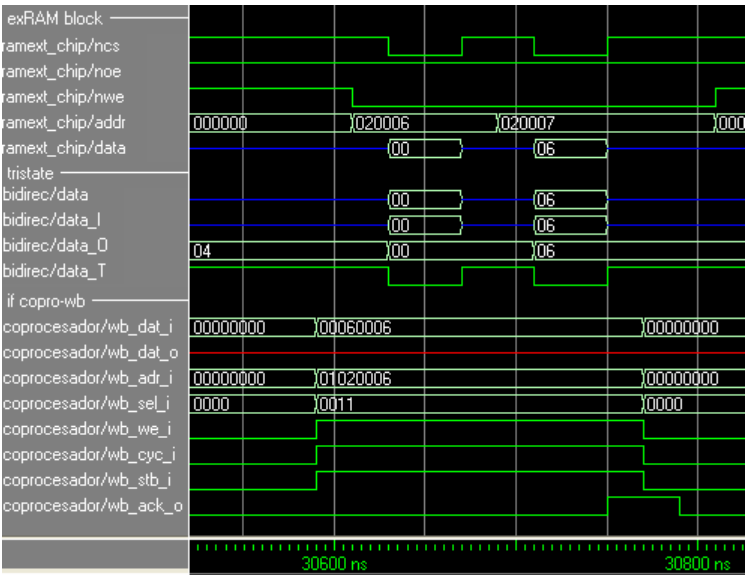
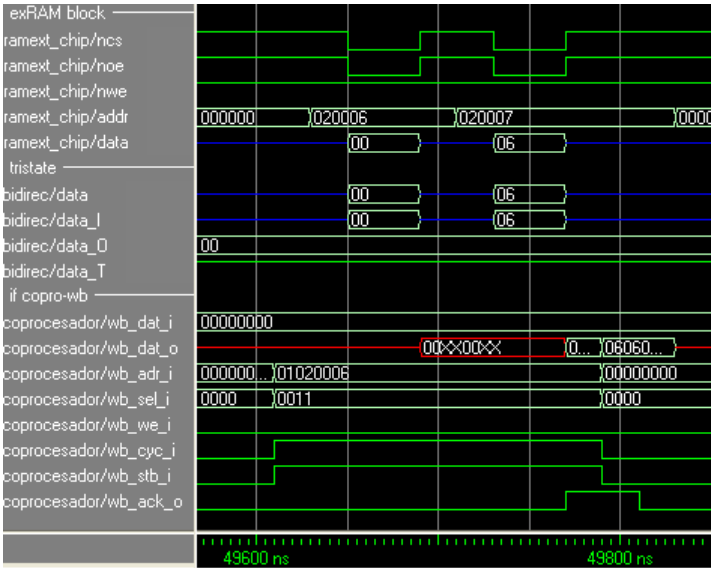


Imagen 61 - Detalle de un acceso de escritura de 16 bits a memoria externa de 8 bits

Para la lectura, los datos de 16 bits son de la misma manera replicados en el bus de datos de salida `wb_dat_o`.

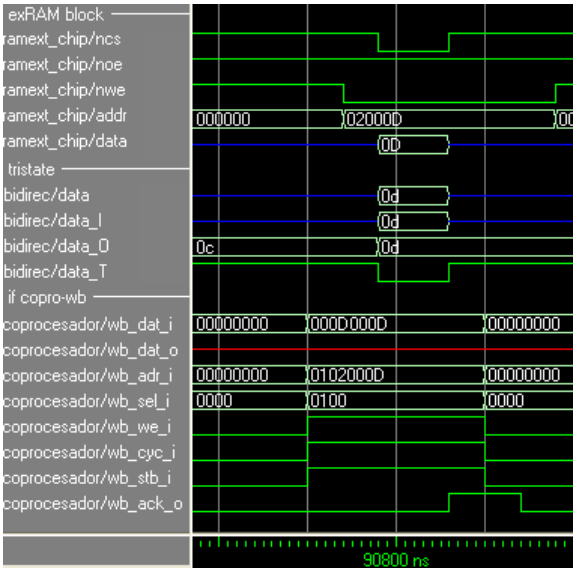


**Imagen 62** - Detalle de un acceso de lectura de 16 bits a memoria externa de 8 bits

Tras ocho accesos de escritura de 16 bits, se comprueba mediante los accesos de lectura que el bloque de memoria accedido queda de la siguiente manera:

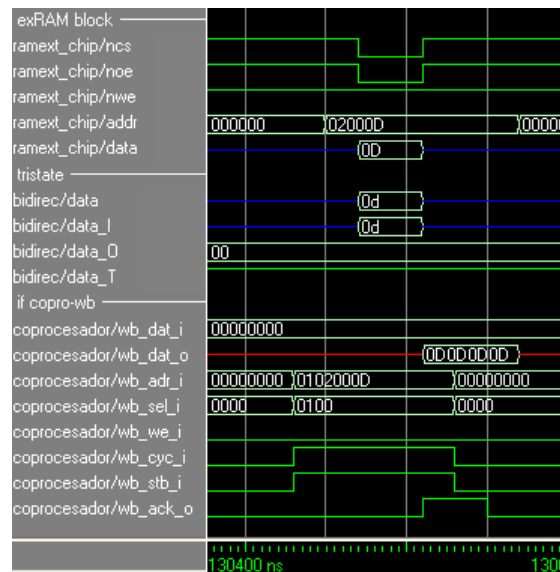
|                 | +3 | +2 | +1 | +0 |
|-----------------|----|----|----|----|
| <b>0102000c</b> | 0e | 00 | 0c | 00 |
| <b>01020008</b> | 0a | 00 | 08 | 00 |
| <b>01020004</b> | 06 | 00 | 04 | 00 |
| <b>01020000</b> | 02 | 00 | 00 | 00 |

En el caso de el acceso de escritura de un *byte* (8 bits) éste es replica en el bus de entrada `wb_dat_i` y validados solamente 8 de los 32 bits mediante la señal `wb_sel_i`. En este caso solamente se realiza un acceso de escritura a memoria.



**Imagen 63** - Detalle de acceso de escritura de 8 bits a memoria externa de 8 bits

El *byte* accedido en el acceso de lectura es replicado de la misma manera en el bus de datos de salida `wb_dat_o` y validado mediante la señal de control `wb_ack_o`.



**Imagen 64** - Detalle de acceso de lectura de 8 bits a memoria externa de 8 bits

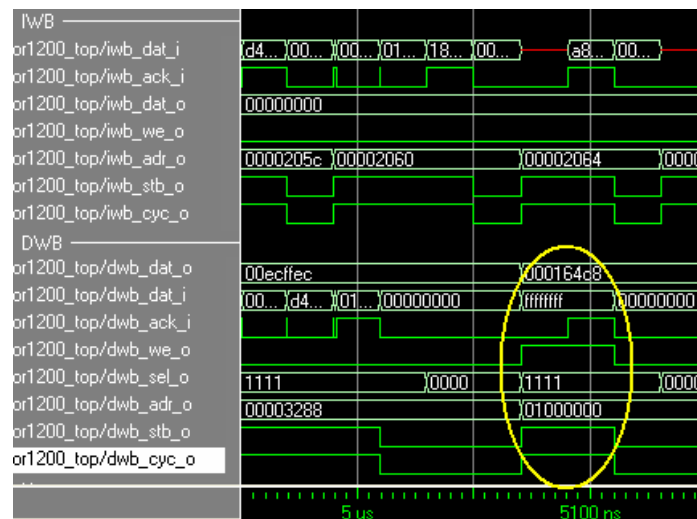
Tras 16 accesos de escritura de 8 bits, se comprueba mediante los accesos de lectura que el bloque de memoria accedido queda de la siguiente manera:

|                 | +3 | +2 | +1 | +0 |
|-----------------|----|----|----|----|
| <b>0102000c</b> | 0f | 0e | 0d | 0c |
| <b>01020008</b> | 0b | 0a | 09 | 08 |
| <b>01020004</b> | 07 | 06 | 05 | 04 |
| <b>01020000</b> | 03 | 02 | 01 | 00 |

En resumen, se ha comprobado el buen funcionamiento del controlador de memoria embebido al coprocesador en su contexto original. Realiza correctamente accesos de escritura y de lectura a memoria externa con tamaños de datos de 32, 16 y 8 bits. En el siguiente punto se comprueba que el coprocesador realiza la iteración del algoritmo correctamente.

## b) Modo coprocesador

Para simular el comportamiento del coprocesador, se añade al proyecto del *ModelSim* el fichero *vhdl* obtenido a partir del programa *checkcopro.cpp* y se compila. Esta operación nos permite trabajar con la memoria interna del sistema inicializada con los datos del programa. En la simulación, el sistema responde de forma idéntica a la reinicialización (rutina de servicio a la excepción *reset*) e inicia la ejecución del programa inicializado en memoria. Los interfaces de instrucciones (*iwb*) y de datos (*dwb*) del *openrisc* realizan el *fetch* de las instrucciones del programa y acceden a los datos necesarios mapeados en memoria interna. En el tiempo de simulación de 5070ns el *dwb* del microprocesador transmite al interface *wishbone* del coprocesador las señales necesarias para realizar un acceso de escritura al registro de propósito especial del coprocesador mapeado en la dirección 0x01000000 de la memoria externa. El dato escrito en el registro es 0x000164c8.



**Imagen 65** - Detalle del acceso de escritura del dwb del openrise

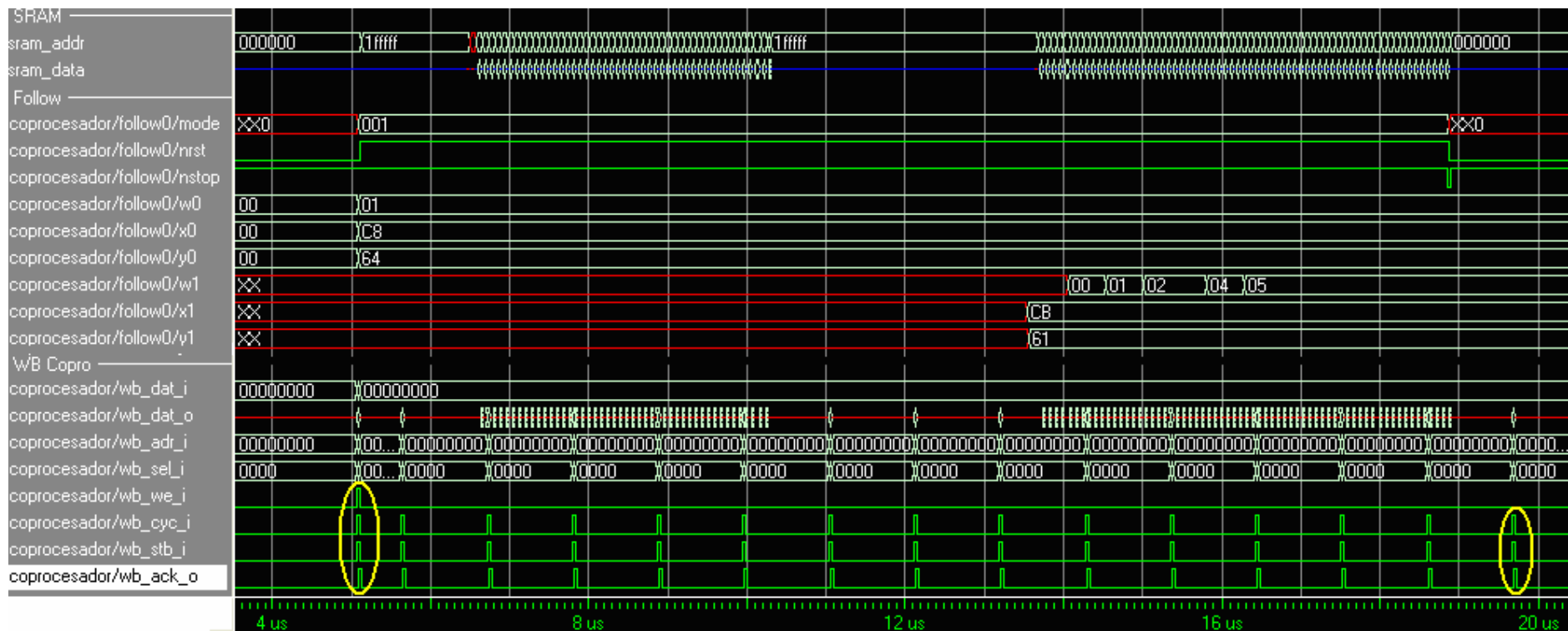


Imagen 66 - Detalle del inicio de la operación del coprocesador y posterior espera de resultados

El *dwb* del microprocesador realiza un acceso inicial de escritura (*wb\_we\_i* = 1) al registro de propósito especial que inicia la operación del coprocesador (señal *nrst* = 1). El interface de instrucciones (*iwb*) sigue haciendo el *fetch* de las instrucciones del programa, que ha entrado en un bucle de espera, forzando al *dwb* a realizar una secuencia de accesos de lectura del mismo registro hasta que, aproximadamente en el tiempo de simulación de 19us, el coprocesador finaliza su operación (*nrst* = 0 y *nstop* = 0) y, en el tiempo de simulación de 20us, el *dwb* obtiene un valor diferente a 0xffffffff (valor -1). En ese preciso momento, el programa sale del bucle de espera y continua la ejecución del programa.

Sin embargo, se observa que el coprocesador realiza dos series de accesos consecutivos independientes de los accesos de lectura realizados por el *dwb*. Se trata de los accesos de lectura efectuados durante la ejecución de las etapas de filtrado de la sección (*filt*) y de cálculo de la nueva dirección (*tgdir*) para el seguimiento de la huella dactilar.

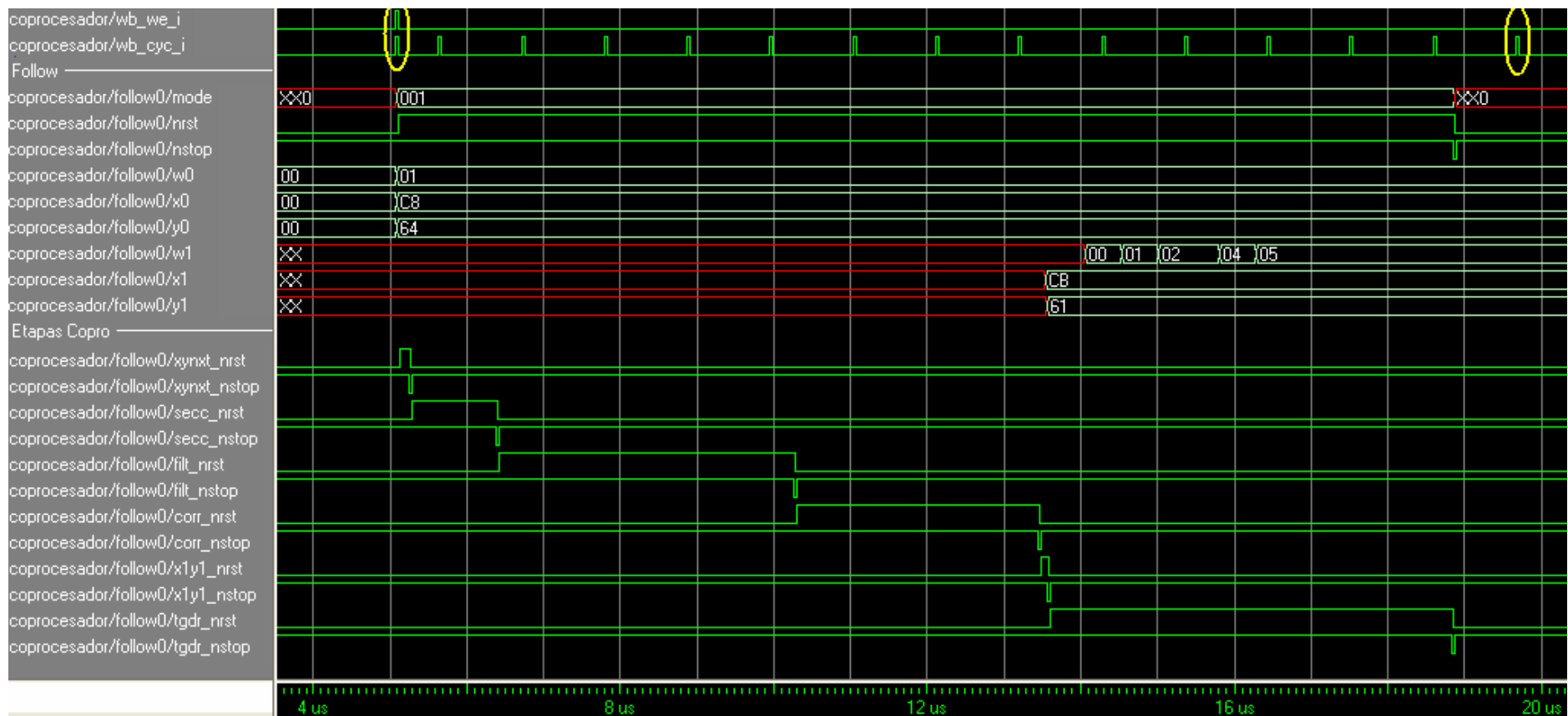
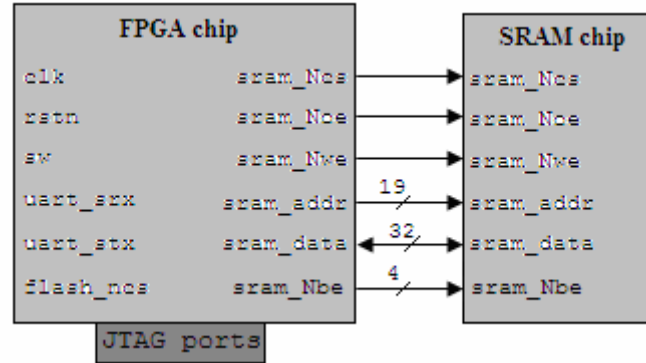


Imagen 67 - Detalle de la operación del coprocesador y de sus 6 etapas

A partir del acceso de escritura del dato 0x000164c8 en el registro de propósito especial, el coprocesador inicia su operación ( $nrst = 1$ ) y con los valores iniciales asignados a las señales:  $mode = 001$ ,  $w0 = 01$ ,  $x0 = c8$  y  $y0 = 64$ . Las señales  $nrst$  específicas de cada etapa indican el inicio y la duración de cada etapa, mientras que las señales  $nstop$  específicas determinan el final de la etapa. Los resultados obtenidos por el coprocesador a partir de los datos iniciales y de la huella dactilar almacenada en memoria son:  $w1 = 05$ ,  $x1 = cb$  y  $y1 = 61$ . De esta forma, la palabra de 32 bits transmitida al microprocesador por el  $dwb$  es 0x000561cb.

### 3.5.6.5 Adaptación del sistema al contexto real

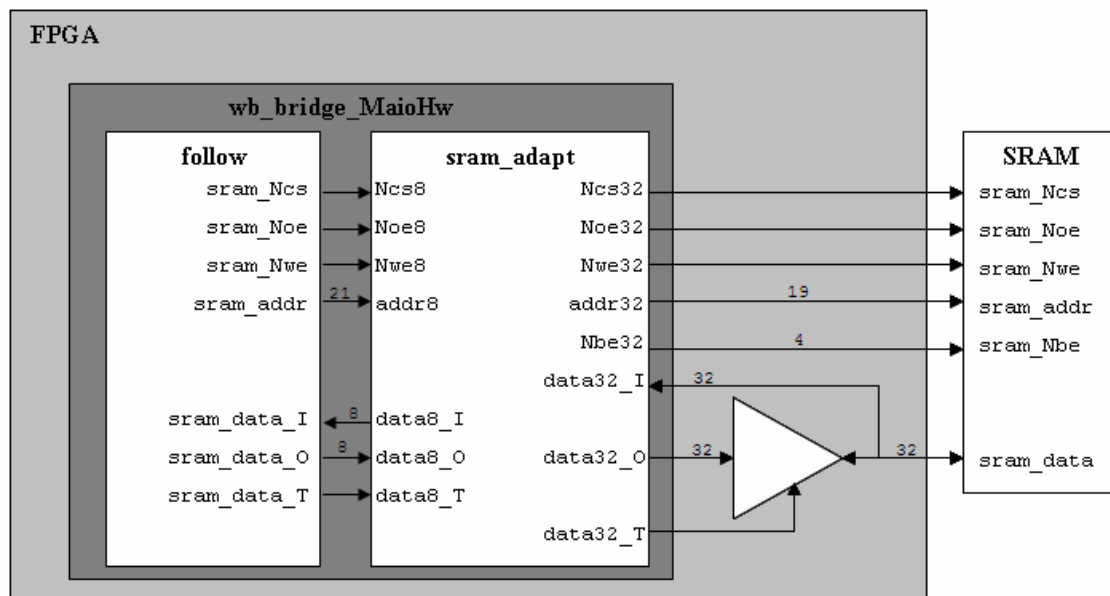
El objetivo de este punto es realizar la adaptación de la lógica que mantiene al coprocesador y la memoria externa conectados, para que el sistema pueda trabajar en el contexto real del proyecto, es decir, accediendo a una memoria externa de 32 bits.



**Imagen 68** - Detalle de la conexión entre FPGA y memoria externa de 32 bits

Como se ha dicho anteriormente, el coprocesador fue desarrollado para trabajar con memorias de 8 bits, realizando accesos consecutivos para obtener palabras de mayor tamaño. Algunas etapas del algoritmo realizan múltiples accesos a memoria externa, por lo que el coprocesador fue diseñado para realizar de forma síncrona diferentes operaciones paralelas. En definitiva, la arquitectura del coprocesador está condicionada de manera importante por el patrón de accesos consecutivos de 8 bits.

Sin embargo, el desarrollo y/o actualización del coprocesador está fuera de los límites del presente proyecto. Por esta razón, se pretende trabajar con el diseño original del coprocesador sin modificar su código fuente. Para poder conectar el coprocesador a una memoria externa de 32 bits, se pretende diseñar un módulo que contenga la *glue logic* necesaria para adaptar las señales ambos dispositivos.



**Imagen 69** - Descripción esquemática de la conexión coprocesador-memoria externa

El proceso de adaptación del sistema se resume en los pasos descritos a continuación:

- Ajuste de la anchura de los buses externos al coprocesador:



- El *buffer tristate* dedicado a la conversión de las líneas de datos se convierte a una anchura de 32 bits (archivo *tristate.v*).
- Se añade el cableado y los puertos de I/O (`sram_Nbe[3:0]`) necesarios y se actualiza la anchura de los buses de datos (32) y de direcciones (19).
- Conexión del módulo de adaptación *sram\_adapt* dentro del coprocesador:
  - Se crean las señales intermedias (`sram8_...`) que conectan el módulo principal del coprocesador (*follow*) con el módulo de adaptación.
  - El módulo *sram\_adapt* se puede ver como una caja negra de la que solamente se conocen los puertos de entrada/salida. El código siguiente, contenido en la arquitectura del módulo *user\_maioHw*, describe la conexión (ver *Imagen 69*) del módulo de adaptación dentro del módulo puente de *wishbone*.

```

component sram_adapt is
port(
 sram8_Ncs,sram8_Noel,sram8_Nwe: in std_logic;
 sram8_addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
 sram8_data_I: out std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram8_data_O: in std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram8_data_T: in std_logic;
 sram32_Ncs,sram32_Noel,sram32_Nwe: out std_logic;
 sram32_Nbe: out std_logic_vector(3 downto 0);
 sram32_addr: out std_logic_vector(ADDR_WIDTH-1 downto 2);
 sram32_data_I: in std_logic_vector(31 downto 0);
 sram32_data_O: out std_logic_vector(31 downto 0);
 sram32_data_T: out std_logic
);
end component;

sram_adapt0: sram_adapt port map(
 sram8_Ncs,sram8_Noel,sram8_Nwe,sram8_addr,
 sram8_data_I,sram8_data_O,sram8_data_T,
 sram32_Ncs,sram32_Noel,sram32_Nwe,sram32_Nbe,
 sram32_addr,sram32_data_I,sram32_data_O,sram32_data_T);

```

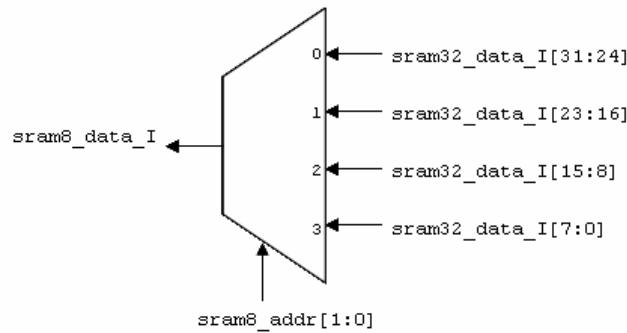
- Diseño del módulo de adaptación (*sram\_adapt*) en el archivo *sram\_adapt.vhd* mediante lenguaje *vhdl*. A continuación, se describe la interconexión de las líneas:
  - Conexión directa de las líneas
 

|                         |               |                          |
|-------------------------|---------------|--------------------------|
| <code>sram8_Ncs</code>  | $\rightarrow$ | <code>sram32_Ncs</code>  |
| <code>sram8_Noel</code> | $\rightarrow$ | <code>sram32_Noel</code> |
| <code>sram8_Nwe</code>  | $\rightarrow$ | <code>sram32_Nwe</code>  |
  - Segmentación del bus de direcciones *sram8\_addr* de 21 bits. Los dos bits de menor peso se utilizan como bits internos de control y los 19 restantes se conectan al bus de direcciones de salida (*sram32\_addr*).
  - En cada acceso de lectura a memoria externa, el módulo de adaptación debe seleccionar a cual de los cuatro bytes que componen la palabra obtenida se quiere acceder. Se define un multiplexor que selecciona el byte accedido según los dos bits de menor peso del bus de direcciones:

```

process (sram32_data_I,addr_be) begin
 case (addr_be) is
 when "00" => sram8_data_I<=sram32_data_I(31 downto 24);
 when "01" => sram8_data_I<=sram32_data_I(23 downto 16);
 when "10" => sram8_data_I<=sram32_data_I(15 downto 8);
 when "11" => sram8_data_I<=sram32_data_I(7 downto 0);
 when others => sram8_data_I<=(others=>'Z');
 end case;
end process;

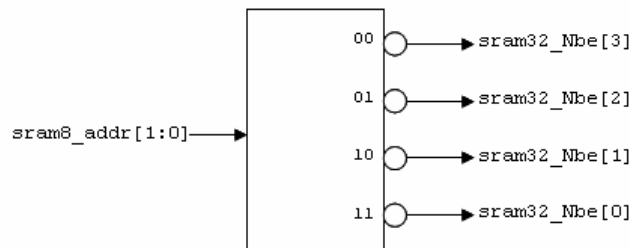
```



**Imagen 70** - Esquema del MUX de 4 entradas

- En cada acceso de escritura, la entidad *sram\_adapt* replica el byte obtenido en su puerto de entrada *sram8\_data\_O* en los cuatro bytes del bus de datos de salida *sram32\_data\_O*. En este caso, se define un decodificador que realiza la gestión de las señales de *byte enable* de los cuatro bloques de memoria utilizados para la simulación. La gestión se realiza a partir de los dos bits de menor peso del bus de direcciones:

```
process (addr_be) begin
 case (addr_be) is
 when "00" => sram32_Nbe <= "0111";
 when "01" => sram32_Nbe <= "1011";
 when "10" => sram32_Nbe <= "1101";
 when "11" => sram32_Nbe <= "1110";
 when others => sram32_Nbe <= "1111";
 end case;
end process;
```



**Imagen 71** - Esquema del decodificador 2a4

- Adaptación del modelo de memoria externa *sram\_sim*. Para realizar las posteriores simulaciones del comportamiento del sistema, se debe poder inicializar la memoria externa con los datos de la huella dactilar. Sin embargo, la memoria externa está formada por cuatro bloques de memoria (hasta ahora idénticos) que deben ser inicializados con datos diferentes. El problema está en que el lenguaje *verilog* no permite definir diferentes arquitecturas en una misma entidad. La solución es describir un modelo de memoria diferente para cada bloques que llame a una función de inicialización diferente. No se trata de la solución más óptima, pero resulta útil para realizar la simulación:
  - Diseño de los cuatro modelos *sram\_sim0*, *sram\_sim1*, *sram\_sim2* y *sram\_sim3*. Se trata de entidades similares que llaman a funciones de inicialización diferentes.
  - Creación de las cuatro funciones de inicialización. Se trata de funciones similares en las que varía la condición de escritura en el *array* de memoria. La función recorre cada *byte* de la huella dactilar escribiendo solamente los que corresponden con el bloque de memoria.

```

function init_sram3(ADDR_IMG0:std_logic_vector; fingerprint: T_FINGERPRINT)
 return t_sram is
variable i,j,k: integer:=0;
variable val_ram: t_sram;
variable ADDR0_IMG: unsigned(sram_addr_width-1 downto 0):=unsigned(ADDR_IMG0
 (sram_addr_width-1 downto 0));
variable ADDR1_IMG: unsigned(sram_addr_width-1 downto 0):=ADDR0_IMG+X"FFFF";
begin
for i in conv_integer(ADDR0_IMG) to conv_integer(ADDR1_IMG) loop
 if (j=0) then
 val_ram(conv_integer(ADDR0_IMG)+k):=fingerprint(i-
 conv_integer(ADDR0_IMG));
 k:=k+1;
 end if;
 j:=j+1;
 if (j=4) then
 j:=0;
 end if;
end loop;
return val_ram;
end function;

```

Se debe tener en cuenta que tanto el microprocesador como el coprocesador acceden a memoria de forma coherente con el formato *big endian*, es decir, la palabra de 32 bits 0x12345678 se escribe en la dirección 0x2000 de memoria de la forma siguiente:

|        |    |    |    |    |
|--------|----|----|----|----|
|        | +3 | +2 | +1 | +0 |
| 0x2000 | 78 | 56 | 34 | 12 |

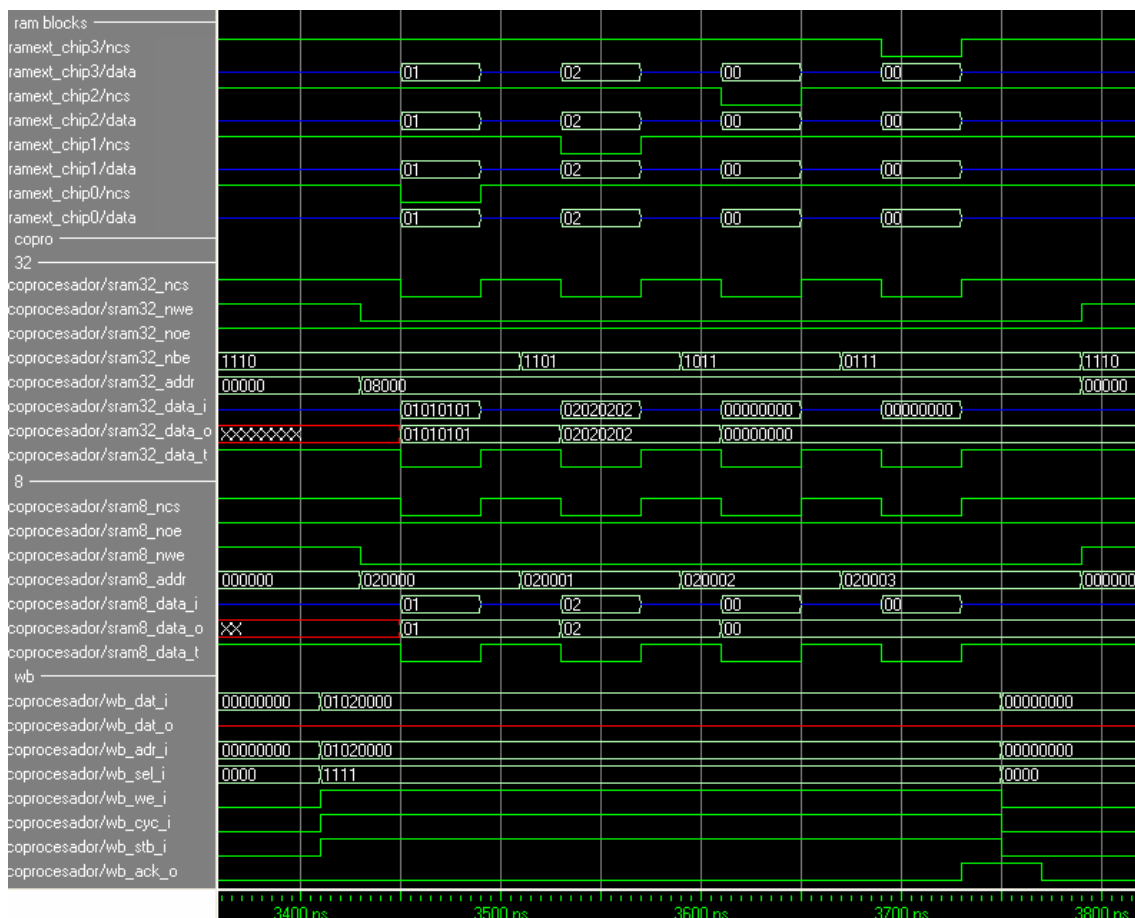
### 3.5.6.6 Comprobación del funcionamiento en el contexto real

Para realizar la simulación funcional del sistema avanzado con coprocesador trabajando en el contexto real (con una memoria externa de 32 bits), se ha creado otro proyecto de simulación. A partir un duplicado de la versión del proyecto de simulación ubicada en `./modelsim/sim_funcional/copro` se crea el nuevo proyecto *copro\_adaptado*. Mediante la edición del fichero de proyecto *modelsim.mpf* se modifican las rutas de todos los ficheros fuente para que *ModelSim* utilice los ficheros fuente del sistema avanzado con el coprocesador adaptado, es decir, se reemplaza *rtl\_SA\_copro* por *rtl\_SA\_copro\_adaptado*. Además, los ficheros que se encuentran en la nueva ruta definen el modelo de memoria externa de 32 bits descrito en el apartado anterior. En este punto se añade al proyecto el fichero *sram\_adapt.vhd* que contiene la lógica de adaptación.

La simulación de comportamiento del coprocesador en ambos modos de trabajo (modo controlador de memoria y modo iteración del coprocesador) resulta muy similar a la obtenida en el apartado 3.5.4. Los resultados de simulación difieren en los accesos a memoria externa. En la *Imagen 72* se comprueba la correcta adaptación de las señales en el caso de un ciclo de escritura. Se aprecian los cuatro accesos consecutivos de escritura de los cuatro *bytes* que componen la palabra 0x01020000. Los datos se escriben en memoria según el formato *big endian*.

- 1) sram8\_nwe=0 → sram32\_nwe=0  
sram8\_addr=0x20000 → sram32\_addr=0x8000  
sram8\_data\_O=0x01 → sram32\_data\_O=0x01010101 y sram32\_Nbe=1110  
escritura en el chip0

- 2) sram8\_nwe=0 → sram32\_nwe=0  
sram8\_addr=0x20001 → sram32\_addr=0x8000  
sram8\_data\_O=0x02 → sram32\_data\_O=0x02020202 y sram32\_Nbe=1101  
escritura en el chip1
- 3) sram8\_nwe=0 → sram32\_nwe=0  
sram8\_addr=0x20002 → sram32\_addr=0x8000  
sram8\_data\_O=0x00 → sram32\_data\_O=0x00000000 y sram32\_Nbe=1011  
escritura en el chip2
- 4) sram8\_nwe=0 → sram32\_nwe=0  
sram8\_addr=0x20003 → sram32\_addr=0x8000  
sram8\_data\_O=0x00 → sram32\_data\_O=0x00000000 y sram32\_Nbe=0111  
escritura en el chip3



**Imagen 72** - Detalle de un acceso de escritura de 32 bits a memoria externa de 32 bits

En la *Imagen 73* se comprueba que el controlador embedido en el coprocesador obtiene de memoria externa la palabra de 32 bits correcta. En esta caso, el controlador realiza cuatro accesos consecutivos de lectura de los cuatro *bytes* escritos en anteriormente con el formato *big endian*. La palabra de 32 bits obtenida en el bus de datos de salida del interface *wishbone* del coprocesador (marcada en la simulación) es 0x01020000.



### 3.5.7.1 Creación del proyecto

En la primera sección del proyecto se ha creado el proyecto de síntesis a partir de los ficheros *hdl* fuente del sistema básico. Los resultados del proceso de síntesis han demostrado que se trata de un sistema sintetizable que utiliza los recursos de *block ram* de la *fpga* para sintetizar la memoria interna del sistema. Para crear el proyecto de síntesis del sistema avanzado con coprocesador se siguen los pasos siguientes:

- Copiar el proyecto de síntesis completo del sistema básico (*wb\_soc\_SB.ise*) almacenado en la ruta *'../ISE/wb\_soc\_SB/'* al directorio del nuevo proyecto de síntesis del sistema avanzado en la ruta *'../ISE/wb\_soc\_SA\_copro/'*.
- Renombrar el fichero de proyecto con *wb\_soc\_SA\_copro.ise*.
- Reemplazar los ficheros fuente modificados durante el proceso de desarrollo del sistema avanzado con coprocesador. Los ficheros que han sido reemplazados en el directorio de trabajo son:

|                             |                           |
|-----------------------------|---------------------------|
| <i>xsv_fpga_top.v</i>       | <i>xsv_fpga_defines.v</i> |
| <i>wb_onchip_4sram8.vhd</i> | <i>pack_sram8.vhd</i>     |

- Añadir los ficheros fuente generados durante el proceso de desarrollo del sistema avanzado. El fichero *tristate.v* contiene la definición *hardware* del *buffer tri-state* utilizado para la adaptación del bus de datos bidireccional. Los ficheros utilizados para la definición y la adaptación del coprocesador son:

|                             |                             |
|-----------------------------|-----------------------------|
| <i>wb_bridge_maioHw.vhd</i> | <i>pack_maioHw.vhd</i>      |
| <i>sram_adapt.vhd</i>       | <i>follow.vhd</i>           |
| <i>sramcontrol_8_8.vhd</i>  | <i>sramcontrol_8_32.vhd</i> |
| <i>corrmax.vhd</i>          | <i>filtsec.vhd</i>          |
| <i>nextpoint.vhd</i>        | <i>section.vhd</i>          |
| <i>tangdir.vhd</i>          | <i>xynext.vhd</i>           |

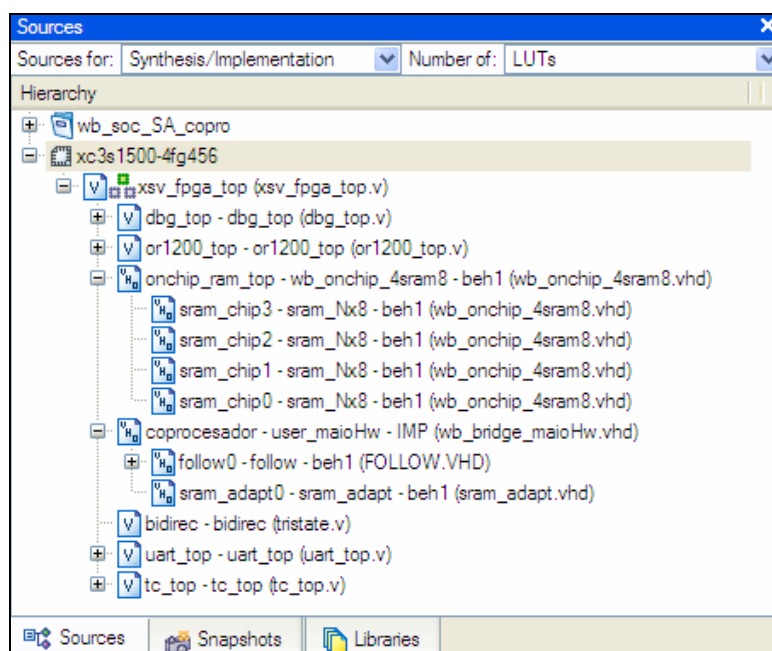


Imagen 75 - Ventana *sources* del proyecto de síntesis en el entorno *ISE*

### 3.5.7.2 Procesado para el dispositivo xc3s1500

Se selecciona el dispositivo destino en el control de propiedades del proyecto, al que se accede mediante el menú emergente con el botón derecho del *mouse* en el objeto del dispositivo de la ventana *sources*. Se trata de un dispositivo *fpga* de la familia *Spartan3* de *Xilinx* con 1.5 millones de puertas lógicas (*xc3s1500*). La herramienta de síntesis y simulación son *XST* y *ModelSim SE*, ambas compatibles con los lenguajes *verilog* y *vhdl* utilizados para definir los diferentes componentes del sistema.

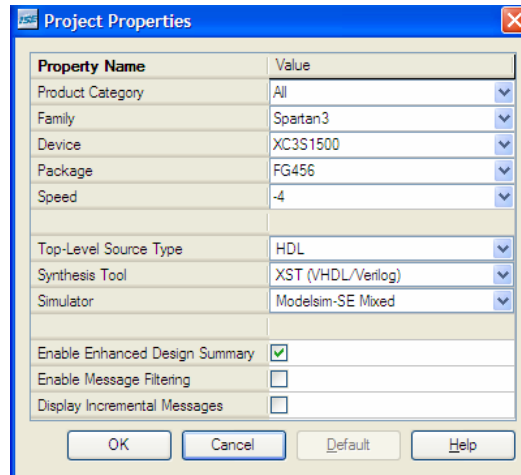


Imagen 76 - Control *Properties* del entorno *ISE*

El proceso de síntesis del sistema se ejecuta mediante la selección del módulo de mayor jerarquía en la ventana de *sources* y el posterior accionamiento del proceso *Synthesize – XST* en la ventana *processes*. Al final del proceso, el programa genera un informe llamado *Synthesis Report*. El índice de contenidos del informe es el mostrado en el cuadro de texto siguiente:

|                                    |
|------------------------------------|
| TABLE OF CONTENTS                  |
| 1) Synthesis Options Summary       |
| 2) HDL Compilation                 |
| 3) Design Hierarchy Analysis       |
| 4) HDL Analysis                    |
| 5) HDL Synthesis                   |
| 5.1) HDL Synthesis Report          |
| 6) Advanced HDL Synthesis          |
| 6.1) Advanced HDL Synthesis Report |
| 7) Low Level Synthesis             |
| 8) Partition Report                |
| 9) Final Report                    |
| 9.1) Device utilization summary    |
| 9.2) TIMING REPORT                 |

En definitiva, el informe se resume con la correcta síntesis del diseño para el dispositivo *xc3s1500* realizada por la herramienta *XST*. Otra información relevante contenida en el informe es el resumen de utilización de recursos del dispositivo destino. Se observa una utilización de recursos importante siendo un punto crítico para la implementación del diseño la ocupación del 75% de las *LUTs* (*Look Up Tables*) disponibles en el dispositivo programable.

| Device Utilization Summary (estimated values) |       |           |             |
|-----------------------------------------------|-------|-----------|-------------|
| Logic Utilization                             | Used  | Available | Utilization |
| Number of Slices                              | 6203  | 13312     | 46%         |
| Number of Slice Flip Flops                    | 3805  | 26624     | 14%         |
| Number of 4 input LUTs                        | 20018 | 26624     | 75%         |
| Number of bonded IOBs                         | 70    | 333       | 21%         |
| Number of BRAMs                               | 8     | 32        | 25%         |
| Number of MULT18X18s                          | 4     | 32        | 12%         |
| Number of GCLKs                               | 2     | 8         | 25%         |

**Imagen 77** - Resumen de ocupación de recursos del informe de síntesis

El proceso de implementación del sistema en el dispositivo destino se ejecuta mediante la selección del módulo de mayor jerarquía en la ventana de *sources* y el posterior accionamiento del proceso *Implement Design* en la ventana *processes*. Este proceso realiza llamadas a varios subprocesos internos como *Translate*, *Map* o *Place&Route*. Tras 11 horas de procesado, la operación auto-termina cuando la herramienta presenta un error en el proceso de rutado del diseño. El informe del error especifica que la herramienta *Par* encuentra problemas de espacio para finalizar el rutado de todas las señales del sistema debido a una elevada complejidad del diseño. Además, propone simplificar el diseño o seleccionar un dispositivo destino de tamaño superior.

### 3.5.7.3 Particiones

Puesto que el diseño presenta una elevada utilización de recursos del dispositivo y un elevado tiempo de procesado en la implementación del diseño, se decide dividir el diseño en diferentes particiones que coinciden con los principales módulos *hardware* del sistema. El entorno *ISE* permite el uso de particiones para preservación del diseño. Crear una partición para una instancia del diseño indica que la implementación de dicha instancia puede ser reutilizada cuando las condiciones lo permitan. Sin embargo, una modificación del código fuente de la instancia o de las opciones de síntesis o implementación forzarán que la instancia sea reimplementada. Las particiones pueden ser instancias en fuentes *hdl*, *rtl*, o *edif*. De modo que, en este caso, las particiones se especifican sobre instancias de módulos en lenguaje *verilog* o sobre arquitecturas de entidades en código *vhdl*. La propiedad principal de las particiones es su nivel de preservación (*Preserve Partition Property*), que especifica qué datos de la implementación serán preservados en caso que la partición no haya sido modificada:

- *Synthesis*: se preserva la *netlist* sintetizada
- *Placement*: se preserva la *netlist* sintetizada y el *placement*
- *Routing*: se preserva la *netlist* sintetizada, el *placement* y el *routing*

Se especifican las siguientes particiones mediante el accionamiento de la opción *New Partition* en el menú emergente con el botón derecho del *mouse* sobre las instancias en la ventana *sources* del proyecto: *dbg\_top*, *orl200\_top*, *onchip\_ram\_top*, *coprocesador*, *uart\_top*, *tc\_top*. A continuación se edita la propiedad de preservación a la opción *routing* en todas las particiones definidas.

Cada partición ha sido sintetizada por separado para comprobar su peso lógico en el diseño. A continuación se presenta el resumen de utilización de recursos resultado de cada proceso independiente. Se debe tener en cuenta que las particiones han sido



sintetizadas de forma independiente al diseño completo, por lo que la herramienta *XST* no ha realizado una optimización global del diseño. Por lo tanto, los resultados de ocupación de recursos de las particiones resultan ligeramente inexactos, puesto que la herramienta es capaz de detectar los algoritmos o bloques lógicos no conectados al diseño y obviar su implementación. Un claro ejemplo se observa en el bloque del árbitro del bus:

- en la síntesis de la partición la herramienta tiene en cuenta para la síntesis todos los posibles nodos de conexión: 9 *masters* y 8 *slaves*
- en la síntesis del diseño completo la herramienta solamente sintetiza los nodos conectados a alguna red, por lo se obvia el procesamiento de 6 *masters* y 5 *slaves*

Los siguientes resultados han sido obtenidos mediante la herramienta *XST* del entorno *ISE* de *Xilinx*. Cada partición ha sido resintetizada utilizando la herramienta *LeonardoSpectrum* pero se han obtenido peores resultados. Por lo tanto se ha descartado el uso de diferentes sintetizadores para conseguir una menor utilización de recursos del dispositivo destino.

a) Memoria Interna (*onchip\_ram\_top*)

| Logic Utilization          | Used |
|----------------------------|------|
| Number of Slices           | 6    |
| Number of Slice Flip Flops | 1    |
| Number of 4 input LUTs     | 11   |
| Number of bonded IOBs      | 97   |
| Number of BRAMs            | 16   |
| Number of GCLKs            | 1    |

b) OpenRisc (*or1200\_top*)

| Logic Utilization          | Used  |
|----------------------------|-------|
| Number of Slices           | 4658  |
| Number of Slice Flip Flops | 2331  |
| Number of 4 input LUTs     | 17172 |
| Number of bonded IOBs      | 369   |
| Number of MULT18X18s       | 4     |
| Number of GCLKs            | 3     |

c) Interface del Depurador (*dbg\_top*)

| Logic Utilization          | Used |
|----------------------------|------|
| Number of Slices           | 247  |
| Number of Slice Flip Flops | 276  |
| Number of 4 input LUTs     | 328  |
| Number of bonded IOBs      | 223  |
| Number of GCLKs            | 3    |

d) Coprocesador (*user\_maioHw*)

| Logic Utilization          | Used |
|----------------------------|------|
| Number of Slices           | 740  |
| Number of Slice Flip Flops | 860  |
| Number of 4 input LUTs     | 1187 |
| Number of bonded IOBs      | 198  |
| Number of GCLKs            | 1    |

e) UART 16650 (*uart\_top*)

| Logic Utilization          | Used |
|----------------------------|------|
| Number of Slices           | 483  |
| Number of Slice Flip Flops | 373  |
| Number of 4 input LUTs     | 931  |
| Number of bonded IOBs      | 86   |
| Number of GCLKs            | 1    |

f) Traffic Cop (*tc\_top*)

| Logic Utilization          | Used |
|----------------------------|------|
| Number of Slices           | 1065 |
| Number of Slice Flip Flops | 16   |
| Number of 4 input LUTs     | 1851 |
| Number of bonded IOBs      | 1804 |
| Number of GCLKs            | 1    |

#### 3.5.7.4 Procesado para el dispositivo xc3s2000

Puesto que el diseño ha agotado los recursos de rutado del dispositivo destino en el caso del *xc3s1500*, se accede al control de propiedades del proyecto para modificar el dispositivo destino. En este caso, se selecciona un dispositivo *fpga* de la familia *Spartan3* de 2 millones de puertas lógicas (*xc3s2000*). Se ejecuta el proceso de síntesis en el que la herramienta *XST* realiza las operaciones descritas a continuación:

```
HDL Compilation:
- Compila todos los archivos vhdl y sus arquitecturas en la
 librería work
- Compila todos los archivos verilog y sus módulos en la
 librería work
Design Hierarchy Analysis:
- Analiza la jerarquía de todos los módulos y entidades del
 diseño
HDL Analysis:
- Analiza todos los módulos y entidades del diseño para
 verificar que son correctos para la síntesis
HDL Synthesis:
- Performing bidirectional port resolution
- Sintetiza todas las unidades del diseño
- HDL Synthesis Report
Advanced HDL Synthesis:
- Analiza y optimiza las FSM (máquina de estado finita) para
 mejor codificación
- Carga el dispositivo para la aplicación Rf_Device
- Report
Low Level Synthesis:
- Optimiza todas las unidades del diseño
- Mapea todas las ecuaciones
- Construye y optimiza la netlist final
- Procesado final de las macros
- Procesa las unidades de jerarquía superior
Informe de particiones
Informe Final:
- Resultados finales
- Estimación de utilización de recursos
Timing Report
```

La herramienta *XST* presenta una estimación de los recursos utilizados y de la temporización al finalizar el proceso de síntesis. Tras un análisis temporal, la herramienta estima la frecuencia máxima de la señal de reloj de entrada al diseño *clk* en 30.72MHz, es decir, un periodo mínimo de 32.55ns. La estimación de utilización de recursos del dispositivo se muestra a continuación:

| Device Utilization Summary (estimated values) |       |           |             |
|-----------------------------------------------|-------|-----------|-------------|
| Logic Utilization                             | Used  | Available | Utilization |
| Number of Slices                              | 6535  | 20480     | 32%         |
| Number of Slice Flip Flops                    | 3788  | 40960     | 10%         |
| Number of 4 input LUTs                        | 20359 | 40960     | 50%         |
| Number of bonded IOBs                         | 70    | 333       | 21%         |
| Number of BRAMs                               | 16    | 32        | 50%         |
| Number of MULT18X18s                          | 4     | 32        | 12%         |
| Number of GCLKs                               | 2     | 8         | 25%         |

**Imagen 78** - Resumen de ocupación de recursos del informe de síntesis

A continuación se ejecuta el proceso *Translate* de la ventana de *processes* del entorno *ISE*. La parte que interesa en este momento es averiguar cuales son los 8 recursos de *block ram* utilizados del dispositivo. Para ello se ejecuta el sub-proceso *Generate Post-Translate Simulation Model* que realiza una llamada a la herramienta *netgen* que, en este caso, genera un fichero en código *hdl* llamado *xsv\_fpga\_top\_translate.v* en el directorio '*netgen/translate*'. Este fichero contiene un modelo de simulación del diseño creado a partir de instancias a recursos del dispositivo destino. Un ejemplo de instancia de un elemento *block ram* contenida en el modelo de simulación post-translate es:

```
X_RAMB16_S4 \onchip_ram_top/sram_chip3/inst_Mram_mem11
(
 .CLK(wb_clk),
 .EN(\onchip_ram_top/N2),
 .SSR(wb_ss_err_o),
 .WE(\onchip_ram_top/sram_chip3/_and0000),
 .ADDR({wb_ss_adr_i[13], wb_ss_adr_i[12], wb_ss_adr_i[11], wb_ss_adr_i[10],
 wb_ss_adr_i[9], wb_ss_adr_i[8], wb_ss_adr_i[7], wb_ss_adr_i[6],
 wb_ss_adr_i[5], wb_ss_adr_i[4], wb_ss_adr_i[3], wb_ss_adr_i[2]}),
 .DI({wb_ss_dat_i[31], wb_ss_dat_i[30], wb_ss_dat_i[29], wb_ss_dat_i[28]}),
 .DO({wb_ss_dat_o[31], wb_ss_dat_o[30], wb_ss_dat_o[29], wb_ss_dat_o[28]})
);
```

A partir de las 16 instancias de este tipo contenidas en el fichero *hdl*, se genera un fichero de formato *bmm* (*Block Ram Memory Map*) que define la organización de los recursos *block ram* del diseño, es decir, que parte del mapa de direcciones contiene cada bloque de memoria. En este caso, cada bloque de memoria embebida en el dispositivo contiene todo el mapa de direcciones de memoria interna para los dos bits definidos en los buses de datos de entrada o salida de la instancia.

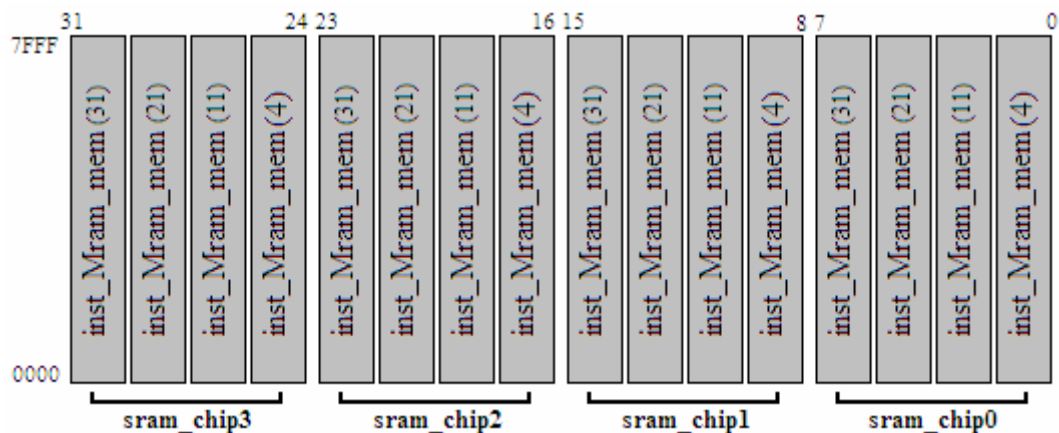


Imagen 79 - Esquema del mapa de direcciones de la memoria interna

El entorno de desarrollo ha determinado un mapeado del mapa de direcciones de la memoria interna como el mostrado en la imagen anterior. Para inicializar los recursos de *block ram* con unos datos coherentes, el fichero *bmm* debe describir un mapa de direcciones equivalente al anterior. El fichero *bmm* que se corresponde con el mapa descrito en el esquema es:

```

ADDRESS_SPACE Mram_mem RAMB16 [0x00000000:0x00007FFF]
 BUS_BLOCK
 onchip_ram_top/sram_chip3/inst_Mram_mem31 [31:30];
 onchip_ram_top/sram_chip3/inst_Mram_mem21 [29:28];
 onchip_ram_top/sram_chip3/inst_Mram_mem11 [27:26];
 onchip_ram_top/sram_chip3/inst_Mram_mem4 [25:24];
 onchip_ram_top/sram_chip2/inst_Mram_mem31 [23:22];
 onchip_ram_top/sram_chip2/inst_Mram_mem21 [21:20];
 onchip_ram_top/sram_chip2/inst_Mram_mem11 [19:18];
 onchip_ram_top/sram_chip2/inst_Mram_mem4 [17:16];
 onchip_ram_top/sram_chip1/inst_Mram_mem31 [15:14];
 onchip_ram_top/sram_chip1/inst_Mram_mem21 [13:12];
 onchip_ram_top/sram_chip1/inst_Mram_mem11 [11:10];
 onchip_ram_top/sram_chip1/inst_Mram_mem4 [9:8];
 onchip_ram_top/sram_chip0/inst_Mram_mem31 [7:6];
 onchip_ram_top/sram_chip0/inst_Mram_mem21 [5:4];
 onchip_ram_top/sram_chip0/inst_Mram_mem11 [3:2];
 onchip_ram_top/sram_chip0/inst_Mram_mem4 [1:0];
 END_BUS_BLOCK;
END_ADDRESS_SPACE;

```

Este fichero de formato *bmm* se añade al proyecto mediante la opción *Add Sources* de la ventana *sources*. Entonces, se ejecuta el proceso *Implement Design* de la ventana *processes* del entorno *ISE*. Este proceso se compone de tres subprocesos principales: *Translate*, *Map* y *Place&Routing*.

En el proceso *Translate* la herramienta *NGDBuild* realiza las siguientes operaciones:

```

Command Line: C:\Xilinx\bin\nt\ngdbuild.exe -ise
.../wb_soc_SA_copro_parts.ise -intstyle ise -dd _ngo -nt timestamp -uc
xsv_fpga_top.ucf -bm wb_onchip_4sram8.bmm -p xc3s2000-fg676-4
xsv_fpga_top.ngc xsv_fpga_top.ngd

```

- Lectura del fichero NGO
- Descarga de los ficheros NGC del módulo principal y de las particiones (módulos de diseño)
- Aplicación de constraints en el diseño (UCF)
- Comprobación de las especificaciones temporales
- Comprobación de las particiones
- Procesado del fichero BMM
- Comprobación del diseño expandido
- Escritura del fichero NGD
- Escritura del fichero BLD (informe)

En el proceso *Map* se realiza la llamada a la herramienta *map* con la línea de comando:

```

C:\Xilinx\bin\nt\map.exe -ise .../wb_soc_SA_copro_parts.ise -intstyle ise -
p xc3s2000-fg676-4 -cm area -pr b -k 4 -c 100 -o xsv_fpga_top_map.ncd
xsv_fpga_top.ngd xsv_fpga_top.pcf

```

En el proceso *Place&Route* se realiza la llamada a la herramienta *par* con la línea de comando:

```

C:\Xilinx\bin\nt\par -w -intstyle ise -ol high -t 1 xsv_fpga_top_map.ncd
xsv_fpga_top.ncd xsv_fpga_top.pcf

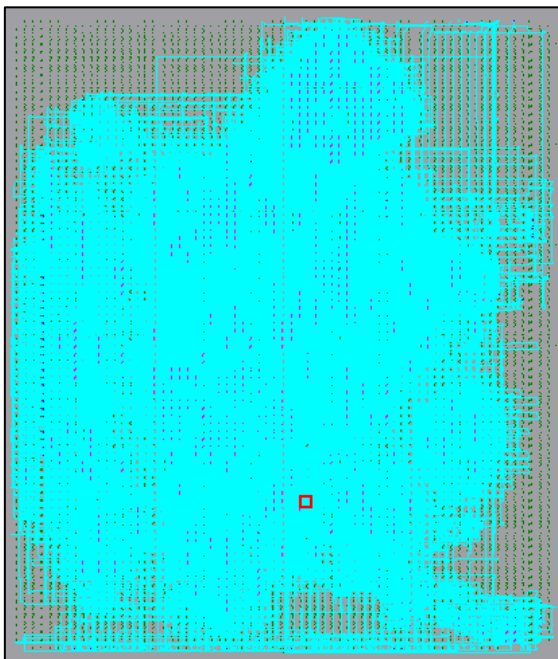
```

La herramienta *par* comienza con un Análisis Temporal inicial y, a continuación, realiza el emplazamiento y enrutamiento del diseño en el dispositivo destino. Al finalizar el proceso, se procede a la escritura del fichero *xsv\_fpga\_top.ncd*. El resumen final de utilización de recursos del dispositivo contenido en el informe es:

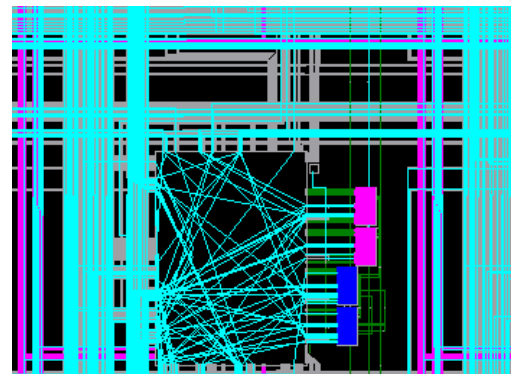
| Device Utilization Summary                     |           |           |             |
|------------------------------------------------|-----------|-----------|-------------|
| Logic Utilization                              | Used      | Available | Utilization |
| Number of Slice Flip Flops                     | 3,777     | 40,960    | 9%          |
| Number of 4 input LUTs                         | 19,841    | 40,960    | 48%         |
| Logic Distribution                             |           |           |             |
| Number of occupied Slices                      | 11,107    | 20,480    | 54%         |
| Number of Slices containing only related logic | 11,107    | 11,107    | 100%        |
| Number of Slices containing unrelated logic    | 0         | 11,107    | 0%          |
| Total Number of 4 input LUTs                   | 20,015    | 40,960    | 48%         |
| Number used as logic                           | 19,841    |           |             |
| Number used as a route-thru                    | 174       |           |             |
| Number used as 16x1 RAMs                       | 8,240     |           |             |
| Number used as Shift registers                 | 16        |           |             |
| Number of bonded IOBs                          | 70        | 489       | 14%         |
| IOB Flip Flops                                 | 21        |           |             |
| Number of RAMB16s                              | 16        | 40        | 40%         |
| Number of MULT18X18s                           | 4         | 40        | 10%         |
| Number of BUFGMUXs                             | 3         | 8         | 37%         |
| Total equivalent gate count for design         | 1,192,516 |           |             |
| Additional JTAG gate count for IOBs            | 3,360     |           |             |

**Imagen 80** - Resumen de utilización de recursos del informe final del entorno *ISE*

La opción *View/Edit Routed Design* de la ventana *processes* del entorno *ISE* abre el programa *FPGA Editor*. Esta aplicación permite al usuario visualizar los resultados del proceso de enrutamiento del diseño.

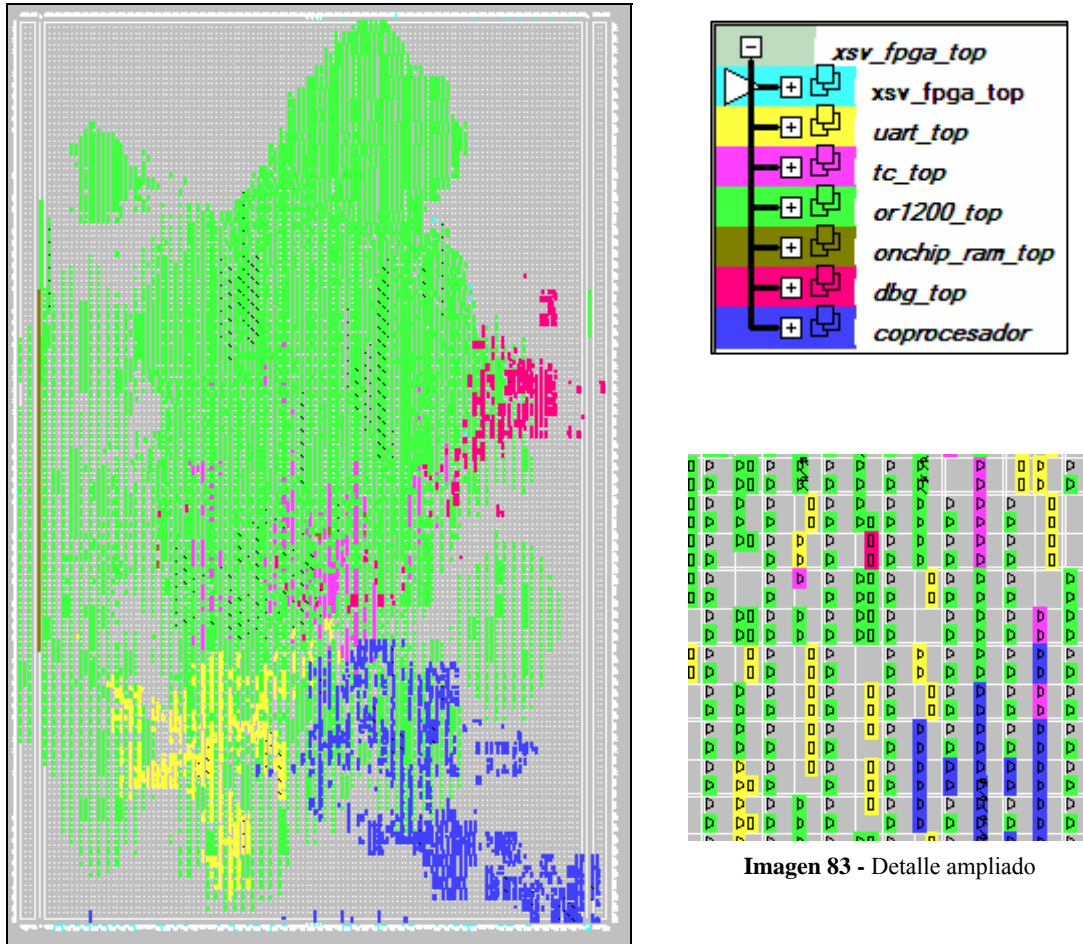


**Imagen 82** - Representación gráfica del diseño enrutado



**Imagen 81** - Detalle ampliado

La opción *View/Edit Placed Design* de la ventana *processes* del entorno *ISE* permite al usuario contemplar gráficamente los resultados del proceso de emplazamiento del diseño. Se abre el programa *Floorplanner* que, a partir del fichero *ncd* generado tras la implementación, permite ver (o editar manualmente) el emplazado de los componentes del diseño en el dispositivo destino.



**Imagen 84** - Detalle del diseño emplazado

### 3.5.8 Simulación Post-Layout

En el presente apartado se pretende verificar el diseño implementado mediante la simulación temporal con el programa *ModelSim*. La *timing simulation* verifica que el diseño opera dentro de las *constraints* especificadas tras el enrutamiento teniendo en cuenta los retardos lógicos. Por lo tanto, el propósito de este método es comprobar el comportamiento dinámico temporal del diseño *hdl* simulado para el dispositivo destino.

#### 3.5.8.1 Modelo de simulación

Tras el proceso de implementación del diseño en el entorno *ISE* se ha obtenido el fichero *ncd* final del proyecto. La herramienta *netgen* permite la generación de un modelo de simulación *post-layout* a partir de la *netlist* obtenida.

En el caso de la simulación funcional, el proyecto de *ModelSim* está formado por diferentes ficheros fuente *hdl* que definen los diferentes módulos y entidades del sistema. Por ello, la memoria interna del sistema puede ser inicializada a partir de funciones *software* no sintetizables que acceden a estructuras de datos con el formato

adecuado. En cambio en el caso de la simulación temporal, el proyecto está formado por un único fichero *hdl* que modela el sistema completo a partir de primitivas de *Xilinx*. Por lo tanto, la memoria interna debe ser inicializada antes de generar el modelo de simulación.

El programa *FPGA Editor* del entorno *ISE* permite actualizar los datos de la memoria *block ram* definida en el fichero *bmm* del sistema especificado por el fichero *ncd* a partir de la información proporcionada por el fichero de datos de formato *elf*. Este fichero es el obtenido a partir de la compilación y el enlazado de un programa mediante la cadena de herramientas *GNU*.  
(copro.or32 = copro.elf)

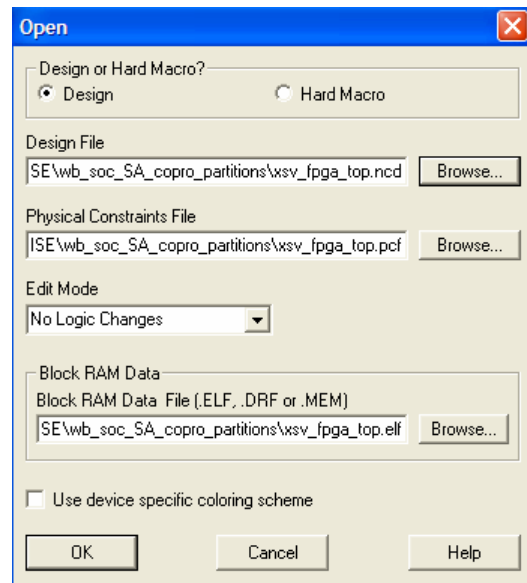


Imagen 85 - Control utilizado para la inicialización

Tras inicializar el contenido de la memoria interna del sistema, se ejecuta el proceso *Generate Post-Place&Route Simulation Model* de la ventana *processes* del entorno *ISE*. Este proceso realiza una llamada a la herramienta *netgen* con la línea de comandos siguiente:

```
netgen -intstyle ise -s 4 -bd xsv_fpga_top.elf -pcf
xsv_fpga_top.pcf -sdf_anno true -sdf_path netgen/par -insert_glbl true -w -
dir netgen/par -ofmt verilog -sim xsv_fpga_top.ncd xsv_fpga_top_timesim.v
```

y genera los siguientes ficheros en el directorio '*.../netgen/par*':

- *\_timesim.v* fichero en código *verilog* que contiene el modelo del diseño completo a partir de primitivas específicas de simulación *simprim* de *Xilinx*
- *\_timesim.sdf* *standard delay format* - fichero asociado que contiene todos los retardos del sistema
- *\_timesim.nlf* contiene el informe del proceso

Puesto que el modelo forma parte de un sistema formado por el diseño embedido en el dispositivo *fpga* y una memoria externa, las rutas de las instancias contenidas en el fichero *sdf* deben ser adaptadas para la simulación temporal. El texto '*chip\_fpga*' debe ser añadido a todas las instancias.

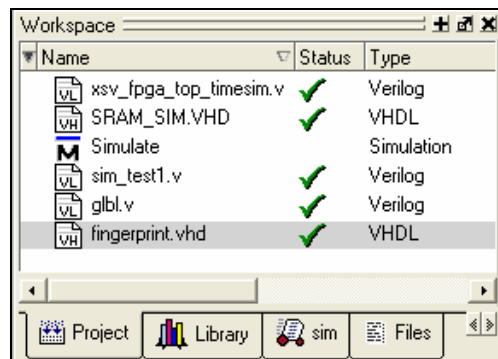
### 3.5.8.2 Creación del proyecto de simulación

La ubicación del nuevo proyecto de simulación para el programa *ModelSim* es la ruta '*/modelsim/sim\_postlayout/copro*'. Los ficheros fuente añadidas al proyecto son:

- el modelo de simulación del diseño (*xsv\_fpga\_top\_timesim.v*)
- el modelo de la memoria ram externa (*sram\_sim.vhd*)



- el paquete de inicialización de la memoria externa (*fingerprint.vhd*)
- el fichero de *testbench* (*sim\_test1.v*)
- el módulo *glbl.v*, requerido para la simulación temporal con modelo en lenguaje *verilog*



**Imagen 86** - Ventana *workspace* del proyecto de simulación

Para obtener una correcta compilación y simulación del proyecto se debe utilizar la librería de simulación *simprim*. Por lo tanto, se crea una nueva librería llamada *simprim* mediante la opción *New* → *Library* del menú emergente de botón derecho en el control *Library* de la ventana *workspace* de *ModelSim*. Mediante la opción *Edit* se asocia la nueva librería a la ruta '*c:/Modeltech\_6.0d/xilinx/verilog/simprims\_ver*'. Cuando finaliza el proceso de compilación, la librería puede ser utilizada en el proyecto de simulación.

El fichero *verilog* del modelo de simulación debe encontrarse en el directorio raíz del proyecto de *ModelSim*. Sin embargo, para que el proyecto simule correctamente, el fichero de retardos de formato *sdf* debe encontrarse en la ruta '*./netgen/par*'.

### 3.5.8.3 Simulación Temporal

En este caso, el sistema no puede ser simulado mediante el *script* (opción *Simulate* → *Start* del menú principal del programa) utilizado para simulaciones funcionales. Se debe introducir un comando de simulación específico en la ventana *Transcript* del programa. La línea de comandos es:

```
vsim -L C:/Modeltech_6.0d/xilinx/verilog/simprims_ver -sdftyp
/=C:/User/Doctor/wishbone/orp_soc_tutorial/modelsim/sim_postlayout/
copro/netgen/par/xsv_fpga_top_timesim.sdf work.sim_test1 work.glbl
```

El comando *vsim* se utiliza para invocar el simulador *VSIM*. Es posible simular una configuración *vhdl* o una pareja entidad/arquitectura; una configuración o módulo en *verilog*; un módulo *systemC*; o un diseño optimizado. Durante la elaboración *vsim* determina si la fuente ha sido modificada desde la última compilación. Los argumentos utilizados en este caso son:

`-L <library_name> ...`

Especifica la librería en la que buscar unidades de diseño instanciadas en *verilog*. Si se especifican múltiples librerías, cada una debe ser precedida por la opción `-L`. Las librerías son accedidas en el orden de aparición en la línea de comandos.

`-sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=<instance>] <sdf_filename>`



Anota celdas *verilog* en el fichero *sdf* especificado con mínima, típica o máxima temporización. El argumento opcional *delayScale* escala todos los valores de tiempo según el valor especificado.

```
<library_name>.<design_unit>
```

Especifica una librería y una unidad de diseño asociada. Múltiples librería/unidad de diseño pueden ser especificadas. En este caso, se especifican dos módulos de mayor jerarquía en el diseño: *sim\_test1* y *glbl*.

Cuando el comando anterior es ejecutado, se obtienen los siguientes resultados de elaboración del diseño en la ventana *Transcript*:

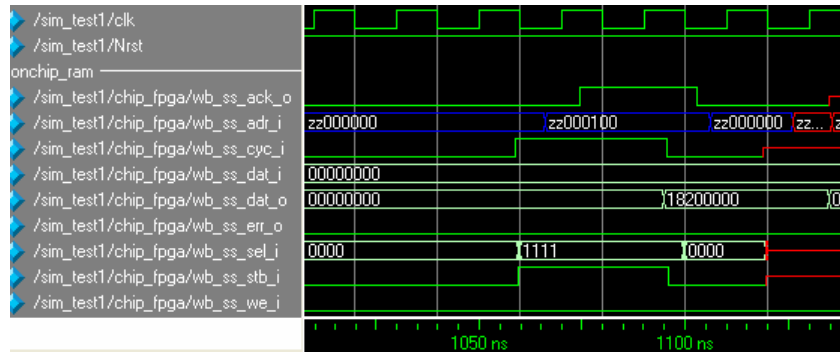
```
Loading work.sim_test1
Loading work.xsv_fpga_top
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_BUF
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_IPAD
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_OPAD
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_OBUF
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_BPAD
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_OBUFT
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_BUFMUX
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_INV
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_FF
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_ZERO
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_ONE
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_LUT4
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_SFF
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_MUX2
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_XOR2
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_AND2
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_RAMD16
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_MULT18X18
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_RAMB16_S4
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.X_SRLC16E
Loading work.glbl
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.ffa_sce
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.sffa_sce
Loading C:/Modeltech_6.0d/xilinx/verilog/simprims_ver.mux

Loading C:\Modeltech_6.0d\win32\../std.standard
Loading C:\Modeltech_6.0d\win32\../ieee.std_logic_1164(body)
Loading C:\Modeltech_6.0d\win32\../ieee.std_logic_arith(body)
Loading C:\Modeltech_6.0d\win32\../ieee.std_logic_signed(body)
Loading work.pack_fingerprint(body)
Loading work.pack_sram_sim(body)
Loading work.sram_sim3(beh3)
Loading work.sram_sim2(beh2)
Loading work.sram_sim1(beh1)
Loading work.sram_sim0(beh0)
Loading
C:/User/Doctor/wishbone/orp_soc_tutorial/modelsim/sim_postlayout/copro/netg
en/par/xsv_fpga_top_timesim.sdf
Loading netgen/par/xsv_fpga_top_timesim.sdf
** Note: (vsim-3587) SDF Backannotation Successfully Completed.
Time: 0 ps Iteration: 0 Region: /sim_test1 File:
../.../modelsim/sim_postlayout/copro/sim_test1.v
```

Para visualizar la simulación, se accede al submenú *View* → *Debug Windows* del menú principal del programa y se visualizan las ventanas de depurado *objects* y *wave*. La ventana de depurado *wave* permite visualizar los resultados de la simulación en forma de ondas y sus valores. El panel *objects* muestra el nombre y el valor actual de los objetos de datos declarados en la región seleccionada en la ventana *workspace*. Esto incluye las señales, redes, registros, constantes, variables no declaradas en un proceso,

genéricos y parámetros. Tras una búsqueda minuciosa en las regiones del diseño mostradas en la ventana *workspace*, todos los objetos de interés son añadidos a la ventana *wave*. La configuración de la ventana de depurado *wave* se guarda en un fichero llamado por defecto *wave.do*. Para poder acceder de nuevo a la configuración almacenada se selecciona la opción *File* → *Load* del menú principal.

Se ejecuta la simulación mediante las opciones de depurado en el menú *Simulate* del programa principal y se obtiene:



**Imagen 87** - Detalle de simulación temporal. Acceso a memoria interna.

La *Imagen 87* muestra un acceso a la dirección de memoria 0x100, relativa a la primera instrucción de la rutina de servicio a la excepción de reset del sistema. Se observa que la señal *wb\_ss\_ack\_o* se activa antes de que los datos hayan sido introducidos en el bus de datos *wb\_ss\_dat\_o* por lo que el microcontrolador obtiene un valor incorrecto en su interface de instrucciones. Se trata de un problema típico en la simulación temporal y se debe a una frecuencia de reloj demasiado alta para el sistema. Se comprueba que la frecuencia de reloj en el fichero *testbench* del proyecto es de 50MHz:

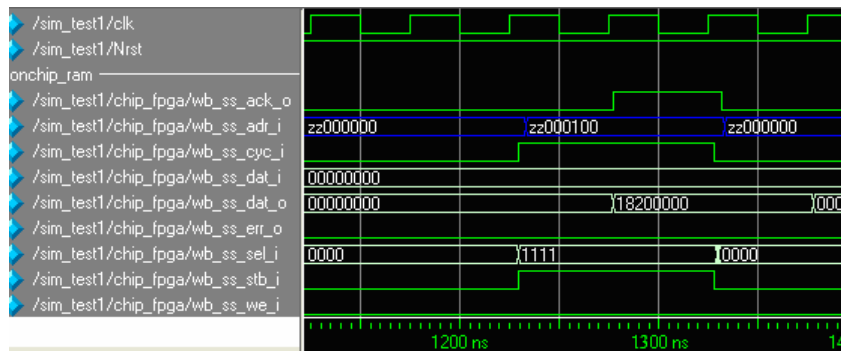
```
#10 clk=!clk; // f = 1/(2·10ns) = 50MHz
```

El informe del proceso de *Place&Route* del entorno de diseño *ISE* indica el valor mínimo posible de la señal de reloj del sistema y, por lo tanto, la frecuencia máxima a la que el sistema es capaz de trabajar. Por otro lado, el informe del proceso de *Synthesis* presenta una estimación de la frecuencia máxima de la señal de reloj. De esta manera se comprueba que la frecuencia máxima del sistema implementado es de 32MHz aproximadamente.

Se elige un frecuencia de 20MHz para simular el sistema. Se trata de un valor que se encuentra dentro de los márgenes de trabajo del diseño y se encuentra bastante cercano al máximo impuesto por la arquitectura. Por lo tanto, el fichero de *testbench* (*sim\_test1.v*) debe ser modificado con la siguiente línea de código *verilog*:

```
#25 clk=!clk; // f = 1/(2·25ns) = 20MHz
```

Tras recompilar la fuente modificada (fichero *testbench*) se vuelve a ejecutar la línea de comando de simulación temporal. Los resultados obtenidos son:



**Imagen 88** - Detalle de simulación temporal. Acceso a memoria interna.

Para el mismo acceso mostrado en la *Imagen 87*, se comprueba en la *Imagen 88* que el ciclo de bus se ejecuta de forma correcta. La activación de la señal *wb\_ss\_ack\_o* se produce de manera posterior a la aparición de los datos en el bus *wb\_ss\_dat\_o* provenientes del controlador de memoria interna del sistema. Por lo tanto, la interfaz *wishbone* de instrucciones del microprocesador obtiene los datos de forma correcta.

En la simulación temporal, se observa que tras el reset hardware inicial del sistema la interfaz *wishbone* de instrucciones del *openrisc* accede a la dirección 0x100 y ejecuta la rutina de servicio a la excepción de reset. Tras la rutina, el *program counter* (PC) salta a la primera instrucción del programa principal.

Se pueden utilizar diferentes comandos para comprobar en qué direcciones de memoria han sido ensambladas cada una de las funciones o instrucciones del programa. Las líneas de comandos recomendadas son:

- `or32-elf-readelf -a copro.or32 > program.txt`: genera un fichero de texto que contiene información sobre el contenido de un fichero objeto de formato *elf*. El argumento *-a* provoca que el fichero contenga la información siguiente: cabecera del fichero *elf*, cabecera de los programas, cabeceras de las secciones, tabla de símbolos, reubicaciones, sección dinámica, información de versión, información específica de la arquitectura e histograma. Este fichero permite conocer la ubicación de cada objeto en el mapa de memoria y su tamaño.
- `or32-elf-objdump -d -S copro.or32 > disassemble.txt`: genera un fichero de texto que contiene el desensamblado del programa contenido en el fichero objeto de formato *elf*. Los argumentos *-d* y *-S* provocan que el fichero contenga el desensamblado de las secciones ejecutables del fichero objeto y que se combine el código máquina con el código fuente del programa respectivamente. Este fichero permite conocer la ubicación exacta de cada instrucción ensamblada en el mapa de memoria y su codificación hexadecimal.

### 3.5.9 Configuración de la FPGA

En el presente apartado se realiza la programación del sistema diseñado sobre el dispositivo destino y su posterior verificación mediante la ejecución del *software* específico del algoritmo de extracción de minutia de *Maio-Maltoni*. Los objetivos principales son:

- estudiar el *Xilinx Spartan3 Development Kit* utilizado como placa de desarrollo del dispositivo destino.
- realizar los ajustes del *hardware* del sistema para adaptar el diseño al dispositivo destino



| FPGA     | Logic Gates | Logic Cells | Array CLB<br>(4 celdas lógicas) |     |            | BRAM<br>(2KB) | Mult | DCM |
|----------|-------------|-------------|---------------------------------|-----|------------|---------------|------|-----|
|          |             |             | Fil                             | Col | Total CLBs |               |      |     |
| XC3S1500 | 1.5M        | 29,952      | 64                              | 52  | 3,328      | 32            | 32   | 4   |
| XC3S2000 | 2M          | 46,080      | 80                              | 64  | 5,120      | 40            | 40   | 4   |

### 3.5.9.1.2 Configuración

La placa de desarrollo soporta tanto *Boundary-Scan* como *Master/Slave* serie o paralelo usando las memorias *PROM* de la placa. Todos los pines de configuración han sido portados al conector *J1* para el uso de métodos alternativos de programación. Programar el dispositivo *fpga* mediante *Boundary-Scan* requiere el uso de un cable *JTAG*. La placa contiene los conectores *J1* y *J5* para la conexión de los cables *Parallel III* y *Parallel IV* respectivamente.

Para configurar el dispositivo en modo *Boundary-Scan* se debe especificar mediante el uso de los *Mode Select jumpers* del *JP2*. Las posiciones de los *jumpers* están etiquetadas como *M0*, *M1* y *M2* y se encuentran a nivel bajo por defecto. Se deben poner *jumpers* en las posiciones *M0* y *M2* (nivel alto) para deshabilitar la memoria *PROM* y seleccionar el modo de configuración *Boundary-Scan*.

| JP2<br>Config Mode | 1-2 | 3-4 | 5-6 |
|--------------------|-----|-----|-----|
|                    | M0  | M1  | M2  |
| Master Serial      | 0   | 0   | 0   |
| Slave Serial       | 1   | 1   | 1   |
| Master SelectMAP   | 1   | 1   | 0   |
| Slave SelectMAP    | 0   | 1   | 1   |
| Boundary Scan      | 1   | 0   | 1   |

Imagen 90 - Selección de modo de configuración

El conector *J5* está destinado a la conexión de un cable plano de 14 pines suministrado con el cable *Xilinx Parallel IV*. Se debe conectar un *jumper* en las posiciones 2-3 del *JP6* para configurar la cadena de *JTAG* en modo *standalone*.

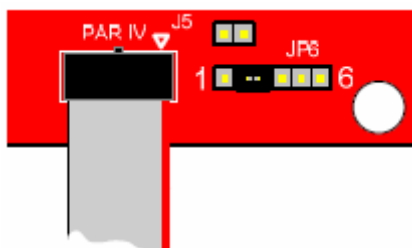


Imagen 91 – Conexión del cable Xilinx Parallel IV

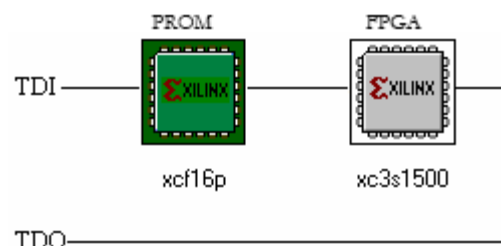


Imagen 92 - Cadena JTAG en modo standalone

### 3.5.9.1.3 Señal de reloj

La placa de desarrollo tiene diferentes fuentes de reloj disponibles. En la tabla siguiente se muestran las entradas de reloj recomendadas para la señal de reloj del sistema:

| Frecuencia | #pin FPGA | Observaciones |
|------------|-----------|---------------|
| 66MHz      | AD13      |               |
| 100MHz     | A13       |               |
| 40MHz typ. | AE14      | Socket        |
| -          | AE13      | SMA           |
| -          | AF14      | Header        |

|             |          |                          |
|-------------|----------|--------------------------|
| 125MHz typ. | C14, B14 | No instalado por defecto |
|-------------|----------|--------------------------|

#### 3.5.9.1.4 Memoria SRAM

La memoria *SRAM* es uno de los tipos de memoria disponibles en la placa de desarrollo (además de memoria *Flash* y *DDR SDRAM*). Se trata de un dispositivo de *SRAM* asíncrona conectado al bus compartido de datos con anchura de 32 bits. Proporciona 2MB de memoria *SRAM* en un *IC* único organizada como 512KB x 32.

#### 3.5.9.1.5 Comunicación R232

Las señales de transmisión/recepción están conectadas a un banco de entrada/salida de la *fpga* (banco 6). Los puertos de entrada/salida utilizados para las líneas *RX* y *TX* son los pines *P4* y *P5* respectivamente.

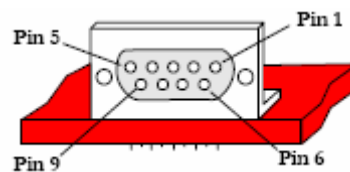


Imagen 93 - Conector serie DB9

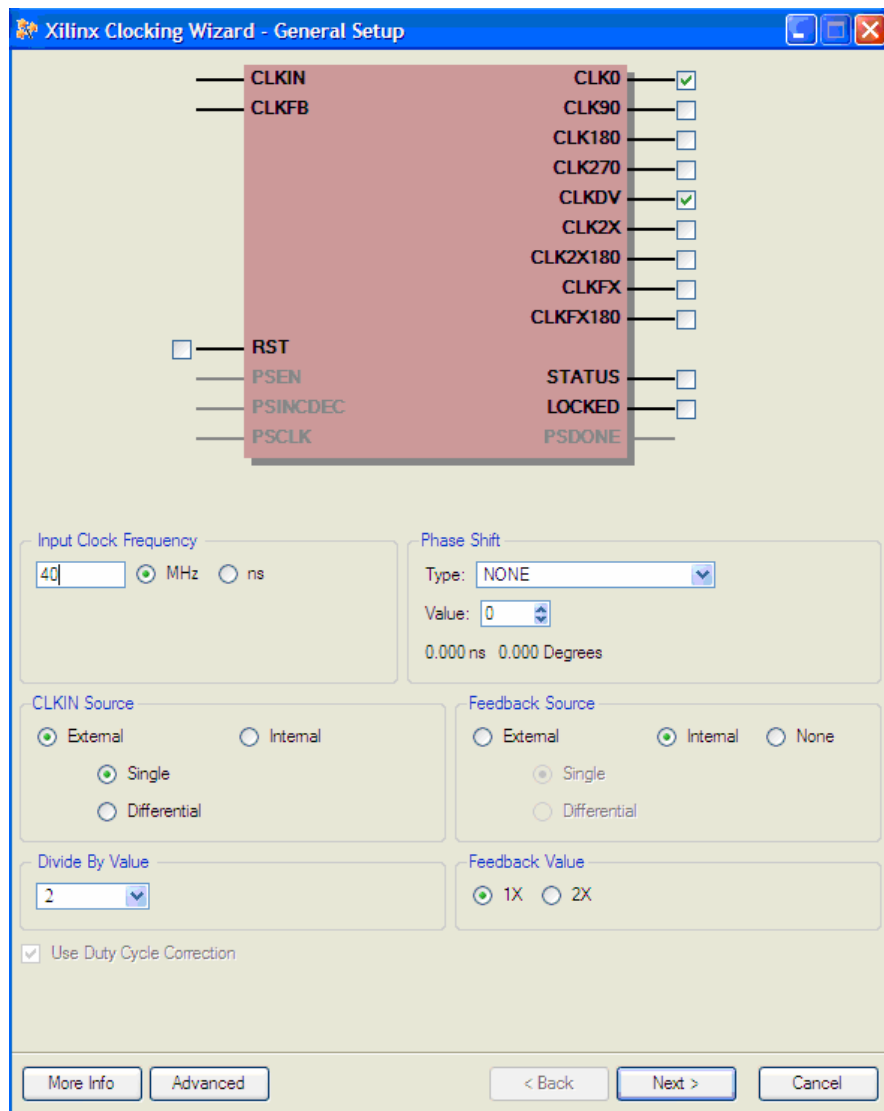
#### 3.5.9.1.6 Header JP1

El conector J17 de 50 pines de la placa de desarrollo está conectado a 47 pines de entrada/salida de la *fpga*. Los pines del *header JP1* están compartidos con muchos otros periféricos mediante interruptores de bus. Estos interruptores pueden ser deshabilitados eliminando la conexión *JP20*. De esta manera, el conector se aísla de los periféricos permaneciendo conectado a los pines I/O del dispositivo *fpga*.

### 3.5.9.2 Digital Clock Manager (DCM)

El primer problema encontrado para implementar el diseño sobre el dispositivo final es la adaptación de las señales de reloj de la placa para cumplir las restricciones del diseño. El informe generado por el entorno *ISE* tras la implementación del diseño en el dispositivo seleccionado detalla un estudio de la temporización del sistema y, entre otros parámetros, especifica la frecuencia máxima del reloj del sistema en 30.72MHz. En el apartado 3.5.6.1.3 se muestra una tabla con las señales de reloj integradas en la placa de desarrollo y se comprueba que el oscilador que más se aproxima a esta frecuencia genera una señal de reloj de 40MHz. Por lo tanto, se debe implementar un módulo *DCM* en el sistema para adaptar la frecuencia de la señal generada.

El entorno *ISE* proporciona la herramienta *Architecture Wizard* que utiliza una serie de librerías de *Xilinx* para implementar una serie de arquitecturas customizables con primitivas específicas del dispositivo de *Xilinx* seleccionado. Al ejecutar la aplicación se visualiza un control en el que se selecciona el nombre del componente especificado por el usuario, el lenguaje utilizado y el dispositivo destino sobre el que se implementará el sistema. En el siguiente control aparece una lista de las arquitecturas disponibles y se selecciona *Clocking Wizard* → *Single DCM*. En la Imagen 94 se muestra el control utilizado para la configuración general de la arquitectura, donde se pueden customizar los puertos de entrada/salida y las características funcionales.



**Imagen 94** - Control de configuración general de la herramienta *Xilinx Clocking Wizard*

El puerto de salida *Locked* debe ser deshabilitado pero, para asegurar una señal de reloj estabilizada en el *buffer* de salida, la opción avanzada *Wait for DCM lock before DONE signal goes high* debe ser activada. En el siguiente control, se debe especificar el uso de *buffers* globales para todas los puertos de salida de reloj. Un resumen del *core* generado por la herramienta es:

```
library ieee;use ieee.std_logic_1164.ALL;use ieee.numeric_std.ALL;
library UNISIM;use UNISIM.Vcomponents.ALL;

entity wb_dcm is
 port (CLKIN_IN : in std_logic;
 CLKDV_OUT : out std_logic;
 CLKIN_IBUFG_OUT : out std_logic;
 CLK0_OUT : out std_logic);
end wb_dcm;

architecture BEHAVIORAL of wb_dcm is
 //declara señales CLKDV_BUF, CLKFB_IN, CLKIN_IBUFG, CLK0_BUF, GND1
 //declara componentes BUFG, IBUFG, DCM
 begin
 GND1 <= '0';
 CLKIN_IBUFG_OUT <= CLKIN_IBUFG;
```

```

CLK0_OUT <= CLKFB_IN;
CLKDV_BUF_INST : BUFG
 port map (I=>CLKDV_BUF,
 O=>CLKDV_OUT);

CLKIN_IBUF_INST : IBUF
 port map (I=>CLKIN_IN,
 O=>CLKIN_IBUF);

CLK0_BUF_INST : BUFG
 port map (I=>CLK0_BUF,
 O=>CLKFB_IN);

DCM_INST : DCM
 generic map(...)
 port map (CLKFB=>CLKFB_IN,
 CLKIN=>CLKIN_IBUF,
 DSSEN=>GND1,
 PSCLK=>GND1,
 PSEN=>GND1,
 PSINCDEC=>GND1,
 RST=>GND1,
 CLKDV=>CLKDV_BUF,
 CLKFX=>open,
 CLKFX180=>open,
 CLK0=>CLK0_BUF,
 CLK2X=>open,
 CLK2X180=>open,
 CLK90=>open,
 CLK180=>open,
 CLK270=>open,
 LOCKED=>open,
 PSDONE=>open,
 STATUS=>open);
end BEHAVIORAL;

```

En el código se define la entidad `wb_dcm` customizada durante la configuración de la arquitectura con la herramienta *Architecture Wizard* que instancia componentes genéricos de la librería *unisim* de *Xilinx* para obtener el comportamiento esperado.

Para que el componente creado sea correctamente implementado sobre el dispositivo destino, las siguientes líneas de parámetros deben ser añadidas al fichero *UCF* (*user constraints file*) del proyecto:

```

INST DCM_INST CLK_FEEDBACK = 1X;
INST DCM_INST CLKDV_DIVIDE = 2;
INST DCM_INST CLKFX_DIVIDE = 1;
INST DCM_INST CLKFX_MULTIPLY = 4;
INST DCM_INST CLKIN_DIVIDE_BY_2 = FALSE;
INST DCM_INST CLKIN_PERIOD = 25.0;
INST DCM_INST CLKOUT_PHASE_SHIFT = NONE;
INST DCM_INST DESKEW_ADJUST = SYSTEM_SYNCHRONOUS;
INST DCM_INST DFS_FREQUENCY_MODE = LOW;
INST DCM_INST DLL_FREQUENCY_MODE = LOW;
INST DCM_INST DUTY_CYCLE_CORRECTION = TRUE;
INST DCM_INST FACTORY_JF = C080;
INST DCM_INST PHASE_SHIFT = 0;
INST DCM_INST STARTUP_WAIT = TRUE;

```

Para conectar el módulo *DCM* a la señal de reloj del sistema, se debe instanciar y conectar la entidad en el fichero de mayor jerarquía del proyecto (*xsv\_fpga\_top.v*). Se sustituye la línea de código que conectaba la señal de reloj externa directamente a la señal de reloj interna del sistema (`assign wb_clk = clk;`) por el bloque de código:



```
// Instantation del DCM
wb_dcm wb_dcm (
 .CLKIN_IN (clk), //40MHz
 .CLKDV_OUT (wb_clk), //20MHz
 .CLKIN_IBUFG_OUT (),
 .CLK0_OUT()
);
```

Para comprobar el efecto de los cambios efectuados, se ejecuta de nuevo el proceso de síntesis e implementación del proyecto en el entorno *ISE* obteniendo la información siguiente en el informe de temporización:

```
Timing constraint: NET "wb_dcm/CLKIN_IBUFG_OUT" PERIOD = 25 ns HIGH 50%;

Timing constraint: PERIOD analysis for net "wb_dcm/CLKDV_BUF" derived from
NET "wb_dcm/CLKIN_IBUFG_OUT" PERIOD = 25 ns HIGH 50%;

multiplied by 2.00 and duty cycle corrected to 50 ns HIGH 25 ns

1391623923 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold
errors)

Minimum period is 46.760ns.
```

En definitiva, la herramienta ha implementado la red de salida `wb_dcm/CLKDV_BUF` a la frecuencia de 20MHz (periodo = 50ns) a partir de la red de entrada `wb_dcm/CLKIN_IBUFG_OUT` de frecuencia 40MHz. Por otro lado, se determina la frecuencia máxima del sistema en 21.4MHz tras los cambios efectuados.

Para comprobar el correcto funcionamiento del módulo *DCM*, se genera de nuevo el modelo de simulación *post-layout* con el entorno *ISE* y se realiza la simulación temporal del sistema. Cabe resaltar que la frecuencia de la señal de reloj definida en el fichero *testbench* debe ser de 40MHz. Se observa en el detalle mostrado que la frecuencia de la señal de reloj `dcm/CLK_DV` ha sido reducida a la mitad de la frecuencia de la señal externa `dcm/CLK_IN`.



**Imagen 95** - Detalle de la simulación *post-layout*

### 3.5.9.3 Fichero UCF

El fichero de formato *User Constraints File* del proyecto del entorno *ISE* es un fichero ASCII que especifica las restricciones lógicas del diseño. Es posible crear o editar este fichero de forma manual o utilizar la herramienta *Constraints Editor* para definir restricciones en el fichero. Estas restricciones afectan a la implementación del diseño lógico en el dispositivo destino.

El fichero *UCF* es una entrada a la herramienta *NGDBuild* utilizada durante el proceso *Translate* en la implementación. Las restricciones en el fichero *UCF* pasan a formar parte de la información contenida en el fichero *NGD* generado por la herramienta *NGDBuild*. Algunas de estas restricciones son utilizadas durante el mapeado del diseño en el proceso *Map* mientras que otras son escritas en el fichero *PFC* (*Physical Constraints File*) producido por la herramienta *Map*. Las restricciones del fichero *PFC* son utilizadas

por cada una de las herramientas de diseño físico (como *Par*) ejecutadas tras el mapeado del diseño.

Existen otras metodologías para introducir restricciones en el diseño:

- En código *vhdl* se pueden especificar restricciones mediante *attribute* del lenguaje *vhdl*
- En el caso de *verilog* se utiliza la primitiva específica del sintetizador *//synthesis attribute*
- Herramientas como *Floorplanner* o *PACE (Pin Assignment Constraints Editor)* pueden ser utilizadas para especificar restricciones de forma gráfica y manual.

En definitiva, el fichero *UCF* utilizado para nuestro diseño es muy simple, y se limita a especificar las restricciones de asignación de pines del dispositivo y a especificar las restricciones temporales de la red relativa al reloj del sistema, es decir, la frecuencia de la señal externa y los parámetros del módulo *DCM*. En alguno de los apartados siguientes se informa de la adición de algunas líneas al fichero mostrado:

```
NET "clk" LOC = "AE14";
NET "clk" PERIOD = 25000 ps;
NET "flash_Ncs" LOC = "H16";
NET "rstn" LOC = "H14";
NET "sram_Nbe[0]" LOC = "AD5";
NET "sram_Nbe[1]" LOC = "AE5";
NET "sram_Nbe[2]" LOC = "AF6";
NET "sram_Nbe[3]" LOC = "AE4";
NET "sram_Ncs" LOC = "AE6";
NET "sram_Noe" LOC = "AD4";
NET "sram_Nwe" LOC = "AF4";
NET "uart_srx" LOC = "P4";
NET "uart_stx" LOC = "P5";
NET "sram_addr[2]" LOC = "AE10";
NET "sram_addr[3]" LOC = "AD9";
NET "sram_addr[4]" LOC = "AE9";
NET "sram_addr[5]" LOC = "AB8";
NET "sram_addr[6]" LOC = "AC8";
NET "sram_addr[7]" LOC = "AF8";
NET "sram_addr[8]" LOC = "AB7";
NET "sram_addr[9]" LOC = "AF7";
NET "sram_addr[10]" LOC = "AB6";
NET "sram_addr[11]" LOC = "AD10";
NET "sram_addr[12]" LOC = "AF10";
NET "sram_addr[13]" LOC = "AB9";
NET "sram_addr[14]" LOC = "AC9";
NET "sram_addr[15]" LOC = "AD8";
NET "sram_addr[16]" LOC = "AE8";
NET "sram_addr[17]" LOC = "AC7";
NET "sram_addr[18]" LOC = "AE7";
NET "sram_addr[19]" LOC = "AC6";
NET "sram_addr[20]" LOC = "AD6";
NET "sram_data[0]" LOC = "AB20";
NET "sram_data[1]" LOC = "AC20";
NET "sram_data[2]" LOC = "AA19";
NET "sram_data[3]" LOC = "AD19";
NET "sram_data[4]" LOC = "AC19";
```

```
NET "sram_data[5]" LOC = "AC18";
NET "sram_data[6]" LOC = "AA18";
NET "sram_data[7]" LOC = "AD17";
NET "sram_data[8]" LOC = "AB17";
NET "sram_data[9]" LOC = "AC16";
NET "sram_data[10]" LOC = "AB16";
NET "sram_data[11]" LOC = "AB15";
NET "sram_data[12]" LOC = "AE12";
NET "sram_data[13]" LOC = "AF12";
NET "sram_data[14]" LOC = "AC11";
NET "sram_data[15]" LOC = "AE11";
NET "sram_data[16]" LOC = "AA20";
NET "sram_data[17]" LOC = "Y19";
NET "sram_data[18]" LOC = "AB19";
NET "sram_data[19]" LOC = "AF19";
NET "sram_data[20]" LOC = "Y18";
NET "sram_data[21]" LOC = "AD18";
NET "sram_data[22]" LOC = "AB18";
NET "sram_data[23]" LOC = "AF17";
NET "sram_data[24]" LOC = "AC17";
NET "sram_data[25]" LOC = "AF16";
NET "sram_data[26]" LOC = "AF15";
NET "sram_data[27]" LOC = "AD15";
NET "sram_data[28]" LOC = "AF13";
NET "sram_data[29]" LOC = "AD12";
NET "sram_data[30]" LOC = "AB11";
NET "sram_data[31]" LOC = "AF11";
NET "gpio[7]" LOC = "T5";
NET "gpio[6]" LOC = "T6";
NET "gpio[5]" LOC = "T7";
NET "gpio[4]" LOC = "T8";
NET "gpio[3]" LOC = "W7";
NET "gpio[2]" LOC = "V6";
NET "gpio[1]" LOC = "W4";
NET "gpio[0]" LOC = "U4";
```

#### 3.5.9.4 Interruptores, botones y leds

Existen en la placa de desarrollo dos botones del tipo *push-button* (SW2 y SW3) propuestos para propósito general. En nuestro diseño asignamos el botón SW2 como generador del *reset hardware* del sistema. En reposo el botón fuerza un '0' lógico en la

línea y cuando es presionado genera un '1' lógico. Por lo tanto se trata de lógica directa, en contraste con el control de lógica negada que se ha desarrollado dentro del sistema. Esto se resuelve mediante la adaptación de las siguientes líneas del fichero de mayor jerarquía:

```
// Reset debounce
always @(posedge wb_clk or negedge rstn)
 if (~rstn)
 rst_r <= 1'b1;
 else
 rst_r <= #1 1'b0;
```

El puerto de entrada/salida de la *fpga* conectado al interruptor *SW2* es el etiquetado como *H14*. La línea debe 'NET "rstn" LOC = "H14";' ser añadida al fichero *UCF* para un correcto mapeado y enrutado de los componentes.

Por otro lado, resulta interesante el uso de los botones y/o leds integrados en la placa de desarrollo para controlar y visualizar el proceso de un programa en el dispositivo destino de una manera sencilla. El componente utilizado se ha descargado del directorio del proyecto *S\_GPIO* del *CVS* de *OpenCores*. Se trata de un fichero escrito en lenguaje *verilog* que define el comportamiento del *core* de un *GPIO* de 8 bits. El componente se conecta al bus mediante una interfaz *wishbone* y accede a dos registros de 8 bits para realizar su operación:

```
0x00 Control Register <io[7:0]>
bits 7:0 R/W Input/Output '1' = output mode
 '0' = input mode
```

Define el sentido (entrada/salida) de las líneas

```
0x01 Line Register
bits 7:0 R Status Current GPIO pin level
 W Output GPIO pin output level
```

Define el nivel de salida (0/1) de las líneas

A continuación se detalla el procedimiento para conectar un *GPIO* (*general purpose input/output*) a nuestro diseño.

- Agregar el fichero *simple\_gpio.v* al directorio de fuentes del proyecto
- Asignar el componente a la dirección de memoria del sistema 0x91000000. Se añade la línea ``define APP_ADDR_GPIO `APP_ADDR_DEC_W'h91` al fichero *xsv\_fpga\_defines.v*
- Conectar el componente al bus *wishbone*. Se sustituye la constante `APP_ADDR_GPIO` en la posición adecuada de la lista de genéricos de la instancia del *TrafficCop* en el fichero de mayor jerarquía (*xsv\_fpga\_top.v*). Además se conecta la interfaz *wishbone* del componente al puerto del *slave 6* del *Traffic Cop* con:

```
// WISHBONE Target 6 (simple_GPIO)
.t6_wb_cyc_o (wb_gp_cyc_i),
.t6_wb_stb_o (wb_gp_stb_i),
.t6_wb_cab_o (wb_gp_cab_i),
.t6_wb_adr_o (wb_gp_adr_i),
.t6_wb_sel_o (wb_gp_sel_i),
.t6_wb_we_o (wb_gp_we_i),
.t6_wb_dat_o (wb_gp_dat_i),
.t6_wb_dat_i (wb_gp_dat_o),
.t6_wb_ack_i (wb_gp_ack_o),
.t6_wb_err_i (wb_gp_err_o),
```

- Añadir el puerto de entrada/salida *gpio* al módulo de la *fpga*:

```

module xsv_fpga_top (
 clk, rstn,
 uart_stx, uart_srx,
 sram_Ncs, sram_Noce, sram_Nwe, sram_Nbe, sram_addr, sram_data,
 flash_Ncs,
 jtag_tck, jtag_tms, jtag_tdi, jtag_trst,
 jtag_tvref, jtag_tgnd, jtag_tdo,
 gpio);

inout [7:0] gpio;

```

- Instanciar el componente en el fichero *xsv\_fpga\_top.v* con el siguiente trozo de código:

```

simple_gpio simple_gpio (
 .clk_i (wb_clk),
 .rst_i (wb_rst),
 .cyc_i (wb_gp_cyc_i),
 .stb_i (wb_gp_stb_i),
 .adr_i (wb_gp_adr_i[0]),
 .we_i (wb_gp_we_i),
 .dat_i (wb_gp_dat_i[7:0]),
 .dat_o (wb_gp_dat_o[7:0]),
 .ack_o (wb_gp_ack_o),
 .gpio (gpio));

```

- Asignar los puertos de entrada/salida de la *fpga* a los pines conectados a los elementos (leds o interruptores) deseados. Se añaden las líneas siguientes al fichero *UCF* del proyecto:

```

NET "LED0" LOC ="U4";
NET "LED1" LOC ="W4";
NET "LED2" LOC ="V6";
NET "LED3" LOC ="W7";
NET "LED4" LOC ="T8";
NET "LED5" LOC ="T7";
NET "LED6" LOC ="T6";
NET "LED7" LOC ="T5";

```

- El programa *software* accede a los registros del componente con las líneas de código siguientes:

```

en fichero board.h:
#define GPIO_BASE 0x91000000
#define REG8 (add) *((volatile unsigned char *) (add))

en programa principal (ejemplo):
REG8(GPIO_BASE) = 0xFF; // SR --> 8 salidas (leds)
REG8(GPIO_BASE + 0x1) = 0x01; // LR

```

### 3.5.9.5 Configuración

El establecimiento de la conexión del equipo de desarrollo (ordenador personal) con el dispositivo destino se realiza mediante la herramienta *Impact* del entorno de desarrollo *ISE* de *Xilinx*. Por otro lado, la comunicación se consigue físicamente mediante la conexión del entrenador *UWPC4* al puerto paralelo del ordenador y al conector *J5* para cables *XPC-IV* de la placa de desarrollo.

La herramienta *Impact* utiliza un fichero de formato *bitstream* y extensión *bit* para configurar el dispositivo destino. Este fichero se obtiene con la ejecución del proceso *Generate Programming File* de la ventana de procesos del *Project Navigator*. Una vez

obtenido el fichero de configuración, se ejecuta la herramienta *Impact* y se procede a la programación del dispositivo destino mediante *Boundary-Scan* (*JTAG*).

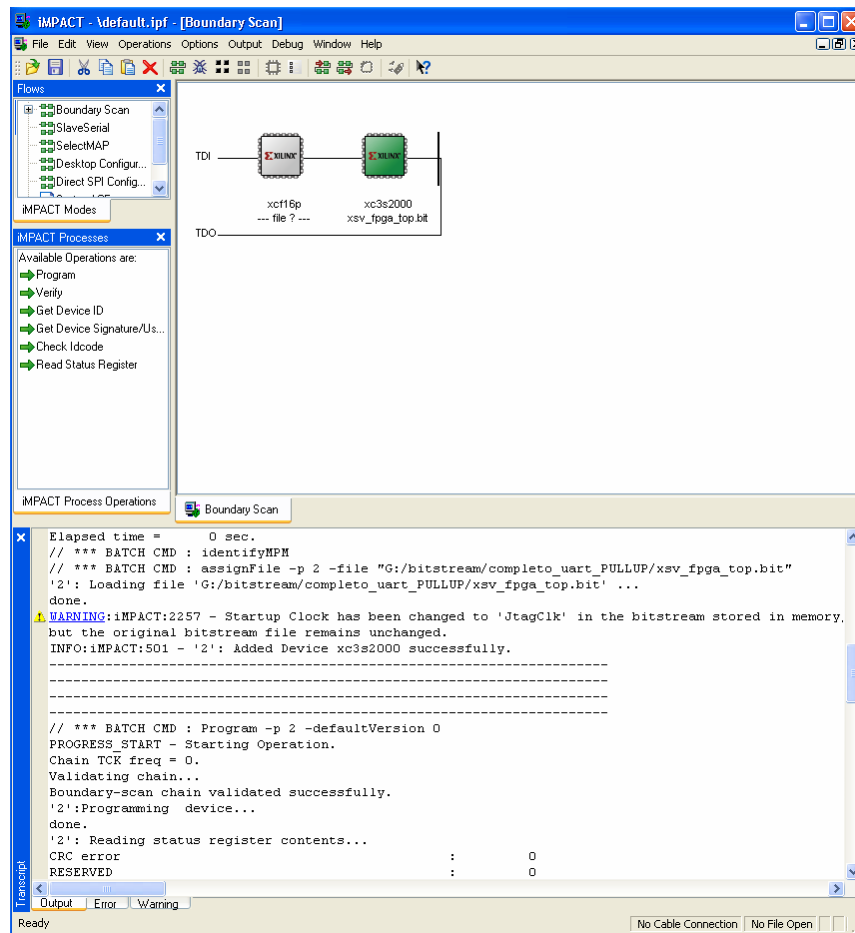


Imagen 96 - Pantalla principal de la herramienta *Impact*

La cadena detectada automáticamente por el *Boundary Scan* está compuesta por dos puertos *JTAG* diferentes. El primero corresponde con la memoria *PROM* de la placa de desarrollo utilizada para almacenar de forma permanente la configuración del dispositivo *fpga*, normalmente utilizada para diseños definitivos. El segundo puerto (configurado en el apartado 3.5.9.1.2) es el utilizado para la configuración del dispositivo programable desde el equipo de desarrollo. Se especifica el fichero *bitstream* para el puerto adecuado y se selecciona la opción *Program* del menú emergente del *mouse*.

### 3.5.9.6 Conexión *JTAG*

Para descargar el fichero *ELF*, que contiene los datos e instrucciones de programa con que se debe inicializar la memoria interna y externa, no se utiliza el conector *J5* de *JTAG* específico de la placa de desarrollo. Nuestro diseño tiene el *hardware* de *JTAG* embedido en la interfaz de depurado para asegurar compatibilidad con cualquier arquitectura *ASIC* o *FPGA* destino. Por lo tanto, se utilizan pines libres del conector *J17* de 50 pines de la placa de desarrollo. La asignación de pines es la mostrada en la tabla siguiente:

| Señal | Pin fpga | Pin header#50 |
|-------|----------|---------------|
| tck   | A19      | 1             |
| tdi   | A22      | 2             |
| tdo   | A20      | 3             |

|       |     |        |
|-------|-----|--------|
| tgnd  | A23 | 4      |
| tms   | D19 | 5      |
| trst  | A21 | 6 (nc) |
| tvref | E19 | 7      |

Para realizar la conexión de los pines asignados se diseña una placa de topes simple que conecte mediante soldadura de cableado los pines asignados del *Header JP1* de la placa de desarrollo con un conector para cables *XPC3 (Xilinx Parallel Cable III)*.

Las líneas añadidas al fichero *UCF* para la asignación de pines son:

```
JTAG mapeado en 50 pin header
NET "jtag_tck" LOC = "A19";
NET "jtag_tdi" LOC = "A22";
NET "jtag_tdo" LOC = "A20";
NET "jtag_tgnd" LOC = "A23";
NET "jtag_tms" LOC = "D19";
NET "jtag_trst" LOC = "A21";
NET "jtag_tvref" LOC = "E19";
```

### 3.5.9.7 Software JTAG

Para establecer una conexión *JTAG* desde el equipo de desarrollo se debe instalar un *software* específico con unas librerías que lo complementan. Los paquetes son los siguientes:

- Los ficheros fuente del programa *jp1* se pueden descargar del directorio *JTAG* del proyecto *or1k* de la web de *OpenCores*.
- La librería *ioperm 0.4.1* se puede instalar del directorio *devel* de la aplicación de instalación de *Cygwin*. Esta librería establece los permisos en los puertos de entrada/salida del ordenador.
- La librería *popt 1.6.4.4* se puede instalar del directorio *lib* de la aplicación de instalación de *Cygwin*. Se encarga de el análisis sintáctico de los parámetros de las líneas de comandos (*cmdline*).

Antes de la generación del programa *jp1* se deben resolver algunos problemas ocasionados por *bugs* de la versión. En el código del fichero *jp1.c* se sustituyen las llamadas a la función *error(...)* por la llamada a la función *exit(-1)*. Por otro lado, se debe añadir la librería *ioperm* instalada a las líneas de comandos de compilación del fichero *makefile* del proyecto de la manera siguiente:

```
jp1: Makefile jp1.o jp-io.o jp.h
 rm -f $@
 $(CC) -o $@ $(CFLAGS) jp1.o jp-io.o -lioperm

jp2: Makefile jp2.o gdb2.o jp2.h jp-io.o jp.h
 rm -f $@
 $(CC) -o $@ $(CFLAGS) jp2.o jp-io.o gdb2.o -lioperm
```

Ejecutada la línea de comando *make clean all* en el directorio de fuentes del programa desde una consola de *Cygwin*, se copian los ejecutables (*jp1.exe* y *jp2.exe*) al directorio de ejecutables del sistema */bin*. Para establecer una conexión *JTAG* se ejecuta la línea de comandos *jp1 xpc3 9999*, que establece una conexión para el cable *XPC3* en el puerto 9999, y se deja abierta la consola.

```

Enric@lab333 /cygdrive/c/cygwin
$ jpl xpc3 9999
Connected to parallel port at 378
Dropping root privileges.
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 4000000c ppc = 40000024 r1 = 00000005
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 4000000c ppc = 40000024 r1 = 00000008
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 40000024 ppc = 40000020 r1 = 0000000b
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 40000020 ppc = 4000001c r1 = 00000018
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 4000001c ppc = 40000018 r1 = 00000031
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 40000020 ppc = 4000001c r1 = 00000032
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 40000010 ppc = 4000000c r1 = 00000063
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 40000024 ppc = 40000020 r1 = 00000065
Read npc = 00000000 ppc = 00000000 r1 = 000015b4
Expected npc = 4000000c ppc = 40000024 r1 = 000000c9
result = 5eae9dfd
Dropping root privileges.
JTAG Proxy server started on port 9999
Press CTRL+c to exit.

```

Imagen 97 - Detalle del establecimiento de una conexión *JTAG*

### 3.5.9.8 Descarga del fichero *ELF*

Una vez conectado el cable *XPC3* al conector de la placa de topes y establecida la conexión *JTAG* mediante el programa *JPI*, se procede a la descarga del fichero *ELF* (*copro.or32*) sobre el dispositivo destino. Se consigue mediante la ejecución del depurador cruzado con la línea línea de comandos *or32-elf-gdb copro.or32*.

```

Floy@lestat /cygdrive/c/cygwin/proyectos/programs/test_fpga_simpuart
$ or32-elf-gdb copro.or32
GNU gdb 5.3
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=or32-elf"...
(gdb)

```

Una vez dentro de la interfaz del depurador, se procede a la conexión con el puerto 9999 y a la descarga del fichero *ELF* con las líneas de comandos específicas del depurador:

```

<gdb> target jtag jtag://localhost:9999
<gdb> load

```

Antes de ejecutar el programa, se conecta el cable serie del equipo de desarrollo al conector serie *J3* de la placa de desarrollo. Se abre una terminal (en este caso *HyperTerminal*) con los parámetros de conexión establecidos en el *core* de la *uart16550*, es decir, un *baudrate* de 19200, sin bit de paridad y un bit de parada. Se ejecuta el programa desde la interfaz del depurador con las líneas de comandos:

```

<gdb> set $pc = 0x100
<gdb> c

```

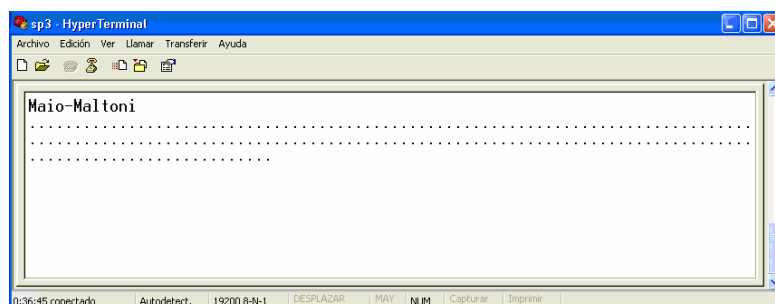


Imagen 98 - Detalle de la terminal durante la ejecución del programa

Se comprueba visualmente que el programa se ejecuta parcialmente, y el sistema se bloquea en un punto del código. Esto se observa debido a la iluminación de los *leds* al inicio del programa y a los mensajes recibidos mediante comunicación serie por la terminal. Sin embargo, para una depuración minuciosa del programa y del sistema se debe instalar una interfaz gráfica para el depurador cruzado. Este tema se trata en profundidad en el siguiente apartado del trabajo.

### 3.5.10 Depurado

Este apartado pretende ser un tutorial de la instalación de la interfaz gráfica para el depurador cruzado *GDB*, además de una descripción de la metodología de depurado para verificar el diseño y la aplicación ejecutada sobre el microprocesador del sistema embebido. Como se ha mencionado en el texto anterior, se recomienda la utilización de una interfaz gráfica para poder realizar un depurado minucioso a la vez que efectivo del sistema. La interfaz más comúnmente utilizada para la depuración es *DDD* y, por lo tanto, es la elegida para nuestro proyecto. Algunas de las definiciones relacionadas con la interfaz gráfica para *GDB* son:

- *DDD: Data Display Debugger* es una popular interfaz de usuario gráfica para depuradores en línea de comandos, como *GDB*. Se rige bajo licencia *GNU GPL* y es *software* libre. *DDD* tiene una *GUI* (interfaz gráfica de usuario) que permite visualizar el código fuente y una interfaz gráfica interactiva para visualizar los datos, donde las estructuras de datos son visualizadas como gráficos. Usando *DDD*, se puede determinar como se comporta la aplicación visualizando sus datos, y no viendo como se ejecutan sus líneas de código. Utilizada principalmente en sistemas *UNIX*, su funcionalidad se puede complementar con *plug-in* desarrollados por programadores de la comunidad de código abierto.
- *X11*: el sistema de ventanas *X* fue desarrollado en el *MIT* para dotar de una interfaz gráfica a los sistemas *Unix*. Este protocolo permite la interacción gráfica en red entre un usuario y una o más computadoras haciendo transparente la red para éste. Generalmente se refiere a la versión *X11* de este protocolo, el que está en uso actualmente. *X* es el encargado de mostrar la información gráfica y es totalmente independiente del sistema operativo. El sistema de ventanas *X* distribuye el procesamiento de aplicaciones especificando enlaces cliente-servidor. El servidor provee servicios para acceder a la pantalla, teclado y ratón, mientras que los clientes son las aplicaciones que utilizan estos recursos para interacción con el usuario. De este modo, mientras el servidor se ejecuta de manera local, las aplicaciones pueden ejecutarse remotamente desde otras máquinas, proporcionando así el concepto de transparencia de la red.
- *LessTif*: se trata de una reimplementación o clon del kit de herramientas de programación *Motif*. Desarrollado por la agrupación *Hungry Programmers* para ser la alternativa compatible de software libre para *Motif*. La aplicación se encuentra en actual estado de desarrollo. *Motif* es una librería para la creación de entornos gráficos bajo *X Window System* en sistemas *Unix*. Se trata de un estándar de la industria bajo el código IEEE 1295. Es utilizado como infraestructura del entorno de escritorio *CDE*.

Los requerimientos para instalar la interfaz *DDD* son:

- La colección de compilación de *GNU (GCC)* de la versión 3.0 o superior.



- La colección de herramientas *LessTif* de la versión 0.89 o superior. Se requiere el uso de la librería *X11* para la utilización de *LessTif*.

Se pueden descargar los ficheros binarios de las librerías y herramientas necesarias de la aplicación de instalación de *Cygwin* e instalarlos automáticamente. En el directorio '*X11*' de la *ftp* se encuentran los ficheros binarios siguientes:

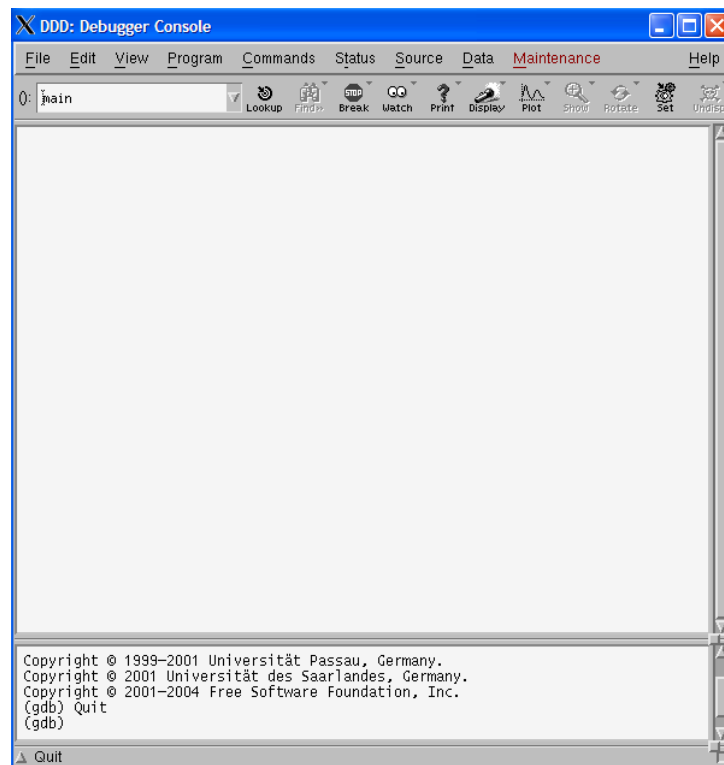
- *xorg-x11-base*: es un simple *script* de instalación de la librería *X11* que selecciona automáticamente un set básico de paquetes para la instalación típica que permitirá el funcionamiento de *Cygwin/X*
- *ddd-3.3.9*: es la distribución binaria de la versión 3.3.9 de la aplicación *DDD*
- *lesstif-0.94.4*: es la distribución binaria de la versión 0.94.4 del paquete de herramientas *LessTif*

Para ejecutar el depurador *GDB* en el entorno gráfico instalado, se abre una consola de *cygwin* y se introducen las líneas de comandos siguientes:

```
$ cd c:/cygwin/usr/x11r6/bin
$ startxwin.sh
```

Con esto, se abre una ventana de *Cygwin/X* desde la que se ejecuta la aplicación *DDD* tras acceder al directorio donde se encuentra el ejecutable a depurar (*copro.or32*):

```
$ ddd --debugger /opt/or32-elf/bin/or32-elf-gdb.exe
```



**Imagen 99** - Pantalla principal del entorno gráfico de depuración *DDD*

Desde la aplicación *DDD* mostrada en la *Imagen 99*, se procede al depurado del sistema. Se recuerda que el cable *XPC3* debe estar conectado del puerto paralelo del equipo de desarrollo al conector específico soldado a la placa de topes adjuntada a la placa de desarrollo. Además, la aplicación *JPI* debe encontrarse ejecutada en una consola *Cygwin* para tener una conexión *JTAG* establecida. Para abrir el programa a depurar, se accede a la opción *File* → *OpenProgram* del menú principal del entorno *DDD*. Se

selecciona el fichero *ELF (copro.or32)* y se confirma la operación. Debe tenerse en cuenta que los ficheros fuente utilizados para la generación del fichero ejecutable deben encontrarse en el mismo directorio de trabajo para el depurado.

Ahora se procede a la especificación del dispositivo destino mediante la indicación del puerto *JTAG* utilizado. La línea de comando especifica una conexión del tipo *JTAG* en el puerto 9999, es decir, utiliza la conexión establecida por el programa *JPI*.

```
target jtag jtag://localhost:9999
load
```

El comando *load* descarga los datos contenidos en el fichero *ELF* sobre el mapa de direcciones de nuestro sistema. Como se ha observado en el apartado 3.5.9, el programa realiza una serie de operaciones hasta que el sistema se bloquea en un punto del código desconocido. Por la información obtenida de los *leds* y los datos recibidos por el puerto serie en la terminal establecida, se puede tratar de aislar el punto del programa donde se bloquea el sistema. Por lo tanto, se fuerza un punto de ruptura (*breakpoint*) en la instrucción `followridge(j_start,i_start);` contenida en la función `Estraccio()` del fichero fuente *estraccio.cpp*. Cada vez que el puntero del programa ejecuta la instrucción, se realiza una iteración del algoritmo *follow* y se envía un mensaje por el puerto serie. De esta forma, comparando las iteraciones ejecutadas con los mensajes recibidos por el puerto serie se puede acceder a la ejecución paso a paso de la iteración que bloquea el sistema.

Una vez establecido el punto de ruptura en la instrucción adecuada, se procede a la ejecución del programa con la selección de la opción *Program → Step* e indicando el punto inicial de la ejecución con la opción *Set Execution Position* del menú emergente del *mouse* en la primera instrucción de la función *main()*. En este punto se selecciona la opción *Program → Continue* hasta que se determina a partir de los mensajes obtenidos en la terminal del equipo de desarrollo que el programa se encuentra en la ejecución de la iteración del algoritmo *follow* que provoca el bloqueo del sistema. Mediante la activación de puntos de ruptura y la ejecución paso a paso se aísla la instrucción que provoca el problema. Se trata de la instrucción de lenguaje de alto nivel `int f_mean = suma1 / (numitems);` contenida en la función *contrast()* del fichero fuente *funcions.cpp*.

El compilador cruzado de *GNU* ha ensamblado una instrucción *jal (jump and link*, en definitiva, un salto absoluto) a la posición cero en lugar de la división entera. El tema de la división en el microprocesador *OpenRisc* es bastante interesante. El *or1200* permite la implementación opcional de un divisor *hardware* que realiza las operaciones *l.div* y *l.divu* del código ensamblador mediante la habilitación opcional de la primitiva ``define OR1200_IMPL_DIV`. Por defecto, las instrucciones de división no están implementadas en el sistema ya que la arquitectura específica del divisor consume un área importante en el dispositivo destino y reduce la frecuencia máxima del reloj del sistema.

Por otro lado, el compilador *GCC* puede utilizar la librería *software* llamada *libgcc.a* para la implementación de la división. Para conseguirlo se añade la opción *-msoft-div* a la línea de compilación del fichero *funcions.cpp* en el fichero *makefile* del proyecto. El compilador entonces ensambla una llamada a la función `__udivsi3`, mapeada al final del fichero *ELF copro.or32* por el enlazador cruzado.

Una vez regenerado el fichero *ELF* y descargado sobre el dispositivo destino mediante el depurador cruzado, el programa se ejecuta de forma correcta. El programa realiza 3451 ejecuciones *hardware* del coprocesador en las 9801 iteraciones del algoritmo *follow*.

## 4 Conclusiones

### 4.1 Resultados Experimentales

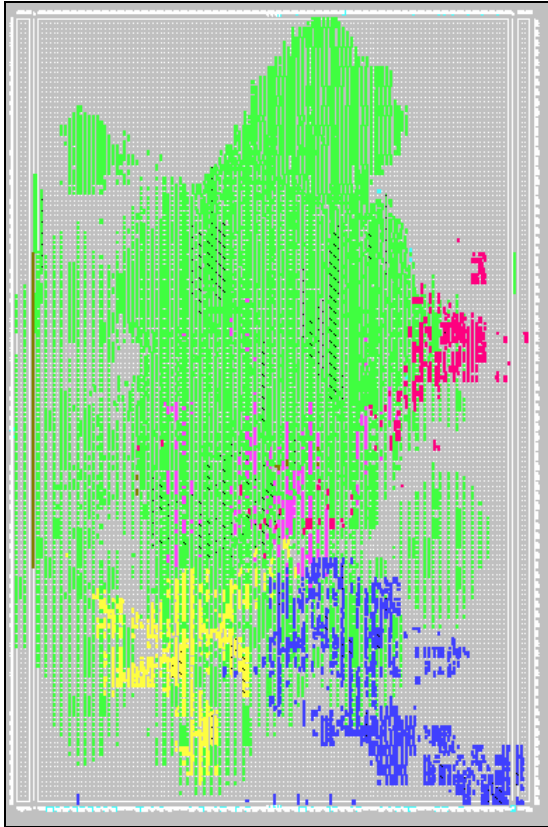
El objetivo principal de este trabajo es estudiar la factibilidad de la implementación de un sistema de reconocimiento de huella dactilar basado en coprocesador biométrico sobre una arquitectura *SoC Wishbone* para un dispositivo *FPGA*. A lo largo del trabajo se ha descrito el proceso de desarrollo y verificación para, en los últimos apartados, testear el sistema obtenido sobre un dispositivo *xc3s2000* de la familia *Spartan3* de *Xilinx*. Este objetivo ha sido alcanzado con resultados positivos.

El microprocesador *OpenRisc* se encuentra en actual proceso de desarrollo. La comunidad *OpenCores* está trabajando en el comportamiento funcional del *core*, dejando para el futuro la optimización de utilización de recursos y de temporización. Por lo tanto, los resultados obtenidos son poco óptimos aunque no definitivos. Los resultados de síntesis obtenidos con el entorno *ISE* de *Xilinx* para cada *core* sintetizado independientemente son:

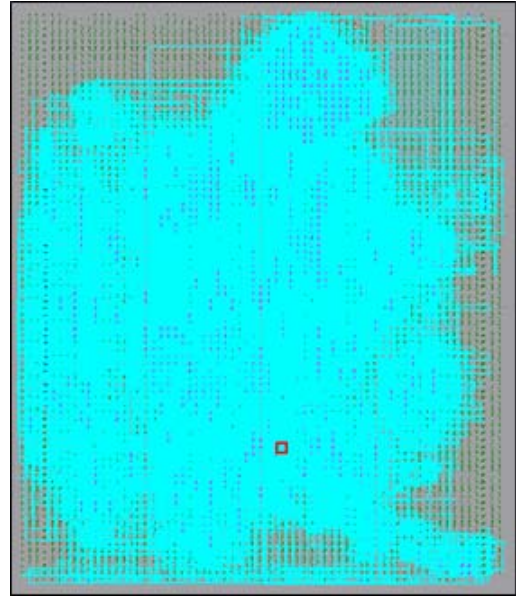
| Partición          | Slices | LUTs  | Otros          |
|--------------------|--------|-------|----------------|
| Coprocesador       | 866    | 1222  |                |
| Interfaz depurador | 268    | 341   |                |
| Memoria interna    | 7      | 10    | Block RAM = 16 |
| OpenRISC           | 9221   | 25301 |                |
| UART 16550         | 473    | 856   |                |
| Traffic Cop        | 378    | 731   |                |

El diseño completo ha sido sintetizado en 11231 *slices* utilizando 20045 *LUT*'s con una utilización aproximada de el 50% de los recursos del dispositivo. A su vez, utiliza recursos específicos *Block Ram* para implementar la memoria interna del sistema. El número total de puertas lógicas utilizadas para la implementación del diseño es de 1.7M aproximadamente.

La implementación sobre el dispositivo destino ha sido costosa por la complejidad del diseño. Los procesos de mapeado y enrutado han descartado la posibilidad de utilizar un dispositivo *FPGA* de menor tamaño aunque el diseño ocupa el 50% de sus recursos solamente. Los resultados gráficos de la implementación del sistema se muestran a continuación:



**Imagen 100** - Resultado del mapeado del diseño



**Imagen 101** - Resultado del enrutado del diseño

Durante la ejecución del algoritmo *software* sobre el microprocesador embebido, el sistema ha realizado 3451 ejecuciones *hardware* del coprocesador a lo largo de las 9801 iteraciones del algoritmo *follow*. La matriz  $M$  de resultados donde se almacenan los datos de la minutia queda:

| i   | j   | w   | tipo |
|-----|-----|-----|------|
| 58  | 111 | 4   | 2    |
| 74  | 177 | 0   | 2    |
| 84  | 60  | -9  | 2    |
| 85  | 113 | 5   | 2    |
| 87  | 41  | -10 | 2    |
| 98  | 138 | 7   | 2    |
| 105 | 165 | -2  | 2    |
| 110 | 140 | -3  | 2    |
| 114 | 195 | 12  | 2    |
| 114 | 120 | 6   | 2    |
| 122 | 76  | -1  | 2    |
| 127 | 123 | -11 | 2    |
| 128 | 88  | 0   | 2    |
| 143 | 158 | -2  | 1    |
| 146 | 25  | 0   | 2    |
| 151 | 135 | 12  | 2    |
| 152 | 111 | 6   | 2    |
| 155 | 239 | -1  | 2    |
| 157 | 97  | 5   | 2    |
| 159 | 75  | 0   | 2    |
| 166 | 209 | 8   | 2    |
| 172 | 200 | -4  | 2    |

| i          | j          | w         | tipo |
|------------|------------|-----------|------|
| <b>172</b> | <b>229</b> | <b>-2</b> | 2    |
| 172        | 36         | 0         | 2    |
| 173        | 95         | 5         | 2    |
| 174        | 36         | 0         | 2    |
| 178        | 205        | -3        | 2    |
| 179        | 34         | 1         | 2    |
| 180        | 81         | 0         | 2    |
| 182        | 80         | 1         | 2    |
| 185        | 66         | 5         | 2    |
| 186        | 214        | 10        | 2    |
| 186        | 51         | 0         | 2    |
| 189        | 216        | 8         | 2    |
| 109        | 76         | 1         | 2    |
| 203        | 26         | 5         | 1    |
| 204        | 133        | -8        | 2    |
| 206        | 20         | 2         | 2    |
| 213        | 81         | 7         | 2    |
| 221        | 209        | -6        | 2    |
| 221        | 50         | -1        | 2    |
| 226        | 73         | 5         | 2    |
| 234        | 184        | -6        | 2    |

Las valores de  $i$  y  $j$  representan las coordenadas del punto en un sistema cartesiano ( $x,y$ ). El valor  $w$  codifica el ángulo ( $-180^\circ$  a  $180^\circ$ ) mediante factores enteros ( $-12$  a  $12$ ) que

representan 15° cada unidad. La codificación de la columna *tipo* especifica si un punto determinado es un final de cresta (1) o una bifurcación (2).

Una iteración *hardware* del coprocesador requiere 750 ciclos de reloj, mientras las operaciones *software* que debería ejecutar el microcontrolador para obtener la misma funcionalidad requieren 144K ciclos de reloj. Se deduce que el tiempo específico para el procesamiento del cálculo realizado por el coprocesador se reduce en un factor superior al 190 en comparación del mismo cálculo realizado sin coprocesador. Al comparar los ciclos necesarios para la extracción de la minutía de la imagen completa usando el coprocesador, el tiempo de ejecución se reduce en un 52% respecto a la ejecución puramente *software*.

## 4.2 Propuestas

### a) Reducción de la memoria interna

Puede ocurrir que el programa principal que debe ejecutar el microprocesador no pueda ser mapeado íntegramente en la memoria interna embebida. Esto puede darse debido a la utilización de un dispositivo destino de menor tamaño (posible tras la optimización del *OpenRisc* por parte de la comunidad *OpenCores*) y/o por la necesidad de ejecutar un *software* de dimensiones elevadas. En este caso se propone el mapeado del programa en la memoria externa, manteniendo las rutinas de servicio a las excepciones y la pila en memoria interna. Esto puede provocar una reducción importante del *throughput* del sistema, aunque se puede mejorar mediante el mapeado de las rutinas críticas en una pequeña zona de la memoria embebida. Para ello se debe generar una sección específica en el fichero *ELF* para las rutinas críticas. La declaración de las rutinas críticas debe realizarse de forma similar a:

```
void rutina_critica(...) __attribute__ ((section(".seccion_critica")));
```

El fichero de *scripts* de enlazado (*ram.ld*) debe tener un formato similar a:

```
MEMORY
{
 vectors : ORIGIN = 0x00000000, LENGTH = 0x00002000
 ram_int : ORIGIN = 0x00002000, LENGTH = 0x00002000
 ram_ext : ORIGIN = 0x01000000, LENGTH = 0x00030000
}

SECTIONS
{
 .vectors :
 {*(.vectors)} > vectors

 .text :
 {*(.text)} > ram_ext

 .data :
 {*(.data)} > ram_ext

 .rodata :
 {*(.rodata)} > ram_ext

 .bss :
 {*(.bss)} > ram_ext

 .stack :
 {*(.stack)
 _src_addr = .;} > ram_int

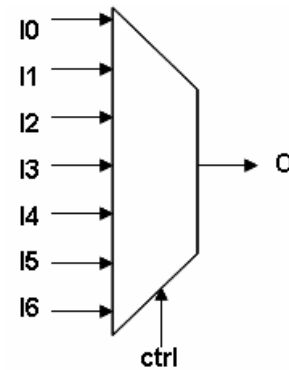
 .seccion_critica :
 {*(.seccion_critica)} > ram_int
```

}

### b) Optimización del *Traffic Cop*

La arquitectura del árbitro del bus utilizado en el presente proyecto se describe en el apartado 3.2.2 del documento. Se trata del fichero de *Traffic Cop* original adjunto al proyecto *OPRSoC*. Los resultados temporales obtenidos con la herramienta *ISE* han sido óptimos permitiendo una frecuencia máxima aproximada de 170MHz. Sin embargo, se ha detectado en el código un punto crítico en la gestión de múltiples señales para un solo canal de datos. El trozo de código conflictivo es:

```
assign t0_out = (req_won == 3'd0) ? i0_in :
 (req_won == 3'd1) ? i1_in :
 (req_won == 3'd2) ? i2_in :
 (req_won == 3'd3) ? i3_in :
 (req_won == 3'd4) ? i4_in :
 (req_won == 3'd5) ? i5_in :
 (req_won == 3'd6) ? i6_in :
 (req_won == 3'd7) ? i7_in : {'TC_IIN_W{1'b0}};
```



El código implementa un decodificador de prioridad que comprueba las condiciones de forma individual y descendente hasta que encuentra el valor adecuado para la señal de salida.

Esta configuración tiene el problema de generar un excesivo retardo combinacional cuando la condición del periférico destino no se encuentra entre las primeras de la lista. Por lo tanto, se propone sustituir el trozo de código por una arquitectura AND-OR sin prioridad:

```
assign t0_out = ({'TC_IIN_W{req_won[0]}}&i0_in) |
 ({'TC_IIN_W{req_won[1]}}&i1_in) | ({'TC_IIN_W{req_won[2]}}&i2_in) |
 ({'TC_IIN_W{req_won[3]}}&i3_in) | ({'TC_IIN_W{req_won[4]}}&i4_in) |
 ({'TC_IIN_W{req_won[5]}}&i5_in) | ({'TC_IIN_W{req_won[6]}}&i6_in) |
 ({'TC_IIN_W{req_won[7]}}&i7_in);
```

La línea implementa ocho puertas lógicas que realizan la función AND de la señal de  $n$  bits de entrada con el bit de control replicado  $n$  veces, con las salidas conectadas a una función lógica OR. Por lo tanto, el retardo se establece en dos niveles lógicos mientras que, en el caso del decodificador, el retardo máximo era el producido por 8 niveles lógicos y un multiplexor. Para introducir este cambio, se debe realizar la gestión de las señales de control *req\_won* y *req\_r* mediante registros de 8 bits, en lugar de utilizar una codificación binaria en tres bits. El mismo procedimiento se debe seguir para sustituir el decodificador de prioridad implementado en el módulo *tc\_si\_to\_mt*.

### c) Generación del *Traffic Cop*

Cualquier sistema embebido basado en una arquitectura de bus *Wishbone* debe contener un fichero que describa la interconexión de los *cores* que componen el sistema *hardware* llamado *traffic cop* (árbitro del bus) que debe cumplir la especificación de interconexión *wishbone*. Este fichero puede ser especificado manualmente a partir de una plantilla como se ha realizado en el presente proyecto. Sin embargo, existe una herramienta destinada a la generación automatizada de interconexiones *wishbone* llamada *Wishbone Builder*. Este es el tema tratado en este apartado del trabajo.

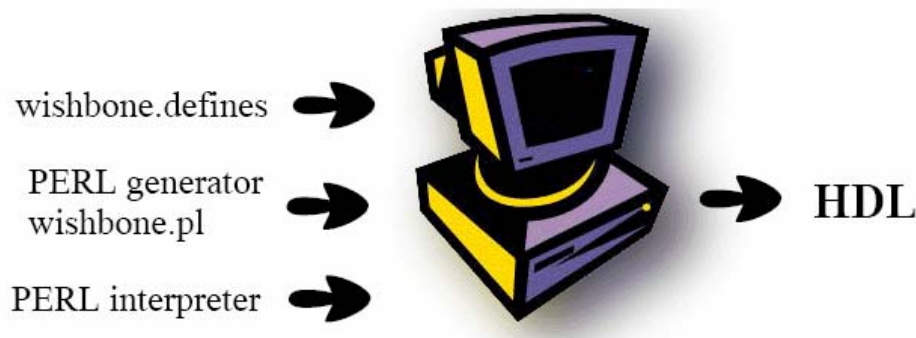


Imagen 102 - Flujo del generador

El generador de *wishbone* está escrito en lenguaje *PERL*. Esto se debe a que *PERL*:

- es un lenguaje de programación para plataformas cruzadas estable
- se utiliza para proyectos de misión crítica para el ámbito público y el privado
- es *software* de código abierto con licencia *GNU GPL* o similar

La configuración del árbitro del bus *wishbone* se especifica en un fichero de definición llamado típicamente *wishbone.defines*. El generador se llama *wishbone.pl* y es invocado mediante la introducción de la línea de comandos siguiente en una consola de comandos:

```
perl wishbone.pl [-nogui] [wishbone.defines]
```

La opción *-nogui* ejecuta el generador para una generación automatizada. De otra manera, una interfaz gráfica *GUI* es utilizada para la confección manual del árbitro.

La interfaz *wishbone* habitual puede ser modificada mediante señales definidas por el usuario. Esto se hace mediante la técnica llamada *tagging*. Las etiquetas son un concepto muy utilizado en la industria de los buses para microcontroladores. Permiten asociar información definida por el usuario con una dirección, una palabra de datos o un ciclo de bus. Sin embargo, este aspecto tiene poca relevancia en el presente proyecto, por lo que la información específica será omitida.

Todas las opciones de configuración deben residir dentro del fichero *wishbone.defines*. Los comentarios empiezan por el carácter *#* en el fichero de configuración. El número de master y slaves presentes en el sistema debe ser definido. El mapa de memoria utilizado por la herramienta coincide con el proporcionado con el proyecto *OPRSoC*.

#### a) Configuración global

| Parámetro     | Función                                                         | Valores                   | Por defecto |
|---------------|-----------------------------------------------------------------|---------------------------|-------------|
| syscon        | Define el nombre del módulo/entidad del controlador del sistema | -                         | syscon      |
| intercon      | Define el nombre del módulo/entidad de la interconexión         | -                         | intercon    |
| filename      | Define el nombre del fichero                                    | -                         | wb_arbiter  |
| hdl           | Define el lenguaje de salida                                    | vhdl, verilog, perlilog   | vhdl        |
| signal_groups | Define si el agrupamiento de señales debe ser utilizado         | 0,1                       | 0           |
| target        | Define la tecnología destino                                    | generic, xilinx, altera   | generic     |
| dat_size      | Define la anchura del bus de datos                              | 8, 16, 32, 64             | 32          |
| adr_size      | Define la anchura del bus de direcciones                        | -                         | 32          |
| interconnect  | Especifica la topología de bus                                  | sharedbus, crossbarswitch | sharedbus   |

|          |                                                         |                      |       |
|----------|---------------------------------------------------------|----------------------|-------|
| mux_type | Define el estilo de implementación de los multiplexores | andor, tristate, mux | andor |
| optimize | Optimiza para area o velocidad                          | speed, area          | speed |

### b) Configuración del puerto de un Master

La configuración debe estar contenida entre las palabras clave *master* y *end master* seguidas ambas por el nombre del componente.

| Parámetro       | Función                                                     | Valores              | Por defecto       |
|-----------------|-------------------------------------------------------------|----------------------|-------------------|
| dat_size        | Define la anchura del bus <i>dat_i</i>                      | 0, 8, 16, 24, 32, 64 | dat_size          |
| type            | Define la funcionalidad de lectura y/o escritura            | ro, wo, rw           | rw                |
| adr_o           | Define la anchura del bus de direcciones                    | 0-63                 | adr_size          |
| lock_o          | Define si bloquear la salida está permitido                 | 0,1                  | 1                 |
| err_i           | Define si la entrada error está soportada                   | 0,1                  | 1                 |
| rty_i           | Define si reintentar está soportado                         | 0,1                  | 1                 |
| tga_o           | Define si se usan las etiquetas de direcciones              | 0,1                  | 0                 |
| tgc_o           | Define si se usan las etiquetas de ciclo                    | 0,1                  | 0                 |
| priority        | Define el acceso al bus para sistemas <i>sharedbus</i>      | -                    | 1                 |
| priority_%slave | Define el acceso al bus para sistemas <i>crossbarswitch</i> | -                    | debe ser definido |

### c) Configuración del puerto de un Slave

La configuración debe estar contenida entre las palabras clave *slave* y *end slave* seguidas ambas por el nombre del componente.

| Parámetro | Función                                                                         | Valores              | Por defecto |
|-----------|---------------------------------------------------------------------------------|----------------------|-------------|
| dat_size  | Define la anchura del bus de datos                                              | 0, 8, 16, 24, 32, 64 | dat_size    |
| type      | Define la funcionalidad de lectura y/o escritura                                | ro, wo, rw           | rw          |
| adr_i_hi  | Anchura del bus de direcciones, límite superior                                 | 0-31                 | 31          |
| adr_i_lo  | Anchura del bus de direcciones, límite inferior                                 | 0-31                 | 2           |
| lock_i    | Define si bloquear la entrada está permitido                                    | 0,1                  | 0           |
| err_o     | Indica una terminación de ciclo anormal                                         | 0,1                  | 0           |
| rty_o     | Indica que la interfaz no está preparada y que el ciclo debería ser reintentado | 0,1                  | 0           |
| tga_i     | Define si se usan las etiquetas de direcciones                                  | 0,1                  | 0           |
| tgc_i     | Define si se usan las etiquetas de ciclo                                        | 0,1                  | 0           |
| baseaddr* | Dirección base del modulo                                                       | -                    | -           |
| size*     | Tamaño de memoria utilizado por el módulo                                       | -                    | 0x0010_0000 |

\* Estas opciones pueden ser múltiples



## 5 Anexos

### 5.1 Anexos de Software

#### 5.1.1 Anexo 1: Programa hello-uart (software)

Ficheros:       hello.c  
                  reset.S  
                  board.h  
                  mc.h  
                  uart.h  
                  makefile  
                  ram.ld

##### ***Fichero hello.c***

```
#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)

void uart_init(void)
{
 int divisor;
 /* Reset receiver and transmitter */
 REG8(UART_BASE + UART_FCR) = UART_FCR_ENABLE_FIFO | UART_FCR_CLEAR_RCVR |
 UART_FCR_CLEAR_XMIT | UART_FCR_TRIGGER_14;
 /* Disable all interrupts */
 REG8(UART_BASE + UART_IER) = 0x00;
 /* Set 8 bit char, 1 stop bit, no parity */
 REG8(UART_BASE + UART_LCR) = UART_LCR_WLEN8 & ~(UART_LCR_STOP |
 UART_LCR_PARITY);
 /* Set baud rate */
 divisor = IN_CLK/(16 * UART_BAUD_RATE);
 REG8(UART_BASE + UART_LCR) |= UART_LCR_DLAB;
 REG8(UART_BASE + UART_DLL) = divisor & 0x000000ff;
 REG8(UART_BASE + UART_DLM) = (divisor >> 8) & 0x000000ff;
 REG8(UART_BASE + UART_LCR) &= ~(UART_LCR_DLAB);
}

void uart_putc(char c)
{
 unsigned char lsr;
 WAIT_FOR_THRE;
```

```

REG8(UART_BASE + UART_TX) = c;
if(c == '\n')
{
 WAIT_FOR_THRE;
 REG8(UART_BASE + UART_TX) = '\r';
}
WAIT_FOR_XMITR;
}

char uart_getc(void)
{
 unsigned char lsr;
 char c;
 WAIT_FOR_CHAR;
 c = REG8(UART_BASE + UART_RX);
 return c;
}

char *str = "Hello world!!!\n";
int main (void)
{
 char *s;
 uart_init ();
 for (s = str; *s; s++)
 uart_putc (*s);
 while (1)
 uart_putc (uart_getc () + 1);
 return 0;
}

```

### ***Fichero reset.S***

```

#include "board.h"
#include "mc.h"

 .global __main
 .section .stack, "aw", @nobits
.space STACK_SIZE
_stack:

 .section .vectors, "ax"
 .org 0x100
_reset:
 l.movhi r1,hi(_stack-4)
 l.ori r1,r1,lo(_stack-4)
 l.addi r2,r0,-3
 l.and r1,r1,r2

 l.movhi r2,hi(_main)
 l.ori r2,r2,lo(_main)
 l.jr r2
 l.addi r2,r0,0

__main:
 l.jr r9
 l.nop

```

### ***Fichero board.h***

```

#ifndef _BOARD_H_
#define _BOARD_H_

```

```

#define IN_CLK 20000000
#define STACK_SIZE 0x1000
#define UART_BAUD_RATE 19200 /* 115200 */
#define UART_BASE 0x90000000
/* Register access macros */
#define REG8(add) *((volatile unsigned char *) (add))
#define REG16(add) *((volatile unsigned short *) (add))
#define REG32(add) *((volatile unsigned long *) (add))
#endif

```

### ***Fichero mc.h***

```

/* Prototypes */
#ifndef __MC_H
#define __MC_H

#define N_CE (8)
#define MC_CSR (0x00)
#define MC_POC (0x04)
#define MC_BA_MASK (0x08)
#define MC_CSC(i) (0x10 + (i) * 8)
#define MC_TMS(i) (0x14 + (i) * 8)
#define MC_ADDR_SPACE (MC_CSC(N_CE))

/* POC register field definition */
#define MC_POC_EN_BW_OFFSET 0
#define MC_POC_EN_BW_WIDTH 2
#define MC_POC_EN_MEMTYPE_OFFSET 2
#define MC_POC_EN_MEMTYPE_WIDTH 2

/* CSC register field definition */
#define MC_CSC_EN_OFFSET 0
#define MC_CSC_MEMTYPE_OFFSET 1
#define MC_CSC_MEMTYPE_WIDTH 2
#define MC_CSC_BW_OFFSET 4
#define MC_CSC_BW_WIDTH 2
#define MC_CSC_MS_OFFSET 6
#define MC_CSC_MS_WIDTH 2
#define MC_CSC_WP_OFFSET 8
#define MC_CSC_BAS_OFFSET 9
#define MC_CSC_KRO_OFFSET 10
#define MC_CSC_PEN_OFFSET 11
#define MC_CSC_SEL_OFFSET 16
#define MC_CSC_SEL_WIDTH 8

#define MC_CSC_MEMTYPE_SDRAM 0
#define MC_CSC_MEMTYPE_SSRAM 1
#define MC_CSC_MEMTYPE_ASYNC 2
#define MC_CSC_MEMTYPE_SYNC 3

#define MC_CSR_VALID 0xFF000703LU
#define MC_POC_VALID 0x00000000FLU
#define MC_BA_MASK_VALID 0x00000000FFLU
#define MC_CSC_VALID 0x00FF0FFFFLU
#define MC_TMS_SDRAM_VALID 0x0FFF83FFLU
#define MC_TMS_SSRAM_VALID 0x00000000LU
#define MC_TMS_ASYNC_VALID 0x03FFFFFFLU
#define MC_TMS_SYNC_VALID 0x01FFFFFFLU
#define MC_TMS_VALID 0xFFFFFFFFFLU /* reg test compat. */

/* TMS register field definition SDRAM */

```

```

#define MC_TMS_SDRAM_TRFC_OFFSET 24
#define MC_TMS_SDRAM_TRFC_WIDTH 4
#define MC_TMS_SDRAM_TRP_OFFSET 20
#define MC_TMS_SDRAM_TRP_WIDTH 4
#define MC_TMS_SDRAM_TRCD_OFFSET 17
#define MC_TMS_SDRAM_TRCD_WIDTH 4
#define MC_TMS_SDRAM_TWR_OFFSET 15
#define MC_TMS_SDRAM_TWR_WIDTH 2
#define MC_TMS_SDRAM_WBL_OFFSET 9
#define MC_TMS_SDRAM_OM_OFFSET 7
#define MC_TMS_SDRAM_OM_WIDTH 2
#define MC_TMS_SDRAM_CL_OFFSET 4
#define MC_TMS_SDRAM_CL_WIDTH 3
#define MC_TMS_SDRAM_BT_OFFSET 3
#define MC_TMS_SDRAM_BL_OFFSET 0
#define MC_TMS_SDRAM_BL_WIDTH 3

/* TMS register field definition ASYNC */
#define MC_TMS_ASYNC_TWWD_OFFSET 20
#define MC_TMS_ASYNC_TWWD_WIDTH 6
#define MC_TMS_ASYNC_TWD_OFFSET 16
#define MC_TMS_ASYNC_TWD_WIDTH 4
#define MC_TMS_ASYNC_TWPW_OFFSET 12
#define MC_TMS_ASYNC_TWPW_WIDTH 4
#define MC_TMS_ASYNC_TRDZ_OFFSET 8
#define MC_TMS_ASYNC_TRDZ_WIDTH 4
#define MC_TMS_ASYNC_TRDV_OFFSET 0
#define MC_TMS_ASYNC_TRDV_WIDTH 8

/* TMS register field definition SYNC */
#define MC_TMS_SYNC_TTO_OFFSET 16
#define MC_TMS_SYNC_TTO_WIDTH 9
#define MC_TMS_SYNC_TWR_OFFSET 12
#define MC_TMS_SYNC_TWR_WIDTH 4
#define MC_TMS_SYNC_TRDZ_OFFSET 8
#define MC_TMS_SYNC_TRDZ_WIDTH 4
#define MC_TMS_SYNC_TRDV_OFFSET 0
#define MC_TMS_SYNC_TRDV_WIDTH 8
#endif

```

### ***Fichero uart.h***

```

#ifndef _UART_H_
#define _UART_H_

#define UART_RX 0 /* In: Receive buffer (DLAB=0) */
#define UART_TX 0 /* Out: Transmit buffer (DLAB=0) */
#define UART_DLL 0 /* Out: Divisor Latch Low (DLAB=1) */
#define UART_DLM 1 /* Out: Divisor Latch High (DLAB=1) */
#define UART_IER 1 /* Out: Interrupt Enable Register */
#define UART_IIR 2 /* In: Interrupt ID Register */
#define UART_FCR 2 /* Out: FIFO Control Register */
#define UART_EFR 2 /* I/O: Extended Features Register */
 /* (DLAB=1, 16C660 only) */
#define UART_LCR 3 /* Out: Line Control Register */
#define UART_MCR 4 /* Out: Modem Control Register */
#define UART_LSR 5 /* In: Line Status Register */
#define UART_MSR 6 /* In: Modem Status Register */
#define UART_SCR 7 /* I/O: Scratch Register */

// These are the definitions for the FIFO Control Register

```

```

// (16650 only)
#define UART_FCR_ENABLE_FIFO 0x01 /* Enable the FIFO */
#define UART_FCR_CLEAR_RCVR 0x02 /* Clear the RCVR FIFO */
#define UART_FCR_CLEAR_XMIT 0x04 /* Clear the XMIT FIFO */
#define UART_FCR_DMA_SELECT 0x08 /* For DMA applications */
#define UART_FCR_TRIGGER_MASK 0xC0 /* Mask for the FIFO trigger range */
#define UART_FCR_TRIGGER_1 0x00 /* Mask for trigger set at 1 */
#define UART_FCR_TRIGGER_4 0x40 /* Mask for trigger set at 4 */
#define UART_FCR_TRIGGER_8 0x80 /* Mask for trigger set at 8 */
#define UART_FCR_TRIGGER_14 0xC0 /* Mask for trigger set at 14 */
/* 16650 redefinitions */
#define UART_FCR6_R_TRIGGER_8 0x00 /* Mask for receive trigger set at 1 */
#define UART_FCR6_R_TRIGGER_16 0x40 /* Mask for receive trigger set at 4 */
#define UART_FCR6_R_TRIGGER_24 0x80 /* Mask for receive trigger set at 8 */
#define UART_FCR6_R_TRIGGER_28 0xC0 /* Mask for receive trigger set at 14 */
#define UART_FCR6_T_TRIGGER_16 0x00 /* Mask for transmit trigger set at 16 */
#define UART_FCR6_T_TRIGGER_8 0x10 /* Mask for transmit trigger set at 8 */
#define UART_FCR6_T_TRIGGER_24 0x20 /* Mask for transmit trigger set at 24 */
#define UART_FCR6_T_TRIGGER_30 0x30 /* Mask for transmit trigger set at 30 */

/* These are the definitions for the Line Control Register
 * Note: if the word length is 5 bits (UART_LCR_WLEN5), then setting
 * UART_LCR_STOP will select 1.5 stop bits, not 2 stop bits.*/
#define UART_LCR_DLAB 0x80 /* Divisor latch access bit */
#define UART_LCR_SBC 0x40 /* Set break control */
#define UART_LCR_SPAR 0x20 /* Stick parity (?) */
#define UART_LCR_EPAR 0x10 /* Even parity select */
#define UART_LCR_PARITY 0x08 /* Parity Enable */
#define UART_LCR_STOP 0x04 /* Stop bits: 0=1 stop bit, 1= 2 stop bits */
#define UART_LCR_WLEN5 0x00 /* Wordlength: 5 bits */
#define UART_LCR_WLEN6 0x01 /* Wordlength: 6 bits */
#define UART_LCR_WLEN7 0x02 /* Wordlength: 7 bits */
#define UART_LCR_WLEN8 0x03 /* Wordlength: 8 bits */

// These are the definitions for the Line Status Register
#define UART_LSR_TEMT 0x40 /* Transmitter empty */
#define UART_LSR_THRE 0x20 /* Transmit-hold-register empty */
#define UART_LSR_BI 0x10 /* Break interrupt indicator */
#define UART_LSR_FE 0x08 /* Frame error indicator */
#define UART_LSR_PE 0x04 /* Parity error indicator */
#define UART_LSR_OE 0x02 /* Overrun error indicator */
#define UART_LSR_DR 0x01 /* Receiver data ready */

// These are the definitions for the Interrupt Identification Register
#define UART_IIR_NO_INT 0x01 /* No interrupts pending */
#define UART_IIR_ID 0x06 /* Mask for the interrupt ID */

#define UART_IIR_MSI 0x00 /* Modem status interrupt */
#define UART_IIR_THRI 0x02 /* Transmitter holding register empty */
#define UART_IIR_TOI 0x0c /* Receive time out interrupt */
#define UART_IIR_RDI 0x04 /* Receiver data interrupt */
#define UART_IIR_RLSI 0x06 /* Receiver line status interrupt */

// These are the definitions for the Interrupt Enable Register
#define UART_IER_MSI 0x08 /* Enable Modem status interrupt */
#define UART_IER_RLSI 0x04 /* Enable receiver line status interrupt */
#define UART_IER_THRI 0x02 /* Enable Transmitter holding register int. */
#define UART_IER_RDI 0x01 /* Enable receiver data interrupt */

// These are the definitions for the Modem Control Register
#define UART_MCR_LOOP 0x10 /* Enable loopback test mode */
#define UART_MCR_OUT2 0x08 /* Out2 complement */

```

```

#define UART_MCR_OUT1 0x04 /* Out1 complement */
#define UART_MCR_RTS 0x02 /* RTS complement */
#define UART_MCR_DTR 0x01 /* DTR complement */

// These are the definitions for the Modem Status Register
#define UART_MSR_DCD 0x80 /* Data Carrier Detect */
#define UART_MSR_RI 0x40 /* Ring Indicator */
#define UART_MSR_DSR 0x20 /* Data Set Ready */
#define UART_MSR_CTS 0x10 /* Clear to Send */
#define UART_MSR_DDCD 0x08 /* Delta DCD */
#define UART_MSR_TERI 0x04 /* Trailing edge ring indicator */
#define UART_MSR_DDSR 0x02 /* Delta DSR */
#define UART_MSR_DCTS 0x01 /* Delta CTS */
#define UART_MSR_ANY_DELTA 0x0F /* Any of the delta bits! */

// These are the definitions for the Extended Features Register
// (StarTech 16C660 only, when DLAB=1)
#define UART_EFR_CTS 0x80 /* CTS flow control */
#define UART_EFR_RTS 0x40 /* RTS flow control */
#define UART_EFR_SCD 0x20 /* Special character detect */
#define UART_EFR_ENI 0x10 /* Enhanced Interrupt */
#endif /* _UART_H_ */

```

### ***Fichero makefile***

```

ifndef CROSS_COMPILE
CROSS_COMPILE = or32-elf-
CC = $(CROSS_COMPILE)gcc
LD = $(CROSS_COMPILE)ld
NM = $(CROSS_COMPILE)nm
endif

export CROSS_COMPILE
all: hello.or32
reset.o: reset.S Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
hello.o: hello.c Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
hello.or32: reset.o hello.o Makefile
 $(LD) -Tram.ld -o $@ reset.o hello.o $(LIBS)

System.map: hello.or32
 @$ (NM) $< | \
 grep -v '\(compiled\)\|\|(\.o$$\)\|\| ([aUw]
\)\|\|(\.\.ng$$\)\|\|(LASH[RL]DI\)' | \
 sort > System.map

#####
clean:
 find . -type f \
 \(-name 'core' -o -name '*.bak' -o -name '*~' \
 -o -name '*.o' -o -name '*.a' -o -name '*.tmp' \
 -o -name '*.or32' -o -name '*.bin' -o -name '*.srec' \
 -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
 -o -name '*.aux' -o -name '*.log' \) -print \
 | xargs rm -f
rm -f System.map
distclean: clean
 find . -type f \
 \(-name .depend -o -name '*.srec' -o -name '*.bin' \
 -o -name '*.pdf' \) \

```

```

 -print | xargs rm -f
rm -f $(OBSJ) *.bak tags TAGS
rm -fr *.*~

```

### ***Fichero ram.ld***

```

MEMORY
{
 vectors : ORIGIN = 0x00000000, LENGTH = 0x00002000
 ram : ORIGIN = 0x00002000, LENGTH = 0x00002000
}

SECTIONS
{
 .vectors :
 {
 *(.vectors)
 } > vectors

 .text :
 {
 *(.text)
 } > ram

 .data :
 {
 *(.data)
 } > ram

 .rodata :
 {
 *(.rodata)
 } > ram

 .bss :
 {
 *(.bss)
 } > ram

 .stack :
 {
 *(.stack)
 _src_addr = .;
 } > ram
}

```

### 5.1.2 Anexo 2: Programa checkmem (software)

Ficheros:      checkmem.cpp  
                 reset.S  
                 board.h  
                 makefile  
                 ram.ld

#### ***Fichero checkmem.cpp***

```
#define CHECK_I 0x01020000
#define CHECK_F 0x0102000F
typedef unsigned short int uword;
typedef unsigned int uint;
typedef unsigned char uchar;

int check_memoria(void *direccion_i, void *direccion_f)
{
 uint *p32;
 uword *p16;
 uchar *p8;
 for(p32=((uint*)direccion_i); p32<=((uint*)direccion_f); p32++)
 *p32=(uint)p32;
 for(p32=((uint*)direccion_i); p32<=((uint*)direccion_f); p32++)
 if(*p32!=(uint)p32)
 return -1;
 for(p16=((uword*)direccion_i); p16<=((uword*)direccion_f); p16++)
 *p16=(uword)p16;
 for(p16=((uword*)direccion_i); p16<=((uword*)direccion_f); p16++)
 if(*p16!=(uword)p16)
 return -1;
 for(p8=((uchar*)direccion_i); p8<=((uchar*)direccion_f); p8++)
 *p8=(unsigned char)p8;
 for(p8=((uchar*)direccion_i); p8<=((uchar*)direccion_f); p8++)
 if(*p8!=(uchar)p8)
 return -1;
 return 0;
}

int main()
{
 if (check_memoria((void*)CHECK_I, (void*)CHECK_F)==-1) return -1;
 return 0;
}
```

#### ***Fichero makefile***

```
ifndef CROSS_COMPILE
CROSS_COMPILE = or32-elf-
CC = $(CROSS_COMPILE)gcc
LD = $(CROSS_COMPILE)ld
NM = $(CROSS_COMPILE)nm
endif

export CROSS_COMPILE
all: controller.or32
checkmem.o: checkmem.cpp Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
reset.o: reset.S Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
```



```

controller.or32: checkmem.o reset.o Makefile
$(LD) -Tram.ld -o $@ checkmem.o reset.o $(LIBS)

System.map: controller.or32
 @$(NM) $< | \
 grep -v '\(compiled\)\|\|\(.o$$\)\|\|([aUw]
\)\|\|\(.ng$$\)\|\|(LASH[RL]DI\)' | \
 sort > System.map
#####
clean:
 find . -type f \
 \(-name 'core' -o -name '*.bak' -o -name '*~' \
 -o -name '*.o' -o -name '*.a' -o -name '*.tmp' \
 -o -name '*.or32' -o -name '*.bin' -o -name '*.srec' \
 -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
 -o -name '*.aux' -o -name '*.log'\) -print \
 | xargs rm -f
 rm -f System.map
distclean: clean
 find . -type f \
 \(-name .depend -o -name '*.srec' -o -name '*.bin' \
 -o -name '*.pdf'\) \
 -print | xargs rm -f
 rm -f $(OBJS) *.bak tags TAGS
 rm -fr *.*~

```

### 5.1.3 Anexo 3: Programa checkcopro (software)

Ficheros:      checkcopro.cpp  
                 reset.S  
                 board.h  
                 makefile  
                 ram.ld

#### ***Fichero checkcopro.cpp***

```
#define MAIOHW_REG 0x01000000
#define FINGER_DATA 0x01080000 //0x01080000 - 0x0108ffff
typedef struct T_MAIOHW_FIELD
{
 unsigned char mode; //0=normal_follow 1=init_follow
 signed char a; //angle
 unsigned char y; //x
 unsigned char x; //y
};
typedef union T_MAIOHW_REG //estructura de datos del registro
 //(union --> .data (general)
 // -->.field.? (específica un campo)
{
 unsigned int data;
 T_MAIOHW_FIELD field;
};

T_MAIOHW_REG *p_copro_reg; //puntero al registro mapeado en memoria externa
unsigned char *imatge; //puntero a huella mapeada en memoria externa
T_MAIOHW_REG check_copro(unsigned int data)
{
 p_copro_reg=(T_MAIOHW_REG*)MAIOHW_REG;
 p_copro_reg->data=data;
 while(p_copro_reg->data==--1);
 return *p_copro_reg;
}

int main ()
{
 T_MAIOHW_REG registro;
 registro.field.mode=0;
 registro.field.a=1;
 registro.field.y=100;
 registro.field.x=200;
 check_copro(registro.data);
 return 0;
}
```

#### 5.1.4 Anexo 4: Programa Maio-Maltoni (software)

Ficheros:        actualitza.cpp  
                 angle.cpp  
                 estraccio.cpp  
                 followridge.cpp  
                 funcions.cpp  
                 next\_point.cpp  
                 uart.cpp  
                 updateT.cpp  
                 reset.S  
                 board.h  
                 maio.h  
                 maioHw.h  
                 uart.h  
                 makefile  
                 ram.ld

##### *Fichero actualitza.cpp*

```
#include "maio.h"

void actualitza(int i_f, int j_f, int angle, int tipus)
{
 int proximitat = proximitat2;
 int i, afegir;
 afegir = 1;
 if (NumMinuties>=99) return;
 for (i=0; i<NumMinuties; i++)
 {
 // mirem si hi ha alguna altra minutia propera
 if ((M[i][0] >= i_f - proximitat)&&(M[i][0] <= i_f + proximitat))
 {
 if ((M[i][1] >= j_f - proximitat)&&(M[i][1] <= j_f + proximitat))
 {
 // si la minutia propera no és una bifurcació la filtrem.
 if (M[i][3]!=2)
 {
 UpdateT(M[i][1],M[i][0],j_f,i_f);
 // per a filtrar eliminen una fila de M desplaçant la resta cap amunt
 //memmove(&M[i][0], &M[i+1][0],sizeof(int)*4*(NumMinuties-i));
 for (int fila=i;fila<NumMinuties;fila++)
 {
 M[fila][0]=M[fila+1][0];
 M[fila][1]=M[fila+1][1];
 M[fila][2]=M[fila+1][2];
 M[fila][3]=M[fila+1][3];
 }
 NumMinuties--;
 afegir = 0;
 }
 }
 }
 }
 if (afegir)
 {
 // fem una inserció ordenada
 i=0;
 while ((M[i][0]<i_f)&&(i<NumMinuties)) i++;
 if (M[i][0]==j_f) // hem de mirar per columnes
 while (M[i][i]<j_f) i++;
 }
}
```

```

// fem espai abans d'afegir
//memmove(&M[i+1][0], &M[i][0],sizeof(int)*4*(NumMinuties-i));
for (int fila=NumMinuties;fila>=i;fila--)
{
 M[fila+1][0]=M[fila][0];
 M[fila+1][1]=M[fila][1];
 M[fila+1][2]=M[fila][2];
 M[fila+1][3]=M[fila][3];
}
M[i][0]=i_f; M[i][1]=j_f; M[i][2]=angle; M[i][3]=tipus;
NumMinuties++;
}
}

```

### ***Fichero angle.cpp***

```

#include "maio.h"

int rect2polar(int j, int i)
{
 int w;
 int y = abs(j);
 int x = abs(i);
 if (j==0) w=0;
 else if (i==0) w=6;
 else {
 int mesde45=0;
 if (x==y) w=3;
 else {
 if (y>x) { mesde45=1; int aux=y; y=x; x=aux; }
 // ja sols cal triar entre 15 o 30 graus
 switch (x)
 {
 case 2:
 w=2;
 break;
 case 3:
 if (y==1) w=1;
 else w=2;
 break;
 case 4:
 if (y<2) w=1;
 else if (y<3) w=2;
 else w=3;
 break;
 default:
 if (y<2) w=1;
 else if (y<4) w=2;
 else w=3;
 break;
 }
 if (mesde45) w=6-w;
 }
 }
 if (i<0) w=12-w;
 if (j<0) w=-w;
 return w;
}

bool angleerror(int dir, int previ, int j, int i)
{
 int w = rect2polar(j, i);
}

```

```

int dif = abs(w-dir);
if (dif>12) dif = 24 -dif;
if (dif>beta) return true;
dif = abs(w-previ);
if (dif>12) dif = 24 -dif;
if (dif>psi)
 return true;
return false;
}

```

### ***Fichero estraccio.cpp***

```

#include "maio.h"
#include "maioHw.h"
#include "uart.h"
#include <cstdio>

T_MAIOHW_REG *p_copro_reg;

unsigned short posicio;
char *hi, *lo;
int proximitat,proximitat2,margeperbifurcacio,marge,j_c,i_c,xmax,ymax;
int iter,follows;

unsigned char copy_imatge[256*256] __attribute__
((section(".fingerprint"))) = {...};
bool T[256][256] __attribute__ ((section(".Tmatrix"))) = {...};

int NumMinuties,M[100][4];
int beta,psi,psi_start,psi_c,psi_n;
bool *test;
int sigma, nu, margeinicial, numcols,numfiles,mu,tgw;

bool Estraccio()
{
 InitVars();
 for (int j_start=margeinicial; j_start<=numcols-margeinicial; j_start+=nu)
 for (int i_start=margeinicial; i_start<=numfiles-margeinicial; i_start+=nu)
 {
 follows++;
 followridge(j_start,i_start);
 }
 return true;
}

int main()
{
 char *s;
 imatge=(unsigned char*)FINGER_DATA;
 p_copro_reg=(T_MAIOHW_REG*)MAIOHW_REG;
 uart_init ();
 Estraccio();
 /* char st_iter[50];
 sprintf (st_iter,"\n#Copro_iters = %d",iter);
 for (s = st_iter; *s; s++)
 uart_putc (*s);
 char st_follows[50];
 sprintf (st_follows,"\n#Follow_iters = %d",follows);
 for (s = st_follows; *s; s++)
 uart_putc (*s);
 */
}

```

### ***Fichero followridge.cpp***

```
#include "maio.h"
#include "maioHw.h"
#include "uart.h"

extern int proximitat,proximitat2,margeperbifurcacio,marge,j_c,i_c,xmax,ymax;
extern int beta,psi,psi_start,psi_c,psi_n;
extern bool T[256][256], *test;
extern int sigma, nu, margeinicial, numcols,numfiles,mu,tgw;
char *str2 = ".";
extern int iter;

void followridge(int j_start, int i_start)
{
 //segueix el crestall fins trobar final
 //afegeix la minutia trobada a la matriu M
 //T es la imatge auxiliar per a anar marcant el seguiment de la cresta
 //i poder completar la deteccio de tota la minutia
 T_MAIOHW_REG maiohw_reg;
 int seccioX[15],seccioY[15],previ;
 int h1[15];
 int continua,passades;
 int delta_i,delta_j,i_n,j_n,index,afegir,bifurca,tipus;
 int i_final,j_final,angle;
 int address, start;

 if (imatge[ndx(j_start,i_start)]<135) return;
 address=((j_start-proximitat)*numcols)+(i_start-proximitat);
 start=address;
 for (int k=proximitat;k<=proximitat;k++)
 {
 for (int m=proximitat;m<=proximitat;m++)
 {
 if (checkT(address)) return;
 address++;
 }
 start+=numcols;
 address=start;
 }
 continua = 1; passades=1;
 maiohw_reg.field.mode=1;
 maiohw_reg.field.a=0;
 maiohw_reg.field.y=j_start;
 maiohw_reg.field.x=i_start;
 p_copro_reg->data=maiohw_reg.data;
 while(p_copro_reg->data!=-1);
 maiohw_reg.data=p_copro_reg->data;
 iter++;
 i_c=maiohw_reg.field.x;
 j_c=maiohw_reg.field.y;
 psi_start=maiohw_reg.field.a;

 address=((j_c-proximitat2)*numcols)+(i_c-proximitat2);
 start=address;
 for (int k=-proximitat2;k<=proximitat2;k++)
 {
 for (int m=-proximitat2;m<=proximitat2;m++)
 {
 if (checkT(address)) return;
 address++;
 }
 }
}
```

```

 start+=numcols;
 address=start;
 }
 maiohw_reg.field.mode=3;
 maiohw_reg.field.a=0;
 maiohw_reg.field.y=j_c;
 maiohw_reg.field.x=i_c;
 p_copro_reg->data=maiohw_reg.data;
 while(p_copro_reg->data!=-1);
 maiohw_reg.data=p_copro_reg->data;
 iter++;
 psi_c=maiohw_reg.field.a;
 i_start=i_c; j_start=j_c; psi_start=psi_c; previ=psi_c; psi_n=psi_c;
 while (continua)
 {
 maiohw_reg.field.mode=0;
 maiohw_reg.field.a=psi_c;
 maiohw_reg.field.y=j_c;
 maiohw_reg.field.x=i_c;
 p_copro_reg->data=maiohw_reg.data;
 while(p_copro_reg->data!=-1);
 maiohw_reg.data=p_copro_reg->data;
 iter++;
 i_n=maiohw_reg.field.x;
 j_n=maiohw_reg.field.y;
 psi_n=maiohw_reg.field.a;
 afegir=0;

// FINAL DE SEGUIMENT (opcions)
// CAS 1: sortim de l'area de la imatge
// modifiquem per a mirar bifurcacio abans que final de cresta
 if ((i_n>xmax)|(i_n<marge)|(j_n>ymax)|(j_n<marge))
 {continua=0; bifurca=1; tipus=0;}
// CAS 3: bifurcacio, l'angle hauria de ser el de la vall (recalcular)
// si no ens hem allunyat gaire del punt anterior podriem detectar
// una bifurcacio erronia amb el mateix crestall
 else
 {
 if ((abs(j_c-j_n)>2)|(abs(i_n-i_c)>2))
 for (int k=-margeperbifurcacio;k<=margeperbifurcacio; k++)
 for (int m=-margeperbifurcacio;m<=margeperbifurcacio;m++)
 // en cas de bifurcacio si que actualitzem T
 if (checkT(j_n+k,i_n+m))
 {
 continua=0; i_final=i_n; j_final=j_n;
 angle=psi_n;bifurca=1; afegir=1; tipus=2;}
 }
// CAS 2: final de cresta, igual que excessive binding - hauria de ser
// lleugerament diferent
 if ((continua)& angleerror(psi_c,previ,j_c-j_n,i_n-i_c))
 {
 continua=0; i_final=i_c; j_final=j_c; angle=psi_c;bifurca=0;
 afegir=1; tipus=1;
 }
 if ((continua==1)|((continua==0)&(bifurca==1)))
 //actualitzar la imatge auxiliar i representacio opcional
 UpdateT(j_c,i_c,j_n,i_n);
 //sols actualitzem si continuem, sino la linea efegida es erronia
 //en cas de bifurcacio si que s'ha d'afegir

 if (afegir)

```

```

 //no considerem els punts amb mitges baixes i desviacions elevades
 if (contrast(j_final,i_final) > 35*35)
 //rutina per a actualitzar la minutia
 actualitza(i_final,j_final,angle,tipus);
 if (continua==0)
 {
 if (passades==1)
 {
 continua=1; passades=2; i_c=i_start; j_c=j_start;
 if (psi_start>0) psi_c=psi_start-12;
 else psi_c=psi_start+12;
 previ=psi_c;
 }
 }
 else {previ=psi_c; i_c=i_n; j_c= j_n; psi_c=psi_n;}
}

```

### ***Fichero funciones.cpp***

```

#include "maio.h"
//#include <string.h>
//#include <stdlib.h>

void InitVars()
{
 numfiles=256;
 numcols=256;
 NumMinuties=0;
 lo = (char *)&posicio;
 hi = lo +1;
 nu=2;
 margeinicial=30;
 sigma=SIGMA;
 mu=4;
 beta=3;
 psi=3;
 tgw=4;
 proximitat=3;
 proximitat2=2;
 margeperbifurcacio=1;
 marge=15;
 for (int i=0;i<256;i++)
 for(int j=0;j<256;j++)
 T[i][j]=false;
 test = &T[0][0];
 ymax=numfiles-marge; //rows
 xmax=numcols-marge;
 iter=0;
 follows=0;
}

bool checkT(int fila, int columna)
{
 return T[fila][columna];
}

bool checkT(int index)
{
 return test[index];
}

```



```

int contrast(int fila, int columna)
{
 int numitems = 0;
 int suma1 = 0, suma2 = 0;
 for (int i=fila-5; i<=fila+5; i++)
 {
 for (int j=columna-5; j<=columna+5; j++)
 {
 numitems++;
 suma1 += imatge[ndx(i,j)];
 suma2 += imatge[ndx(i,j)]*imatge[ndx(i,j)];
 }
 }
 int f_mean = suma1 / (numitems);
 int f_std = (((suma2 - (suma1*suma1 / numitems)) / numitems));
 return (int)((f_mean*f_mean)-f_std);
}

```

### ***Fichero next\_point.cpp***

```

#include "maio.h"

void next_point(int w, int increment, int *deltaX, int *deltaY)
{
 static int delta_x[8][7]={
 1,1,1,1,1,0,0,
 2,2,2,2,1,1,0,
 3,3,3,3,1,1,0,
 4,4,3,4,2,1,0,
 5,5,4,5,3,1,0,
 6,6,5,6,3,2,0,
 7,7,6,7,4,2,0,
 8,8,7,8,4,2,0
 };
 int angle=abs(w);
 increment--;
 if (angle>6)
 {
 *deltaX = -delta_x[increment][12-angle];
 *deltaY = delta_x[increment][angle-6];
 }
 else
 {
 *deltaX = delta_x[increment][angle];
 *deltaY = delta_x[increment][6-angle];
 }
 if (w>0) *deltaY = -*deltaY;
}

```

### ***Fichero updateT.cpp***

```

#include "maio.h"

extern int proximitat,proximitat2,margeperbifurcacio,marge,j_c,i_c,xmax,ymax;
extern int beta,psi,psi_start,psi_c,psi_n;
extern bool T[256][256], *test;
extern int sigma, nu, margeinicial, numcols,numfiles,mu,tgw;

void UpdateT(int j_c, int i_c, int j_n, int i_n)
{
 int j,nouX,nouY;
 if ((j_c==j_n)&&(i_n==i_c)) return;
 int Xdif=i_n-i_c;

```

```

int Ydif=j_c-j_n;
int angle = rect2polar(Ydif,Xdif);
Xdif= abs(Xdif); Ydif=abs(Ydif);
j=1; nouX=i_c; nouY=j_c;
while (true)
{
 for (int m=-1;m<=1;m++)
 for (int n=-1;n<=1;n++)
 T[nouY+m][nouX+n]=true;
 next_point(angle,j,&nouX, &nouY);
 if ((abs(nouX)>Xdif)|| (abs(nouY)>Ydif)) break;
 nouY+=j_c; nouX+=i_c;
 j=j+1;
}
}

```

### ***Fichero maio.h***

```

#ifndef maioH
#define maioH
#define pi 3.141592654
#define SIGMA 7
#define ndx(y,x) ((x)|((y)<<8))
#define abs(n) __builtin_abs ((n))

extern unsigned short posicio;
extern char *hi, *lo;
extern int proximitat,proximitat2,margeperbifurcacio,marge,j_c,i_c,xmax,ymax;
extern int M[100][4], NumMinuties;
extern int beta,psi,psi_start,psi_c,psi_n;
extern bool T[256][256] __attribute__((section(".Tmatrix"))), *test;
extern int sigma, nu, margeinicial, numcols,numfiles,mu,tgw;
extern unsigned char *imatge;
extern int iter, follows;

bool Estraccio();
void actualitza(int,int,int,int);
bool angleerror(int,int,int,int);
bool checkT(int,int);
bool checkT(int);
int contrast(int, int);
void followridge(int, int);
void InitVars();
int rect2polar(int, int);
void UpdateT(int,int,int,int);
void next_point(int, int, int *, int *);
void uart_init(void);
void uart_putc(char);
char uart_getc(void);
#endif

```

### ***Fichero maioHw.h***

```

#ifndef maioHwH
#define maioHwH
#define MAIOHW_REG 0x01000000
#define FINGER_DATA 0x010F0000

typedef struct T_MAIOHW_FIELD
{
 unsigned char mode; //0=normal_follow 1=init_follow
 signed char a; //angle
 unsigned char y; //x

```

```

 unsigned char x; //y
 };
typedef union T_MAIOHW_REG
{
 unsigned int data;
 T_MAIOHW_FIELD field;
};
extern T_MAIOHW_REG *p_copro_reg;
#endif

```

### ***Fichero makefile***

```

ifndef CROSS_COMPILE
CROSS_COMPILE = or32-elf-
CC = $(CROSS_COMPILE)gcc
LD = $(CROSS_COMPILE)ld
NM = $(CROSS_COMPILE)nm
endif

export CROSS_COMPILE
all: copro.or32
reset.o: reset.S Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
actualitza.o: actualitza.cpp Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
angle.o: angle.cpp Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
estraccio.o: estraccio.cpp Makefile
 $(CC) -I /opt/or32-uclinux/include -g -c -o $@ $< $(CFLAGS)
followridge.o: followridge.cpp Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
funcions.o: funciones.cpp Makefile
 $(CC) -msoft-div -g -c -o $@ $< $(CFLAGS)
next_point.o: next_point.cpp Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
UpdateT.o: UpdateT.cpp Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)
uart.o: uart.cpp Makefile
 $(CC) -g -c -o $@ $< $(CFLAGS)

copro.or32: reset.o actualitza.o angle.o estraccio.o followridge.o
funcions.o next_point.o UpdateT.o uart.o Makefile
 $(LD) -Tram.ld reset.o actualitza.o angle.o estraccio.o
followridge.o funciones.o next_point.o UpdateT.o uart.o -L /opt/or32-
elf/lib -L /opt/or32-elf/lib/gcc/or32-elf/3.4.4 -lc -lgcc -o $@ $(LIBS)

System.map: copro.or32
 @$ (NM) $< | \
 grep -v '\(compiled\)\|\|(\.o$$)\|\|([aUw]
\)\|\|(\.\.ng$$)\|\|(LASH[RL]DI)\)' | \
 sort > System.map

clean:
 find . -type f \
 \(-name 'core' -o -name '*.bak' -o -name '*~' \
 -o -name '*.o' -o -name '*.a' -o -name '*.tmp' \
 -o -name '*.or32' -o -name '*.bin' -o -name '*.srec' \
 -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
 -o -name '*.aux' -o -name '*.log' \) -print \
 | xargs rm -f
 rm -f System.map
distclean: clean

```

```

find . -type f \
 \(-name .depend -o -name '*.srec' -o -name '*.bin' \
 -o -name '*.pdf' \) \
 -print | xargs rm -f
rm -f $(OBJDUMP -t | grep -E '\.bak$' | cut -d: -f1)
rm -fr *.~

```

### ***Fichero ram.ld***

#### MEMORY

```

{
 vectors : ORIGIN = 0x00000000, LENGTH = 0x00001000
 stack : ORIGIN = 0x00001000, LENGTH = 0x00001000
 ram : ORIGIN = 0x00002000, LENGTH = 0x00005FFF
 Tmatrix : ORIGIN = 0x010E0000, LENGTH = 0x00010000
 fingerprint : ORIGIN = 0x010F0000, LENGTH = 0x00010000
}

```

#### SECTIONS

```

{
 .Tmatrix :
 {
 *(.Tmatrix)
 } > Tmatrix

 .fingerprint :
 {
 *(.fingerprint)
 } > fingerprint

 .vectors :
 {
 *(.vectors)
 } > vectors

 .text :
 {
 *(.text)
 } > ram

 .data :
 {
 *(.data)
 } > ram

 .rodata :
 {
 *(.rodata)
 } > ram

 .rodata.str1.1 :
 {
 *(.rodata.str1.1)
 } > ram

 .bss :
 {
 *(.bss)
 } > ram

 .stack :
 {

```

```

 *(.stack)
 _src_addr = .;
 } > stack
}

```

## 5.2 Anexos de Hardware

El código contenido en este anexo del trabajo corresponde con el código *HDL* (*verilog* o *vhdl*) de los ficheros del diseño considerados relevantes para el propósito del proyecto. Todos los ficheros pertenecientes al proyecto *ORP\_SoC* de *OpenCores* contienen la declaración de la licencia *GNU LGPL* mostrada a continuación:

```

///
//// ////
//// Copyright (C) 2001 Authors ////
//// ////
//// This source file may be used and distributed without ////
//// restriction provided that this copyright statement is not ////
//// removed from the file and that any derivative work contains ////
//// the original copyright notice and the associated disclaimer. ////
//// ////
//// This source file is free software; you can redistribute it ////
//// and/or modify it under the terms of the GNU Lesser General ////
//// Public License as published by the Free Software Foundation; ////
//// either version 2.1 of the License, or (at your option) any ////
//// later version. ////
//// ////
//// This source is distributed in the hope that it will be ////
//// useful, but WITHOUT ANY WARRANTY; without even the implied ////
//// warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR ////
//// PURPOSE. See the GNU Lesser General Public License for more ////
//// details. ////
//// ////
//// You should have received a copy of the GNU Lesser General ////
//// Public License along with this source; if not, download it ////
//// from http://www.opencores.org/lgpl.shtml ////
//// ////
///

```

### 5.2.1 Anexo 5: Jerarquía superior del diseño

Ficheros: xsv\_fpga\_top.v  
xsv\_fpga\_defines.v

#### ***Fichero xsv\_fpga\_top.v***

```

`include "xsv_fpga_defines.v"
`include "bench_defines.v"
`define SRAM_EXT_AW 19

module xsv_fpga_top (
 clk, rstn,
 uart_stx, uart_srx,
 sram_Ncs, sram_Noce, sram_Nwe, sram_Nbe, sram_addr, sram_data,
 flash_Ncs,
 jtag_tck, jtag_tms, jtag_tdi, jtag_trst,
 jtag_tvref, jtag_tgnd, jtag_tdo,
 gpio
);

///
// I/O Ports

```

```

////////////////////////////////////

// Global
input clk;
input rstn;

// UART
input uart_srx;
output uart_stx;

// external SRAM & FLASH (enables de las memorias externas)
output sram_Ncs, sram_Noe, sram_Nwe;
output [3:0] sram_Nbe;
output [`SRAM_EXT_AW+1:2] sram_addr;
inout [31:0] sram_data;
output flash_Ncs;

// JTAG
input jtag_tck, jtag_tms, jtag_tdi, jtag_trst;
output jtag_tvref, jtag_tgnd, jtag_tdo;

//GPIO
inout [7:0] gpio;

////////////////////////////////////
// Internal wires
////////////////////////////////////

// Debug core master i/f wires
(wb_dm_*=WishBone_DebugMaster_*)
wire [31:0] wb_dm_adr_o;
wire [31:0] wb_dm_dat_i;
wire [31:0] wb_dm_dat_o;
wire [3:0] wb_dm_sel_o;
wire wb_dm_we_o;
wire wb_dm_stb_o;
wire wb_dm_cyc_o;
wire wb_dm_cab_o;
wire wb_dm_ack_i;
wire wb_dm_err_i;

// Debug <-> RISC wires
wire [3:0] dbg_lss;
wire [1:0] dbg_is;
wire [10:0] dbg_wp;
wire dbg_bp;
wire [31:0] dbg_dat_dbg;
wire [31:0] dbg_dat_risc;
wire [31:0] dbg_adr;
wire dbg_ewt;
wire dbg_stall;
wire [2:0] dbg_op;

// RISC instruction master i/f wires
(wb_rim_*=WishBone_RiscInstructionMaster_*)
wire [31:0] wb_rim_adr_o;
wire wb_rim_cyc_o;
wire [31:0] wb_rim_dat_i;
wire [31:0] wb_rim_dat_o;
wire [3:0] wb_rim_sel_o;
wire wb_rim_ack_i;

```

```

wire wb_rim_err_i;
wire wb_rim_rty_i = 1'b0;
wire wb_rim_we_o;
wire wb_rim_stb_o;
wire wb_rim_cab_o;
wire [31:0] wb_rif_adr;
reg prefix_flash;

// RISC data master i/f wires
(wb_rdm_*=WishBone_RiscDataMaster-*)
wire [31:0] wb_rdm_adr_o;
wire wb_rdm_cyc_o;
wire [31:0] wb_rdm_dat_i;
wire [31:0] wb_rdm_dat_o;
wire [3:0] wb_rdm_sel_o;
wire wb_rdm_ack_i;
wire wb_rdm_err_i;
wire wb_rdm_rty_i = 1'b0;
wire wb_rdm_we_o;
wire wb_rdm_stb_o;
wire wb_rdm_cab_o;
wire wb_rdm_ack;

// RISC PIC (entrada al Peripheral Interrupt Controller)
wire [19:0] pic_ints;

// onchip-SRAM controller slave i/f wires
wire [31:0] wb_ss_dat_i;
wire [31:0] wb_ss_dat_o;
wire [31:0] wb_ss_adr_i;
wire [3:0] wb_ss_sel_i;
wire wb_ss_we_i;
wire wb_ss_cyc_i;
wire wb_ss_stb_i;
wire wb_ss_ack_o;
wire wb_ss_err_o;

// Copro - slave i/f wires
wire [31:0] wb_cop_dat_i;
wire [31:0] wb_cop_dat_o;
wire [31:0] wb_cop_adr_i;
wire [3:0] wb_cop_sel_i;
wire wb_cop_we_i;
wire wb_cop_cyc_i;
wire wb_cop_stb_i;
wire wb_cop_ack_o;
wire wb_cop_err_o;

// UART16550 core slave i/f wires
(wb_us_*=WishBone_UartSlave-*)
wire [31:0] wb_us_dat_i;
wire [31:0] wb_us_dat_o;
wire [31:0] wb_us_adr_i;
wire [3:0] wb_us_sel_i;
wire wb_us_we_i;
wire wb_us_cyc_i;
wire wb_us_stb_i;
wire wb_us_ack_o;
wire wb_us_err_o;

```

```

// UART external i/f wires
wire uart_stx;
wire uart_srx;

// GPIO core slave i/f wires
wire [31:0] wb_gp_dat_i;
wire [31:0] wb_gp_dat_o;
wire [31:0] wb_gp_adr_i;
wire [3:0] wb_gp_sel_i;
wire wb_gp_we_i;
wire wb_gp_cyc_i;
wire wb_gp_stb_i;
wire wb_gp_ack_o;
wire wb_gp_err_o;
// GPIO external i/f wires
wire [7:0] gpio;

// external-SRAM & FLASH
wire sram_Ncs, sram_Noe, sram_Nwe;
wire [3:0] sram_Nbe;
wire [`SRAM_EXT_AW+1:2] sram_addr;
wire [31:0] sram_data;
wire [31:0] sram_data_I;
wire [31:0] sram_data_O;
wire sram_data_T;
wire flash_Ncs;

// JTAG wires
wire jtag_tdi;
wire jtag_tms;
wire jtag_tck;
wire jtag_trst;
wire jtag_tdo;

// Reset debounce
reg rst_r;
reg wb_rst;

// Global clock
wire wb_clk;

////////////////////////////////////
// Architecture
////////////////////////////////////

// Reset debounce
always @(posedge wb_clk or posedge rstn)
 if (rstn)
 rst_r <= 1'b1;
 else
 rst_r <= #1 1'b0;

always @(posedge wb_clk)
 wb_rst <= #1 rst_r;

// Wishbone clk
//assign wb_clk = clk; // substituido para utilizar DCM

// Unused WISHBONE signals
assign wb_us_err_o = 1'b0;

```



```

// OR1200 PIC
assign pic_ints[`APP_INT_RES1] = 'b0;
assign pic_ints[`APP_INT_RES2] = 'b0;

// External SRAM & FLASH
assign flash_Ncs = 1'b1;

// JTAG
assign jtag_tvref = 1'b1;
assign jtag_tgnd = 1'b0;

//
// Instantation del DCM
//
wb_dcm wb_dcm (
 .CLKIN_IN (clk),
 .CLKDV_OUT (wb_clk), //30MHz
 .CLKIN_IBUFG_OUT (),
 .CLK0_OUT()
);

//
// Instantiation of the development i/f model
//
// Used only for simulations.
//
`ifdef DBG_IF_MODEL
dbg_if_model dbg_if_model (

 // JTAG pins
 .tms_pad_i (jtag_tms),
 .tck_pad_i (jtag_tck),
 .trst_pad_i (jtag_trst),
 .tdi_pad_i (jtag_tdi),
 .tdo_pad_o (jtag_tdo),

 // Boundary Scan signals
 .capture_dr_o (),
 .shift_dr_o (),
 .update_dr_o (),
 .extest_selected_o (),
 .bs_chain_i (1'b0),

 // RISC signals
 .risc_clk_i (wb_clk),
 .risc_data_i (dbg_dat_risc),
 .risc_data_o (dbg_dat_dbg),
 .risc_addr_o (dbg_adr),
 .wp_i (dbg_wp),
 .bp_i (dbg_bp),
 .opselect_o (dbg_op),
 .lsstatus_i (dbg_lss),
 .istatus_i (dbg_is),
 .risc_stall_o (dbg_stall),
 .reset_o (),

 // WISHBONE common
 .wb_clk_i (wb_clk),
 .wb_rst_i (wb_rst),

```

```

 // WISHBONE master interface
 .wb_adr_o (wb_dm_adr_o),
 .wb_dat_i (wb_dm_dat_i),
 .wb_dat_o (wb_dm_dat_o),
 .wb_sel_o (wb_dm_sel_o),
 .wb_we_o (wb_dm_we_o),
 .wb_stb_o (wb_dm_stb_o),
 .wb_cyc_o (wb_dm_cyc_o),
 .wb_cab_o (wb_dm_cab_o),
 .wb_ack_i (wb_dm_ack_i),
 .wb_err_i (wb_dm_err_i)
);
`else
//
// Instantiation of the development i/f
//
dbg_top dbg_top (

 // JTAG pins
 .tms_pad_i (jtag_tms),
 .tck_pad_i (jtag_tck),
 .trst_pad_i (jtag_trst),
 .tdi_pad_i (jtag_tdi),
 .tdo_pad_o (jtag_tdo),
 .tdo_padoen_o (),

 // Boundary Scan signals
 .capture_dr_o (),
 .shift_dr_o (),
 .update_dr_o (),
 .extest_selected_o (),
 .bs_chain_i (1'b0),
 .bs_chain_o (),

 // RISC signals
 .risc_clk_i (wb_clk),
 .risc_addr_o (dbg_adr),
 .risc_data_i (dbg_dat_risc),
 .risc_data_o (dbg_dat_dbg),
 .wp_i (dbg_wp),
 .bp_i (dbg_bp),
 .opselect_o (dbg_op),
 .lsstatus_i (dbg_lss),
 .istatus_i (dbg_is),
 .risc_stall_o (dbg_stall),
 .reset_o (),

 // WISHBONE common
 .wb_clk_i (wb_clk),
 .wb_rst_i (wb_rst),

 // WISHBONE master interface
 .wb_adr_o (wb_dm_adr_o),
 .wb_dat_i (wb_dm_dat_i),
 .wb_dat_o (wb_dm_dat_o),
 .wb_sel_o (wb_dm_sel_o),
 .wb_we_o (wb_dm_we_o),
 .wb_stb_o (wb_dm_stb_o),
 .wb_cyc_o (wb_dm_cyc_o),
 .wb_cab_o (wb_dm_cab_o),
 .wb_ack_i (wb_dm_ack_i),

```

```

 .wb_err_i (wb_dm_err_i)
);
`endif

// Instantiation of the OR1200 RISC
orl200_top orl200_top (

 // Common
 .rst_i (wb_rst),
 .clk_i (wb_clk),
 .clmode_i (2'b00), //reloj 1:1???

 // WISHBONE Instruction Master
 .iwb_clk_i (wb_clk),
 .iwb_rst_i (wb_rst),
 .iwb_cyc_o (wb_rim_cyc_o),
 .iwb_adr_o (wb_rim_adr_o),
 .iwb_dat_i (wb_rim_dat_i),
 .iwb_dat_o (wb_rim_dat_o),
 .iwb_sel_o (wb_rim_sel_o),
 .iwb_ack_i (wb_rim_ack_i),
 .iwb_err_i (wb_rim_err_i),
 .iwb_rty_i (wb_rim_rty_i),
 .iwb_we_o (wb_rim_we_o),
 .iwb_stb_o (wb_rim_stb_o),
 .iwb_cab_o (wb_rim_cab_o),

 // WISHBONE Data Master
 .dwb_clk_i (wb_clk),
 .dwb_rst_i (wb_rst),
 .dwb_cyc_o (wb_rdm_cyc_o),
 .dwb_adr_o (wb_rdm_adr_o),
 .dwb_dat_i (wb_rdm_dat_i),
 .dwb_dat_o (wb_rdm_dat_o),
 .dwb_sel_o (wb_rdm_sel_o),
 .dwb_ack_i (wb_rdm_ack_i),
 .dwb_err_i (wb_rdm_err_i),
 .dwb_rty_i (wb_rdm_rty_i),
 .dwb_we_o (wb_rdm_we_o),
 .dwb_stb_o (wb_rdm_stb_o),
 .dwb_cab_o (wb_rdm_cab_o),

 // Debug
 .dbg_stall_i (dbg_stall),
 .dbg_dat_i (dbg_dat_dbg),
 .dbg_adr_i (dbg_adr),
 .dbg_ewt_i (1'b0),
 .dbg_lss_o (dbg_lss),
 .dbg_is_o (dbg_is),
 .dbg_wp_o (dbg_wp),
 .dbg_bp_o (dbg_bp),
 .dbg_dat_o (dbg_dat_risc),
 .dbg_ack_o (),
 .dbg_stb_i (dbg_op[2]),
 .dbg_we_i (dbg_op[0]),

 // Power Management
 .pm_clk_sd_o (),
 .pm_cpustall_i (1'b0),
 .pm_dc_gate_o (),
 .pm_ic_gate_o (),

```

```

 .pm_dmmu_gate_o (),
 .pm_immu_gate_o (),
 .pm_tt_gate_o (),
 .pm_cpu_gate_o (),
 .pm_wakeup_o (),
 .pm_lvolt_o (),

 // Interrupts
 .pic_ints_i (20'b0)
);

 // Instancia del controlador de memoria RAM interna
 wb_onchip_4sram8 onchip_ram_top(
 // // WISHBONE common
 .wb_clk_i (wb_clk),
 .wb_rst_i (wb_rst),

 // WISHBONE slave
 .wb_dat_i (wb_ss_dat_i),
 .wb_dat_o (wb_ss_dat_o),
 .wb_adr_i (wb_ss_adr_i),
 .wb_sel_i (wb_ss_sel_i),
 .wb_we_i (wb_ss_we_i),
 .wb_cyc_i (wb_ss_cyc_i),
 .wb_stb_i (wb_ss_stb_i),
 .wb_ack_o (wb_ss_ack_o),
 .wb_err_o (wb_ss_err_o)
);

 // Instancia del coprocesador (puente wb-->ip)
 user_maioHw coprocesador (
 .wb_clk_i (wb_clk),
 .wb_rst_i (wb_rst),
 .wb_dat_i (wb_cop_dat_i),
 .wb_dat_o (wb_cop_dat_o),
 .wb_adr_i (wb_cop_adr_i),
 .wb_sel_i (wb_cop_sel_i),
 .wb_we_i (wb_cop_we_i),
 .wb_cyc_i (wb_cop_cyc_i),
 .wb_stb_i (wb_cop_stb_i),
 .wb_ack_o (wb_cop_ack_o),
 .wb_err_o (wb_cop_err_o),
 .sram32_Ncs (sram_Ncs),
 .sram32_NoE (sram_NoE),
 .sram32_Nwe (sram_Nwe),
 .sram32_Nbe (sram_Nbe),
 .sram32_addr (sram_addr),
 .sram32_data_I(sram_data_I),
 .sram32_data_O(sram_data_O),
 .sram32_data_T(sram_data_T)
);

 // Instancia del buffer tri-state
 // conecta el bus de datos bidireccional de la memoria externa con las
 // señales del controlador (I+O+T)
 bidirec bidirec (
 .data_T (sram_data_T),
 .data_I (sram_data_I),
 .data_O (sram_data_O),
 .data (sram_data)
);

```

```

);

// Instancia del GPIO
simple_gpio simple_gpio (
 .clk_i (wb_clk),
 .rst_i (wb_rst),
 .cyc_i (wb_gp_cyc_i),
 .stb_i (wb_gp_stb_i),
 .adr_i (wb_gp_adr_i[0]),
 .we_i (wb_gp_we_i),
 .dat_i (wb_gp_dat_i[7:0]),
 .dat_o (wb_gp_dat_o[7:0]),
 .ack_o (wb_gp_ack_o),
 .gpio (gpio)
);

// Instantiation of the UART16550
uart_top uart_top (
 // WISHBONE common
 .wb_clk_i (wb_clk),
 .wb_rst_i (wb_rst),

 // WISHBONE slave
 .wb_adr_i (wb_us_adr_i[4:0]),
 .wb_dat_i (wb_us_dat_i),
 .wb_dat_o (wb_us_dat_o),
 .wb_we_i (wb_us_we_i),
 .wb_stb_i (wb_us_stb_i),
 .wb_cyc_i (wb_us_cyc_i),
 .wb_ack_o (wb_us_ack_o),
 .wb_sel_i (wb_us_sel_i),

 // Interrupt request
 .int_o (pic_ints[`APP_INT_UART]),

 // UART signals
 // serial input/output
 .stx_pad_o (uart_stx),
 .srx_pad_i (uart_srx),

 // modem signals
 .rts_pad_o (),
 .cts_pad_i (1'b0),
 .dtr_pad_o (),
 .dsr_pad_i (1'b0),
 .ri_pad_i (1'b0),
 .dcd_pad_i (1'b0)
);

// Instantiation of the Traffic COP
tc_top #(`APP_ADDR_DEC_W,
 `APP_ADDR_SRAM,
 `APP_ADDR_DEC_W,
 `APP_ADDR_RAMEXT,
 `APP_ADDR_DECP_W,
 `APP_ADDR_PERIP,
 `APP_ADDR_DEC_W,
 `APP_ADDR_VGA,
 `APP_ADDR_ETH,
 `APP_ADDR_AUDIO,
 `APP_ADDR_UART,

```

```

`APP_ADDR_GPIO,
`APP_ADDR_RES1,
`APP_ADDR_RES2
) tc_top (

// WISHBONE common
.wb_clk_i (wb_clk),
.wb_rst_i (wb_rst),

// WISHBONE Initiator 0
.i0_wb_cyc_i (1'b0),
.i0_wb_stb_i (1'b0),
.i0_wb_cab_i (1'b0),
.i0_wb_adr_i (32'h0000_0000),
.i0_wb_sel_i (4'b0000),
.i0_wb_we_i (1'b0),
.i0_wb_dat_i (32'h0000_0000),
.i0_wb_dat_o (),
.i0_wb_ack_o (),
.i0_wb_err_o (),

// WISHBONE Initiator 1
.i1_wb_cyc_i (1'b0),
.i1_wb_stb_i (1'b0),
.i1_wb_cab_i (1'b0),
.i1_wb_adr_i (32'h0000_0000),
.i1_wb_sel_i (4'b0000),
.i1_wb_we_i (1'b0),
.i1_wb_dat_i (32'h0000_0000),
.i1_wb_dat_o (),
.i1_wb_ack_o (),
.i1_wb_err_o (),

// WISHBONE Initiator 2
.i2_wb_cyc_i (1'b0),
.i2_wb_stb_i (1'b0),
.i2_wb_cab_i (1'b0),
.i2_wb_adr_i (32'h0000_0000),
.i2_wb_sel_i (4'b0000),
.i2_wb_we_i (1'b0),
.i2_wb_dat_i (32'h0000_0000),
.i2_wb_dat_o (),
.i2_wb_ack_o (),
.i2_wb_err_o (),

// WISHBONE Initiator 3 (Debug Module)
.i3_wb_cyc_i (wb_dm_cyc_o),
.i3_wb_stb_i (wb_dm_stb_o),
.i3_wb_cab_i (wb_dm_cab_o),
.i3_wb_adr_i (wb_dm_adr_o),
.i3_wb_sel_i (wb_dm_sel_o),
.i3_wb_we_i (wb_dm_we_o),
.i3_wb_dat_i (wb_dm_dat_o),
.i3_wb_dat_o (wb_dm_dat_i),
.i3_wb_ack_o (wb_dm_ack_i),
.i3_wb_err_o (wb_dm_err_i),

// WISHBONE Initiator 4 (Risc Data Master)
.i4_wb_cyc_i (wb_rdm_cyc_o),
.i4_wb_stb_i (wb_rdm_stb_o),
.i4_wb_cab_i (wb_rdm_cab_o),

```

```

.i4_wb_adr_i (wb_rdm_adr_o),
.i4_wb_sel_i (wb_rdm_sel_o),
.i4_wb_we_i (wb_rdm_we_o),
.i4_wb_dat_i (wb_rdm_dat_o),
.i4_wb_dat_o (wb_rdm_dat_i),
.i4_wb_ack_o (wb_rdm_ack_i),
.i4_wb_err_o (wb_rdm_err_i),

// WISHBONE Initiator 5 (Risc Instruction Master)
.i5_wb_cyc_i (wb_rim_cyc_o),
.i5_wb_stb_i (wb_rim_stb_o),
.i5_wb_cab_i (wb_rim_cab_o),
.i5_wb_adr_i (wb_rim_adr_o),
.i5_wb_sel_i (wb_rim_sel_o),
.i5_wb_we_i (wb_rim_we_o),
.i5_wb_dat_i (wb_rim_dat_o),
.i5_wb_dat_o (wb_rim_dat_i),
.i5_wb_ack_o (wb_rim_ack_i),
.i5_wb_err_o (wb_rim_err_i),

// WISHBONE Initiator 6
.i6_wb_cyc_i (1'b0),
.i6_wb_stb_i (1'b0),
.i6_wb_cab_i (1'b0),
.i6_wb_adr_i (32'h0000_0000),
.i6_wb_sel_i (4'b0000),
.i6_wb_we_i (1'b0),
.i6_wb_dat_i (32'h0000_0000),
.i6_wb_dat_o (),
.i6_wb_ack_o (),
.i6_wb_err_o (),

// WISHBONE Initiator 7
.i7_wb_cyc_i (1'b0),
.i7_wb_stb_i (1'b0),
.i7_wb_cab_i (1'b0),
.i7_wb_adr_i (32'h0000_0000),
.i7_wb_sel_i (4'b0000),
.i7_wb_we_i (1'b0),
.i7_wb_dat_i (32'h0000_0000),
.i7_wb_dat_o (),
.i7_wb_ack_o (),
.i7_wb_err_o (),

// WISHBONE Target 0 (onchip Sram Controller)
.t0_wb_cyc_o (wb_ss_cyc_i),
.t0_wb_stb_o (wb_ss_stb_i),
.t0_wb_cab_o (wb_ss_cab_i),
.t0_wb_adr_o (wb_ss_adr_i),
.t0_wb_sel_o (wb_ss_sel_i),
.t0_wb_we_o (wb_ss_we_i),
.t0_wb_dat_o (wb_ss_dat_i),
.t0_wb_dat_i (wb_ss_dat_o),
.t0_wb_ack_i (wb_ss_ack_o),
.t0_wb_err_i (wb_ss_err_o),

// WISHBONE Target 1 (coprocesador)
.t1_wb_cyc_o (wb_cop_cyc_i),
.t1_wb_stb_o (wb_cop_stb_i),
.t1_wb_cab_o (wb_cop_cab_i),
.t1_wb_adr_o (wb_cop_adr_i),

```

```

.t1_wb_sel_o (wb_cop_sel_i),
.t1_wb_we_o (wb_cop_we_i),
.t1_wb_dat_o (wb_cop_dat_i),
.t1_wb_dat_i (wb_cop_dat_o),
.t1_wb_ack_i (wb_cop_ack_o),
.t1_wb_err_i (wb_cop_err_o),

// WISHBONE Target 2
.t2_wb_cyc_o (),
.t2_wb_stb_o (),
.t2_wb_cab_o (),
.t2_wb_adr_o (),
.t2_wb_sel_o (),
.t2_wb_we_o (),
.t2_wb_dat_o (),
.t2_wb_dat_i (32'h0000_0000),
.t2_wb_ack_i (1'b0),
.t2_wb_err_i (1'b1),

// WISHBONE Target 3
.t3_wb_cyc_o (),
.t3_wb_stb_o (),
.t3_wb_cab_o (),
.t3_wb_adr_o (),
.t3_wb_sel_o (),
.t3_wb_we_o (),
.t3_wb_dat_o (),
.t3_wb_dat_i (32'h0000_0000),
.t3_wb_ack_i (1'b0),
.t3_wb_err_i (1'b1),

// WISHBONE Target 4
.t4_wb_cyc_o (),
.t4_wb_stb_o (),
.t4_wb_cab_o (),
.t4_wb_adr_o (),
.t4_wb_sel_o (),
.t4_wb_we_o (),
.t4_wb_dat_o (),
.t4_wb_dat_i (32'h0000_0000),
.t4_wb_ack_i (1'b0),
.t4_wb_err_i (1'b1),

// WISHBONE Target 5 (UartSlave)
.t5_wb_cyc_o (wb_us_cyc_i),
.t5_wb_stb_o (wb_us_stb_i),
.t5_wb_cab_o (wb_us_cab_i),
.t5_wb_adr_o (wb_us_adr_i),
.t5_wb_sel_o (wb_us_sel_i),
.t5_wb_we_o (wb_us_we_i),
.t5_wb_dat_o (wb_us_dat_i),
.t5_wb_dat_i (wb_us_dat_o),
.t5_wb_ack_i (wb_us_ack_o),
.t5_wb_err_i (wb_us_err_o),

// WISHBONE Target 6 (simple_GPIO)
.t6_wb_cyc_o (wb_gp_cyc_i),
.t6_wb_stb_o (wb_gp_stb_i),
.t6_wb_cab_o (wb_gp_cab_i),
.t6_wb_adr_o (wb_gp_adr_i),
.t6_wb_sel_o (wb_gp_sel_i),

```



```

 .t6_wb_we_o (wb_gp_we_i),
 .t6_wb_dat_o (wb_gp_dat_i),
 .t6_wb_dat_i (wb_gp_dat_o),
 .t6_wb_ack_i (wb_gp_ack_o),
 .t6_wb_err_i (wb_gp_err_o),

// WISHBONE Target 7
 .t7_wb_cyc_o (),
 .t7_wb_stb_o (),
 .t7_wb_cab_o (),
 .t7_wb_adr_o (),
 .t7_wb_sel_o (),
 .t7_wb_we_o (),
 .t7_wb_dat_o (),
 .t7_wb_dat_i (32'h0000_0000),
 .t7_wb_ack_i (1'b0),
 .t7_wb_err_i (1'b1),

// WISHBONE Target 8
 .t8_wb_cyc_o (),
 .t8_wb_stb_o (),
 .t8_wb_cab_o (),
 .t8_wb_adr_o (),
 .t8_wb_sel_o (),
 .t8_wb_we_o (),
 .t8_wb_dat_o (),
 .t8_wb_dat_i (32'h0000_0000),
 .t8_wb_ack_i (1'b0),
 .t8_wb_err_i (1'b1)
);

//initial begin
// $dumpvars(0);
// $dumpfile("dump.vcd");
//end

```

endmodule

### ***Fichero xsv\_fpga\_defines.v***

```

//
// Define to target to Xilinx Virtex
//
`define TARGET_VIRTEX

//
// Interrupts
//
`define APP_INT_RES1 1:0
`define APP_INT_UART 2
`define APP_INT_RES2 19:3

//
// Address map
//
`define APP_ADDR_DEC_W 8
`define APP_ADDR_SRAM `APP_ADDR_DEC_W'h00
`define APP_ADDR_RAMEXT `APP_ADDR_DEC_W'h01
`define APP_ADDR_FLASH `APP_ADDR_DEC_W'h04
`define APP_ADDR_DECP_W 4
`define APP_ADDR_PERIP `APP_ADDR_DEC_W'h9
`define APP_ADDR_VGA `APP_ADDR_DEC_W'h97

```

```

`define APP_ADDR_ETH `APP_ADDR_DEC_W'h92
`define APP_ADDR_AUDIO `APP_ADDR_DEC_W'h9d
`define APP_ADDR_UART `APP_ADDR_DEC_W'h90
`define APP_ADDR_PS2 `APP_ADDR_DEC_W'h94
`define APP_ADDR_RES1 `APP_ADDR_DEC_W'h9e
`define APP_ADDR_RES2 `APP_ADDR_DEC_W'h9f
`define APP_ADDR_GPIO `APP_ADDR_DEC_W'h91
`define APP_ADDR_FAKEMC 4'h6

```

### 5.2.2 Traffic Cop

Lestat

### 5.2.3 Anexo 6: Simulation Testbench

Ficheros:       sim\_test1.v

```

`timescale 1ns/1ps
`define SRAM_EXT_AW 19

module sim_test1;

reg clk, Nrst;
wire uart_srx;
wire uart_stx;
wire sram_Ncs, sram_No, sram_Nwe;
wire [3:0] sram_Nbe;
wire [`SRAM_EXT_AW+1:2] sram_addr;
wire [31:0] sram_data;
wire flash_Ncs;
wire jtag_tck, jtag_tms, jtag_tdi, jtag_trst;
wire jtag_tvref, jtag_tgnd, jtag_tdo;
wire [3:0] bank_Ncs;
wire [7:0] gpio;
assign uart_srx=1'b0;
assign {jtag_tck,jtag_tms,jtag_tdi,jtag_trst}='b0;
assign bank_Ncs[3]= sram_Ncs | sram_Nbe[3];
assign bank_Ncs[2]= sram_Ncs | sram_Nbe[2];
assign bank_Ncs[1]= sram_Ncs | sram_Nbe[1];
assign bank_Ncs[0]= sram_Ncs | sram_Nbe[0];

xsv_fpga_top chip_fpga (
 .clk(clk),
 .jtag_trst(jtag_trst),
 .jtag_tck(jtag_tck),
 .jtag_tdi(jtag_tdi),
 .uart_srx(uart_srx),
 .jtag_tms(jtag_tms),
 .rstn(Nrst),
 .sram_Ncs(sram_Ncs),
 .jtag_tdo(jtag_tdo),
 .sram_No(sram_No),
 .uart_stx(uart_stx),
 .jtag_tgnd(jtag_tgnd),
 .sram_Nwe(sram_Nwe),
 .jtag_tvref(jtag_tvref),
 .flash_Ncs(flash_Ncs),
 .sram_data(sram_data),
 .sram_addr(sram_addr),
 .sram_Nbe(sram_Nbe),
 .gpio(gpio)
)

```

```

);

sram_sim3 ramext_chip3(
 .Ncs(bank_Ncs[3]),
 .Noe(sram_No),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[31:24])
);

sram_sim2 ramext_chip2(
 .Ncs(bank_Ncs[2]),
 .Noe(sram_No),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[23:16])
);

sram_sim1 ramext_chip1(
 .Ncs(bank_Ncs[1]),
 .Noe(sram_No),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[15:8])
);

sram_sim0 ramext_chip0(
 .Ncs(bank_Ncs[0]),
 .Noe(sram_No),
 .Nwe(sram_Nwe),
 .addr(sram_addr),
 .data(sram_data[7:0])
);

initial begin
 clk=1'b0;
 Nrst=1'b1;
 #1000 Nrst=1'b0;
end

always begin
 #12.5 clk=!clk;
end

endmodule

```

#### 5.2.4 Anexo 7: Modelo de memoria ROM interna

Ficheros:        rom32x32.v (*wrapper* generado por *Xilinx Core Generator*)  
                  rc203\_romcontroller (controlador de *OpenCores*)

##### **Fichero rom32x32.v**

```

/*****
* This file is owned and controlled by Xilinx and must be used
* solely for design, simulation, implementation and creation of
* design files limited to Xilinx devices or technologies. Use
* with non-Xilinx devices or technologies is expressly prohibited
* and immediately terminates your license.
*
* XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"
* SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR
* XILINX DEVICES. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION
* AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION
* OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS

```

```

* IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
* AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
* FOR YOUR IMPLEMENTATION. XILINX EXPRESSLY DISCLAIMS ANY
* WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
* IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
* REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
* INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
* FOR A PARTICULAR PURPOSE.
*
* Xilinx products are not intended for use in life support
* appliances, devices, or systems. Use in such applications are
* expressly prohibited.
*
* (c) Copyright 1995-2006 Xilinx, Inc.
* All rights reserved.
*****/
// You must compile the wrapper file rom32x32.v when simulating
// the core, rom32x32. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".
`timescale 1ns/1ps

module rom32x32(
 addr,
 clk,
 dout);

input [14 : 0] addr;
input clk;
output [31 : 0] dout;

// synopsys translate_off
 BLKMEMSP_V6_2 #(
 15, // c_addr_width
 "0", // c_default_data
 32768, // c_depth
 0, // c_enable_rlocs
 0, // c_has_default_data
 0, // c_has_din
 0, // c_has_en
 0, // c_has_limit_data_pitch
 0, // c_has_nd
 0, // c_has_rdy
 0, // c_has_rfd
 0, // c_has_sinit
 0, // c_has_we
 18, // c_limit_data_pitch
 "rom32x32.mif", // c_mem_init_file
 0, // c_pipe_stages
 0, // c_reg_inputs
 "0", // c_sinit_value
 32, // c_width
 0, // c_write_mode
 "0", // c_ybottom_addr
 1, // c_yclock_is_rising
 1, // c_yen_is_high
 "hierarchy1", // c_yhierarchy
 0, // c_ymake_bmm
 "16kx1", // c_yprimitive_type
 1, // c_ysinit_is_high
 "1024", // c_ytop_addr

```

```

 0, // c_yuse_single_primitive
 1, // c_ywe_is_high
 1) // c_yydisable_warnings
 inst (
 .ADDR(addr),
 .CLK(clk),
 .DOUT(dout),
 .DIN(),
 .EN(),
 .ND(),
 .RFD(),
 .RDY(),
 .SINIT(),
 .WE());
// synopsys translate_on
endmodule

Fichero rc203_romcontroller.v

// synopsys translate_off
`include "timescale.v"
// synopsys translate_on

module wb_rom_controller(clk,reset,
 wb_stb_i,wb_dat_o,wb_dat_i,
 wb_ack_o,wb_adr_i,wb_we_i,
 wb_cyc_i,wb_sel_i,
 address,data);

input clk;
input reset;
input wb_stb_i;
output [31:0] wb_dat_o;
input [31:0] wb_dat_i;
output wb_ack_o;
input [31:0] wb_adr_i;
input wb_we_i;
input wb_cyc_i;
input [3:0] wb_sel_i;
output [14:0] address;
input [31:0] data;

reg [31:0] wb_dat_o;
reg wb_ack_o;
reg [14:0] address;
reg next_reading;
reg reading;

//read_data:
always @(posedge clk or posedge reset)
begin
 if(reset==1)
 begin
 wb_ack_o<=#1 1'b0;
 wb_ack_o<=#1 1'b0;
 wb_dat_o<=#1 1'b0;
 end
 else
 begin
 wb_dat_o <= #1 1'b0;
 wb_ack_o <= #1 1'b0;
 if (reading)

```

```

 begin
 wb_ack_o <= #1 1'b1;
 wb_dat_o <= #1 data;
 end
 end
end

reg [31:0] addr_var;

always @(wb_adr_i or wb_stb_i or wb_we_i or wb_cyc_i
 or reading or wb_ack_o)
 begin
 next_reading = 1'b0;
 addr_var = wb_adr_i >> 2;
 address = addr_var[14:0];

 if(~reading && ~wb_ack_o)
 begin
 if (wb_cyc_i && wb_stb_i && !wb_we_i)
 begin
 addr_var = wb_adr_i >> 2;
 address = addr_var[14:0];
 next_reading = 1'b1;
 end
 end
 end
 if(reading)
 next_reading=1'b0;
 end
end

//register_proc:
always @(posedge clk or posedge reset)
 begin
 if (reset)
 begin
 reading <= #1 1'b0;
 end
 else
 begin
 reading <= #1 next_reading;
 end
 end
end

rom32x32 simu_rom(
 .addr(address),
 .clk(clk),
 .dout(data));
endmodule

```

### ***5.2.5 Anexo 8: Modelo de memoria RAM interna no sintetizable***

Ficheros:       wb\_onchip\_sram32.vhd  
                   pack\_sram32.vhd

#### ***Fichero wb\_onchip\_sram32.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_sram32.all;
--use work.pack_bin_hello_uart.all;

```

```

entity sram_Nx32 is
 port(
 clk: in std_logic;
 e: in std_logic;
 addr: in std_logic_vector(ADDR_WIDTH-1-2 downto 0);
 we: in std_logic;
 be: in std_logic_vector(3 downto 0);
 idata: in std_logic_vector(31 downto 0);
 odata: out std_logic_vector(31 downto 0));
end entity;

architecture beh1 of sram_Nx32 is
 --constant BINIMG_SRAM: T_SRAM_Nx32:=F_BINIMG(BIN_HELLO_UART);
 --constant BINIMG_SRAM: T_SRAM_Nx32:=BINIMG_U;
begin
 sram: process
 variable idx: integer range 0 to NWORDS-1;
 variable sram: T_SRAM_Nx32; --:=BINIMG_SRAM;
 begin
 wait until rising_edge(clk);
 idx:=conv_integer(unsigned(addr));
 odata<=(others=>'X');
 if e='1' then
 case(we) is
 when '0'=>
 if BIG_ENDIAN=1 then
 if be(0)='1' then odata(7 downto 0)<=sram(0)(idx);end if;
 if be(1)='1' then odata(15 downto 8)<=sram(1)(idx);end if;
 if be(2)='1' then odata(23 downto 16)<=sram(2)(idx);end if;
 if be(3)='1' then odata(31 downto 24)<=sram(3)(idx); end if;
 else
 if be(3)='1' then odata(7 downto 0)<=sram(0)(idx); end if;
 if be(2)='1' then odata(15 downto 8)<=sram(1)(idx); end if;
 if be(1)='1' then odata(23 downto 16)<=sram(2)(idx); end if;
 if be(0)='1' then odata(31 downto 24)<=sram(3)(idx); end if;
 end if;
 when '1'=>
 if BIG_ENDIAN=1 then
 if be(0)='1' then sram(0)(idx):=idata(7 downto 0); end if;
 if be(1)='1' then sram(1)(idx):=idata(15 downto 8); end if;
 if be(2)='1' then sram(2)(idx):=idata(23 downto 16); end if;
 if be(3)='1' then sram(3)(idx):=idata(31 downto 24); end if;
 else
 if be(3)='1' then sram(0)(idx):=idata(7 downto 0); end if;
 if be(2)='1' then sram(1)(idx):=idata(15 downto 8); end if;
 if be(1)='1' then sram(2)(idx):=idata(23 downto 16); end if;
 if be(0)='1' then sram(3)(idx):=idata(31 downto 24); end if;
 end if;
 when others=>
 end case;
 end if;
 end process;
 end architecture;

 library ieee;
 use ieee.std_logic_1164.all;
 use ieee.std_logic_arith.all;
 use work.pack_sram32.all;

 entity wb_onchip_sram32 is
 port(
 wb_clk_i : in std_logic;
 wb_rst_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_sel_i : in std_logic_vector(3 downto 0);
 wb_we_i : in std_logic;
 wb_cyc_i : in std_logic;
 wb_stb_i : in std_logic;
 wb_ack_o : out std_logic;
 wb_err_o : out std_logic);
end entity;

architecture beh1 of wb_onchip_sram32 is
function or_bits(bits: std_logic_vector) return std_logic is
variable result: std_logic:='0';
begin
 for k in bits'range loop
 result:=result or bits(k);
 end loop;
 return result;
end function;

signal strobe,enabled,ack_re,ack_we: std_logic;
begin
 onchip_sram32: entity work.sram_Nx32(beh1) port map(
 clk=> wb_clk_i,
 e=> enabled,
 addr=> wb_adr_i(ADDR_WIDTH-1 downto 2),
 we=> wb_we_i,
 be=> wb_sel_i,
 idata=> wb_dat_i,
 odata=> wb_dat_o);

 strobe<=(not wb_rst_i) and wb_cyc_i and wb_stb_i;
 wb_ack_o<=ack_re or ack_we;
 wb_err_o<=strobe and or_bits(wb_adr_i(23 downto ADDR_WIDTH)) ;
 enabled<=strobe and not or_bits(wb_adr_i(23 downto ADDR_WIDTH)) ;

 ack: process (enabled,wb_we_i,wb_clk_i) begin
 ack_we<=enabled and wb_we_i;
 if enabled='0' then
 ack_re<='0';
 elsif rising_edge(wb_clk_i) then
 ack_re<=not wb_we_i;
 end if;
 end process;
end architecture;

```

### ***Fichero pack\_sram32.vhd***

```

library ieee;
use ieee.std_logic_1164.all;

package pack_sram32 is
 constant BIG_ENDIAN: integer:=1;
 constant ADDR_WIDTH: integer:=15;
 constant NWORDS: integer:=(2**(ADDR_WIDTH-2));
 type T_SRAM_Nx8 is array(0 to NWORDS-1) of std_logic_vector(7 downto 0);
 type T_SRAM_Nx32 is array(0 to 3) of T_SRAM_Nx8;
 constant BINIMG_U: T_SRAM_Nx32:=(others=>(others=>(others=>'U')));
 constant BINIMG_0: T_SRAM_Nx32:=(others=>(others=>(others=>'0')));
 constant BINIMG_1: T_SRAM_Nx32:=(others=>(others=>(others=>'1')));
 type T_RAWDATA is array(natural range<>) of std_logic_vector(31 downto 0);

```



```

 function F_BINIMG(rawdata: T_RAWDATA) return T_SRAM_Nx32;
end package;

package body pack_sram32 is
 function F_BINIMG(rawdata: T_RAWDATA) return T_SRAM_Nx32 is
 variable k: integer;
 variable word32: std_logic_vector(31 downto 0);
 variable binimg: T_SRAM_Nx32:=BINIMG_U;
 begin
 for k in rawdata'range loop
 word32:=rawdata(k);
 binimg(0)(k):=word32(7 downto 0);
 binimg(1)(k):=word32(15 downto 8);
 binimg(2)(k):=word32(23 downto 16);
 binimg(3)(k):=word32(31 downto 24);
 end loop;
 return binimg;
 end function;
end package body;

```

### 5.2.6 Anexo 9: Modelo de memoria RAM interna sintetizable

Ficheros:      wb\_onchip\_4sram8.vhd  
                  pack\_sram8.vhd

#### ***Fichero wb\_onchip\_4sram8.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_sram8.all;

entity sram_Nx8 is
 port(
 clk: in std_logic;
 e: in std_logic;
 addr: in std_logic_vector(ADDR_WIDTH-1 downto 2);
 we: in std_logic;
 idata: in std_logic_vector(7 downto 0);
 odata: out std_logic_vector(7 downto 0));
end entity;

architecture beh1 of sram_Nx8 is
 signal ram: T_SRAM_Nx8;
begin
 process begin
 wait until rising_edge(clk);
 odata<=(others=>'X');
 if e='1' then
 if we='1' then
 ram(conv_integer(unsigned(addr)))<=idata;
 else
 odata<=ram(conv_integer(unsigned(addr)));
 end if;
 end if;
 end process;
end architecture;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

```

```

use work.pack_sram8.all;

entity wb_onchip_4sram8 is
 port(
 wb_clk_i : in std_logic;
 wb_rst_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_sel_i : in std_logic_vector(3 downto 0);
 wb_we_i : in std_logic;
 wb_cyc_i : in std_logic;
 wb_stb_i : in std_logic;
 wb_ack_o : out std_logic;
 wb_err_o : out std_logic);
end entity;

architecture beh1 of wb_onchip_4sram8 is
 --funcion temporal (hay que cambiarla) para seleccionar la memoria
 --int/ext segun los bits altos de wb_adr_i
 function select_mem(bits: std_logic_vector) return std_logic is
 variable result: std_logic:='0';
 begin
 for k in bits'range loop
 result:=result or bits(k);
 end loop;
 return result;
 end function;

 signal strobe,enabled,ack_re,ack_we: std_logic;
begin
 strobe<=(not wb_rst_i) and wb_cyc_i and wb_stb_i;
 wb_ack_o<=ack_re or ack_we;
 --
 wb_err_o<=strobe and select_mem(wb_adr_i(23 downto ADDR_WIDTH));
 --no tenemos en cuenta la señal de error
 wb_err_o<='0';
 enabled<=strobe and not select_mem(wb_adr_i(23 downto ADDR_WIDTH));
 --se sustituiria por unas lineas de codigo para elegir memoria

 ack: process (enabled,wb_we_i,wb_clk_i) begin
 ack_we<=enabled and wb_we_i;
 if enabled='0' then
 ack_re<='0';
 elsif rising_edge(wb_clk_i) then
 ack_re<=not wb_we_i;
 end if;
 end process;

 sram_chip3: entity work.sram_Nx8(beh1) port map(
 clk=> wb_clk_i,
 e=> wb_sel_i(3),
 addr=> wb_adr_i(ADDR_WIDTH-1 downto 2),
 we=> wb_we_i,
 idata=> wb_dat_i(31 downto 24),
 odata=> wb_dat_o(31 downto 24));

 sram_chip2: entity work.sram_Nx8(beh1) port map(
 clk=> wb_clk_i,
 e=> wb_sel_i(2),
 addr=> wb_adr_i(ADDR_WIDTH-1 downto 2),
 we=> wb_we_i,

```

```

 idata=> wb_dat_i(23 downto 16),
 odata=> wb_dat_o(23 downto 16));

sram_chip1: entity work.sram_Nx8(beh1) port map(
 clk=> wb_clk_i,
 e=> wb_sel_i(1),
 addr=> wb_adr_i(ADDR_WIDTH-1 downto 2),
 we=> wb_we_i,
 idata=> wb_dat_i(15 downto 8),
 odata=> wb_dat_o(15 downto 8));

sram_chip0: entity work.sram_Nx8(beh1) port map(
 clk=> wb_clk_i,
 e=> wb_sel_i(0),
 addr=> wb_adr_i(ADDR_WIDTH-1 downto 2),
 we=> wb_we_i,
 idata=> wb_dat_i(7 downto 0),
 odata=> wb_dat_o(7 downto 0));
end architecture;

```

### ***Fichero pack\_sram8.vhd***

```

library ieee;
use ieee.std_logic_1164.all;

package pack_sram8 is
 constant BIG_ENDIAN: integer:=1;
 constant ADDR_WIDTH: integer:=14;
 constant NBYTES: integer:=(2**ADDR_WIDTH);
 type T_SRAM_Nx8 is array(0 to NBYTES-1) of std_logic_vector(7 downto 0);
 constant BINIMG_U: T_SRAM_Nx8:=(others=>(others=>'U'));
 constant BINIMG_0: T_SRAM_Nx8:=(others=>(others=>'0'));
 constant BINIMG_1: T_SRAM_Nx8:=(others=>(others=>'1'));
end package;

```

## **5.2.7 Anexo 10: Modelo de memoria RAM externa**

Ficheros:        sram\_sim.vhd

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use work.pack_fingerprint.all;

package pack_sram_sim is
 constant sram_addr_width: integer:=19; --2MB<->21bits=(1 bloque)
 constant sram_data_width: integer:=8;
 subtype sram_word is std_logic_vector(sram_data_width-1 downto 0);
 constant sram_word_U: sram_word:=(others=>'U');
 type t_sram is array(0 to (2**sram_addr_width)-1) of sram_word;
 function init_sram(ADDR_IMG0:std_logic_vector;
fingerprint:T_FINGERPRINT) return t_sram;
 function init_sram3(ADDR_IMG0:std_logic_vector;
fingerprint:T_FINGERPRINT) return t_sram;
 function init_sram2(ADDR_IMG0:std_logic_vector;
fingerprint:T_FINGERPRINT) return t_sram;
 function init_sram1(ADDR_IMG0:std_logic_vector;
fingerprint:T_FINGERPRINT) return t_sram;
 function init_sram0(ADDR_IMG0:std_logic_vector;
fingerprint:T_FINGERPRINT) return t_sram;
 constant FINGERPRINT_EXAMPLE:T_FINGERPRINT:=CONV_FINGERPRINT(INT_FINGERPRINT);

```

```

 constant ADDRESS0: std_logic_vector(sram_addr_width-1 downto
0):="01000000000000000000";
end pack_sram_sim;

package body pack_sram_sim is
 function init_sram(ADDR_IMG0:std_logic_vector;
 fingerprint:T_FINGERPRINT) return t_sram is
 variable i: integer;
 variable val_ram: t_sram;
 variable ADDR0_IMG: unsigned(sram_addr_width-1 downto
 0):=unsigned(ADDR_IMG0(sram_addr_width-1 downto 0));
 variable ADDR1_IMG: unsigned(sram_addr_width-1 downto
 0):=ADDR0_IMG+X"FFFF"; --64kB/fingerprint
 begin
 for i in conv_integer(ADDR0_IMG) to conv_integer(ADDR1_IMG) loop
 val_ram(i):=fingerprint(i-conv_integer(ADDR0_IMG));
 end loop;
 return val_ram;
 end function;

 function init_sram3(ADDR_IMG0:std_logic_vector;
 fingerprint:T_FINGERPRINT) return t_sram is
 variable i,j,k: integer:=0;
 variable val_ram: t_sram;

 variable ADDR0_IMG: unsigned(sram_addr_width-1 downto
 0):=unsigned(ADDR_IMG0(sram_addr_width-1 downto 0));
 variable ADDR1_IMG: unsigned(sram_addr_width-1 downto
 0):=ADDR0_IMG+X"FFFF"; --64kB/fingerprint
 begin
 for i in conv_integer(ADDR0_IMG) to conv_integer(ADDR1_IMG) loop
 if (j=3) then
 val_ram(conv_integer(ADDR0_IMG)+k):=fingerprint(i-
conv_integer(ADDR0_IMG));
 k:=k+1;
 end if;
 j:=j+1;
 if (j=4) then
 j:=0;
 end if;
 end loop;
 return val_ram;
 end function;

 function init_sram2(ADDR_IMG0:std_logic_vector;
 fingerprint:T_FINGERPRINT) return t_sram is
 variable i,j,k: integer:=0;
 variable val_ram: t_sram;
 variable ADDR0_IMG: unsigned(sram_addr_width-1 downto
 0):=unsigned(ADDR_IMG0(sram_addr_width-1 downto 0));
 variable ADDR1_IMG: unsigned(sram_addr_width-1 downto
 0):=ADDR0_IMG+X"FFFF"; --64kB/fingerprint
 begin
 for i in conv_integer(ADDR0_IMG) to conv_integer(ADDR1_IMG) loop
 if (j=2) then
 val_ram(conv_integer(ADDR0_IMG)+k):=fingerprint(i-
conv_integer(ADDR0_IMG));
 k:=k+1;
 end if;
 j:=j+1;
 if (j=4) then
 j:=0;
 end if;
 end loop;
 end function;
end package body pack_sram_sim;

```

```

 end if;
 end loop;
 return val_ram;
end function;

function init_sram1(ADDR_IMG0:std_logic_vector;
fingerprint:T_FINGERPRINT) return t_sram is
 variable i,j,k: integer:=0;
 variable val_ram: t_sram;
 variable ADDR0_IMG: unsigned(sram_addr_width-1 downto
 0):=unsigned(ADDR_IMG0(sram_addr_width-1 downto 0));
 variable ADDR1_IMG: unsigned(sram_addr_width-1 downto
 0):=ADDR0_IMG+X"FFFF"; --64kB/fingerprint
begin
 for i in conv_integer(ADDR0_IMG) to conv_integer(ADDR1_IMG) loop
 if (j=1) then
 val_ram(conv_integer(ADDR0_IMG)+k):=fingerprint(i-
conv_integer(ADDR0_IMG));
 k:=k+1;
 end if;
 j:=j+1;
 if (j=4) then
 j:=0;
 end if;
 end loop;
 return val_ram;
end function;

function init_sram0(ADDR_IMG0:std_logic_vector;
fingerprint:T_FINGERPRINT) return t_sram is
 variable i,j,k: integer:=0;
 variable val_ram: t_sram;
 variable ADDR0_IMG: unsigned(sram_addr_width-1 downto
 0):=unsigned(ADDR_IMG0(sram_addr_width-1 downto 0));
 variable ADDR1_IMG: unsigned(sram_addr_width-1 downto
 0):=ADDR0_IMG+X"FFFF"; --64kB/fingerprint
begin
 for i in conv_integer(ADDR0_IMG) to conv_integer(ADDR1_IMG) loop
 if (j=0) then
 val_ram(conv_integer(ADDR0_IMG)+k):=fingerprint(i-
conv_integer(ADDR0_IMG));
 k:=k+1;
 end if;
 j:=j+1;
 if (j=4) then
 j:=0;
 end if;
 end loop;
 return val_ram;
end function;
end pack_sram_sim;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use work.pack_sram_sim.all;

entity sram_sim0 is
 generic(

```

```

 delay: time:=0 ns);
 port(
 Ncs,Noe: in std_logic;
 Nwe: in std_logic;
 addr: in std_logic_vector(sram_addr_width-1 downto 0);
 data: inout std_logic_vector(sram_data_width-1 downto 0));
end sram_sim0;

```

```

architecture beh0 of sram_sim0 is
begin
 process
 --variable ram: t_sram:=init;
 variable ram: t_sram:=init_sram0(ADDRESS0,FINGERPRINT_EXAMPLE);
 variable index: integer;
 begin
 wait on Ncs,Noe,Nwe,addr,data;
 data<=(others=>'X');
 if Ncs='0' then
 index:=conv_integer(unsigned(addr));
 if Nwe='0' then
 data<=(others=>'Z');
 wait for delay;
 ram(index):=data;
 else
 if Noe='0' then
 data<=ram(index) after delay;
 else
 data<=(others=>'Z');
 end if;
 end if;
 else
 data<=(others=>'Z');
 end if;
 end process;
end beh0;

```

---

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use work.pack_sram_sim.all;

entity sram_sim1 is
 generic(
 delay: time:=0 ns);
 port(
 Ncs,Noe: in std_logic;
 Nwe: in std_logic;
 addr: in std_logic_vector(sram_addr_width-1 downto 0);
 data: inout std_logic_vector(sram_data_width-1 downto 0));
end sram_sim1;

```

```

architecture beh1 of sram_sim1 is
begin
 process
 --variable ram: t_sram:=init;
 variable ram: t_sram:=init_sram1(ADDRESS0,FINGERPRINT_EXAMPLE);
 variable index: integer;
 begin
 wait on Ncs,Noe,Nwe,addr,data;
 data<=(others=>'X');
 end process;
end beh1;

```

```

 if Ncs='0' then
 index:=conv_integer(unsigned(addr));
 if Nwe='0' then
 data<=(others=>'Z');
 wait for delay;
 ram(index):=data;
 else
 if Noe='0' then
 data<=ram(index) after delay;
 else
 data<=(others=>'Z');
 end if;
 end if;
 else
 data<=(others=>'Z');
 end if;
 end process;
end beh1;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use work.pack_sram_sim.all;

entity sram_sim2 is
 generic(
 delay: time:=0 ns);
 port(
 Ncs,Noe: in std_logic;
 Nwe: in std_logic;
 addr: in std_logic_vector(sram_addr_width-1 downto 0);
 data: inout std_logic_vector(sram_data_width-1 downto 0));
end sram_sim2;

architecture beh2 of sram_sim2 is
begin
 process
 --variable ram: t_sram:=init;
 variable ram: t_sram:=init_sram2(ADDRESS0,FINGERPRINT_EXAMPLE);
 variable index: integer;
 begin
 wait on Ncs,Noe,Nwe,addr,data;
 data<=(others=>'X');
 if Ncs='0' then
 index:=conv_integer(unsigned(addr));
 if Nwe='0' then
 data<=(others=>'Z');
 wait for delay;
 ram(index):=data;
 else
 if Noe='0' then
 data<=ram(index) after delay;
 else
 data<=(others=>'Z');
 end if;
 end if;
 else
 data<=(others=>'Z');
 end if;
 end process;
end beh2;

```

```

end beh2;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use work.pack_sram_sim.all;

entity sram_sim3 is
 generic(
 delay: time:=0 ns);
 port(
 Ncs,Noe: in std_logic;
 Nwe: in std_logic;
 addr: in std_logic_vector(sram_addr_width-1 downto 0);
 data: inout std_logic_vector(sram_data_width-1 downto 0));
end sram_sim3;

architecture beh3 of sram_sim3 is
begin
 process
 --variable ram: t_sram:=init;
 variable ram: t_sram:=init_sram3(ADDRESS0,FINGERPRINT_EXAMPLE);
 variable index: integer;
 begin
 wait on Ncs,Noe,Nwe,addr,data;
 data<=(others=>'X');
 if Ncs='0' then
 index:=conv_integer(unsigned(addr));
 if Nwe='0' then
 data<=(others=>'Z');
 wait for delay;
 ram(index):=data;
 else
 if Noe='0' then
 data<=ram(index) after delay;
 else
 data<=(others=>'Z');
 end if;
 end if;
 else
 data<=(others=>'Z');
 end if;
 end process;
end beh3;

```

### 5.2.8 Anexo 11: Controlador de memoria RAM externa

Ficheros:      wb\_bridge\_sramcontrol.vhd  
                  sramcontrol\_entity.vhd  
                  sramcontrol\_beh2.vhd  
                  sramcontrol\_beh3.vhd  
                  utils.vhd

#### ***Fichero wb\_bridge\_sramcontrol.vhd***

```

library ieee;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_utils.all;

```



```

entity wb_sramcontrol is
 generic
 (
 C_BASEADDR : std_logic_vector(31 downto 0) := X"01000000
 C_HIGHADDR : std_logic_vector(31 downto 0) := X"010FFFFF
 C_WB_AWIDTH : integer := 32;
 C_WB_DWIDTH : integer := 32;
 C_ADDR_WIDTH : integer:=20;
 C_RAM_WIDTH : integer:=32;
 C_LOG2_RAM_WIDTH_DIV8 : integer:=2;
 C_ARCHITECTURE : integer:=3;
 C_START_STATE : integer:=0;
 C_END_STATE : integer:=0;
 C_WAIT_STATES : integer:=2
);
 port
 (
 wb_clk_i : in std_logic;
 wb_rst_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_sel_i : in std_logic_vector(3 downto 0);
 wb_we_i : in std_logic;
 wb_cyc_i : in std_logic;
 wb_stb_i : in std_logic;
 wb_ack_o : out std_logic;
 wb_err_o : out std_logic;
 sram_Ncs, sram_Nwe, sram_Noe: out std_logic;
 sram_Nbe : out std_logic_vector((C_RAM_WIDTH/8)-1 downto 0);
 sram_addr : out std_logic_vector(C_ADDR_WIDTH-1 downto
C_LOG2_RAM_WIDTH_DIV8);
 sram_data_I : in std_logic_vector(C_RAM_WIDTH-1 downto 0);
 sram_data_O : out std_logic_vector(C_RAM_WIDTH-1 downto 0);
 sram_data_T : out std_logic
);
end entity;

architecture beh1 of wb_sramcontrol is
 signal ip_initrst,ip_strt,ip_rdy, ip_rNw, ip_doutack: std_logic;
 signal ip_size: std_logic_vector(1 downto 0);
 signal ip_addr: std_logic_vector(C_ADDR_WIDTH-1 downto 0);
 signal ip_din: std_logic_vector(31 downto 0);
 signal ip_dout: std_logic_vector(31 downto 0);
 signal cycstb: std_logic;
 signal strobe: std_logic;
 signal cs,waiting: std_logic;
 signal dataout: std_logic_vector(0 to 31);
begin
 ip_din<=wb_dat_i;
 ip_rnw<=not wb_we_i;
 wb_err_o<='0';
 cycstb<=wb_cyc_i and wb_stb_i;

 process(wb_sel_i,wb_adr_i,ip_dout)
 variable wb_addr: std_logic_vector(31 downto 0);
 begin
 case wb_sel_i is
 when "0001"|"0010"|"0100"|"1000" =>
 ip_size<="00";
 dataout<=ip_dout(7 downto 0)&ip_dout(7 downto
 0)&ip_dout(7 downto 0)&ip_dout(7 downto 0);

```

```

 when "0011"|"1100" =>
 ip_size<="01";
 dataout<=ip_dout(15 downto 0)&ip_dout(15 downto 0);
 when others =>
 ip_size<="11";
 dataout<=ip_dout;
 end case;
 wb_addr:=wb_adr_i;
 ip_addr<=wb_addr(ip_addr'range);
end process;

--CS del la memoria externa
process(wb_adr_i,cycstb) begin
 cs<='0';
 if cycstb='1' then
 if unsigned(wb_adr_i)>=unsigned(C_BASEADDR) and unsigned(wb_adr_i) <=
 unsigned(C_HIGHADDR) then
 cs<='1';
 end if;
 end if;
end process;

process(wb_rst_i,wb_clk_i,cs,ip_rdy)
variable started: std_logic;
begin
 ip_initrst<='0';
 ip_strt<='0';
 waiting<='0';
 if wb_rst_i='1' then
 ip_initrst<='1';
 started:= '0';
 elsif rising_edge(wb_clk_i) then
 if started='0' and ip_strt='1' then
 started:= '1';
 elsif started='1' and cs='0' then
 started:= '0';
 end if;
 end if;
 if cs='1' and started='0' then
 if ip_rdy='1' then
 ip_strt<='1';
 else
 waiting<='1';
 end if;
 end if;
end process;

process(wb_clk_i,cs,wb_we_i,waiting,dataout,ip_strt)
begin
 wb_dat_o<=(others=>'0');
 wb_ack_o<='0';
 if cs='1' then
 if wb_we_i='1' then
 wb_ack_o<=not waiting;--'1';
 else
 wb_dat_o<=dataout;
 wb_ack_o<=ip_doutack;
 end if;
 end if;
end process;

gen_beh2: if C_ARCHITECTURE=2 generate

```

```

sramcontrol0: entity work.sramcontrol(beh2)
 generic map(
 C_ADDR_WIDTH,32,C_RAM_WIDTH,C_START_STATE,C_END_STATE,C_WAIT_STATES)
 port map(
 wb_clk_i,ip_initrst,ip_strt,ip_rdy,ip_size,ip_rNw,ip_addr,ip_
 din,ip_dout,ip_doutack,
 sram_Ncs,sram_Nwe,sram_Noe,sram_Nbe,sram_addr,sram_data_I,sra
 m_data_O,sram_data_T);
end generate;

gen_beh3: if C_ARCHITECTURE=3 generate
 sramcontrol0: entity work.sramcontrol(beh3)
 generic map(
 C_ADDR_WIDTH,32,C_RAM_WIDTH,C_START_STATE,C_END_STATE,C_WAIT_STATES)
 port map(
 wb_clk_i,ip_initrst,ip_strt,ip_rdy,ip_size,ip_rNw,ip_addr,ip_
 din,ip_dout,ip_doutack,
 sram_Ncs,sram_Nwe,sram_Noe,sram_Nbe,sram_addr,sram_data_I,sra
 m_data_O,sram_data_T);
 end generate;
end architecture;

```

### ***Fichero sramcontrol\_entity.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_utils.all;

entity sramcontrol is
 generic(
 ADDRUP_WIDTH: integer:=16;
 DATAUP_WIDTH: integer:=32;
 DATARAM_WIDTH: integer:=8;
 START_STATE: integer:=0;
 END_STATE: integer:=0;
 WAIT_STATES: integer:=2);
 port(
 clk: in std_logic;
 initrst: in std_logic;
 strt: in std_logic;
 rdy: out std_logic;
 size: in std_logic_vector(1 downto 0);
 rNw: in std_logic;
 addr: in std_logic_vector(ADDRUP_WIDTH-1 downto 0);
 din: in std_logic_vector(DATAUP_WIDTH-1 downto 0);
 dout: out std_logic_vector(DATAUP_WIDTH-1 downto 0);
 doutack: out std_logic;
 sram_Ncs,sram_Nwe,sram_Noe: out std_logic;
 sram_Nbe: out std_logic_vector((DATARAM_WIDTH/8)-1 downto 0);
 sram_addr: out std_logic_vector(ADDRUP_WIDTH-1 downto
 num_bits(DATARAM_WIDTH/8));
 sram_data_I: in std_logic_vector(DATARAM_WIDTH-1 downto 0);
 sram_data_O: out std_logic_vector(DATARAM_WIDTH-1 downto 0);
 sram_data_T: out std_logic);
end entity;

```

### ***Fichero sramcontrol\_beh2.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_utils.all;

```

```

architecture beh2 of sramcontrol is
 type STATES is (STATE_STOP, STATE_RUN);
 signal nxtstate, state: STATES;
 signal start, ready0: std_logic;
 signal sram_Nbe0: std_logic_vector(sram_Nbe'range);
 signal en_addr, en_read, en_write: std_logic;
 subtype T_NUMACCESS is integer range (DATAUP_WIDTH/DATARAM_WIDTH)-1 downto 0;
 function numaccess(bytes: std_logic_vector(size'range)) return T_NUMACCESS is
begin
 if DATAUP_WIDTH>DATARAM_WIDTH then
 case DATARAM_WIDTH is
 when 32=>
 return (32/DATARAM_WIDTH)-1;
 when 16=>
 case bytes is
 when "11"=> return (32/DATARAM_WIDTH)-1;
 when "01"=> return (16/DATARAM_WIDTH)-1;
 when others=> return 0;
 end case;
 when 8=>
 case bytes is
 when "11"=> return (32/DATARAM_WIDTH)-1;
 when "01"=> return (16/DATARAM_WIDTH)-1;
 when "00"=> return (8/DATARAM_WIDTH)-1;
 when others=> return 0;
 end case;
 when others=>
 return 0;
 end case;
 else
 return 0;
 end if;
end function;

begin
 rdy<=ready0;
 sram_Nbe<=sram_Nbe0;

 UC0: process (initrst, strt, ready0, clk)
 begin
 start<='0';
 if strt='1' then
 start<='1';
 nxtstate<=STATE_RUN;
 elsif ready0='0' then
 nxtstate<=STATE_RUN;
 else
 nxtstate<=STATE_STOP;
 end if;
 if initrst='1' then
 start<='0';
 state<=STATE_STOP;
 nxtstate<=STATE_STOP;
 elsif rising_edge(clk) then
 state<=nxtstate;
 end if;
 end process;

 UC: process
 subtype T_WAITCOUNT is integer range wait_states-1 downto 0;
 variable waitcount: T_WAITCOUNT;

```

```

variable indx: T_NUMACCESS;
variable rNw0: std_logic;
begin
 wait until rising_edge(clk);
 en_addr<='0';
 en_read<='0';
 en_write<='0';
 if start='1' then
 indx:=numaccess(size);
 waitcount:=T_WAITCOUNT'high;
 rNw0:=rNw;
 ready0<='0';
 if WAIT_STATES=1 then
 en_addr<='1';
 en_read<=rNw0;
 en_write<=not rNw0;
 if indx=0 then
 ready0<='1';
 end if;
 end if;
 else
 case nxtstate is
 when STATE_RUN=>
 case waitcount is
 when 1=>
 waitcount:=waitcount-1;
 en_addr<='1';
 en_read<=rNw0;
 en_write<=not rNw0;
 if indx=0 then
 ready0<='1';
 end if;
 when 0=>
 if WAIT_STATES=1 then
 en_addr<='1';
 en_read<=rNw0;
 en_write<=not rNw0;
 if indx=0 then
 en_addr<='0';
 en_read<='0';
 en_write<='0';
 end if;
 if indx=1 then
 ready0<='1';
 end if;
 end if;
 waitcount:=T_WAITCOUNT'high;
 if indx/=0 then
 indx:=indx-1;
 end if;
 when others=>
 waitcount:=waitcount-1;
 end case;
 when STATE_STOP=>
 ready0<='1';
 end case;
 end if;
 end process;

 UP1: process
 variable addr0: std_logic_vector(sram_addr'range);

```

```

variable addr0_2lsb: unsigned(sram_addr'right+1 downto
 sram_addr'right);
variable rNw0,Noe0: std_logic;
begin
 wait until rising_edge(clk);
 if start='1' then
 addr0:=addr(sram_addr'range);
 rNw0:=rNw;
 Noe0:=not rNw;
 elsif en_addr='1' then
 addr0_2lsb:=1+unsigned(addr0(addr0_2lsb'range));
 addr0(addr0_2lsb'range):=std_logic_vector(addr0_2lsb);
 end if;
 case nxtstate is
 when STATE_RUN=>
 sram_addr<=addr0;
 sram_Ncs<='0';
 sram_Nwe<=rNw0;
 sram_Noe<=Noe0;
 sram_data_T<=rNw0;
 when STATE_STOP=>
 addr0:=(others=>'X');
 sram_addr<=addr0;
 sram_Ncs<='1';
 sram_Nwe<='1';
 sram_Noe<='1';
 sram_data_T<='1';
 end case;
end process;

UP2_UP32: if DATAUP_WIDTH=32 generate process
variable Nbe0: std_logic_vector(sram_Nbe'range);
variable din0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if start='1' then
 din0:=(others=>'X'); Nbe0:=(others=>'X');
 case DATARAM_WIDTH is
 when 32=>
 case size is
 when "11"=>
 din0(31 downto 0):=din(31 downto 0);
 Nbe0:="0000";
 when "01"=>
 case addr(1) is
 when '0'=> din0(15 downto 0):=din(15 downto 0);
 Nbe0:"1100";
 when '1'=> din0(31 downto 16):=din(15 downto 0);
 Nbe0:"0011";
 when others=>
 end case;
 end case;
 when "00"=>
 case addr(1 downto 0) is
 when "00"=>
 din0(7 downto 0):=din(7 downto 0);
 Nbe0:"1110";
 when "01"=>
 din0(15 downto 8):=din(7 downto 0);
 Nbe0:"1101";
 when "10"=>
 din0(23 downto 16):=din(7 downto 0);
 Nbe0:"1011";
 when "11"=>
 din0(31 downto 24):=din(7 downto 0);
 Nbe0:"0111";
 when others=>
 end case;
 end case;
 end case;
 end case;
 end generate;

```

```

 end case;
 when others=>
 end case;
 when 16=>
 case size is
 when "11"=> din0(31 downto 0):=din(31 downto 0);
 Nbe0:="00";
 when "01"=> din0(31 downto 16):=din(15 downto 0);
 Nbe0:="00";
 when "00"=>
 case addr(0) is
 when '0'=> din0(23 downto 16):=din(7 downto 0);
 Nbe0:"10";
 when '1'=> din0(31 downto 24):=din(7 downto 0);
 Nbe0:"01";
 when others=>
 end case;
 when others=>
 end case;
 end case;
 when 8=>
 case size is
 when "11"=> din0(31 downto 0):=din(31 downto 0);
 Nbe0:"0";
 when "01"=> din0(31 downto 16):=din(15 downto 0);
 Nbe0:"0";
 when "00"=> din0(31 downto 24):=din(7 downto 0);
 Nbe0:"0";
 when others=>
 end case;
 when others=>
 end case;
 end case;
 elsif en_write='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 din0(DATAUP_WIDTH-1 downto DATARAM_WIDTH) :=
 din0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 din0(DATARAM_WIDTH-1 downto 0):=(others=>'X');
 end if;
 end if;
 case nxtstate is
 when STATE_RUN=>
 sram_Nbe0<=Nbe0;
 sram_data_O<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_STOP=>
 sram_Nbe0<=(others=>'1');
 sram_data_O<=(others=>'X');
 end case;
 end process; end generate;

```

```

UP2_UP16: if DATAUP_WIDTH=16 generate process
variable Nbe0: std_logic_vector(sram_Nbe'range);
variable din0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if start='1' then
 din0:=(others=>'X'); Nbe0:=(others=>'X');
 case DATARAM_WIDTH is
 when 16=>
 case size is
 when "01"=> din0(15 downto 0):=din(15 downto 0);
 Nbe0:"00";

```

```

 when "00"=>
 case addr(0) is
 when '0'=> din0(7 downto 0):=din(7 downto 0);
 Nbe0:="10";
 when '1'=> din0(15 downto 8):=din(7 downto 0);
 Nbe0:="01";
 when others=>
 end case;
 when others=>
 end case;
 when 8=>
 case size is
 when "01"=> din0(15 downto 0):=din(15 downto 0);
 Nbe0:="0";
 when "00"=> din0(15 downto 8):=din(7 downto 0);
 Nbe0:="0";
 when others=>
 end case;
 when others=>
 end case;
 elsif en_write='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 din0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 din0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 din0(DATARAM_WIDTH-1 downto 0):=(others=>'X');
 end if;
 end if;
 case nxtstate is
 when STATE_RUN=>
 sram_Nbe0<=Nbe0;
 sram_data_O<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_STOP=>
 sram_Nbe0<=(others=>'1');
 sram_data_O<=(others=>'X');
 end case;
 end process; end generate;

UP2_UP8: if DATAUP_WIDTH=8 generate process
variable Nbe0: std_logic_vector(sram_Nbe'range);
variable din0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if start='1' then
 din0:=(others=>'X'); Nbe0:=(others=>'X');
 case DATARAM_WIDTH is
 when 8=>
 case DATARAM_WIDTH is
 when 8=>
 case size is
 when "00"=> din0(7 downto 0):=din(7 downto 0);
 Nbe0:="0";
 when others=>
 end case;
 when others=>
 end case;
 when others=>
 end case;
 end case;
 when others=>
 end case;
 end if;
 elsif en_write='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 din0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=

```



```

 din0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 din0(DATARAM_WIDTH-1 downto 0):=(others=>'X');
 end if;
end if;
case nxtstate is
 when STATE_RUN=>
 sram_Nbe0<=Nbe0;
 sram_data_O<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_STOP=>
 sram_Nbe0<=(others=>'1');
 sram_data_O<=(others=>'X');
 end case;
end process; end generate;

UP3_RAM32: if DATARAM_WIDTH=32 generate process
variable dout0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if en_read='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 dout0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 dout0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 end if;
 case sram_Nbe0(3 downto 0) is
 when "0000"=> dout0(31 downto 0):=sram_data_I(31 downto 0);
 when "1100"=> dout0(15 downto 0):=sram_data_I(15 downto 0);
 when "0011"=> dout0(15 downto 0):=sram_data_I(31 downto 16);
 when "1110"=> dout0(7 downto 0):=sram_data_I(7 downto 0);
 when "1101"=> dout0(7 downto 0):=sram_data_I(15 downto 8);
 when "1011"=> dout0(7 downto 0):=sram_data_I(23 downto 16);
 when "0111"=> dout0(7 downto 0):=sram_data_I(31 downto 24);
 when others=>
 end case;
 end if;
 dout<=dout0;
end process; end generate;

UP3_RAM16: if DATARAM_WIDTH=16 generate process
variable dout0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if en_read='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 dout0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 dout0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 end if;
 case sram_Nbe0(1 downto 0) is
 when "00"=> dout0(15 downto 0):=sram_data_I(15 downto 0);
 when "10"=> dout0(7 downto 0):=sram_data_I(7 downto 0);
 when "01"=> dout0(7 downto 0):=sram_data_I(15 downto 8);
 when others=>
 end case;
 end if;
 dout<=dout0;
end process; end generate;

UP3_RAM8: if DATARAM_WIDTH=8 generate process
variable dout0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);

```

```

 if en_read='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 dout0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 dout0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 end if;
 case sram_Nbe0(0) is
 when '0'=> dout0(7 downto 0):=sram_data_I(7 downto 0);
 when others=>
 end case;
 end if;
 dout<=dout0;
 end process; end generate;

UP4: process
begin
 wait until rising_edge(clk);
 doutack<='0';
 if en_read='1' and ready0='1' then
 doutack<='1';
 end if;
end process;

CHECK1: process(clk,strt,addr,size)
variable numaccess0: integer;
begin
 if rising_edge(clk) then
 if strt='1' then
 numaccess0:=numaccess(size);
 case DATARAM_WIDTH is
 when 8=>
 assert (numaccess0=3 and addr(1 downto 0)="00") or
 (numaccess0=1 and addr(0)='0') or (numaccess0=0)
 report "Error en sramcontrol" severity FAILURE;
 when 16=>
 assert (numaccess0=1 and addr(1)='0') or
 (numaccess0=0)
 report "Error en sramcontrol" severity FAILURE;
 when 32=>
 assert numaccess0=0
 report "Error en sramcontrol" severity FAILURE;
 when others=>
 assert false
 report "Error en sramcontrol" severity FAILURE;
 end case;
 end if;
 end if;
end process;

CHECK2:
 assert (DATARAM_WIDTH=8 or DATARAM_WIDTH=16 or DATARAM_WIDTH=32)
 and (DATAUP_WIDTH=8 or DATAUP_WIDTH=16 or DATAUP_WIDTH=32) and
 (DATAUP_WIDTH>=DATARAM_WIDTH)
 report "Error en instancia de sramcontrol";
end architecture;

```

### ***Fichero sramcontrol\_beh3.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_utils.all;

```

```

architecture beh3 of sramcontrol is
type STATES is (STATE_STOP, STATE_START, STATE_RUN, STATE_END);
signal nxtstate, state: STATES;
signal start, ready0: std_logic;
signal sram_Nbe0: std_logic_vector(sram_Nbe'range);
signal en_addr, en_read, en_write: std_logic;
signal cycle_completed, last_read: std_logic;
subtype T_NUMACCESS is integer range (DATAUP_WIDTH/DATARAM_WIDTH)-1
downto 0;

function numaccess(bytes: std_logic_vector(size'range)) return
T_NUMACCESS IS
begin
 if DATAUP_WIDTH>DATARAM_WIDTH then
 case DATARAM_WIDTH is
 when 32=>
 return (32/DATARAM_WIDTH)-1;
 when 16=>
 case bytes is
 when "11"=> return (32/DATARAM_WIDTH)-1;
 when "01"=> return (16/DATARAM_WIDTH)-1;
 when others=> return 0;
 end case;
 when 8=>
 case bytes is
 when "11"=> return (32/DATARAM_WIDTH)-1;
 when "01"=> return (16/DATARAM_WIDTH)-1;
 when "00"=> return (8/DATARAM_WIDTH)-1;
 when others=> return 0;
 end case;
 when others=>
 return 0;
 end case;
 else
 return 0;
 end if;
end function;

begin
 rdy<=ready0;

 UC0:
 process (initrst, strt, ready0, nxtstate, state, cycle_completed, clk)
 begin
 start<='0';
 if strt='1' then
 start<='1';
 if START_STATE=1 then
 nxtstate<=STATE_START;
 else
 nxtstate<=STATE_RUN;
 end if;
 else case state is
 when STATE_START=>
 nxtstate<=STATE_RUN;
 when STATE_RUN=>
 if ready0='1' then
 if END_STATE=1 then
 nxtstate<=STATE_END;
 else
 nxtstate<=STATE_STOP;
 end if;
 end if;
 end case;
 end if;
 end process UC0;
end architecture beh3;

```

```

 end if;
 elsif cycle_completed='1' then
 if END_STATE=1 then
 nxtstate<=STATE_END;
 elsif START_STATE=1 then
 nxtstate<=STATE_START;
 else
 nxtstate<=STATE_RUN;
 end if;
 else
 nxtstate<=STATE_RUN;
 end if;
when STATE_END=>
 if ready0='1' then
 nxtstate<=STATE_STOP;
 else
 if START_STATE=1 then
 nxtstate<=STATE_START;
 else
 nxtstate<=STATE_RUN;
 end if;
 end if;
when STATE_STOP=>
 nxtstate<=STATE_STOP;
end case;
end if;

if initrst='1' then
 state<=STATE_STOP;
 nxtstate<=STATE_STOP;
elsif rising_edge(clk) then
 state<=nxtstate;
end if;
end process;

UC: process
subtype T_WAITCOUNT is integer range WAIT_STATES-1 downto 0;
variable waitcount: T_WAITCOUNT;

variable indx: T_NUMACCESS;
variable rNw0: std_logic;
begin
 wait until rising_edge(clk);
 cycle_completed<='0';
 en_addr<='0';
 en_read<='0';
 en_write<='0';
 last_read<='0';

 if start='1' then
 indx:=numaccess(size);
 waitcount:=T_WAITCOUNT'high;
 rNw0:=rNw;
 ready0<='0';
 end if;

 if nxtstate=STATE_RUN and waitcount=0 then
 cycle_completed<='1';
 en_read<=rNw0;
 if rNw0='1' and indx=0 then
 last_read<='1';

```

```

 end if;
 end if;

 if END_STATE=0 then
 if nxtstate=STATE_RUN and waitcount=0 and indx/=0 then
 en_write<=not rNw0;
 en_addr<='1';
 end if;
 end if;

 if END_STATE=1 then
 if nxtstate=STATE_END and cycle_completed='1' then
 en_write<=not rNw0;
 en_addr<='1';
 end if;
 end if;

 if END_STATE=0 then
 if nxtstate=STATE_RUN then
 if waitcount=0 then
 if indx/=0 then
 indx:=indx-1;
 waitcount:=T_WAITCOUNT'high;
 else
 ready0<='1';
 end if;
 else
 waitcount:=waitcount-1;
 end if;
 end if;
 end if;

 if END_STATE=1 then
 if nxtstate=STATE_RUN then
 if waitcount/=0 then
 waitcount:=waitcount-1;
 end if;
 end if;
 if nxtstate=STATE_END then
 if waitcount=0 then
 if indx/=0 then
 waitcount:=T_WAITCOUNT'high;
 indx:=indx-1;
 else
 ready0<='1';
 end if;
 end if;
 end if;
 end if;

 if nxtstate=STATE_STOP then
 ready0<='1';
 end if;

end process;

UP1: process
 variable addr0: std_logic_vector(sram_addr'range);
 variable addr0_2lsb: unsigned(sram_addr'right+1 downto
sram_addr'right);
 variable rNw0,Noe0: std_logic;

```

```

begin
 wait until rising_edge(clk);
 if start='1' then
 addr0:=addr(sram_addr'range);
 rNw0:=rNw;
 Noe0:=not rNw;
 elsif en_addr='1' then
 addr0_2lsb:=1+unsigned(addr0(addr0_2lsb'range));
 addr0(addr0_2lsb'range):=std_logic_vector(addr0_2lsb);
 end if;
 case nxtstate is
 when STATE_START|STATE_END=>
 sram_addr<=addr0;
 sram_Ncs<='0';
 sram_Nwe<='1';
 sram_Noe<='1';
 sram_data_T<=rNw0;
 when STATE_RUN=>
 sram_addr<=addr0;
 sram_Ncs<='0';
 sram_Nwe<=rNw0;
 sram_Noe<=Noe0;
 sram_data_T<=rNw0;
 when STATE_STOP=>
 addr0:=(others=>'X');
 sram_addr<=addr0;
 sram_Ncs<='1';
 sram_Nwe<='1';
 sram_Noe<='1';
 sram_data_T<='1';
 end case;
end process;

UP2_UP32: if DATAUP_WIDTH=32 generate process
variable Nbe0: std_logic_vector(sram_Nbe'range);
variable din0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if start='1' then
 din0:=(others=>'X'); Nbe0:=(others=>'X');
 case DATARAM_WIDTH is
 when 32=>
 case size is
 when "11"=> din0(31 downto 0):=din(31 downto 0);
 Nbe0:="0000";
 when "01"=>
 case addr(1) is
 when '0'=> din0(15 downto 0):=din(15 downto 0);
 Nbe0:"1100";
 when '1'=> din0(31 downto 16):=din(15 downto 0);
 Nbe0:"0011";
 when others=>
 end case;
 end case;
 when "00"=>
 case addr(1 downto 0) is
 when "00"=> din0(7 downto 0):=din(7 downto 0);
 Nbe0:"1110";
 when "01"=> din0(15 downto 8):=din(7 downto 0);
 Nbe0:"1101";
 when "10"=> din0(23 downto 16):=din(7 downto 0);
 Nbe0:"1011";
 end case;
 end case;
 end case;
 end generate;

```

```

 when "11"=> din0(31 downto 24):=din(7 downto 0);
 Nbe0:="0111";
 when others=>
 end case;
 when others=>
 end case;
 when 16=>
 case size is
 when "11"=> din0(31 downto 0):=din(31 downto 0);
 Nbe0:="00";
 when "01"=> din0(31 downto 16):=din(15 downto 0);
 Nbe0:="00";
 when "00"=>
 case addr(0) is
 when '0'=> din0(23 downto 16):=din(7 downto 0);
 Nbe0:="10";
 when '1'=> din0(31 downto 24):=din(7 downto 0);
 Nbe0:="01";
 when others=>
 end case;
 when others=>
 end case;
 end case;
 when 8=>
 case size is
 when "11"=> din0(31 downto 0):=din(31 downto 0);
 Nbe0:="0";
 when "01"=> din0(31 downto 16):=din(15 downto 0);
 Nbe0:="0";
 when "00"=> din0(31 downto 24):=din(7 downto 0);
 Nbe0:="0";
 when others=>
 end case;
 when others=>
 end case;
 end case;
 elsif en_write='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 din0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 din0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 din0(DATARAM_WIDTH-1 downto 0):=(others=>'X');
 end if;
 end if;
 case nxtstate is
 when STATE_START|STATE_END=>
 sram_Nbe<=Nbe0;
 sram_data_O<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_RUN=>
 sram_Nbe<=Nbe0;
 sram_data_O<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_STOP=>
 sram_Nbe<=(others=>'1');
 sram_data_O<=(others=>'X');
 end case;
 sram_Nbe0<=Nbe0;
end process; end generate;

UP2_UP16: if DATAUP_WIDTH=16 generate process
variable Nbe0: std_logic_vector(sram_Nbe'range);
variable din0: std_logic_vector(din'range);
begin

```

```

wait until rising_edge(clk);
if start='1' then
 din0:=(others=>'X'); Nbe0:=(others=>'X');
 case DATARAM_WIDTH is
 when 16=>
 case size is
 when "01"=> din0(15 downto 0):=din(15 downto 0);
 Nbe0:="00";
 when "00"=>
 case addr(0) is
 when '0'=> din0(7 downto 0):=din(7 downto 0);
 Nbe0:"10";
 when '1'=> din0(15 downto 8):=din(7 downto 0);
 Nbe0:"01";
 when others=>
 end case;
 end case;
 when others=>
 end case;
 when 8=>
 case size is
 when "01"=> din0(15 downto 0):=din(15 downto 0);
 Nbe0:"0";
 when "00"=> din0(15 downto 8):=din(7 downto 0);
 Nbe0:"0";
 when others=>
 end case;
 when others=>
 end case;
 end case;
 elsif en_write='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 din0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 din0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 din0(DATARAM_WIDTH-1 downto 0):=(others=>'X');
 end if;
 end if;
 case nxtstate is
 when STATE_START|STATE_END=>
 sram_Nbe<=Nbe0;
 sram_data_0<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_RUN=>
 sram_Nbe<=Nbe0;
 sram_data_0<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_STOP=>
 sram_Nbe<=(others=>'1');
 sram_data_0<=(others=>'X');
 end case;
 sram_Nbe0<=Nbe0;
 end process; end generate;

```

```

UP2_UP8: if DATAUP_WIDTH=8 generate process
variable Nbe0: std_logic_vector(sram_Nbe'range);
variable din0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if start='1' then
 din0:=(others=>'X'); Nbe0:=(others=>'X');
 case DATARAM_WIDTH is
 when 8=>
 case DATARAM_WIDTH is

```



```

 when 8=>
 case size is
 when "00"=> din0(7 downto 0):=din(7 downto 0);
 Nbe0:="0";
 when others=>
 end case;
 when others=>
 end case;
 when others=>
 end case;
 elsif en_write='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 din0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 din0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 din0(DATARAM_WIDTH-1 downto 0):=(others=>'X');
 end if;
 end if;
 case nxtstate is
 when STATE_START|STATE_END=>
 sram_Nbe<=Nbe0;
 sram_data_O<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_RUN=>
 sram_Nbe<=Nbe0;
 sram_data_O<=din0(DATAUP_WIDTH-1 downto DATAUP_WIDTH-
 DATARAM_WIDTH);
 when STATE_STOP=>
 sram_Nbe<=(others=>'1');
 sram_data_O<=(others=>'X');
 end case;
 sram_Nbe0<=Nbe0;
end process; end generate;

UP3_RAM32: if DATARAM_WIDTH=32 generate process
variable dout0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if en_read='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 dout0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 dout0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 end if;
 case sram_Nbe0(3 downto 0) is
 when "0000"=> dout0(31 downto 0):=sram_data_I(31 downto 0);
 when "1100"=> dout0(15 downto 0):=sram_data_I(15 downto 0);
 when "0011"=> dout0(15 downto 0):=sram_data_I(31 downto 16);
 when "1110"=> dout0(7 downto 0):=sram_data_I(7 downto 0);
 when "1101"=> dout0(7 downto 0):=sram_data_I(15 downto 8);
 when "1011"=> dout0(7 downto 0):=sram_data_I(23 downto 16);
 when "0111"=> dout0(7 downto 0):=sram_data_I(31 downto 24);
 when others=>
 end case;
 end if;
 dout<=dout0;
 end process; end generate;

UP3_RAM16: if DATARAM_WIDTH=16 generate process
variable dout0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if en_read='1' then

```

```

 if DATAUP_WIDTH>DATARAM_WIDTH then
 dout0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):=
 dout0(DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 end if;
 case sram_Nbe0(1 downto 0) is
 when "00"=> dout0(15 downto 0):=sram_data_I(15 downto 0);
 when "10"=> dout0(7 downto 0):=sram_data_I(7 downto 0);
 when "01"=> dout0(7 downto 0):=sram_data_I(15 downto 8);
 when others=>
 end case;
 end if;
 dout<=dout0;
 end process; end generate;

UP3_RAM8: if DATARAM_WIDTH=8 generate process
variable dout0: std_logic_vector(din'range);
begin
 wait until rising_edge(clk);
 if en_read='1' then
 if DATAUP_WIDTH>DATARAM_WIDTH then
 dout0(DATAUP_WIDTH-1 downto DATARAM_WIDTH):= dout0
 (DATAUP_WIDTH-DATARAM_WIDTH-1 downto 0);
 end if;
 case sram_Nbe0(0) is
 when '0'=> dout0(7 downto 0):=sram_data_I(7 downto 0);
 when others=>
 end case;
 end if;
 dout<=dout0;
 end process; end generate;

UP4: process
begin
 wait until rising_edge(clk);
 doutack<='0';
 if last_read='1' then
 doutack<='1';
 end if;
end process;

CHECK1: process(clk,strt,addr,size)
variable numaccess0: integer;
begin
 if rising_edge(clk) then
 if strt='1' then
 numaccess0:=numaccess(size);
 case DATARAM_WIDTH is
 when 8=>
 assert (numaccess0=3 and addr(1 downto 0)="00")or
 (numaccess0=1 and addr(0)='0')or(numaccess0=0)
 report "Error en sramcontrol" severity FAILURE;
 when 16=>
 assert (numaccess0=1 and addr(1)='0')or
 (numaccess0=0)
 report "Error en sramcontrol" severity FAILURE;
 when 32=>
 assert numaccess0=0
 report "Error en sramcontrol" severity FAILURE;
 when others=>
 assert false
 report "Error en sramcontrol" severity FAILURE;
 end case;
 end if;
 end if;
end process;

```

```

 end case;
 end if;
end if;
end process;

CHECK2: assert (DATARAM_WIDTH=8 or DATARAM_WIDTH=16 or DATARAM_WIDTH=32)
 and (DATAUP_WIDTH=8 or DATAUP_WIDTH=16 or DATAUP_WIDTH=32)
 and (DATAUP_WIDTH>=DATARAM_WIDTH)
report "Error en instancia de sramcontrol" severity FAILURE;

CHECK3: assert WAIT_STATES>=1 and (START_STATE=0 or START_STATE=1) and
 (END_STATE=0 or END_STATE=1)
report "Error en instancia de sramcontrol" severity FAILURE;
end architecture;

```

### ***Fichero utils.vhd***

```

package pack_utils is
function num_bits(val: integer) return integer;
end package;

package body pack_utils is
function num_bits(val: integer) return integer is
begin
 case val is
 when 2**0=> return 0;
 when 2**1=> return 1;
 when 2**2=> return 2;
 when 2**3=> return 2;
 when 2**4=> return 2;
 when 2**5=> return 2;
 when 2**6=> return 2;
 when 2**7=> return 2;
 when 2**8=> return 8;
 when others=> return -1;
 end case;
end function;
end package body;

```

## **5.2.9 Anexo 12: Coprocesador**

Ficheros:      wb\_bridge\_maioHw.vhd  
                  sram\_adapt.vhd  
                  pack\_maioHw.vhd  
                  follow.vhd  
                  corrmx.vhd  
                  filtsec.vhd  
                  nextpoint.vhd  
                  section.vhd  
                  tangdir.vhd  
                  xynext.vhd  
                  sramcontrol\_8\_8.vhd  
                  sramcontrol\_8\_32.vhd

### ***Fichero wb\_bridge\_maoiHw.vhd***

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

```

```

use work.pack_maioHw.all;

entity user_maioHw is
 generic (
 C_ADDR_WIDTH: integer:=21; --USER
 C_ADDR_REGISTER: std_logic_vector(0 to 31):=X"0100_0000"
);
 port (
 wb_clk_i : in std_logic;
 wb_rst_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_sel_i : in std_logic_vector(3 downto 0);
 wb_we_i : in std_logic;
 wb_cyc_i : in std_logic;
 wb_stb_i : in std_logic;
 wb_ack_o : out std_logic;
 wb_err_o : out std_logic;

 sram32_Ncs,sram32_Noel,sram32_Nwe: out std_logic;
 sram32_Nbe: out std_logic_vector(3 downto 0);
 sram32_addr: out std_logic_vector(ADDR_WIDTH-1 downto 2);
 sram32_data_I: in std_logic_vector(31 downto 0);
 sram32_data_O: out std_logic_vector(31 downto 0);
 sram32_data_T: out std_logic
);
end entity;

architecture IMP of user_maioHw is
 signal CS: std_logic;
 signal running_maioHw,access_register: boolean;
 signal emc_ack,maioHw_ack: std_logic;
 signal emc_Nrst,maioHw_Nrst: std_logic;
 signal reg_command,reg_result,result: std_logic_vector(31 downto 0);
 signal mode: std_logic_vector(2 downto 0);
 signal Nrst,Nstop,emc_rNw: std_logic;
 signal x0,y0,x1,y1: std_logic_vector(7 downto 0);
 signal w0,w1: signed(4 downto 0);
 signal emc_addr: std_logic_vector(ADDR_WIDTH-1 downto 0);
 signal emc_din,emc_dout,emc_dout0: std_logic_vector(MICRO_DATA_WIDTH-1
 downto 0);
 signal emc_numbytes: unsigned(1 downto 0);
 signal sram8_Ncs,sram8_Noel,sram8_Nwe: std_logic;
 signal sram8_addr: std_logic_vector(ADDR_WIDTH-1 downto 0);
 signal sram8_data_I: std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 signal sram8_data_O: std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 signal sram8_data_T: std_logic;

 component follow is
 port(
 clk: in std_logic;
 mode: in std_logic_vector(2 downto 0);
 --XX0:emc, X01:copro iter#n, 011:copro iter#0, 111:angle
 Nrst: in std_logic;
 Nstop: out std_logic;
 w0: in signed(4 downto 0);
 x0,y0: in std_logic_vector(7 downto 0);
 w1: out signed(4 downto 0);
 x1,y1: out std_logic_vector(7 downto 0);
 emc_numbytes: in unsigned(1 downto 0);
 emc_rNw: in std_logic;

```

```

 emc_addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
 emc_din: in std_logic_vector(MICRO_DATA_WIDTH-1 downto 0);
 emc_dout: out std_logic_vector(MICRO_DATA_WIDTH-1 downto 0);
 sram_Ncs,sram_Noel,sram_Nwe: out std_logic;
 sram_addr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
 sram_data_I: in std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram_data_O: out std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram_data_T: out std_logic
);
end component;

component sram_adapt is
 port(
 sram8_Ncs,sram8_Noel,sram8_Nwe: in std_logic;
 sram8_addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
 sram8_data_I: out std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram8_data_O: in std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram8_data_T: in std_logic;
 sram32_Ncs,sram32_Noel,sram32_Nwe: out std_logic;
 sram32_Nbe: out std_logic_vector(3 downto 0);
 sram32_addr: out std_logic_vector(ADDR_WIDTH-1 downto 2);
 sram32_data_I: in std_logic_vector(31 downto 0);
 sram32_data_O: out std_logic_vector(31 downto 0);
 sram32_data_T: out std_logic
);
end component;

begin
 -- architecture IMP
 wb_err_o<='0';
 CS<=wb_cyc_i and wb_stb_i;
 running_maioHw<=Nstop='1' and maioHw_Nrst='1';
 access_register<=wb_adr_i=C_ADDR_REGISTER and wb_sel_i="1111";
 --simplificar a los ADDR_WIDTH bits de menor peso
 wb_ack_o<=emc_ack or maioHw_ack;
 result(31 downto 24)<=(others=>'0');

 --mode en proceso
 Nrst<=emc_Nrst or maioHw_Nrst;
 --Nstop en instancia
 w0<=signed(reg_command(20 downto 16));
 x0<=reg_command(7 downto 0);
 y0<=reg_command(15 downto 8);
 result(23 downto 16)<=sxt(std_logic_vector(w1),8);
 result(7 downto 0)<=x1;
 result(15 downto 8)<=y1;
 --emc_Nrst en proceso
 --emc_numbytes en proceso
 emc_rNw<=not wb_we_i;
 --emc_addr en proceso
 emc_din<=wb_dat_i;
 --emc_dout en proceso

 follow0: follow port map(
 wb_clk_i,mode,
 Nrst,Nstop,w0,x0,y0,w1,x1,y1,
 emc_numbytes,emc_rNw,emc_addr,emc_din,emc_dout,
 sram8_Ncs,sram8_Noel,sram8_Nwe,sram8_addr,sram8_data_I,sram8_data_O,sram8_data_T);

 sram_adapt0: sram_adapt port map(
 sram8_Ncs,sram8_Noel,sram8_Nwe,sram8_addr,sram8_data_I,sram8_data_O,
 sram8_data_T,

```

```

sram32_Ncs,sram32_Noel,sram32_Nwe,sram32_Nbe,sram32_addr,sram32_data
_I,sram32_data_O,sram32_data_T);

pMode: process begin
 wait until rising_edge(wb_clk_i);
 if wb_rst_i='1' then
 mode<="XX0";
 elsif wb_we_i='1' and access_register then
 mode<=wb_dat_i(25 downto 24)&'1';
 elsif not running_maioHw then
 mode<="XX0";
 end if;
end process;

pBE: process(wb_sel_i,wb_adr_i,emc_dout) begin
 emc_addr<=wb_adr_i(ADDR_WIDTH-1 downto 0);
 case wb_sel_i is
 when "1111"=>
 emc_numbytes<="11";
 emc_addr(1 downto 0)<="00";
 emc_dout0<=emc_dout;
 when "1100"|"0011"=>
 emc_numbytes<="01";
 emc_addr(0)<='0';
 emc_dout0<=emc_dout(15 downto 0)&emc_dout(15 downto 0);
 when others=>
 emc_numbytes<="00";
 emc_dout0<=emc_dout(7 downto 0)&emc_dout(7 downto 0)&
 emc_dout(7 downto 0)&emc_dout(7 downto 0);
 end case;
end process;

pNeAck: process
 begin
 wait until rising_edge(wb_clk_i);
 if wb_rst_i='1' or CS='0' then
 emc_ack<='0';
 emc_Nrst<='0';
 elsif not (running_maioHw or access_register) then
 emc_Nrst<='1';
 if emc_Nrst='1' and Nstop='0' then
 emc_ack<='1';
 end if;
 end if;
 end process;

pMaioHw: process
 variable maioHw_Nrst0: std_logic;
 begin
 wait until rising_edge(wb_clk_i);
 if wb_rst_i = '1' then
 reg_command<=(others=>'0');
 reg_result<=(others=>'1');
 maioHw_Nrst<='0';
 maioHw_Nrst0:='0';
 else
 maioHw_ack<='0';
 if maioHw_Nrst0='1' then
 maioHw_Nrst<='1';
 end if;
 if CS = '1' and access_register then --Bus2IP_CS quitar
 maioHw_ack<='1';
 end if;
 end if;
 end process;

```

```

 if wb_we_i='1' then
 maioHw_Nrst0:='1';
 reg_command<=wb_dat_i;
 end if;
 end if;
 if running_maioHw then
 reg_result<=(others=>'1');
 elsif maioHw_Nrst='1' then
 reg_result<=result;
 maioHw_Nrst<='0';
 maioHw_Nrst0:='0';
 end if;
end if;
end process;

pRead: process(access_register,reg_result,emc_dout0) is
begin
 if access_register then
 wb_dat_o<=reg_result;
 else
 wb_dat_o<=emc_dout0;
 end if;
end process;
end architecture IMP;

```

### ***Fichero sram\_adapt.vhd***

```

library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;
use work.pack_maioHw.all;

entity sram_adapt is
 port(
 sram8_Ncs,sram8_Noe,sram8_Nwe: in std_logic;
 sram8_addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
 sram8_data_I: out std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram8_data_O: in std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram8_data_T: in std_logic;
 sram32_Ncs,sram32_Noe,sram32_Nwe: out std_logic;
 sram32_Nbe: out std_logic_vector(3 downto 0);
 sram32_addr: out std_logic_vector(ADDR_WIDTH-1 downto 2);
 sram32_data_I: in std_logic_vector(31 downto 0);
 sram32_data_O: out std_logic_vector(31 downto 0);
 sram32_data_T: out std_logic);
end entity;

architecture beh1 of sram_adapt is
 signal addr_be: std_logic_vector (1 downto 0);
begin
 addr_be <= sram8_addr(1 downto 0);
 sram32_Ncs <= sram8_Ncs;
 sram32_Noe <= sram8_Noe;
 sram32_Nwe <= sram8_Nwe;
 sram32_addr <= sram8_addr(ADDR_WIDTH-1 downto 2);
 sram32_data_O <= sram8_data_O&sram8_data_O&sram8_data_O&sram8_data_O;
 sram32_data_T <= sram8_data_T;

 process (sram32_data_I,addr_be) begin
 case (addr_be) is
 when "00" => sram8_data_I<=sram32_data_I(7 downto 0);
 when "01" => sram8_data_I<=sram32_data_I(15 downto 8);
 when "10" => sram8_data_I<=sram32_data_I(23 downto 16);

```

```

 when "11" => sram8_data_I<=sram32_data_I(31 downto 24);
 when others => sram8_data_I<=(others=>'Z');
 end case;
end process;
process (addr_be) begin
 case (addr_be) is
 when "00" => sram32_Nbe <= "1110";
 when "01" => sram32_Nbe <= "1101";
 when "10" => sram32_Nbe <= "1011";
 when "11" => sram32_Nbe <= "0111";
 when others => sram32_Nbe <= "1111";
 end case;
end process;
end beh1;

```

### ***Fichero pack\_maioHw.vhd***

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

package pack_maioHw is
 constant ADDR_WIDTH: integer:=21;
 constant ADDR_IMG0_UNSIGNED_24bits: unsigned(23 downto 0):=X"0F_0000";
 constant ADDR_IMG0_INTEGER: integer:=conv_integer (ADDR_IMG0_UNSIGNED_24bits);
 constant ADDR_IMG0_UNSIGNED: unsigned(ADDR_WIDTH-1 downto 0):=
 conv_unsigned(ADDR_IMG0_INTEGER,ADDR_WIDTH);
 constant ADDR_IMG0: std_logic_vector(ADDR_WIDTH-1 downto 0):=
 std_logic_vector(ADDR_IMG0_UNSIGNED);
 constant MICRO_DATA_WIDTH: integer:=32; --Valores 8 o 32
 constant SRAM_DATA_WIDTH: integer:=8; --Valor 8
 type array_pixel_med is array(0 to 14) of unsigned(9 downto 0);
end pack_maioHw;

```

### ***Fichero follow.vhd***

```

library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;
use work.pack_maioHw.all;

entity follow is
 port(
 clk: in std_logic;
 mode: in std_logic_vector(2 downto 0);
 Nrst: in std_logic;
 Nstop: out std_logic;
 w0: in signed(4 downto 0);
 x0,y0: in std_logic_vector(7 downto 0);
 w1: out signed(4 downto 0);
 x1,y1: out std_logic_vector(7 downto 0);
 emc_numbytes: in unsigned(1 downto 0);
 emc_rNw: in std_logic;
 emc_addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
 emc_din: in std_logic_vector(MICRO_DATA_WIDTH-1 downto 0);
 emc_dout: out std_logic_vector(MICRO_DATA_WIDTH-1 downto 0);
 sram_Ncs,sram_No,sram_Nwe: out std_logic;
 sram_addr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
 sram_data_I: in std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram_data_O: out std_logic_vector(SRAM_DATA_WIDTH-1 downto 0);
 sram_data_T: out std_logic
);
end entity;

```



```

architecture beh1 of follow is
signal copro_Nstop: std_logic;
signal xynxt_Nrst,xynxt_Nstop,secc_Nrst,secc_Nstop,filt_Nrst,filt_Nstop,
 corr_Nrst,corr_Nstop,xlyl_Nrst,xlyl_Nstop,tgdr_Nrst,tgdr_Nstop: std_logic;
signal secc_Ne,filt_secc_Ne,xlyl_secc_Ne: std_logic;
signal secc_addr,filt_secc_addr,xlyl_secc_addr: std_logic_vector(4 downto 0);
signal secc_dout: std_logic_vector(7 downto 0);
signal wlmax: unsigned(3 downto 0);
signal memc_numbytes: unsigned(memc_numbytes'range);
signal memc_Nrst,memc_Nstop,memc_rNw,tgdr_memc_Nrst,tgdr_memc_rNw,
 filt_memc_Nrst,filt_memc_rNw: std_logic;
signal memc_addr,tgdr_memc_addr,filt_memc_addr:
std_logic_vector(ADDR_WIDTH-1 downto 0);
signal memc_dout,memc_din: std_logic_vector(MICRO_DATA_WIDTH-1 downto 0);
signal x0_nxt,y0_nxt,x1_aux,y1_aux: std_logic_vector(7 downto 0);
signal tgdr_x0,tgdr_y0,secc_x0,secc_y0: std_logic_vector(7 downto 0);
signal tgdr_w0,tgdr_w1,secc_w0,filt_w0: signed(4 downto 0);
signal mode_iter0,mode_tgdr,mode_emc: std_logic;
signal bank: array_pixel_med;

begin
x1<=x1_aux; y1<=y1_aux;
xynxt: entity work.xynext
 port map(clk,xynxt_Nrst,xynxt_Nstop,x0,y0,w0,x0_nxt,y0_nxt);
secc: entity work.section
port map(clk,secc_Nrst,secc_Nstop,secc_w0,secc_x0,secc_y0,secc_Ne,
secc_addr, secc_dout);
filt: entity work.filtsec
port
map(clk,filt_Nrst,filt_Nstop,filt_w0,bank,filt_memc_Nrst,filt_memc_rNw,fi
lt_memc_addr,memc_dout(7 downto 0),filt_secc_Ne,filt_secc_addr,
secc_dout);
corr: entity work.corrmax port map(clk,corr_Nrst,corr_Nstop,bank,wlmax);
tgdr: entity work.tangdir
port map(clk,tgdr_Nrst,tgdr_Nstop,tgdr_x0,tgdr_y0,tgdr_w0,tgdr_w1,
tgdr_memc_Nrst,tgdr_memc_rNw,tgdr_memc_addr,memc_dout(7 downto 0));
genmemc32: if MICRO_DATA_WIDTH=32 generate
 memc: entity work.sramcontrol_8_32 generic map(ADDR_WIDTH,8,32)
 port map(clk,memc_Nrst,memc_Nstop,memc_numbytes,memc_rNw,memc_addr,
 memc_din,memc_dout,sram_Nwe,sram_Ncs,sram_Noe,sram_addr,sram_data_I
 ,sram_data_O,sram_data_T);
 end generate;
genmemc8: if MICRO_DATA_WIDTH=8 generate
 memc: entity work.sramcontrol_8_8 generic map(ADDR_WIDTH,8,8)
 port map(clk,memc_Nrst,memc_Nstop,memc_rNw,memc_addr,memc_din,
 memc_dout,sram_Nwe,sram_Ncs,sram_Noe,sram_addr,sram_data_I,sram_dat
 a_O,sram_data_T);
 end generate;

muxsecc: block begin
 secc_Ne<=filt_secc_Ne and xlyl_secc_Ne;
 secc_addr<=filt_secc_addr and xlyl_secc_addr;
end block;
muxmemc: block
 signal copro_busy: std_logic;
 signal emc_memc_Nrst,emc_memc_rNw: std_logic;
 signal emc_memc_addr,copro_busy_ext: std_logic_vector(ADDR_WIDTH-1 downto 0);
 begin
 copro_busy<=copro_Nstop and Nrst;
 copro_busy_ext<=(others=>copro_busy);
 emc_memc_Nrst<=Nrst and not copro_busy;
 emc_memc_rNw<=emc_rNw or copro_busy;

```

```

 emc_memc_addr<=emc_addr or copro_busy_ext;
 memc_numbytes<=emc_numbytes when copro_busy='0' else "00";
 memc_Nrst<=filt_memc_Nrst or tgdr_memc_Nrst or emc_memc_Nrst;
 --Nstop<=memc_Nstop when copro_busy='0' else copro_Nstop;
 memc_rNw<=filt_memc_rNw and tgdr_memc_rNw and emc_memc_rNw;
 memc_addr<=filt_memc_addr and tgdr_memc_addr and emc_memc_addr;
 memc_din<=emc_din;
 emc_dout<=memc_dout;
end block;

muxmode: block
begin
 tgdr_x0<=x1_aux when mode_tgdr='0' else x0;
 tgdr_y0<=y1_aux when mode_tgdr='0' else y0;
 tgdr_w0<=w0;
 w1<=tgdr_w1;
 secc_w0<=w0 when mode_iter0='0' else tgdr_w1;
 secc_x0<=x0_nxt when mode_iter0='0' else x0;
 secc_y0<=y0_nxt when mode_iter0='0' else y0;
 filt_w0<=w0 when mode_iter0='0' else tgdr_w1;
 Nstop<=copro_Nstop when mode_emc='0' else memc_Nstop;
end block;

xly1: process
 variable cnt: unsigned(1 downto 0);
 variable index: unsigned(3 downto 0);
begin
 wait until rising_edge(clk);
 if xly1_Nrst='0' then
 cnt:=(others=>'0');
 xly1_Nstop<='1';
 elsif xly1_Nstop='1' then
 index:=3+wlmax;
 case conv_integer(cnt) is
 when 0=> xly1_secc_addr<='0'&std_logic_vector(index);
 xly1_secc_Ne<='0';
 when 1=> xly1_secc_addr<='1'&std_logic_vector(index);
 xly1_secc_Ne<='0';
 when 2=> x1_aux<=secc_dout;
 when others=> y1_aux<=secc_dout; xly1_Nstop<='0';
 end case;
 cnt:=cnt+1;
 end if;
 if xly1_Nrst='0' or not(xly1_Nrst='1' and xly1_Nstop='1') then
 xly1_secc_Ne<='1';
 xly1_secc_addr<=(others=>'1');
 end if;
end process;

UC: process is
 type T_STATES is (STATE_MEMC, STATE_XYNXT, STATE_SECC, STATE_FILT,
 STATE_CORR, STATE_XLY1, STATE_TGDR);
 variable state: T_STATES;
begin
 wait until rising_edge(clk);
 if Nrst='0' then
 xynxt_Nrst<='0'; secc_Nrst<='0'; filt_Nrst<='0'; corr_Nrst<='0';
 xly1_Nrst<='0'; tgdr_Nrst<='0';
 mode_iter0<='0'; mode_tgdr<='0'; mode_emc<='0';
 case mode is
 when "111"=>

```

```

 state:=STATE_TGDR; copro_Nstop<='1'; mode_tgdr<='1';
when "011"=>
 state:=STATE_TGDR; copro_Nstop<='1'; mode_iter0<='1';
 mode_tgdr<='1';
when "001"|"101"=>
 state:=STATE_XYNXT; copro_Nstop<='1';
when others=>
 state:=STATE_MEMC; copro_Nstop<='0'; mode_emc<='1';
end case;
else
 if copro_Nstop='1' then
 case state is
 when STATE_MEMC=>
 state:=STATE_MEMC;
 when STATE_XYNXT=>
 xynxt_Nrst<='1';
 if xynxt_Nstop='0' then
 xynxt_Nrst<='0';
 state:=STATE_SECC;
 end if;
 when STATE_SECC=>
 secc_Nrst<='1';
 if secc_Nstop='0' then
 secc_Nrst<='0';
 state:=STATE_FILT;
 end if;
 when STATE_FILT=>
 filt_Nrst<='1';
 if filt_Nstop='0' then
 filt_Nrst<='0';
 state:=STATE_CORR;
 end if;
 when STATE_CORR=>
 corr_Nrst<='1';
 if corr_Nstop='0' then
 corr_Nrst<='0';
 state:=STATE_X1Y1;
 end if;
 when STATE_X1Y1=>
 x1y1_Nrst<='1';
 if x1y1_Nstop='0' then
 x1y1_Nrst<='0';
 if mode_iter0='0' then
 state:=STATE_TGDR;
 else
 copro_Nstop<='0';
 end if;
 end if;
 when STATE_TGDR=>
 tgdr_Nrst<='1';
 if tgdr_Nstop='0' then
 tgdr_Nrst<='0';
 if mode_iter0='0' then
 copro_Nstop<='0';
 else
 state:=STATE_SECC;
 end if;
 end if;
 when others=>
 end case;
 end if;
 end if;
end if;

```

```

 end if;
 end process;
end beh1;

```

### ***Fichero corrmax.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.pack_maioHw.all;

entity corrmax is
 port(
 clk,Nrst: in std_logic;
 Nstop: out std_logic;
 pixel_med: in array_pixel_med;
 wlmax: out unsigned(3 downto 0));
end corrmax;

architecture beh1 of corrmax is
 signal Nstop_aux: std_logic;
 signal max_stop: std_logic;
 signal clkcnt: unsigned(1 downto 0);
 signal j: unsigned(2 downto 0);
 signal j_is0,j_is0_1,j_is0_2,j_is0_3,j_is0_4: std_logic;
 signal k,jk: unsigned(3 downto 0); --k_prv
 signal d: unsigned(2 downto 0);
 signal down: std_logic;
 signal corr,acc,acc_2,acc_3,acc_4: unsigned(14 downto 0);
 --Valor máximo=(2*1+2*2*2*4+1*8)*(3*$FF)=$41BE (15-bits)
 signal carry: std_logic;
 signal pixel: unsigned(9 downto 0);
 signal xh,xh_2: unsigned(9+3 downto 0);
 --10 bits + 3 bits desplazamiento (multiplicar *8) (13-bits)
 subtype t_filter_cte is unsigned(1 downto 0);
 signal filter_cte: t_filter_cte;
 constant *1\: t_filter_cte:="00";
 constant *2\: t_filter_cte:="01";
 constant *4\: t_filter_cte:="10";
 constant *8\: t_filter_cte:="11";
 type t_filter is array(0 to 6) of t_filter_cte;
 constant filter: t_filter:=(*1\,*2\,*4\,*8\,*4\,*2\,*1\);
begin
 Nstop<=Nstop_aux;

 INDEX: process
 variable kr,kl: unsigned(k'range);
 begin
 wait until rising_edge(clk);
 if Nrst='0' then
 Nstop_aux<='1';
 clkcnt<=(others=>'0');
 j<=(others=>'0');
 j_is0<='1';
 k<=conv_unsigned(4,4);
 d<=conv_unsigned(0,3);
 jk<=conv_unsigned(4,4);
 kr:=conv_unsigned(4,4); kl:=conv_unsigned(4,4);
 down<='1';
 elsif Nstop_aux='1' then
 clkcnt<=clkcnt+1;
 Nstop_aux<=not max_stop;
 end if;
 end process;
end beh1;

```

```

 if clkcnt=2 then
 clkcnt<=(others=>'0');
 j<=j+1;
 j_is0<='0';
 jk<=jk+1;
 if j=6 then
 j<=(others=>'0');
 j_is0<='1';
 if down='1' then
 kl:=kl-1;
 k<=kl;
 jk<=kl;
 else
 kr:=kr+1;
 k<=kr;
 jk<=kr;
 end if;
 if down='0' then d<=d+1; end if;
 down<=not down;
 if k=8 then
 k<=conv_unsigned(4,4);
 jk<=conv_unsigned(4,4);
 end if;
 end if;
 end if;
 end if;
end process;

PIPE0: process
begin
 wait until rising_edge(clk);
 pixel<=pixel_med(conv_integer(jk));
 filter_cte<=filter(conv_integer(j));
 j_is0_1<=j_is0;
end process;

PIPE1: process
begin
 wait until rising_edge(clk);
 case filter_cte is
 when *2\=>
 xh<="00"&pixel&"0";
 when *4\=>
 xh<="0"&pixel&"00";
 when *8\=>
 xh<=pixel&"000";
 when others=>
 xh<="000"&pixel;
 end case;
 j_is0_2<=j_is0_1;
end process;

PIPE2: process
begin
 wait until rising_edge(clk);
 acc(14 downto 0)<=(others=>'0');
 if j_is0_2='1' then
 acc(7 downto 0)<=xh(7 downto 0);
 else
 acc(7 downto 0)<=acc_3(7 downto 0)+xh(7 downto 0);
 end if;
end process;

```

```

 xh_2<=xh;
 acc_4<=acc_3;
 j_is0_3<=j_is0_2;
end process;

PIPE3: process
begin
 wait until rising_edge(clk);
 acc_2(7 downto 0)<=acc(7 downto 0);
 if j_is0_3='1' then
 acc_2(14 downto 8)<=("00"& xh_2(12 downto 8));
 carry<='0';
 else
 acc_2(14 downto 8)<=acc_4(14 downto 8)+("00"& xh_2(12
 downto 8));
 carry<=(acc_4(7) and xh_2(7))or(not acc(7) and
 acc_4(7))or(not acc(7) and xh_2(7)) ;
 end if;
 j_is0_4<=j_is0_3;
end process;

PIPE4: process
variable acc_3aux: unsigned(acc_3'range);
begin
 wait until rising_edge(clk);
 acc_3aux(7 downto 0):=acc_2(7 downto 0);
 if carry='1' then
 acc_3aux(14 downto 8):=acc_2(14 downto 8)+1;
 else
 acc_3aux(14 downto 8):=acc_2(14 downto 8);
 end if;
 acc_3<=acc_3aux;
 if j_is0_1='1' then
 corr<=acc_3aux;

 end if;
end process;

PIPE5: process
type t_max is array(0 to 1) of unsigned(corr'range);
type t_dist is array(0 to 1) of unsigned(2 downto 0);
type t_cont is array(0 to 1) of boolean;
variable max: t_max;
variable dist: t_dist;
variable found,cont: t_cont;
variable lr: unsigned(0 downto 0);
variable sub: signed(corr'left+1 downto 0);
variable a,b,c: unsigned(7 downto 0);
variable borrow: std_logic;
variable max0: unsigned(corr'range);
variable phase: unsigned(2 downto 0);
variable max_rst,max_start: std_logic;
variable max_break,wlmax0: boolean;
begin
 wait until rising_edge(clk);
 if max_start='1' then
 max0:=max(conv_integer(lr));
 case phase is
 when "000"=>
 a:=corr(7 downto 0);
 b:=max0(7 downto 0);

```

```

 c:=a-b;
 sub(7 downto 0):=signed(c);
when "001"=>
 borrow:=not((a(7) and not b(7))or(not c(7)
 and a(7))or(not c(7) and not b(7)));
 a:='0'&corr(14 downto 8);
 b:='0'&max0(14 downto 8);
 c:=a-b;
 sub(15 downto 8):=signed(c);
when "010"=>
 a:=c;
 b:"0000000"&borrow;
 c:=a-b;
 sub(15 downto 8):=signed(c);
when "011"=>
 if cont(conv_integer(lr)) then
 if sub>0 then
 max(conv_integer(lr)):=corr;
 dist(conv_integer(lr)):=d;
 found(conv_integer(lr)):=true;
 elsif sub<0 then
 cont(conv_integer(lr)):=false;
 end if;
 end if;
 max_break:=(not cont(0) and not
 cont(1))or(found(0) and not
 cont(0))or(found(1) and not cont(1));
 if lr=0 then
 max_break:=false;
 elsif d=4 then
 max_break:=true;
 end if;
when "100"=>
 a:=max(0)(7 downto 0);
 b:=max(1)(7 downto 0);
 c:=a-b;
 sub(7 downto 0):=signed(c);
when "101"=>
 borrow:=not((a(7) and not b(7))or(not c(7)
 and a(7))or(not c(7) and not b(7)));
 a:='0'&max(0)(14 downto 8);
 b:='0'&max(1)(14 downto 8);
 c:=a-b;
 sub(15 downto 8):=signed(c);
when "110"=>
 a:=c;
 b:"0000000"&borrow;
 c:=a-b;
 sub(15 downto 8):=signed(c);
when "111"=>
 max_stop<='1';
 if cont(1) and found(0) and not cont(0) then
 wlmax<=4-('0'&dist(0));
 elsif cont(0) and found(1) and not cont(1) then
 wlmax<=4+('0'&dist(1));
 else
 if sub>=0 then
 wlmax<=4-('0'&dist(0));
 else
 wlmax<=4+('0'&dist(1));
 end if;
 end if;
end if;

```

```

 end if;
 end if;
 when others=>
 end case;
 end if;
 lr:=(others=>not down);
 if Nrst='0' then
 max_start:='0';
 max_stop<='0';
 max_rst:='0';
 elsif max_start='0' then
 if max_rst='0' then
 if d=1 and j=0 and clkcnt=0 then
 cont(0):=true; cont(1):=true;
 found(0):=false; found(1):=false;
 dist(0):=(others=>'0');
 dist(1):=(others=>'0');
 max(0):=corr; max(1):=corr;
 max_rst:='1';
 end if;
 elsif d/=0 and j=0 and clkcnt=1 then
 max_start:='1';
 phase:=(others=>'0');
 end if;
 else
 if (std_logic_vector(phase)="111" or
 (std_logic_vector(phase)="011" and not max_break))
 then
 max_start:='0';
 phase:=(others=>'0');
 else
 phase:=phase+1;
 end if;
 end if;
 end process;
 end architecture;

```

### ***Fichero filtsec.vhd***

```

library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;
use work.pack_maioHw.all;

entity filtsec_core is
 port(
 clk,Nrst: in std_logic;
 Nstop: out std_logic;
 w: in signed(4 downto 0);
 pixel_med: out array_pixel_med;
 memc_Nrst,memc_rNw: out std_logic;
 memc_addr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
 memc_dout: in std_logic_vector(7 downto 0);
 secc_Ne: out std_logic;
 secc_addr: out std_logic_vector(4 downto 0);
 secc_dout: in std_logic_vector(7 downto 0));
end entity;

architecture beh1 of filtsec_core is
 signal status: unsigned(1 downto 0);
 signal k: unsigned(1 downto 0);
 signal iter,iter2: signed(4 downto 0); --iteración -1 a 0:14
 signal xoff,yoff: unsigned(7 downto 0);

```



```

signal table1_delta_x,table1_delta_y,table2_delta_x,table2_delta_y:
signed(1 downto 0);
signal table_delta_w: signed(4 downto 0);
constant \+0\: signed(1 downto 0):=conv_signed(0,2);
constant \+1\: signed(1 downto 0):=conv_signed(1,2);
constant \-1\: signed(1 downto 0):=conv_signed(-1,2);
constant \+X\: signed(1 downto 0):=(others=>'X');
signal Nstop_aux,running: std_logic;
begin
 Nstop<=Nstop_aux;
 running<=Nrst and Nstop_aux;

 UC: process
 begin
 wait until rising_edge(clk);
 if Nrst='0' then
 iter<=conv_signed(-1,5);
 iter2<=conv_signed(0,5);
 status<=(others=>'0');
 k<=(others=>'0');
 Nstop_aux<='1';
 elsif Nstop_aux='1' then
 k<=k+1;
 if k=3 then
 if status=2 then
 status<=(others=>'0');
 iter<=iter+1;
 iter2<=iter2+1;
 if iter=14 then
 Nstop_aux<='0';
 end if;
 else
 status<=status+1;
 end if;
 end if;
 end if;
 end process;

 PIPE0: process
 variable addr_iter2: std_logic_vector(iter2'left-1 downto 0);
 begin
 wait until rising_edge(clk);
 if status=0 and iter2/=15 then

 addr_iter2:=std_logic_vector(iter2(addr_iter2'range));
 case conv_integer(k) is
 when 0=> secc_addr<='0'&addr_iter2; secc_Ne<='0';
 when 1=> secc_addr<='1'&addr_iter2; secc_Ne<='0';
 when 2=> xoff<=unsigned(secc_dout);
 when others=> yoff<=unsigned(secc_dout);
 end case;
 end if;

 if running/= '1' then
 secc_addr<=(others=>'1'); secc_Ne<='1';
 end if;
 end process;

 PIPE1A: process(table_delta_w)
 begin
 case conv_integer(table_delta_w) is

```

```

when 0 => table1_delta_x<=\+1\; table1_delta_y<=\+0\;
--1Q
when 1 => table1_delta_x<=\+1\; table1_delta_y<=\+0\;
when 2 => table1_delta_x<=\+1\; table1_delta_y<=\-1\;
when 3 => table1_delta_x<=\+1\; table1_delta_y<=\-1\;
when 4 => table1_delta_x<=\+1\; table1_delta_y<=\-1\;
when 5 => table1_delta_x<=\+0\; table1_delta_y<=\-1\;

when 6 => table1_delta_x<=\+0\; table1_delta_y<=\-1\;
--2Q
when 7 => table1_delta_x<=\+0\; table1_delta_y<=\-1\;
when 8 => table1_delta_x<=\-1\; table1_delta_y<=\-1\;
when 9 => table1_delta_x<=\-1\; table1_delta_y<=\-1\;
when 10 => table1_delta_x<=\-1\; table1_delta_y<=\-1\;
when 11 => table1_delta_x<=\-1\; table1_delta_y<=\+0\;

when 12 => table1_delta_x<=\-1\; table1_delta_y<=\+0\;
--3Q
when -11=> table1_delta_x<=\-1\; table1_delta_y<=\+0\;
when -10=> table1_delta_x<=\-1\; table1_delta_y<=\+1\;
when -9 => table1_delta_x<=\-1\; table1_delta_y<=\+1\;
when -8 => table1_delta_x<=\-1\; table1_delta_y<=\+1\;
when -7 => table1_delta_x<=\+0\; table1_delta_y<=\+1\;

when -6 => table1_delta_x<=\+0\; table1_delta_y<=\+1\;
--4Q
when -5 => table1_delta_x<=\+0\; table1_delta_y<=\+1\;
when -4 => table1_delta_x<=\+1\; table1_delta_y<=\+1\;
when -3 => table1_delta_x<=\+1\; table1_delta_y<=\+1\;
when -2 => table1_delta_x<=\+1\; table1_delta_y<=\+1\;
when -1 => table1_delta_x<=\+1\; table1_delta_y<=\+0\;

when others=> table1_delta_x<=\+X\;
table1_delta_y<=\+X\;
end case;
end process;

PIPE1B: process(table_delta_w)--simplificar ya que es la misma
tabla con diferentes indices
begin
case conv_integer(table_delta_w) is
when 12 => table2_delta_x<=\+1\; table2_delta_y<=\+0\;
--1Q
when -11=> table2_delta_x<=\+1\; table2_delta_y<=\+0\;
when -10=> table2_delta_x<=\+1\; table2_delta_y<=\-1\;
when -9 => table2_delta_x<=\+1\; table2_delta_y<=\-1\;
when -8 => table2_delta_x<=\+1\; table2_delta_y<=\-1\;
when -7 => table2_delta_x<=\+0\; table2_delta_y<=\-1\;

when -6 => table2_delta_x<=\+0\; table2_delta_y<=\-1\;
--2Q
when -5 => table2_delta_x<=\+0\; table2_delta_y<=\-1\;
when -4 => table2_delta_x<=\-1\; table2_delta_y<=\-1\;
when -3 => table2_delta_x<=\-1\; table2_delta_y<=\-1\;
when -2 => table2_delta_x<=\-1\; table2_delta_y<=\-1\;
when -1 => table2_delta_x<=\-1\; table2_delta_y<=\+0\;

```

```

when 0 => table2_delta_x<=\-1\; table2_delta_y<=\+0\;
--3Q
when 1 => table2_delta_x<=\-1\; table2_delta_y<=\+0\;
when 2 => table2_delta_x<=\-1\; table2_delta_y<=\+1\;
when 3 => table2_delta_x<=\-1\; table2_delta_y<=\+1\;
when 4 => table2_delta_x<=\-1\; table2_delta_y<=\+1\;
when 5 => table2_delta_x<=\+0\; table2_delta_y<=\+1\;

when 6 => table2_delta_x<=\+0\; table2_delta_y<=\+1\;
--4Q
when 7 => table2_delta_x<=\+0\; table2_delta_y<=\+1\;
when 8 => table2_delta_x<=\+1\; table2_delta_y<=\+1\;
when 9 => table2_delta_x<=\+1\; table2_delta_y<=\+1\;
when 10 => table2_delta_x<=\+1\; table2_delta_y<=\+1\;
when 11 => table2_delta_x<=\+1\; table2_delta_y<=\+0\;

when others=> table2_delta_x<=\+X\;
 table2_delta_y<=\+X\;
 end case;
end process;

PIPE1C:process
variable delta_x,delta_y: signed(1 downto 0);
variable addr_pixel: std_logic_vector(15 downto 0);
variable xoff0,yoff0: unsigned(xoff'range);
variable x,y: unsigned(8 downto 0);
begin
 table_delta_w<=w;
 wait until rising_edge(clk);
 case conv_integer(k) is
 when 0=>
 case conv_integer(status) is
 when 0=>
 delta_x:=table2_delta_x; delta_y:=table2_delta_y;
 when 1=>
 delta_x:=\+0\; delta_y:=\+0\;
 when others=>
 delta_x:=table1_delta_x; delta_y:=table1_delta_y;
 end case;
 when 1=>
 case conv_integer(delta_x) is
 when 1=> x:=('0' & xoff)+1;
 when -1=> x:=('0' & xoff)-1;
 when others=> x:=('0' & xoff);
 end case;
 case conv_integer(delta_y) is
 when 1=> y:=('0' & yoff)+1;
 when -1=> y:=('0' & yoff)-1;
 when others=> y:=('0' & yoff);
 end case;
 assert (x(8)/='1' and y(8)/='1') report "###Overflow.
 Truncado MSB en addr_pixel" severity WARNING;
 when 2=>
 addr_pixel(15 downto 8):=std_logic_vector(y(7 downto 0));
 addr_pixel(7 downto 0):=std_logic_vector(x(7 downto 0));
 memc_addr(15 downto 0)<=addr_pixel;
 memc_addr(ADDR_WIDTH-1 downto 16)<=
 ADDR_IMG0(ADDR_WIDTH-1 downto 16);
 end process;

```

```

 when others=>
 end case;
 if running/='1' then
 memc_addr<=(others=>'1');
 end if;
 end process;

 PIPE2: process
 begin
 memc_rNw<='1';
 wait until rising_edge(clk);
 case conv_integer(k) is
 when 2=> memc_Nrst<='0';
 when others=> memc_Nrst<='1';
 end case;
 if running/='1' then
 memc_Nrst<='0';
 end if;
 end process;

 PIPE3: process
 variable pixel_acc: unsigned(9 downto 0);
 begin
 wait until rising_edge(clk);
 if k=3 then
 pixel_acc:=pixel_acc+unsigned(memc_dout);
 elsif k=2 and status=2 then
 if iter>=0 then
 pixel_med(conv_integer(unsigned(iter(3 downto
 0))))<=pixel_acc;
 end if;
 pixel_acc:=(others=>'0');
 end if;
 end process;
end beh1;
--*****
library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;
use work.pack_maioHw.all;

entity filtsec is
 port(
 clk,Nrst: in std_logic;
 Nstop: out std_logic;
 w: in signed(4 downto 0);
 bank: out array_pixel_med;
 memc_Nrst,memc_rNw: out std_logic;
 memc_addr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
 memc_dout: in std_logic_vector(7 downto 0);
 secc_rNw: out std_logic;
 secc_addr: out std_logic_vector(4 downto 0);
 secc_dout: in std_logic_vector(7 downto 0));
end entity;

architecture beh1 of filtsec is
 signal bank_int,bank_out: array_pixel_med;
 begin
 core: entity work.filtsec_core
 port map(clk,Nrst,Nstop,w,bank_out,memc_Nrst,memc_rNw,memc_addr,
 memc_dout,secc_rNw,secc_addr,secc_dout);
 bank_int<=bank_out;

```

```

 bank<=bank_int;
end architecture;

```

### ***Fichero nextpoint.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity nextpoint is
 port(
 clk,Nrst: in std_logic;
 Nstop: out std_logic;
 inc: in unsigned(2 downto 0);
 w: in signed(4 downto 0);
 incx,incy: out signed(3 downto 0));
end nextpoint;

architecture beh1 of nextpoint is
constant \0\: unsigned(2 downto 0):=conv_unsigned(0,3);
constant \1\: unsigned(2 downto 0):=conv_unsigned(1,3);
constant \2\: unsigned(2 downto 0):=conv_unsigned(2,3);
constant \3\: unsigned(2 downto 0):=conv_unsigned(3,3);
constant \4\: unsigned(2 downto 0):=conv_unsigned(4,3);
constant \5\: unsigned(2 downto 0):=conv_unsigned(5,3);
constant \6\: unsigned(2 downto 0):=conv_unsigned(6,3);
constant \7\: unsigned(2 downto 0):=conv_unsigned(7,3);
--constant \8\: unsigned(3 downto 0):=conv_unsigned(8,4);
type TABLE_DELTA is array(0 to 6) of unsigned(2 downto 0);
constant DELTA0: TABLE_DELTA:=(\0\,\0\,\0\,\0\,\0\,\0\,\0\);
constant DELTA1: TABLE_DELTA:=(\1\,\1\,\1\,\1\,\1\,\0\,\0\);
constant DELTA2: TABLE_DELTA:=(\2\,\2\,\2\,\2\,\1\,\1\,\0\);
constant DELTA3: TABLE_DELTA:=(\3\,\3\,\3\,\3\,\1\,\1\,\0\);
constant DELTA4: TABLE_DELTA:=(\4\,\4\,\3\,\4\,\2\,\1\,\0\);
constant DELTA5: TABLE_DELTA:=(\5\,\5\,\4\,\5\,\3\,\1\,\0\);
constant DELTA6: TABLE_DELTA:=(\6\,\6\,\5\,\6\,\3\,\2\,\0\);
constant DELTA7: TABLE_DELTA:=(\7\,\7\,\6\,\7\,\4\,\2\,\0\);
--constant DELTA8: TABLE_DELTA:=(\8\,\8\,\7\,\8\,\4\,\2\,\0\);
type TABLES_DELTA is array(0 to 7) of TABLE_DELTA;
constant DELTAS:
TABLES_DELTA:=(DELTA0,DELTA1,DELTA2,DELTA3,DELTA4,DELTA5,DELTA6,DELTA7);

signal state: std_logic;
signal qindex: unsigned(w'left-1 downto 0);
signal index: unsigned(w'left-2 downto 0);
signal q23,q34: boolean;
signal deltarom: unsigned(2 downto 0);
begin
 PIPE0: process(clk)
 variable w_sig: signed(w'range);
 variable w_abs0: unsigned(w'range); --desde -11 to -1 a 0 to +12 =>
 0 to +12 (0° to 180°)
 variable w_abs: unsigned(w'left-1 downto 0);--0 to +6 (0° to 90°)
 begin
 w_sig:=abs(signed(w));
 w_abs0:=unsigned(w_sig);
 w_abs:=w_abs0(w_abs'range);
 if rising_edge(clk) then
 qindex<=w_abs;
 q23<=w_abs>6;
 q34<=signed(w)<0;
 end if;
 end
 end process PIPE0;

```

```

end process;

PIPE1: process(clk)
variable indexXY, indexX, indexY: unsigned(qindex'range);
begin
 case q23 is
 when true=> indexX:=12-qindex; indexY:=qindex-6;
 when others=> indexX:=qindex; indexY:=6-qindex;
 end case;
 case state is
 when '0'=> indexXY:=indexX;
 when others=> indexXY:=indexY;
 end case;
 if rising_edge(clk) then
 index<=indexXY(index'range);
 end if;
end process;

PIPE2: process(clk)
variable table: TABLE_DELTA;
begin
 table:=DELTAS(conv_integer(inc));
 if rising_edge(clk) then
 deltarom<=table(conv_integer(index));
 end if;
end process;

PIPE3: process(clk)
variable deltarom0: signed(3 downto 0);
variable deltaX, deltaY: signed(3 downto 0);
begin
 deltarom0:=signed('0'&std_logic_vector(deltarom));
 case q34 is
 when true=> deltaY:=deltarom0;
 when others=> deltaY:=-deltarom0;
 end case;
 case q23 is
 when true=> deltaX:=-deltarom0;
 when others=> deltaX:=deltarom0;
 end case;
 if rising_edge(clk) then
 case state is
 when '0'=> incx<=deltaX;
 when others=> incy<=deltaY;
 end case;
 end if;
end process;

UC: process
variable nxtstate: std_logic;
variable nxtstage: unsigned(1 downto 0);
begin
 wait until rising_edge(clk);
 if Nrst='0' then
 nxtstate:='0';
 nxtstage:=(others=>'0');
 Nstop<='1';
 else
 nxtstate:=not nxtstate;
 nxtstage:=nxtstage+1;
 if nxtstage=0 then --if nxtstage=3 and

```

```

 nxtstate='0' then
 Nstop<='0';
 end if;
 end if;
 state<=nxtstate;
 end process;
end architecture;

```

### ***Fichero section.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity section_ram is
 port(
 clk,rNw: in std_logic;
 addr: in std_logic_vector(4 downto 0);
 din: in std_logic_vector(7 downto 0);
 dout: out std_logic_vector(7 downto 0));
end entity;

architecture beh1 of section_ram is
 type RAMXY is array(0 to 14) of std_logic_vector(7 downto 0);
 signal ramx,ramy: RAMXY;
begin
 pram: process(clk,din,addr,rNw,ramx,ramy)
 variable addr0: unsigned(addr'left-1 downto 0);
 variable addrXY: std_logic;
 begin
 addrXY:=addr(addr'left);
 addr0:=unsigned(addr(addr0'range));
 if rising_edge(clk) then
 if rNW='0' then
 if addrXY='0' then
 ramx(conv_integer(addr0))<=din;
 else
 ramy(conv_integer(addr0))<=din;
 end if;
 else
 if addrXY='0' then
 dout<=ramx(conv_integer(addr0));
 else
 dout<=ramy(conv_integer(addr0));
 end if;
 end if;
 end if;
 end process;
end architecture;
--*****
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity section_core is
 port(
 clk,Nrst: in std_logic;
 Nstop: out std_logic;
 w: in signed(4 downto 0);
 x0,y0: in std_logic_vector(7 downto 0);
 rNw: out std_logic;
 addr: out std_logic_vector(4 downto 0);

```

```

 dout: out std_logic_vector(7 downto 0));
end entity;

architecture beh1 of section_core is
 signal i,i2: unsigned(2 downto 0);
 signal left,ycoord: std_logic;
 signal NP1_Nrst,NP1_Nstop: std_logic;
 signal NP1_w: signed(w'range);
 signal NP1_inc: unsigned(2 downto 0);
 signal incx,incy,NP1_incx,NP1_incy: signed(3 downto 0);
 signal Nstop_aux: std_logic;
begin
 Nstop<=Nstop_aux;
 NP1: entity work.nextpoint port map(clk,NP1_Nrst,NP1_Nstop,NP1_inc,
 NP1_w,NP1_incx,NP1_incy);

 PIPE0: process(clk)
 variable wtang: signed(w'range);
 begin
 if w>0 then
 wtang:=w-6;
 else
 wtang:=w+6;
 end if;
 if rising_edge(clk) then
 if Nrst='0' then
 NP1_w<=wtang;
 end if;
 end if;
 end process;

 PIPE1: process
 variable runNP1: boolean;
 begin
 wait until rising_edge(clk);
 if Nrst='0' then
 runNP1:=false;
 NP1_Nrst<='0';
 else
 if not runNP1 then
 runNP1:=true;
 NP1_inc<=i;
 NP1_Nrst<='1';
 elsif NP1_Nstop='0' then
 runNP1:=false;
 NP1_Nrst<='0';
 incx<=NP1_incx; incy<=NP1_incy;
 i2<=i;
 end if;
 end if;
 end process;

 PIPE2: process(clk)
 variable op1: signed(x0'range);
 variable op2: signed(NP1_incx'range);
 variable xy0: signed(7 downto 0);
 variable xy: unsigned(7 downto 0);
 variable idx: signed(3 downto 0);
 variable index0: signed(3 downto 0);
 variable xyrl: std_logic_vector(0 to 1);
 variable addr0: std_logic_vector(addr'range);

```



```

begin
 xyrl:=ycoord&left;
 case xyrl is
 when "00"=>op1:=signed(x0);op2:=incx;idx:=signed('0'&i2);
 when "01"=>op1:=signed(x0);op2:=-incx;idx:=-signed('0'&i2);
 when "10"=>op1:=signed(y0);op2:=incy;idx:=signed('0'&i2);
 when others=>op1:=signed(y0);op2:=-incy;idx:=-signed('0'&i2);
 end case;
 xy0:=op1+op2; xy:=unsigned(xy0);
 index0:=7+idx; --index:=unsigned(index0);
 addr0:=ycoord&std_logic_vector(index0);
 if rising_edge(clk) then
 addr<=addr0;
 dout<=std_logic_vector(xy);
 end if;
end process;

PIPE3: process
begin
 wait until rising_edge(clk);
 if Nstop_aux='1' then
 rNw<='0';
 else
 rNw<='1';
 end if;
end process;

UC: process
variable firstcycle: std_logic;
begin
 wait until rising_edge(clk);
 if Nrst='0' then
 i<=(others=>'0');
 ycoord<='0';
 left<='0';
 Nstop_aux<='1';
 firstcycle:='0';
 elsif Nstop_aux='1' and not(NP1_Nstop='1' and ycoord='1' and
 left='1') then
 ycoord<=not ycoord;
 if ycoord='1' then
 left<=not left;
 if left='1' then
 i<=i+1;
 if i=0 then
 firstcycle:=not firstcycle;
 if firstcycle='0' then
 Nstop_aux<='0';
 end if;
 end if;
 end if;
 end if;
 end if;
end process;
end architecture;

library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;

entity section is
 port(
 clk,Nrst: in std_logic;

```

```

 Nstop: out std_logic;
 w: in signed(4 downto 0);
 x0,y0: in std_logic_vector(7 downto 0);
 secc_Ne: in std_logic;
 secc_addr: in std_logic_vector(4 downto 0);
 secc_dout: out std_logic_vector(7 downto 0));
end entity;

architecture beh1 of section is
 signal core_Nstop: std_logic;
 signal ram_rNw,core_rNw: std_logic;
 signal ram_addr,core_addr: std_logic_vector(secc_addr'range);
 signal ram_din,ram_dout,core_dout: std_logic_vector(secc_dout'range);
begin

 Nstop<=core_Nstop;
 core: entity work.section_core port map(clk,Nrst,core_Nstop,w,
 x0,y0,core_rNw,core_addr,core_dout);
 ram: entity work.section_ram port map(clk,ram_rNw,ram_addr,
 ram_din,ram_dout);
 p1: process(secc_Ne,core_Nstop,secc_addr,ram_dout,core_rNw,
 core_addr,core_dout)
 begin
 if secc_Ne='0' and Nrst='0' then--core_Nstop='0' then
 ram_rNw<='1';
 ram_addr<=secc_addr;
 ram_din<=(others=>'X');
 secc_dout<=ram_dout;
 else
 ram_rNw<=core_rNw;
 ram_addr<=core_addr;
 ram_din<=core_dout;
 secc_dout<=(others=>'X');
 end if;
 end process;
end architecture;

```

### ***Fichero tangdir.vhd***

```

library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;
use work.pack_maioHw.all;

entity tangdir is
 port(
 clk,Nrst: in std_logic;
 Nstop: out std_logic;
 x0,y0: in std_logic_vector(7 downto 0);
 w0: in signed(4 downto 0);
 w1: out signed(4 downto 0);
 memc_Nrst,memc_rNw: out std_logic;
 memc_addr: out std_logic_vector(ADDR_WIDTH-1 downto 0);
 memc_dout: in std_logic_vector(7 downto 0));
end tangdir;

architecture beh1 of tangdir is
 signal iter,iter_2: unsigned(3 downto 0);
 signal status: unsigned(2 downto 0);
 signal k: unsigned(1 downto 0);
 signal aver: std_logic;
 signal Nstop_aux,running: std_logic;
 signal pixelacc: unsigned(9 downto 0);

```

```

subtype t_inc is signed(2 downto 0);
constant \+0\: t_inc:=conv_signed(0,3);
constant \+1\: t_inc:=conv_signed(1,3);
constant \-1\: t_inc:=conv_signed(-1,3);
constant \+2\: t_inc:=conv_signed(2,3);
constant \-2\: t_inc:=conv_signed(-2,3);
constant \+3\: t_inc:=conv_signed(3,3);
constant \-3\: t_inc:=conv_signed(-3,3);
type t_rom_inc is array(0 to 32*2-1) of t_inc;
constant rom_inc_1_2: t_rom_inc:=
 \+2\,\+0\, \+3\,\+0\,
 \+2\,\+0\, \+2\,\-1\, \+3\,\-1\,
 \+2\,\-1\, \+2\,\-2\, \+3\,\-2\,
 \+2\,\-2\, \+3\,\-3\,
 \+1\,\-2\, \+2\,\-2\, \+2\,\-3\,
 \+0\,\-2\, \+1\,\-2\, \+1\,\-3\,
 \+0\,\-2\, \+0\,\-3\,
 \+0\,\+2\, \+1\,\+2\, \+1\,\+3\,
 \+1\,\+2\, \+2\,\+2\, \+2\,\+3\,
 \+2\,\+2\, \+3\,\+3\,
 \+2\,\+1\, \+2\,\+2\, \+3\,\+2\,
 \+2\,\+0\, \+2\,\+1\, \+3\,\+1\);
type t_rom_aver is array(0 to 12) of std_logic;
constant rom_aver_1_2:
t_rom_aver:=('0','1','1','0','1','1','0','1','1','0','1','1','X');
begin
 Nstop<=Nstop_aux;
 running<=Nrst and Nstop_aux;
 STATES: process
 begin
 wait until rising_edge(clk);
 if Nrst='0' then
 k<=(others=>'0');
 status<=(others=>'0');
 iter<=(others=>'0');
 iter_2<=(others=>'1');
 aver<=rom_aver_1_2(conv_integer(0));
 Nstop_aux<='1';
 elsif Nstop_aux='1' then
 k<=k+1;
 if k=3 then
 status<=status+1;
 if (status=3 and aver='0') or (status=5 and
 aver='1') then
 status<=(others=>'0');
 iter<=iter+1;
 iter_2<=iter;
 aver<=rom_aver_1_2(conv_integer(iter+1));
 end if;
 elsif iter=12 and status=1 then --and k=1 then
 Nstop_aux<='0';
 end if;
 end if;
 end process;

 PIPE0: process
 variable index: unsigned(5 downto 0);
 variable xinc,yinc: t_inc;
 variable inc8:signed(7 downto 0);
 variable x,y: unsigned(7 downto 0);
 variable addr_pixel: std_logic_vector(15 downto 0);

```

```

begin
 wait until rising_edge(clk);
 if Nrst='0' then
 index:=(others=>'0');
 memc_addr<=(others=>'1');
 else --elsif iter>=0 then
 case conv_integer(k) is
 when 0=>--1=>
 if status(0)='0' then
 xinc:=rom_inc_1_2(conv_integer(index));
 inc8:=conv_signed(xinc,8);
 index:=index+1;
 else
 inc8:=-conv_signed(xinc,8);
 end if;
 x:=unsigned(x0)+unsigned(inc8);

 when 1=>--2=>
 if status(0)='0' then
 yinc:=rom_inc_1_2(conv_integer(index));
 inc8:=conv_signed(yinc,8);
 index:=index+1;
 else
 inc8:=-conv_signed(yinc,8);
 end if;
 y:=unsigned(y0)+unsigned(inc8);
 when others=>
 addr_pixel(15 downto 8):=std_logic_vector(y);
 addr_pixel(7 downto 0):=std_logic_vector(x);
 memc_addr(15 downto 0)<=addr_pixel;
 memc_addr(ADDR_WIDTH-1 downto
 16)<=ADDR_IMG0(ADDR_WIDTH-1 downto 16);
 end case;
 if running/='1' then
 memc_addr<=(others=>'1');
 end if;
 end if;
 end process;

PIPE1: process
begin
 memc_rNw<='1';
 wait until rising_edge(clk);
 case conv_integer(k) is
 when 2=> memc_Nrst<='0';
 when others=> memc_Nrst<='1';
 end case;
 if running/='1' then
 memc_Nrst<='0';
 end if;
end process;

PIPE2: process
variable acc: unsigned(9 downto 0);
begin
 wait until rising_edge(clk);
 if k=3 then
 case conv_integer(status) is
 when 1=>
 acc:="00"&unsigned(memc_dout);
 when 4=>

```

```

 acc:=acc+unsigned(memc_dout);
 acc:=shr(acc,"1");
 when others=>
 acc:=acc+unsigned(memc_dout);
 end case;
 pixelacc<=acc;
end if;
end process;

PIPE3: process
variable angleacc,max: unsigned(9 downto 0);
variable angle: unsigned(3 downto 0);
variable wlaux: signed(w1'range);
variable abs0: std_logic_vector(w1'range);
begin
 wait until rising_edge(clk);
 if Nrst='0' then
 max:=(others=>'0');
 --w1<=(others=>'1');
 elsif status=1 and iter_2/=15 then
 case conv_integer(k) is
 when 0=>
 angleacc:=pixelacc;
 if angleacc>max then
 max:=angleacc;
 angle:=unsigned(iter_2);
 end if;
 when 1=>
 wlaux:=signed('0'&std_logic_vector(angle));
 abs0:=abs(wlaux-w0);
 if unsigned(abs0)<=6 then
 w1<=wlaux;
 else
 if angle=0 then
 w1<=conv_signed(12,w1'length);
 else
 w1<=wlaux-12;
 end if;
 end if;
 when others=>
 null;
 end case;
 end if;
end process;
end architecture;

```

### ***Fichero xynext.vhd***

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity xynext is
 port(
 clk,Nrst: in std_logic;
 Nstop: out std_logic;
 x0,y0: in std_logic_vector(7 downto 0);
 w: in signed(4 downto 0);
 x1,y1: out std_logic_vector(7 downto 0));
end entity;

architecture beh1 of xynext is

```

```

constant INC_3: unsigned(2 downto 0):=conv_unsigned(3,3);
constant INC_4: unsigned(2 downto 0):=conv_unsigned(4,3);
signal Nstop_aux: std_logic;
signal nxtp_Nrst,nxtp_Nstop: std_logic;
signal incx,incy: signed(3 downto 0);
begin
 Nstop<=Nstop_aux;
 nxtp_Nrst<=Nrst;
 nxtp: entity work.nextpoint port map(clk,nxtp_Nrst,nxtp_Nstop,
 INC_4,w,incx,incy);
 UCUP: process
 variable cnt: std_logic;
 variable nxt,op0: signed(x0'range);
 variable op1: signed(incx'range);
 begin
 wait until rising_edge(clk);
 if Nrst='0' then
 cnt:='0';
 Nstop_aux<='1';
 elsif Nstop_aux='1' then
 if nxtp_Nstop='0' then
 case cnt is
 when '0'=> op0:=signed(x0); op1:=incx;
 when others=> op0:=signed(y0); op1:=incy;
 end case;
 nxt:=op0+op1;
 case cnt is
 when '0'=> x1<=std_logic_vector(nxt);
 when others=> y1<=std_logic_vector(nxt);
 Nstop_aux<='0';
 end case;
 cnt:=not cnt;
 end if;
 end if;
 end process;
end architecture;

```

### ***Fichero sramcontrol\_8\_8.vhd***

```

library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;

entity sramcontrol_8_8 is
 generic(
 addr_width: integer:=16;
 dataram_width: integer:=8;
 dataup_width: integer:=8);
 port(
 clk: in std_logic;
 Nrst: in std_logic;
 Nstop: out std_logic;
 rNw: in std_logic;
 addr: in std_logic_vector(addr_width-1 downto 0);
 din: in std_logic_vector(dataup_width-1 downto 0);
 dout: out std_logic_vector(dataup_width-1 downto 0);
 sram_Nwe: out std_logic;
 sram_Ncs, sram_Noë: out std_logic;
 sram_addr: out std_logic_vector(addr_width-1 downto 0);
 sram_data_I: in std_logic_vector(dataram_width-1 downto 0);
 sram_data_O: out std_logic_vector(dataram_width-1 downto 0);
 sram_data_T: out std_logic);
end entity;

```

```

architecture beh1 of sramcontrol_8_8 is
signal sram_Ncs_aux,sram_Noe_aux,sram_Nwe_aux: std_logic;
begin
 sram_Ncs<=sram_Ncs_aux;
 sram_Noe<=sram_Noe_aux;
 sram_Nwe<=sram_Nwe_aux;
 sram_data_T<=sram_Ncs_aux or sram_Nwe_aux;

 SRAM: process
 variable k: unsigned(1 downto 0);
 variable din0: std_logic_vector(sram_data_I'range);
 begin
 wait until rising_edge(clk);
 if Nrst='0' then
 k:=(others=>'0');
 din0:=din;
 dout<=(others=>'X');
 Nstop<='1';
 else
 if k/=3 then
 k:=k+1;
 end if;
 if k=2 then
 Nstop<='0';
 end if;
 end if;
 case conv_integer(k) is
 when 0=> sram_addr<=addr;
 sram_Nwe_aux<=rNw;
 sram_Noe_aux<='1'; sram_Ncs_aux<='1';
 when 1=> sram_data_O<=din0; sram_Ncs_aux<='0';
 sram_Noe_aux<=not sram_Nwe_aux;
 when 3=> sram_Noe_aux<='1'; sram_Ncs_aux<='1';
 if sram_Noe_aux='0' then
 dout<=std_logic_vector(sram_data_I);
 end if;
 when others=>
 end case;
 end process;
end architecture;

```

### ***Fichero sramcontrol\_8\_32.vhd***

```

library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_arith.all;

entity sramcontrol_8_32 is
generic(
 addr_width: integer:=21;
 dataram_width: integer:=8;
 dataup_width: integer:=32);
port(
 clk: in std_logic;
 Nrst: in std_logic;
 Nstop: out std_logic;
 numbytes: in unsigned(1 downto 0);
 rNw: in std_logic;
 addr: in std_logic_vector(addr_width-1 downto 0);
 din: in std_logic_vector(dataup_width-1 downto 0);
 dout: out std_logic_vector(dataup_width-1 downto 0);
 sram_Nwe: out std_logic;

```

```

 sram_Ncs, sram_Noe: out std_logic;
 sram_addr: out std_logic_vector(addr_width-1 downto 0);
 sram_data_I: in std_logic_vector(dataram_width-1 downto 0);
 sram_data_O: out std_logic_vector(dataram_width-1 downto 0);
 sram_data_T: out std_logic);
end entity;

architecture beh1 of sramcontrol_8_32 is
signal sram_Ncs_aux, sram_Noe_aux, sram_Nwe_aux: std_logic;
begin
 sram_Ncs<=sram_Ncs_aux;
 sram_Noe<=sram_Noe_aux;
 sram_Nwe<=sram_Nwe_aux;
 sram_data_T<=sram_Ncs_aux or sram_Nwe_aux;

 SRAM: process
 variable k: unsigned(1 downto 0);
 variable num: unsigned(numbytes'range);
 variable rNw0: std_logic;
 variable addr0: std_logic_vector(addr'range);
 variable addr0_2lsb: unsigned(1 downto 0);
 type ARRAY_BYTES is array(3 downto 0) of
 std_logic_vector(dataram_width-1 downto 0);
 variable din0,dout0: ARRAY_BYTES;
 begin
 wait until rising_edge(clk);
 if Nrst='0' then
 num:=numbytes;
 k:=(others=>'0');
 rNw0:=rNw;
 addr0:=addr;
 dout0(0):=(others=>'X'); dout0(1):=(others=>'X');
 dout0(2):=(others=>'X'); dout0(3):=(others=>'X');
 din0(3):=din(31 downto 24); din0(2):=din(23 downto 16);
 din0(1):=din(15 downto 8); din0(0):=din(7 downto 0);
 Nstop<='1';
 else
 if k=0 and num=numbytes then
 assert (numbytes=3 and addr0(1 downto 0)="00")or
 (numbytes=1 and addr0(0)='0')or(numbytes=0)
 report "Error en sramcontrol_8_32" severity FAILURE;
 end if;
 if k/=3 then
 k:=k+1;
 elsif num/=0 then
 k:=(others=>'0');
 num:=num-1;
 addr0_2lsb:=1+unsigned(addr0(1 downto 0));
 addr0(1 downto 0):=std_logic_vector(addr0_2lsb);
 end if;
 if k=2 and num=0 then
 Nstop<='0';
 end if;
 end if;
 case conv_integer(k) is
 when 0=> sram_addr<=addr0;
 sram_Nwe_aux<=rNw0;
 sram_Noe_aux<='1'; sram_Ncs_aux<='1';
 when 1=> sram_data_O<=din0(conv_integer(num));
 sram_Ncs_aux<='0';
 sram_Noe_aux<=not sram_Nwe_aux;
 end case;
 end process;
end architecture;

```



```

 when 3=> sram_Noe_aux<='1'; sram_Ncs_aux<='1';
 if sram_Noe_aux='0' then
 dout0(conv_integer(num)):=std_logic_vector
 (sram_data_I);
 end if;
 when others=>
 end case;
 dout<=dout0(3)&dout0(2)&dout0(1)&dout0(0);
 end process;
end architecture;

```

### 5.2.10 Anexo 13: DCM

Ficheros:       wb\_dcm.vhd

```

-- Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;

entity wb_dcm is
 port (CLKIN_IN : in std_logic;
 CLKDV_OUT : out std_logic;
 CLKIN_IBUFG_OUT : out std_logic;
 CLK0_OUT : out std_logic);
end wb_dcm;

architecture BEHAVIORAL of wb_dcm is
 signal CLKDV_BUF : std_logic;
 signal CLKFB_IN : std_logic;
 signal CLKIN_IBUFG : std_logic;
 signal CLK0_BUF : std_logic;
 signal GND1 : std_logic;
 component BUFG
 port (I : in std_logic;
 O : out std_logic);
 end component;
 component IBUFG
 port (I : in std_logic;
 O : out std_logic);
 end component;
 component DCM
 generic(CLK_FEEDBACK : string := "1X";
 CLKDV_DIVIDE : real := 2.0;
 CLKFX_DIVIDE : integer := 1;
 CLKFX_MULTIPLY : integer := 4;
 CLKIN_DIVIDE_BY_2 : boolean := FALSE;
 CLKIN_PERIOD : real := 10.0;
 CLKOUT_PHASE_SHIFT : string := "NONE";
 DESKEW_ADJUST : string := "SYSTEM_SYNCHRONOUS";
 DFS_FREQUENCY_MODE : string := "LOW";
 DLL_FREQUENCY_MODE : string := "LOW";
 DUTY_CYCLE_CORRECTION : boolean := TRUE;
 FACTORY_JF : bit_vector := x"C080";
 PHASE_SHIFT : integer := 0;
 STARTUP_WAIT : boolean := FALSE;
 DSS_MODE : string := "NONE");
 end component;

```

```

 port (CLKIN : in std_logic;
 CLKFB : in std_logic;
 RST : in std_logic;
 PSEN : in std_logic;
 PSINCDEC : in std_logic;
 PSCLK : in std_logic;
 DSSEN : in std_logic;
 CLK0 : out std_logic;
 CLK90 : out std_logic;
 CLK180 : out std_logic;
 CLK270 : out std_logic;
 CLKDV : out std_logic;
 CLK2X : out std_logic;
 CLK2X180 : out std_logic;
 CLKFX : out std_logic;
 CLKFX180 : out std_logic;
 STATUS : out std_logic_vector (7 downto 0);
 LOCKED : out std_logic;
 PSDONE : out std_logic);
end component;

begin
 GND1 <= '0';
 CLKIN_IBUFG_OUT <= CLKIN_IBUFG;
 CLK0_OUT <= CLKFB_IN;
 CLKDV_BUFG_INST : BUFG
 port map (I=>CLKDV_BUF,
 O=>CLKDV_OUT);
 CLKIN_IBUFG_INST : IBUFG
 port map (I=>CLKIN_IN,
 O=>CLKIN_IBUFG);
 CLK0_BUFG_INST : BUFG
 port map (I=>CLK0_BUF,
 O=>CLKFB_IN);
 DCM_INST : DCM
 generic map(CLK_FEEDBACK => "1X",
 CLKDV_DIVIDE => 1.5,
 CLKFX_DIVIDE => 1,
 CLKFX_MULTIPLY => 4,
 CLKIN_DIVIDE_BY_2 => FALSE,
 CLKIN_PERIOD => 25.0,
 CLKOUT_PHASE_SHIFT => "NONE",
 DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
 DFS_FREQUENCY_MODE => "LOW",
 DLL_FREQUENCY_MODE => "LOW",
 DUTY_CYCLE_CORRECTION => TRUE,
 FACTORY_JF => x"C080",
 PHASE_SHIFT => 0,
 STARTUP_WAIT => TRUE)
 port map (CLKFB=>CLKFB_IN,
 CLKIN=>CLKIN_IBUFG,
 DSSEN=>GND1,
 PSCLK=>GND1,
 PSEN=>GND1,
 PSINCDEC=>GND1,
 RST=>GND1,
 CLKDV=>CLKDV_BUF,
 CLKFX=>open,
 CLKFX180=>open,
 CLK0=>CLK0_BUF,
 CLK2X=>open,

```

```

 CLK2X180=>open,
 CLK90=>open,
 CLK180=>open,
 CLK270=>open,
 LOCKED=>open,
 PSDONE=>open,
 STATUS=>open);
end BEHAVIORAL;

```

### 5.2.11 Anexo 14: General Purpose I/O

```

// synopsys translate_off
`include "timescale.v"
// synopsys translate_on

module simple_gpio(
 clk_i, rst_i, cyc_i, stb_i, adr_i, we_i, dat_i, dat_o, ack_o,
 gpio
);

 //
 // Inputs & outputs
 //
 parameter io = 8; // number of GPIOs
 // 8bit WISHBONE bus slave interface
 input clk_i; // clock
 input rst_i; // reset (asynchronous active high)
 input cyc_i; // cycle
 input stb_i; // strobe
 input adr_i; // address adr_i[1]
 input we_i; // write enable
 input [7:0] dat_i; // data output
 output [7:0] dat_o; // data input
 output ack_o; // normal bus termination
 // GPIO pins
 inout [io:1] gpio;
 //
 // Module body
 //
 reg [io:1] ctrl, line; // ControlRegister, LineRegister
 reg [io:1] lgpio, llgpio; // LatchedGPIO pins
 //
 // perform parameter checks
 //
 // synopsys translate_off
 initial
 begin
 if(io > 8)
 $display("simple_gpio: max. 8 GPIOs supported.");
 end
 // synopsys translate_on

 //
 // WISHBONE interface
 wire wb_acc = cyc_i & stb_i; // WISHBONE access
 wire wb_wr = wb_acc & we_i; // WISHBONE write access

 always @(posedge clk_i or posedge rst_i)
 if (rst_i)
 begin
 ctrl <= #1 {{io}}{1'b0}};
 end

```

```

 line <= #1 {{io}}{1'b0}};
 end
 else if (wb_wr)
 if (adr_i)
 line <= #1 dat_i[io-1:0];
 else
 ctrl <= #1 dat_i[io-1:0];

 reg [7:0] dat_o;
 always @(posedge clk_i)
 if (adr_i)
 dat_o <= #1 { {{8-io}}{1'b0}}, llgpio;
 else
 dat_o <= #1 { {{8-io}}{1'b0}}, ctrl};

 reg ack_o;
 always @(posedge clk_i or posedge rst_i)
 if (rst_i)
 ack_o <= #1 1'b0;
 else
 ack_o <= #1 wb_acc & !ack_o;

 //
 // GPIO section

 // latch GPIO input pins
 always @(posedge clk_i)
 lgpio <= #1 gpio;
 // latch again (reduce meta-stability risc)
 always @(posedge clk_i)
 llgpio <= #1 lgpio;
 // assign GPIO outputs
 integer n;
 reg [io:1] igpio; // temporary internal signal

 always @(ctrl or line)
 for(n=1;n<=io;n=n+1)
 igpio[n] <= ctrl[n] ? line[n] : 1'bz;
 assign gpio = igpio;
endmodule

```

### 5.3 Proyecto ORPSoC

El presente trabajo no contiene el código fuente de los ficheros que componen el proyecto *ORP\_SoC* de la comunidad *OpenCores* (excepto el fichero de jerarquía superior) por razones de espacio y relevancia del contenido. Para acceder al código se debe consultar la versión del proyecto en formato electrónico en el directorio de fuentes */rtl/verilog*.

El código fuente del fichero que describe la interconexión de los cores (*Traffic Cop*) es *tc\_top.v*:

```

// synopsys translate_off
`include "timescale.v"
// synopsys translate_on

//
// Width of address bus
//
`define TC_AW 32

```

```

//
// Width of data bus
//
`define TC_DW 32

//
// Width of byte select bus
//
`define TC_BSW 4

//
// Width of WB target inputs (coming from WB slave)
//
// data bus width + ack + err
//
`define TC_TIN_W `TC_DW+1+1

//
// Width of WB initiator inputs (coming from WB masters)
//
// cyc + stb + cab + address bus width +
// byte select bus width + we + data bus width
//
`define TC_IIN_W 1+1+1+`TC_AW+`TC_BSW+1+`TC_DW

//
// Traffic Cop Top
//
module tc_top (
 wb_clk_i,
 wb_rst_i,

 i0_wb_cyc_i,
 i0_wb_stb_i,
 i0_wb_cab_i,
 i0_wb_adr_i,
 i0_wb_sel_i,
 i0_wb_we_i,
 i0_wb_dat_i,
 i0_wb_dat_o,
 i0_wb_ack_o,
 i0_wb_err_o,

 i1_wb_cyc_i,
 i1_wb_stb_i,
 i1_wb_cab_i,
 i1_wb_adr_i,
 i1_wb_sel_i,
 i1_wb_we_i,
 i1_wb_dat_i,
 i1_wb_dat_o,
 i1_wb_ack_o,
 i1_wb_err_o,

 i2_wb_cyc_i,
 i2_wb_stb_i,
 i2_wb_cab_i,
 i2_wb_adr_i,
 i2_wb_sel_i,
 i2_wb_we_i,

```

i2\_wb\_dat\_i,  
i2\_wb\_dat\_o,  
i2\_wb\_ack\_o,  
i2\_wb\_err\_o,

i3\_wb\_cyc\_i,  
i3\_wb\_stb\_i,  
i3\_wb\_cab\_i,  
i3\_wb\_adr\_i,  
i3\_wb\_sel\_i,  
i3\_wb\_we\_i,  
i3\_wb\_dat\_i,  
i3\_wb\_dat\_o,  
i3\_wb\_ack\_o,  
i3\_wb\_err\_o,

i4\_wb\_cyc\_i,  
i4\_wb\_stb\_i,  
i4\_wb\_cab\_i,  
i4\_wb\_adr\_i,  
i4\_wb\_sel\_i,  
i4\_wb\_we\_i,  
i4\_wb\_dat\_i,  
i4\_wb\_dat\_o,  
i4\_wb\_ack\_o,  
i4\_wb\_err\_o,

i5\_wb\_cyc\_i,  
i5\_wb\_stb\_i,  
i5\_wb\_cab\_i,  
i5\_wb\_adr\_i,  
i5\_wb\_sel\_i,  
i5\_wb\_we\_i,  
i5\_wb\_dat\_i,  
i5\_wb\_dat\_o,  
i5\_wb\_ack\_o,  
i5\_wb\_err\_o,

i6\_wb\_cyc\_i,  
i6\_wb\_stb\_i,  
i6\_wb\_cab\_i,  
i6\_wb\_adr\_i,  
i6\_wb\_sel\_i,  
i6\_wb\_we\_i,  
i6\_wb\_dat\_i,  
i6\_wb\_dat\_o,  
i6\_wb\_ack\_o,  
i6\_wb\_err\_o,

i7\_wb\_cyc\_i,  
i7\_wb\_stb\_i,  
i7\_wb\_cab\_i,  
i7\_wb\_adr\_i,  
i7\_wb\_sel\_i,  
i7\_wb\_we\_i,  
i7\_wb\_dat\_i,  
i7\_wb\_dat\_o,  
i7\_wb\_ack\_o,  
i7\_wb\_err\_o,

t0\_wb\_cyc\_o,

t0\_wb\_stb\_o,  
t0\_wb\_cab\_o,  
t0\_wb\_adr\_o,  
t0\_wb\_sel\_o,  
t0\_wb\_we\_o,  
t0\_wb\_dat\_o,  
t0\_wb\_dat\_i,  
t0\_wb\_ack\_i,  
t0\_wb\_err\_i,

t1\_wb\_cyc\_o,  
t1\_wb\_stb\_o,  
t1\_wb\_cab\_o,  
t1\_wb\_adr\_o,  
t1\_wb\_sel\_o,  
t1\_wb\_we\_o,  
t1\_wb\_dat\_o,  
t1\_wb\_dat\_i,  
t1\_wb\_ack\_i,  
t1\_wb\_err\_i,

t2\_wb\_cyc\_o,  
t2\_wb\_stb\_o,  
t2\_wb\_cab\_o,  
t2\_wb\_adr\_o,  
t2\_wb\_sel\_o,  
t2\_wb\_we\_o,  
t2\_wb\_dat\_o,  
t2\_wb\_dat\_i,  
t2\_wb\_ack\_i,  
t2\_wb\_err\_i,

t3\_wb\_cyc\_o,  
t3\_wb\_stb\_o,  
t3\_wb\_cab\_o,  
t3\_wb\_adr\_o,  
t3\_wb\_sel\_o,  
t3\_wb\_we\_o,  
t3\_wb\_dat\_o,  
t3\_wb\_dat\_i,  
t3\_wb\_ack\_i,  
t3\_wb\_err\_i,

t4\_wb\_cyc\_o,  
t4\_wb\_stb\_o,  
t4\_wb\_cab\_o,  
t4\_wb\_adr\_o,  
t4\_wb\_sel\_o,  
t4\_wb\_we\_o,  
t4\_wb\_dat\_o,  
t4\_wb\_dat\_i,  
t4\_wb\_ack\_i,  
t4\_wb\_err\_i,

t5\_wb\_cyc\_o,  
t5\_wb\_stb\_o,  
t5\_wb\_cab\_o,  
t5\_wb\_adr\_o,  
t5\_wb\_sel\_o,  
t5\_wb\_we\_o,  
t5\_wb\_dat\_o,

```

t5_wb_dat_i,
t5_wb_ack_i,
t5_wb_err_i,

t6_wb_cyc_o,
t6_wb_stb_o,
t6_wb_cab_o,
t6_wb_adr_o,
t6_wb_sel_o,
t6_wb_we_o,
t6_wb_dat_o,
t6_wb_dat_i,
t6_wb_ack_i,
t6_wb_err_i,

t7_wb_cyc_o,
t7_wb_stb_o,
t7_wb_cab_o,
t7_wb_adr_o,
t7_wb_sel_o,
t7_wb_we_o,
t7_wb_dat_o,
t7_wb_dat_i,
t7_wb_ack_i,
t7_wb_err_i,

t8_wb_cyc_o,
t8_wb_stb_o,
t8_wb_cab_o,
t8_wb_adr_o,
t8_wb_sel_o,
t8_wb_we_o,
t8_wb_dat_o,
t8_wb_dat_i,
t8_wb_ack_i,
t8_wb_err_i

);

//
// Parameters
//
parameter t0_addr_w = 4;
parameter t0_addr = 4'd8;
parameter t1_addr_w = 4;
parameter t1_addr = 4'd0;
parameter t28c_addr_w = 4;
parameter t28_addr = 4'd0;
parameter t28i_addr_w = 4;
parameter t2_addr = 4'd1;
parameter t3_addr = 4'd2;
parameter t4_addr = 4'd3;
parameter t5_addr = 4'd4;
parameter t6_addr = 4'd5;
parameter t7_addr = 4'd6;
parameter t8_addr = 4'd7;

//
// I/O Ports
//
input wb_clk_i;

```



```

input wb_rst_i;

//
// WB slave i/f connecting initiator 0
//
input i0_wb_cyc_i;
input i0_wb_stb_i;
input i0_wb_cab_i;
input [`TC_AW-1:0] i0_wb_adr_i;
input [`TC_BSW-1:0] i0_wb_sel_i;
input i0_wb_we_i;
input [`TC_DW-1:0] i0_wb_dat_i;
output [`TC_DW-1:0] i0_wb_dat_o;
output i0_wb_ack_o;
output i0_wb_err_o;

//
// WB slave i/f connecting initiator 1
//
input i1_wb_cyc_i;
input i1_wb_stb_i;
input i1_wb_cab_i;
input [`TC_AW-1:0] i1_wb_adr_i;
input [`TC_BSW-1:0] i1_wb_sel_i;
input i1_wb_we_i;
input [`TC_DW-1:0] i1_wb_dat_i;
output [`TC_DW-1:0] i1_wb_dat_o;
output i1_wb_ack_o;
output i1_wb_err_o;

//
// WB slave i/f connecting initiator 2
//
input i2_wb_cyc_i;
input i2_wb_stb_i;
input i2_wb_cab_i;
input [`TC_AW-1:0] i2_wb_adr_i;
input [`TC_BSW-1:0] i2_wb_sel_i;
input i2_wb_we_i;
input [`TC_DW-1:0] i2_wb_dat_i;
output [`TC_DW-1:0] i2_wb_dat_o;
output i2_wb_ack_o;
output i2_wb_err_o;

//
// WB slave i/f connecting initiator 3
//
input i3_wb_cyc_i;
input i3_wb_stb_i;
input i3_wb_cab_i;
input [`TC_AW-1:0] i3_wb_adr_i;
input [`TC_BSW-1:0] i3_wb_sel_i;
input i3_wb_we_i;
input [`TC_DW-1:0] i3_wb_dat_i;
output [`TC_DW-1:0] i3_wb_dat_o;
output i3_wb_ack_o;
output i3_wb_err_o;

//
// WB slave i/f connecting initiator 4
//

```

```

input i4_wb_cyc_i;
input i4_wb_stb_i;
input i4_wb_cab_i;
input [`TC_AW-1:0] i4_wb_adr_i;
input [`TC_BSW-1:0] i4_wb_sel_i;
input i4_wb_we_i;
input [`TC_DW-1:0] i4_wb_dat_i;
output [`TC_DW-1:0] i4_wb_dat_o;
output i4_wb_ack_o;
output i4_wb_err_o;

//
// WB slave i/f connecting initiator 5
//
input i5_wb_cyc_i;
input i5_wb_stb_i;
input i5_wb_cab_i;
input [`TC_AW-1:0] i5_wb_adr_i;
input [`TC_BSW-1:0] i5_wb_sel_i;
input i5_wb_we_i;
input [`TC_DW-1:0] i5_wb_dat_i;
output [`TC_DW-1:0] i5_wb_dat_o;
output i5_wb_ack_o;
output i5_wb_err_o;

//
// WB slave i/f connecting initiator 6
//
input i6_wb_cyc_i;
input i6_wb_stb_i;
input i6_wb_cab_i;
input [`TC_AW-1:0] i6_wb_adr_i;
input [`TC_BSW-1:0] i6_wb_sel_i;
input i6_wb_we_i;
input [`TC_DW-1:0] i6_wb_dat_i;
output [`TC_DW-1:0] i6_wb_dat_o;
output i6_wb_ack_o;
output i6_wb_err_o;

//
// WB slave i/f connecting initiator 7
//
input i7_wb_cyc_i;
input i7_wb_stb_i;
input i7_wb_cab_i;
input [`TC_AW-1:0] i7_wb_adr_i;
input [`TC_BSW-1:0] i7_wb_sel_i;
input i7_wb_we_i;
input [`TC_DW-1:0] i7_wb_dat_i;
output [`TC_DW-1:0] i7_wb_dat_o;
output i7_wb_ack_o;
output i7_wb_err_o;

//
// WB master i/f connecting target 0
//
output t0_wb_cyc_o;
output t0_wb_stb_o;
output t0_wb_cab_o;
output [`TC_AW-1:0] t0_wb_adr_o;
output [`TC_BSW-1:0] t0_wb_sel_o;

```

```

output t0_wb_we_o;
output [`TC_DW-1:0] t0_wb_dat_o;
input [`TC_DW-1:0] t0_wb_dat_i;
input t0_wb_ack_i;
input t0_wb_err_i;

//
// WB master i/f connecting target 1
//
output t1_wb_cyc_o;
output t1_wb_stb_o;
output t1_wb_cab_o;
output [`TC_AW-1:0] t1_wb_adr_o;
output [`TC_BSW-1:0] t1_wb_sel_o;
output t1_wb_we_o;
output [`TC_DW-1:0] t1_wb_dat_o;
input [`TC_DW-1:0] t1_wb_dat_i;
input t1_wb_ack_i;
input t1_wb_err_i;

//
// WB master i/f connecting target 2
//
output t2_wb_cyc_o;
output t2_wb_stb_o;
output t2_wb_cab_o;
output [`TC_AW-1:0] t2_wb_adr_o;
output [`TC_BSW-1:0] t2_wb_sel_o;
output t2_wb_we_o;
output [`TC_DW-1:0] t2_wb_dat_o;
input [`TC_DW-1:0] t2_wb_dat_i;
input t2_wb_ack_i;
input t2_wb_err_i;

//
// WB master i/f connecting target 3
//
output t3_wb_cyc_o;
output t3_wb_stb_o;
output t3_wb_cab_o;
output [`TC_AW-1:0] t3_wb_adr_o;
output [`TC_BSW-1:0] t3_wb_sel_o;
output t3_wb_we_o;
output [`TC_DW-1:0] t3_wb_dat_o;
input [`TC_DW-1:0] t3_wb_dat_i;
input t3_wb_ack_i;
input t3_wb_err_i;

//
// WB master i/f connecting target 4
//
output t4_wb_cyc_o;
output t4_wb_stb_o;
output t4_wb_cab_o;
output [`TC_AW-1:0] t4_wb_adr_o;
output [`TC_BSW-1:0] t4_wb_sel_o;
output t4_wb_we_o;
output [`TC_DW-1:0] t4_wb_dat_o;
input [`TC_DW-1:0] t4_wb_dat_i;
input t4_wb_ack_i;
input t4_wb_err_i;

```

```

//
// WB master i/f connecting target 5
//
output t5_wb_cyc_o;
output t5_wb_stb_o;
output t5_wb_cab_o;
output [`TC_AW-1:0] t5_wb_adr_o;
output [`TC_BSW-1:0] t5_wb_sel_o;
output t5_wb_we_o;
output [`TC_DW-1:0] t5_wb_dat_o;
input [`TC_DW-1:0] t5_wb_dat_i;
input t5_wb_ack_i;
input t5_wb_err_i;

//
// WB master i/f connecting target 6
//
output t6_wb_cyc_o;
output t6_wb_stb_o;
output t6_wb_cab_o;
output [`TC_AW-1:0] t6_wb_adr_o;
output [`TC_BSW-1:0] t6_wb_sel_o;
output t6_wb_we_o;
output [`TC_DW-1:0] t6_wb_dat_o;
input [`TC_DW-1:0] t6_wb_dat_i;
input t6_wb_ack_i;
input t6_wb_err_i;

//
// WB master i/f connecting target 7
//
output t7_wb_cyc_o;
output t7_wb_stb_o;
output t7_wb_cab_o;
output [`TC_AW-1:0] t7_wb_adr_o;
output [`TC_BSW-1:0] t7_wb_sel_o;
output t7_wb_we_o;
output [`TC_DW-1:0] t7_wb_dat_o;
input [`TC_DW-1:0] t7_wb_dat_i;
input t7_wb_ack_i;
input t7_wb_err_i;

//
// WB master i/f connecting target 8
//
output t8_wb_cyc_o;
output t8_wb_stb_o;
output t8_wb_cab_o;
output [`TC_AW-1:0] t8_wb_adr_o;
output [`TC_BSW-1:0] t8_wb_sel_o;
output t8_wb_we_o;
output [`TC_DW-1:0] t8_wb_dat_o;
input [`TC_DW-1:0] t8_wb_dat_i;
input t8_wb_ack_i;
input t8_wb_err_i;

//
// Internal wires & registers
//

```

```

//
// Outputs for initiators from both mi_to_st blocks
//
wire [`TC_DW-1:0] xi0_wb_dat_o;
wire xi0_wb_ack_o;
wire xi0_wb_err_o;
wire [`TC_DW-1:0] xi1_wb_dat_o;
wire xi1_wb_ack_o;
wire xi1_wb_err_o;
wire [`TC_DW-1:0] xi2_wb_dat_o;
wire xi2_wb_ack_o;
wire xi2_wb_err_o;
wire [`TC_DW-1:0] xi3_wb_dat_o;
wire xi3_wb_ack_o;
wire xi3_wb_err_o;
wire [`TC_DW-1:0] xi4_wb_dat_o;
wire xi4_wb_ack_o;
wire xi4_wb_err_o;
wire [`TC_DW-1:0] xi5_wb_dat_o;
wire xi5_wb_ack_o;
wire xi5_wb_err_o;
wire [`TC_DW-1:0] xi6_wb_dat_o;
wire xi6_wb_ack_o;
wire xi6_wb_err_o;
wire [`TC_DW-1:0] xi7_wb_dat_o;
wire xi7_wb_ack_o;
wire xi7_wb_err_o;
wire [`TC_DW-1:0] yi0_wb_dat_o;
wire yi0_wb_ack_o;
wire yi0_wb_err_o;
wire [`TC_DW-1:0] yi1_wb_dat_o;
wire yi1_wb_ack_o;
wire yi1_wb_err_o;
wire [`TC_DW-1:0] yi2_wb_dat_o;
wire yi2_wb_ack_o;
wire yi2_wb_err_o;
wire [`TC_DW-1:0] yi3_wb_dat_o;
wire yi3_wb_ack_o;
wire yi3_wb_err_o;
wire [`TC_DW-1:0] yi4_wb_dat_o;
wire yi4_wb_ack_o;
wire yi4_wb_err_o;
wire [`TC_DW-1:0] yi5_wb_dat_o;
wire yi5_wb_ack_o;
wire yi5_wb_err_o;
wire [`TC_DW-1:0] yi6_wb_dat_o;
wire yi6_wb_ack_o;
wire yi6_wb_err_o;
wire [`TC_DW-1:0] yi7_wb_dat_o;
wire yi7_wb_ack_o;
wire yi7_wb_err_o;

//
// Intermediate signals connecting peripheral channel's
// mi_to_st and si_to_mt blocks.
//
wire z_wb_cyc_i;
wire z_wb_stb_i;
wire z_wb_cab_i;
wire [`TC_AW-1:0] z_wb_adr_i;
wire [`TC_BSW-1:0] z_wb_sel_i;

```

```

wire z_wb_we_i;
wire [`TC_DW-1:0] z_wb_dat_i;
wire [`TC_DW-1:0] z_wb_dat_t;
wire z_wb_ack_t;
wire z_wb_err_t;

//
// Outputs for initiators are ORed from both mi_to_st blocks
//
assign i0_wb_dat_o = xi0_wb_dat_o | yi0_wb_dat_o;
assign i0_wb_ack_o = xi0_wb_ack_o | yi0_wb_ack_o;
assign i0_wb_err_o = xi0_wb_err_o | yi0_wb_err_o;
assign i1_wb_dat_o = xi1_wb_dat_o | yi1_wb_dat_o;
assign i1_wb_ack_o = xi1_wb_ack_o | yi1_wb_ack_o;
assign i1_wb_err_o = xi1_wb_err_o | yi1_wb_err_o;
assign i2_wb_dat_o = xi2_wb_dat_o | yi2_wb_dat_o;
assign i2_wb_ack_o = xi2_wb_ack_o | yi2_wb_ack_o;
assign i2_wb_err_o = xi2_wb_err_o | yi2_wb_err_o;
assign i3_wb_dat_o = xi3_wb_dat_o | yi3_wb_dat_o;
assign i3_wb_ack_o = xi3_wb_ack_o | yi3_wb_ack_o;
assign i3_wb_err_o = xi3_wb_err_o | yi3_wb_err_o;
assign i4_wb_dat_o = xi4_wb_dat_o | yi4_wb_dat_o;
assign i4_wb_ack_o = xi4_wb_ack_o | yi4_wb_ack_o;
assign i4_wb_err_o = xi4_wb_err_o | yi4_wb_err_o;
assign i5_wb_dat_o = xi5_wb_dat_o | yi5_wb_dat_o;
assign i5_wb_ack_o = xi5_wb_ack_o | yi5_wb_ack_o;
assign i5_wb_err_o = xi5_wb_err_o | yi5_wb_err_o;
assign i6_wb_dat_o = xi6_wb_dat_o | yi6_wb_dat_o;
assign i6_wb_ack_o = xi6_wb_ack_o | yi6_wb_ack_o;
assign i6_wb_err_o = xi6_wb_err_o | yi6_wb_err_o;
assign i7_wb_dat_o = xi7_wb_dat_o | yi7_wb_dat_o;
assign i7_wb_ack_o = xi7_wb_ack_o | yi7_wb_ack_o;
assign i7_wb_err_o = xi7_wb_err_o | yi7_wb_err_o;

//
// From initiators to target 0
//
tc_mi_to_st #(t0_addr_w, t0_addr,
 0, t0_addr_w, t0_addr) t0_ch(
 .wb_clk_i(wb_clk_i),
 .wb_rst_i(wb_rst_i),

 .i0_wb_cyc_i(i0_wb_cyc_i),
 .i0_wb_stb_i(i0_wb_stb_i),
 .i0_wb_cab_i(i0_wb_cab_i),
 .i0_wb_adr_i(i0_wb_adr_i),
 .i0_wb_sel_i(i0_wb_sel_i),
 .i0_wb_we_i(i0_wb_we_i),
 .i0_wb_dat_i(i0_wb_dat_i),
 .i0_wb_dat_o(xi0_wb_dat_o),
 .i0_wb_ack_o(xi0_wb_ack_o),
 .i0_wb_err_o(xi0_wb_err_o),

 .i1_wb_cyc_i(i1_wb_cyc_i),
 .i1_wb_stb_i(i1_wb_stb_i),
 .i1_wb_cab_i(i1_wb_cab_i),
 .i1_wb_adr_i(i1_wb_adr_i),
 .i1_wb_sel_i(i1_wb_sel_i),
 .i1_wb_we_i(i1_wb_we_i),
 .i1_wb_dat_i(i1_wb_dat_i),
 .i1_wb_dat_o(xi1_wb_dat_o),

```

```

.i1_wb_ack_o(xi1_wb_ack_o),
.i1_wb_err_o(xi1_wb_err_o),

.i2_wb_cyc_i(i2_wb_cyc_i),
.i2_wb_stb_i(i2_wb_stb_i),
.i2_wb_cab_i(i2_wb_cab_i),
.i2_wb_adr_i(i2_wb_adr_i),
.i2_wb_sel_i(i2_wb_sel_i),
.i2_wb_we_i(i2_wb_we_i),
.i2_wb_dat_i(i2_wb_dat_i),
.i2_wb_dat_o(xi2_wb_dat_o),
.i2_wb_ack_o(xi2_wb_ack_o),
.i2_wb_err_o(xi2_wb_err_o),

.i3_wb_cyc_i(i3_wb_cyc_i),
.i3_wb_stb_i(i3_wb_stb_i),
.i3_wb_cab_i(i3_wb_cab_i),
.i3_wb_adr_i(i3_wb_adr_i),
.i3_wb_sel_i(i3_wb_sel_i),
.i3_wb_we_i(i3_wb_we_i),
.i3_wb_dat_i(i3_wb_dat_i),
.i3_wb_dat_o(xi3_wb_dat_o),
.i3_wb_ack_o(xi3_wb_ack_o),
.i3_wb_err_o(xi3_wb_err_o),

.i4_wb_cyc_i(i4_wb_cyc_i),
.i4_wb_stb_i(i4_wb_stb_i),
.i4_wb_cab_i(i4_wb_cab_i),
.i4_wb_adr_i(i4_wb_adr_i),
.i4_wb_sel_i(i4_wb_sel_i),
.i4_wb_we_i(i4_wb_we_i),
.i4_wb_dat_i(i4_wb_dat_i),
.i4_wb_dat_o(xi4_wb_dat_o),
.i4_wb_ack_o(xi4_wb_ack_o),
.i4_wb_err_o(xi4_wb_err_o),

.i5_wb_cyc_i(i5_wb_cyc_i),
.i5_wb_stb_i(i5_wb_stb_i),
.i5_wb_cab_i(i5_wb_cab_i),
.i5_wb_adr_i(i5_wb_adr_i),
.i5_wb_sel_i(i5_wb_sel_i),
.i5_wb_we_i(i5_wb_we_i),
.i5_wb_dat_i(i5_wb_dat_i),
.i5_wb_dat_o(xi5_wb_dat_o),
.i5_wb_ack_o(xi5_wb_ack_o),
.i5_wb_err_o(xi5_wb_err_o),

.i6_wb_cyc_i(i6_wb_cyc_i),
.i6_wb_stb_i(i6_wb_stb_i),
.i6_wb_cab_i(i6_wb_cab_i),
.i6_wb_adr_i(i6_wb_adr_i),
.i6_wb_sel_i(i6_wb_sel_i),
.i6_wb_we_i(i6_wb_we_i),
.i6_wb_dat_i(i6_wb_dat_i),
.i6_wb_dat_o(xi6_wb_dat_o),
.i6_wb_ack_o(xi6_wb_ack_o),
.i6_wb_err_o(xi6_wb_err_o),

.i7_wb_cyc_i(i7_wb_cyc_i),
.i7_wb_stb_i(i7_wb_stb_i),
.i7_wb_cab_i(i7_wb_cab_i),

```

```

.i7_wb_adr_i(i7_wb_adr_i),
.i7_wb_sel_i(i7_wb_sel_i),
.i7_wb_we_i(i7_wb_we_i),
.i7_wb_dat_i(i7_wb_dat_i),
.i7_wb_dat_o(xi7_wb_dat_o),
.i7_wb_ack_o(xi7_wb_ack_o),
.i7_wb_err_o(xi7_wb_err_o),

.t0_wb_cyc_o(t0_wb_cyc_o),
.t0_wb_stb_o(t0_wb_stb_o),
.t0_wb_cab_o(t0_wb_cab_o),
.t0_wb_adr_o(t0_wb_adr_o),
.t0_wb_sel_o(t0_wb_sel_o),
.t0_wb_we_o(t0_wb_we_o),
.t0_wb_dat_o(t0_wb_dat_o),
.t0_wb_dat_i(t0_wb_dat_i),
.t0_wb_ack_i(t0_wb_ack_i),
.t0_wb_err_i(t0_wb_err_i)

);

//
// From initiators to targets 1-8 (upper part)
//
tc_mi_to_st #(t1_addr_w, t1_addr,
1, t28c_addr_w, t28_addr) t18_ch_upper(
.wb_clk_i(wb_clk_i),
.wb_rst_i(wb_rst_i),

.i0_wb_cyc_i(i0_wb_cyc_i),
.i0_wb_stb_i(i0_wb_stb_i),
.i0_wb_cab_i(i0_wb_cab_i),
.i0_wb_adr_i(i0_wb_adr_i),
.i0_wb_sel_i(i0_wb_sel_i),
.i0_wb_we_i(i0_wb_we_i),
.i0_wb_dat_i(i0_wb_dat_i),
.i0_wb_dat_o(yi0_wb_dat_o),
.i0_wb_ack_o(yi0_wb_ack_o),
.i0_wb_err_o(yi0_wb_err_o),

.i1_wb_cyc_i(i1_wb_cyc_i),
.i1_wb_stb_i(i1_wb_stb_i),
.i1_wb_cab_i(i1_wb_cab_i),
.i1_wb_adr_i(i1_wb_adr_i),
.i1_wb_sel_i(i1_wb_sel_i),
.i1_wb_we_i(i1_wb_we_i),
.i1_wb_dat_i(i1_wb_dat_i),
.i1_wb_dat_o(yi1_wb_dat_o),
.i1_wb_ack_o(yi1_wb_ack_o),
.i1_wb_err_o(yi1_wb_err_o),

.i2_wb_cyc_i(i2_wb_cyc_i),
.i2_wb_stb_i(i2_wb_stb_i),
.i2_wb_cab_i(i2_wb_cab_i),
.i2_wb_adr_i(i2_wb_adr_i),
.i2_wb_sel_i(i2_wb_sel_i),
.i2_wb_we_i(i2_wb_we_i),
.i2_wb_dat_i(i2_wb_dat_i),
.i2_wb_dat_o(yi2_wb_dat_o),
.i2_wb_ack_o(yi2_wb_ack_o),
.i2_wb_err_o(yi2_wb_err_o),

```



```
.i3_wb_cyc_i(i3_wb_cyc_i),
.i3_wb_stb_i(i3_wb_stb_i),
.i3_wb_cab_i(i3_wb_cab_i),
.i3_wb_adr_i(i3_wb_adr_i),
.i3_wb_sel_i(i3_wb_sel_i),
.i3_wb_we_i(i3_wb_we_i),
.i3_wb_dat_i(i3_wb_dat_i),
.i3_wb_dat_o(yi3_wb_dat_o),
.i3_wb_ack_o(yi3_wb_ack_o),
.i3_wb_err_o(yi3_wb_err_o),
```

```
.i4_wb_cyc_i(i4_wb_cyc_i),
.i4_wb_stb_i(i4_wb_stb_i),
.i4_wb_cab_i(i4_wb_cab_i),
.i4_wb_adr_i(i4_wb_adr_i),
.i4_wb_sel_i(i4_wb_sel_i),
.i4_wb_we_i(i4_wb_we_i),
.i4_wb_dat_i(i4_wb_dat_i),
.i4_wb_dat_o(yi4_wb_dat_o),
.i4_wb_ack_o(yi4_wb_ack_o),
.i4_wb_err_o(yi4_wb_err_o),
```

```
.i5_wb_cyc_i(i5_wb_cyc_i),
.i5_wb_stb_i(i5_wb_stb_i),
.i5_wb_cab_i(i5_wb_cab_i),
.i5_wb_adr_i(i5_wb_adr_i),
.i5_wb_sel_i(i5_wb_sel_i),
.i5_wb_we_i(i5_wb_we_i),
.i5_wb_dat_i(i5_wb_dat_i),
.i5_wb_dat_o(yi5_wb_dat_o),
.i5_wb_ack_o(yi5_wb_ack_o),
.i5_wb_err_o(yi5_wb_err_o),
```

```
.i6_wb_cyc_i(i6_wb_cyc_i),
.i6_wb_stb_i(i6_wb_stb_i),
.i6_wb_cab_i(i6_wb_cab_i),
.i6_wb_adr_i(i6_wb_adr_i),
.i6_wb_sel_i(i6_wb_sel_i),
.i6_wb_we_i(i6_wb_we_i),
.i6_wb_dat_i(i6_wb_dat_i),
.i6_wb_dat_o(yi6_wb_dat_o),
.i6_wb_ack_o(yi6_wb_ack_o),
.i6_wb_err_o(yi6_wb_err_o),
```

```
.i7_wb_cyc_i(i7_wb_cyc_i),
.i7_wb_stb_i(i7_wb_stb_i),
.i7_wb_cab_i(i7_wb_cab_i),
.i7_wb_adr_i(i7_wb_adr_i),
.i7_wb_sel_i(i7_wb_sel_i),
.i7_wb_we_i(i7_wb_we_i),
.i7_wb_dat_i(i7_wb_dat_i),
.i7_wb_dat_o(yi7_wb_dat_o),
.i7_wb_ack_o(yi7_wb_ack_o),
.i7_wb_err_o(yi7_wb_err_o),
```

```
.t0_wb_cyc_o(z_wb_cyc_i),
.t0_wb_stb_o(z_wb_stb_i),
.t0_wb_cab_o(z_wb_cab_i),
.t0_wb_adr_o(z_wb_adr_i),
.t0_wb_sel_o(z_wb_sel_i),
```

```

 .t0_wb_we_o(z_wb_we_i),
 .t0_wb_dat_o(z_wb_dat_i),
 .t0_wb_dat_i(z_wb_dat_t),
 .t0_wb_ack_i(z_wb_ack_t),
 .t0_wb_err_i(z_wb_err_t)

);

//
// From initiators to targets 1-8 (lower part)
//
tc_si_to_mt #(t1_addr_w, t1_addr, t28i_addr_w, t2_addr, t3_addr,
 t4_addr, t5_addr, t6_addr, t7_addr, t8_addr) t18_ch_lower(

 .i0_wb_cyc_i(z_wb_cyc_i),
 .i0_wb_stb_i(z_wb_stb_i),
 .i0_wb_cab_i(z_wb_cab_i),
 .i0_wb_adr_i(z_wb_adr_i),
 .i0_wb_sel_i(z_wb_sel_i),
 .i0_wb_we_i(z_wb_we_i),
 .i0_wb_dat_i(z_wb_dat_i),
 .i0_wb_dat_o(z_wb_dat_t),
 .i0_wb_ack_o(z_wb_ack_t),
 .i0_wb_err_o(z_wb_err_t),

 .t0_wb_cyc_o(t1_wb_cyc_o),
 .t0_wb_stb_o(t1_wb_stb_o),
 .t0_wb_cab_o(t1_wb_cab_o),
 .t0_wb_adr_o(t1_wb_adr_o),
 .t0_wb_sel_o(t1_wb_sel_o),
 .t0_wb_we_o(t1_wb_we_o),
 .t0_wb_dat_o(t1_wb_dat_o),
 .t0_wb_dat_i(t1_wb_dat_i),
 .t0_wb_ack_i(t1_wb_ack_i),
 .t0_wb_err_i(t1_wb_err_i),

 .t1_wb_cyc_o(t2_wb_cyc_o),
 .t1_wb_stb_o(t2_wb_stb_o),
 .t1_wb_cab_o(t2_wb_cab_o),
 .t1_wb_adr_o(t2_wb_adr_o),
 .t1_wb_sel_o(t2_wb_sel_o),
 .t1_wb_we_o(t2_wb_we_o),
 .t1_wb_dat_o(t2_wb_dat_o),
 .t1_wb_dat_i(t2_wb_dat_i),
 .t1_wb_ack_i(t2_wb_ack_i),
 .t1_wb_err_i(t2_wb_err_i),

 .t2_wb_cyc_o(t3_wb_cyc_o),
 .t2_wb_stb_o(t3_wb_stb_o),
 .t2_wb_cab_o(t3_wb_cab_o),
 .t2_wb_adr_o(t3_wb_adr_o),
 .t2_wb_sel_o(t3_wb_sel_o),
 .t2_wb_we_o(t3_wb_we_o),
 .t2_wb_dat_o(t3_wb_dat_o),
 .t2_wb_dat_i(t3_wb_dat_i),
 .t2_wb_ack_i(t3_wb_ack_i),
 .t2_wb_err_i(t3_wb_err_i),

 .t3_wb_cyc_o(t4_wb_cyc_o),
 .t3_wb_stb_o(t4_wb_stb_o),
 .t3_wb_cab_o(t4_wb_cab_o),

```

```

.t3_wb_adr_o(t4_wb_adr_o),
.t3_wb_sel_o(t4_wb_sel_o),
.t3_wb_we_o(t4_wb_we_o),
.t3_wb_dat_o(t4_wb_dat_o),
.t3_wb_dat_i(t4_wb_dat_i),
.t3_wb_ack_i(t4_wb_ack_i),
.t3_wb_err_i(t4_wb_err_i),

.t4_wb_cyc_o(t5_wb_cyc_o),
.t4_wb_stb_o(t5_wb_stb_o),
.t4_wb_cab_o(t5_wb_cab_o),
.t4_wb_adr_o(t5_wb_adr_o),
.t4_wb_sel_o(t5_wb_sel_o),
.t4_wb_we_o(t5_wb_we_o),
.t4_wb_dat_o(t5_wb_dat_o),
.t4_wb_dat_i(t5_wb_dat_i),
.t4_wb_ack_i(t5_wb_ack_i),
.t4_wb_err_i(t5_wb_err_i),

.t5_wb_cyc_o(t6_wb_cyc_o),
.t5_wb_stb_o(t6_wb_stb_o),
.t5_wb_cab_o(t6_wb_cab_o),
.t5_wb_adr_o(t6_wb_adr_o),
.t5_wb_sel_o(t6_wb_sel_o),
.t5_wb_we_o(t6_wb_we_o),
.t5_wb_dat_o(t6_wb_dat_o),
.t5_wb_dat_i(t6_wb_dat_i),
.t5_wb_ack_i(t6_wb_ack_i),
.t5_wb_err_i(t6_wb_err_i),

.t6_wb_cyc_o(t7_wb_cyc_o),
.t6_wb_stb_o(t7_wb_stb_o),
.t6_wb_cab_o(t7_wb_cab_o),
.t6_wb_adr_o(t7_wb_adr_o),
.t6_wb_sel_o(t7_wb_sel_o),
.t6_wb_we_o(t7_wb_we_o),
.t6_wb_dat_o(t7_wb_dat_o),
.t6_wb_dat_i(t7_wb_dat_i),
.t6_wb_ack_i(t7_wb_ack_i),
.t6_wb_err_i(t7_wb_err_i),

.t7_wb_cyc_o(t8_wb_cyc_o),
.t7_wb_stb_o(t8_wb_stb_o),
.t7_wb_cab_o(t8_wb_cab_o),
.t7_wb_adr_o(t8_wb_adr_o),
.t7_wb_sel_o(t8_wb_sel_o),
.t7_wb_we_o(t8_wb_we_o),
.t7_wb_dat_o(t8_wb_dat_o),
.t7_wb_dat_i(t8_wb_dat_i),
.t7_wb_ack_i(t8_wb_ack_i),
.t7_wb_err_i(t8_wb_err_i)

);

endmodule

```

```

//
// Multiple initiator to single target
//
module tc_mi_to_st (
 wb_clk_i,
 wb_rst_i,

 i0_wb_cyc_i,
 i0_wb_stb_i,
 i0_wb_cab_i,
 i0_wb_adr_i,
 i0_wb_sel_i,
 i0_wb_we_i,
 i0_wb_dat_i,
 i0_wb_dat_o,
 i0_wb_ack_o,
 i0_wb_err_o,

 i1_wb_cyc_i,
 i1_wb_stb_i,
 i1_wb_cab_i,
 i1_wb_adr_i,
 i1_wb_sel_i,
 i1_wb_we_i,
 i1_wb_dat_i,
 i1_wb_dat_o,
 i1_wb_ack_o,
 i1_wb_err_o,

 i2_wb_cyc_i,
 i2_wb_stb_i,
 i2_wb_cab_i,
 i2_wb_adr_i,
 i2_wb_sel_i,
 i2_wb_we_i,
 i2_wb_dat_i,
 i2_wb_dat_o,
 i2_wb_ack_o,
 i2_wb_err_o,

 i3_wb_cyc_i,
 i3_wb_stb_i,
 i3_wb_cab_i,
 i3_wb_adr_i,
 i3_wb_sel_i,
 i3_wb_we_i,
 i3_wb_dat_i,
 i3_wb_dat_o,
 i3_wb_ack_o,
 i3_wb_err_o,

 i4_wb_cyc_i,
 i4_wb_stb_i,
 i4_wb_cab_i,
 i4_wb_adr_i,
 i4_wb_sel_i,
 i4_wb_we_i,
 i4_wb_dat_i,
 i4_wb_dat_o,
 i4_wb_ack_o,
 i4_wb_err_o,

```

```

 i5_wb_cyc_i,
 i5_wb_stb_i,
 i5_wb_cab_i,
 i5_wb_adr_i,
 i5_wb_sel_i,
 i5_wb_we_i,
 i5_wb_dat_i,
 i5_wb_dat_o,
 i5_wb_ack_o,
 i5_wb_err_o,

 i6_wb_cyc_i,
 i6_wb_stb_i,
 i6_wb_cab_i,
 i6_wb_adr_i,
 i6_wb_sel_i,
 i6_wb_we_i,
 i6_wb_dat_i,
 i6_wb_dat_o,
 i6_wb_ack_o,
 i6_wb_err_o,

 i7_wb_cyc_i,
 i7_wb_stb_i,
 i7_wb_cab_i,
 i7_wb_adr_i,
 i7_wb_sel_i,
 i7_wb_we_i,
 i7_wb_dat_i,
 i7_wb_dat_o,
 i7_wb_ack_o,
 i7_wb_err_o,

 t0_wb_cyc_o,
 t0_wb_stb_o,
 t0_wb_cab_o,
 t0_wb_adr_o,
 t0_wb_sel_o,
 t0_wb_we_o,
 t0_wb_dat_o,
 t0_wb_dat_i,
 t0_wb_ack_i,
 t0_wb_err_i

);

//
// Parameters
//
parameter t0_addr_w = 2;
parameter t0_addr = 2'b00;
parameter multitarg = 1'b0;
parameter t17_addr_w = 2;
parameter t17_addr = 2'b00;

//
// I/O Ports
//
input wb_clk_i;
input wb_rst_i;

```

```

//
// WB slave i/f connecting initiator 0
//
input i0_wb_cyc_i;
input i0_wb_stb_i;
input i0_wb_cab_i;
input [`TC_AW-1:0] i0_wb_adr_i;
input [`TC_BSW-1:0] i0_wb_sel_i;
input i0_wb_we_i;
input [`TC_DW-1:0] i0_wb_dat_i;
output [`TC_DW-1:0] i0_wb_dat_o;
output i0_wb_ack_o;
output i0_wb_err_o;

//
// WB slave i/f connecting initiator 1
//
input i1_wb_cyc_i;
input i1_wb_stb_i;
input i1_wb_cab_i;
input [`TC_AW-1:0] i1_wb_adr_i;
input [`TC_BSW-1:0] i1_wb_sel_i;
input i1_wb_we_i;
input [`TC_DW-1:0] i1_wb_dat_i;
output [`TC_DW-1:0] i1_wb_dat_o;
output i1_wb_ack_o;
output i1_wb_err_o;

//
// WB slave i/f connecting initiator 2
//
input i2_wb_cyc_i;
input i2_wb_stb_i;
input i2_wb_cab_i;
input [`TC_AW-1:0] i2_wb_adr_i;
input [`TC_BSW-1:0] i2_wb_sel_i;
input i2_wb_we_i;
input [`TC_DW-1:0] i2_wb_dat_i;
output [`TC_DW-1:0] i2_wb_dat_o;
output i2_wb_ack_o;
output i2_wb_err_o;

//
// WB slave i/f connecting initiator 3
//
input i3_wb_cyc_i;
input i3_wb_stb_i;
input i3_wb_cab_i;
input [`TC_AW-1:0] i3_wb_adr_i;
input [`TC_BSW-1:0] i3_wb_sel_i;
input i3_wb_we_i;
input [`TC_DW-1:0] i3_wb_dat_i;
output [`TC_DW-1:0] i3_wb_dat_o;
output i3_wb_ack_o;
output i3_wb_err_o;

//
// WB slave i/f connecting initiator 4
//
input i4_wb_cyc_i;

```

```

input i4_wb_stb_i;
input i4_wb_cab_i;
input [`TC_AW-1:0] i4_wb_adr_i;
input [`TC_BSW-1:0] i4_wb_sel_i;
input i4_wb_we_i;
input [`TC_DW-1:0] i4_wb_dat_i;
output [`TC_DW-1:0] i4_wb_dat_o;
output i4_wb_ack_o;
output i4_wb_err_o;

//
// WB slave i/f connecting initiator 5
//
input i5_wb_cyc_i;
input i5_wb_stb_i;
input i5_wb_cab_i;
input [`TC_AW-1:0] i5_wb_adr_i;
input [`TC_BSW-1:0] i5_wb_sel_i;
input i5_wb_we_i;
input [`TC_DW-1:0] i5_wb_dat_i;
output [`TC_DW-1:0] i5_wb_dat_o;
output i5_wb_ack_o;
output i5_wb_err_o;

//
// WB slave i/f connecting initiator 6
//
input i6_wb_cyc_i;
input i6_wb_stb_i;
input i6_wb_cab_i;
input [`TC_AW-1:0] i6_wb_adr_i;
input [`TC_BSW-1:0] i6_wb_sel_i;
input i6_wb_we_i;
input [`TC_DW-1:0] i6_wb_dat_i;
output [`TC_DW-1:0] i6_wb_dat_o;
output i6_wb_ack_o;
output i6_wb_err_o;

//
// WB slave i/f connecting initiator 7
//
input i7_wb_cyc_i;
input i7_wb_stb_i;
input i7_wb_cab_i;
input [`TC_AW-1:0] i7_wb_adr_i;
input [`TC_BSW-1:0] i7_wb_sel_i;
input i7_wb_we_i;
input [`TC_DW-1:0] i7_wb_dat_i;
output [`TC_DW-1:0] i7_wb_dat_o;
output i7_wb_ack_o;
output i7_wb_err_o;

//
// WB master i/f connecting target
//
output t0_wb_cyc_o;
output t0_wb_stb_o;
output t0_wb_cab_o;
output [`TC_AW-1:0] t0_wb_adr_o;
output [`TC_BSW-1:0] t0_wb_sel_o;
output t0_wb_we_o;

```

```

output [`TC_DW-1:0] t0_wb_dat_o;
input [`TC_DW-1:0] t0_wb_dat_i;
input t0_wb_ack_i;
input t0_wb_err_i;

//
// Internal wires & registers
//
wire [`TC_IIN_W-1:0] i0_in, i1_in,
 i2_in, i3_in,
 i4_in, i5_in,
 i6_in, i7_in;
wire [`TC_TIN_W-1:0] i0_out, i1_out,
 i2_out, i3_out,
 i4_out, i5_out,
 i6_out, i7_out;
wire [`TC_IIN_W-1:0] t0_out;
wire [`TC_TIN_W-1:0] t0_in;
wire [7:0] req_i;
wire [2:0] req_won;
reg req_cont;
reg [2:0] req_r;

//
// Group WB initiator 0 i/f inputs and outputs
//
assign i0_in = {i0_wb_cyc_i, i0_wb_stb_i, i0_wb_cab_i, i0_wb_adr_i,
 i0_wb_sel_i, i0_wb_we_i, i0_wb_dat_i};
assign {i0_wb_dat_o, i0_wb_ack_o, i0_wb_err_o} = i0_out;

//
// Group WB initiator 1 i/f inputs and outputs
//
assign i1_in = {i1_wb_cyc_i, i1_wb_stb_i, i1_wb_cab_i, i1_wb_adr_i,
 i1_wb_sel_i, i1_wb_we_i, i1_wb_dat_i};
assign {i1_wb_dat_o, i1_wb_ack_o, i1_wb_err_o} = i1_out;

//
// Group WB initiator 2 i/f inputs and outputs
//
assign i2_in = {i2_wb_cyc_i, i2_wb_stb_i, i2_wb_cab_i, i2_wb_adr_i,
 i2_wb_sel_i, i2_wb_we_i, i2_wb_dat_i};
assign {i2_wb_dat_o, i2_wb_ack_o, i2_wb_err_o} = i2_out;

//
// Group WB initiator 3 i/f inputs and outputs
//
assign i3_in = {i3_wb_cyc_i, i3_wb_stb_i, i3_wb_cab_i, i3_wb_adr_i,
 i3_wb_sel_i, i3_wb_we_i, i3_wb_dat_i};
assign {i3_wb_dat_o, i3_wb_ack_o, i3_wb_err_o} = i3_out;

//
// Group WB initiator 4 i/f inputs and outputs
//
assign i4_in = {i4_wb_cyc_i, i4_wb_stb_i, i4_wb_cab_i, i4_wb_adr_i,
 i4_wb_sel_i, i4_wb_we_i, i4_wb_dat_i};
assign {i4_wb_dat_o, i4_wb_ack_o, i4_wb_err_o} = i4_out;

//
// Group WB initiator 5 i/f inputs and outputs
//

```



```

assign i5_in = {i5_wb_cyc_i, i5_wb_stb_i, i5_wb_cab_i, i5_wb_adr_i,
 i5_wb_sel_i, i5_wb_we_i, i5_wb_dat_i};
assign {i5_wb_dat_o, i5_wb_ack_o, i5_wb_err_o} = i5_out;

//
// Group WB initiator 6 i/f inputs and outputs
//
assign i6_in = {i6_wb_cyc_i, i6_wb_stb_i, i6_wb_cab_i, i6_wb_adr_i,
 i6_wb_sel_i, i6_wb_we_i, i6_wb_dat_i};
assign {i6_wb_dat_o, i6_wb_ack_o, i6_wb_err_o} = i6_out;

//
// Group WB initiator 7 i/f inputs and outputs
//
assign i7_in = {i7_wb_cyc_i, i7_wb_stb_i, i7_wb_cab_i, i7_wb_adr_i,
 i7_wb_sel_i, i7_wb_we_i, i7_wb_dat_i};
assign {i7_wb_dat_o, i7_wb_ack_o, i7_wb_err_o} = i7_out;

//
// Group WB target 0 i/f inputs and outputs
//
assign {t0_wb_cyc_o, t0_wb_stb_o, t0_wb_cab_o, t0_wb_adr_o,
 t0_wb_sel_o, t0_wb_we_o, t0_wb_dat_o} = t0_out;
assign t0_in = {t0_wb_dat_i, t0_wb_ack_i, t0_wb_err_i};

//
// Assign to WB initiator i/f outputs
//
// Either inputs from the target are assigned or zeros.
//
assign i0_out = (req_won == 3'd0) ? t0_in : {`TC_TIN_W{1'b0}};
assign i1_out = (req_won == 3'd1) ? t0_in : {`TC_TIN_W{1'b0}};
assign i2_out = (req_won == 3'd2) ? t0_in : {`TC_TIN_W{1'b0}};
assign i3_out = (req_won == 3'd3) ? t0_in : {`TC_TIN_W{1'b0}};
assign i4_out = (req_won == 3'd4) ? t0_in : {`TC_TIN_W{1'b0}};
assign i5_out = (req_won == 3'd5) ? t0_in : {`TC_TIN_W{1'b0}};
assign i6_out = (req_won == 3'd6) ? t0_in : {`TC_TIN_W{1'b0}};
assign i7_out = (req_won == 3'd7) ? t0_in : {`TC_TIN_W{1'b0}};

//
// Assign to WB target i/f outputs
//
// Assign inputs from initiator to target outputs according to
// which initiator has won. If there is no request for the target,
// assign zeros.
//
assign t0_out = (req_won == 3'd0) ? i0_in :
 (req_won == 3'd1) ? i1_in :
 (req_won == 3'd2) ? i2_in :
 (req_won == 3'd3) ? i3_in :
 (req_won == 3'd4) ? i4_in :
 (req_won == 3'd5) ? i5_in :
 (req_won == 3'd6) ? i6_in :
 (req_won == 3'd7) ? i7_in : {`TC_IIN_W{1'b0}};

//
// Determine if an initiator has address of the target.
//
assign req_i[0] = i0_wb_cyc_i &
 ((i0_wb_adr_i[`TC_AW-1:`TC_AW-t0_addr_w] == t0_addr) |
 multitarg & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==

```

```

 t17_addr));
assign req_i[1] = i1_wb_cyc_i &
 ((i1_wb_addr_i[`TC_AW-1:`TC_AW-t0_addr_w] == t0_addr) |
 multitarg & (i1_wb_addr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
 t17_addr));
assign req_i[2] = i2_wb_cyc_i &
 ((i2_wb_addr_i[`TC_AW-1:`TC_AW-t0_addr_w] == t0_addr) |
 multitarg & (i2_wb_addr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
 t17_addr));
assign req_i[3] = i3_wb_cyc_i &
 ((i3_wb_addr_i[`TC_AW-1:`TC_AW-t0_addr_w] == t0_addr) |
 multitarg & (i3_wb_addr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
 t17_addr));
assign req_i[4] = i4_wb_cyc_i &
 ((i4_wb_addr_i[`TC_AW-1:`TC_AW-t0_addr_w] == t0_addr) |
 multitarg & (i4_wb_addr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
 t17_addr));
assign req_i[5] = i5_wb_cyc_i &
 ((i5_wb_addr_i[`TC_AW-1:`TC_AW-t0_addr_w] == t0_addr) |
 multitarg & (i5_wb_addr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
 t17_addr));
assign req_i[6] = i6_wb_cyc_i &
 ((i6_wb_addr_i[`TC_AW-1:`TC_AW-t0_addr_w] == t0_addr) |
 multitarg & (i6_wb_addr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
 t17_addr));
assign req_i[7] = i7_wb_cyc_i &
 ((i7_wb_addr_i[`TC_AW-1:`TC_AW-t0_addr_w] == t0_addr) |
 multitarg & (i7_wb_addr_i[`TC_AW-1:`TC_AW-t17_addr_w] ==
 t17_addr));

//
// Determine who gets current access to the target.
//
// If current initiator still asserts request, do nothing
// (keep current initiator).
// Otherwise check each initiator's request, starting from initiator 0
// (highest priority).
// If there is no requests from initiators, park initiator 0.
//
assign req_won = req_cont ? req_r :
 req_i[0] ? 3'd0 :
 req_i[1] ? 3'd1 :
 req_i[2] ? 3'd2 :
 req_i[3] ? 3'd3 :
 req_i[4] ? 3'd4 :
 req_i[5] ? 3'd5 :
 req_i[6] ? 3'd6 :
 req_i[7] ? 3'd7 : 3'd0;

//
// Check if current initiator still wants access to the target and if
// it does, assert req_cont.
//
always @(req_r or req_i)
 case (req_r) // synopsys parallel_case
 3'd0: req_cont = req_i[0];
 3'd1: req_cont = req_i[1];
 3'd2: req_cont = req_i[2];
 3'd3: req_cont = req_i[3];
 3'd4: req_cont = req_i[4];
 3'd5: req_cont = req_i[5];
 endcase

```

```

 3'd6: req_cont = req_i[6];
 3'd7: req_cont = req_i[7];
 endcase

//
// Register who has current access to the target.
//
always @(posedge wb_clk_i or posedge wb_rst_i)
 if (wb_rst_i)
 req_r <= #1 3'd0;
 else
 req_r <= #1 req_won;

endmodule

//
// Single initiator to multiple targets
//
module tc_si_to_mt (

 i0_wb_cyc_i,
 i0_wb_stb_i,
 i0_wb_cab_i,
 i0_wb_adr_i,
 i0_wb_sel_i,
 i0_wb_we_i,
 i0_wb_dat_i,
 i0_wb_dat_o,
 i0_wb_ack_o,
 i0_wb_err_o,

 t0_wb_cyc_o,
 t0_wb_stb_o,
 t0_wb_cab_o,
 t0_wb_adr_o,
 t0_wb_sel_o,
 t0_wb_we_o,
 t0_wb_dat_o,
 t0_wb_dat_i,
 t0_wb_ack_i,
 t0_wb_err_i,

 t1_wb_cyc_o,
 t1_wb_stb_o,
 t1_wb_cab_o,
 t1_wb_adr_o,
 t1_wb_sel_o,
 t1_wb_we_o,
 t1_wb_dat_o,
 t1_wb_dat_i,
 t1_wb_ack_i,
 t1_wb_err_i,

 t2_wb_cyc_o,
 t2_wb_stb_o,
 t2_wb_cab_o,
 t2_wb_adr_o,
 t2_wb_sel_o,
 t2_wb_we_o,
 t2_wb_dat_o,

```

t2\_wb\_dat\_i,  
t2\_wb\_ack\_i,  
t2\_wb\_err\_i,

t3\_wb\_cyc\_o,  
t3\_wb\_stb\_o,  
t3\_wb\_cab\_o,  
t3\_wb\_adr\_o,  
t3\_wb\_sel\_o,  
t3\_wb\_we\_o,  
t3\_wb\_dat\_o,  
t3\_wb\_dat\_i,  
t3\_wb\_ack\_i,  
t3\_wb\_err\_i,

t4\_wb\_cyc\_o,  
t4\_wb\_stb\_o,  
t4\_wb\_cab\_o,  
t4\_wb\_adr\_o,  
t4\_wb\_sel\_o,  
t4\_wb\_we\_o,  
t4\_wb\_dat\_o,  
t4\_wb\_dat\_i,  
t4\_wb\_ack\_i,  
t4\_wb\_err\_i,

t5\_wb\_cyc\_o,  
t5\_wb\_stb\_o,  
t5\_wb\_cab\_o,  
t5\_wb\_adr\_o,  
t5\_wb\_sel\_o,  
t5\_wb\_we\_o,  
t5\_wb\_dat\_o,  
t5\_wb\_dat\_i,  
t5\_wb\_ack\_i,  
t5\_wb\_err\_i,

t6\_wb\_cyc\_o,  
t6\_wb\_stb\_o,  
t6\_wb\_cab\_o,  
t6\_wb\_adr\_o,  
t6\_wb\_sel\_o,  
t6\_wb\_we\_o,  
t6\_wb\_dat\_o,  
t6\_wb\_dat\_i,  
t6\_wb\_ack\_i,  
t6\_wb\_err\_i,

t7\_wb\_cyc\_o,  
t7\_wb\_stb\_o,  
t7\_wb\_cab\_o,  
t7\_wb\_adr\_o,  
t7\_wb\_sel\_o,  
t7\_wb\_we\_o,  
t7\_wb\_dat\_o,  
t7\_wb\_dat\_i,  
t7\_wb\_ack\_i,  
t7\_wb\_err\_i

);

```

//
// Parameters
//
parameter t0_addr_w = 3;
parameter t0_addr = 3'd0;
parameter t17_addr_w = 3;
parameter t1_addr = 3'd1;
parameter t2_addr = 3'd2;
parameter t3_addr = 3'd3;
parameter t4_addr = 3'd4;
parameter t5_addr = 3'd5;
parameter t6_addr = 3'd6;
parameter t7_addr = 3'd7;

//
// I/O Ports
//

//
// WB slave i/f connecting initiator 0
//
input i0_wb_cyc_i;
input i0_wb_stb_i;
input i0_wb_cab_i;
input [`TC_AW-1:0] i0_wb_adr_i;
input [`TC_BSW-1:0] i0_wb_sel_i;
input i0_wb_we_i;
input [`TC_DW-1:0] i0_wb_dat_i;
output [`TC_DW-1:0] i0_wb_dat_o;
output i0_wb_ack_o;
output i0_wb_err_o;

//
// WB master i/f connecting target 0
//
output t0_wb_cyc_o;
output t0_wb_stb_o;
output t0_wb_cab_o;
output [`TC_AW-1:0] t0_wb_adr_o;
output [`TC_BSW-1:0] t0_wb_sel_o;
output t0_wb_we_o;
output [`TC_DW-1:0] t0_wb_dat_o;
input [`TC_DW-1:0] t0_wb_dat_i;
input t0_wb_ack_i;
input t0_wb_err_i;

//
// WB master i/f connecting target 1
//
output t1_wb_cyc_o;
output t1_wb_stb_o;
output t1_wb_cab_o;
output [`TC_AW-1:0] t1_wb_adr_o;
output [`TC_BSW-1:0] t1_wb_sel_o;
output t1_wb_we_o;
output [`TC_DW-1:0] t1_wb_dat_o;
input [`TC_DW-1:0] t1_wb_dat_i;
input t1_wb_ack_i;
input t1_wb_err_i;

//

```

```

// WB master i/f connecting target 2
//
output t2_wb_cyc_o;
output t2_wb_stb_o;
output t2_wb_cab_o;
output [`TC_AW-1:0] t2_wb_adr_o;
output [`TC_BSW-1:0] t2_wb_sel_o;
output t2_wb_we_o;
output [`TC_DW-1:0] t2_wb_dat_o;
input [`TC_DW-1:0] t2_wb_dat_i;
input t2_wb_ack_i;
input t2_wb_err_i;

//
// WB master i/f connecting target 3
//
output t3_wb_cyc_o;
output t3_wb_stb_o;
output t3_wb_cab_o;
output [`TC_AW-1:0] t3_wb_adr_o;
output [`TC_BSW-1:0] t3_wb_sel_o;
output t3_wb_we_o;
output [`TC_DW-1:0] t3_wb_dat_o;
input [`TC_DW-1:0] t3_wb_dat_i;
input t3_wb_ack_i;
input t3_wb_err_i;

//
// WB master i/f connecting target 4
//
output t4_wb_cyc_o;
output t4_wb_stb_o;
output t4_wb_cab_o;
output [`TC_AW-1:0] t4_wb_adr_o;
output [`TC_BSW-1:0] t4_wb_sel_o;
output t4_wb_we_o;
output [`TC_DW-1:0] t4_wb_dat_o;
input [`TC_DW-1:0] t4_wb_dat_i;
input t4_wb_ack_i;
input t4_wb_err_i;

//
// WB master i/f connecting target 5
//
output t5_wb_cyc_o;
output t5_wb_stb_o;
output t5_wb_cab_o;
output [`TC_AW-1:0] t5_wb_adr_o;
output [`TC_BSW-1:0] t5_wb_sel_o;
output t5_wb_we_o;
output [`TC_DW-1:0] t5_wb_dat_o;
input [`TC_DW-1:0] t5_wb_dat_i;
input t5_wb_ack_i;
input t5_wb_err_i;

//
// WB master i/f connecting target 6
//
output t6_wb_cyc_o;
output t6_wb_stb_o;
output t6_wb_cab_o;

```

```

output [`TC_AW-1:0] t6_wb_adr_o;
output [`TC_BSW-1:0] t6_wb_sel_o;
output t6_wb_we_o;
output [`TC_DW-1:0] t6_wb_dat_o;
input [`TC_DW-1:0] t6_wb_dat_i;
input t6_wb_ack_i;
input t6_wb_err_i;

//
// WB master i/f connecting target 7
//
output t7_wb_cyc_o;
output t7_wb_stb_o;
output t7_wb_cab_o;
output [`TC_AW-1:0] t7_wb_adr_o;
output [`TC_BSW-1:0] t7_wb_sel_o;
output t7_wb_we_o;
output [`TC_DW-1:0] t7_wb_dat_o;
input [`TC_DW-1:0] t7_wb_dat_i;
input t7_wb_ack_i;
input t7_wb_err_i;

//
// Internal wires & registers
//
wire [`TC_IIN_W-1:0] i0_in;
wire [`TC_TIN_W-1:0] i0_out;
wire [`TC_IIN_W-1:0] t0_out, t1_out,
 t2_out, t3_out,
 t4_out, t5_out,
 t6_out, t7_out;
wire [`TC_TIN_W-1:0] t0_in, t1_in,
 t2_in, t3_in,
 t4_in, t5_in,
 t6_in, t7_in;
wire [7:0] req_t;

//
// Group WB initiator 0 i/f inputs and outputs
//
assign i0_in = {i0_wb_cyc_i, i0_wb_stb_i, i0_wb_cab_i, i0_wb_adr_i,
 i0_wb_sel_i, i0_wb_we_i, i0_wb_dat_i};
assign {i0_wb_dat_o, i0_wb_ack_o, i0_wb_err_o} = i0_out;

//
// Group WB target 0 i/f inputs and outputs
//
assign {t0_wb_cyc_o, t0_wb_stb_o, t0_wb_cab_o, t0_wb_adr_o,
 t0_wb_sel_o, t0_wb_we_o, t0_wb_dat_o} = t0_out;
assign t0_in = {t0_wb_dat_i, t0_wb_ack_i, t0_wb_err_i};

//
// Group WB target 1 i/f inputs and outputs
//
assign {t1_wb_cyc_o, t1_wb_stb_o, t1_wb_cab_o, t1_wb_adr_o,
 t1_wb_sel_o, t1_wb_we_o, t1_wb_dat_o} = t1_out;
assign t1_in = {t1_wb_dat_i, t1_wb_ack_i, t1_wb_err_i};

//
// Group WB target 2 i/f inputs and outputs
//

```

```

assign {t2_wb_cyc_o, t2_wb_stb_o, t2_wb_cab_o, t2_wb_adr_o,
 t2_wb_sel_o, t2_wb_we_o, t2_wb_dat_o} = t2_out;
assign t2_in = {t2_wb_dat_i, t2_wb_ack_i, t2_wb_err_i};

//
// Group WB target 3 i/f inputs and outputs
//
assign {t3_wb_cyc_o, t3_wb_stb_o, t3_wb_cab_o, t3_wb_adr_o,
 t3_wb_sel_o, t3_wb_we_o, t3_wb_dat_o} = t3_out;
assign t3_in = {t3_wb_dat_i, t3_wb_ack_i, t3_wb_err_i};

//
// Group WB target 4 i/f inputs and outputs
//
assign {t4_wb_cyc_o, t4_wb_stb_o, t4_wb_cab_o, t4_wb_adr_o,
 t4_wb_sel_o, t4_wb_we_o, t4_wb_dat_o} = t4_out;
assign t4_in = {t4_wb_dat_i, t4_wb_ack_i, t4_wb_err_i};

//
// Group WB target 5 i/f inputs and outputs
//
assign {t5_wb_cyc_o, t5_wb_stb_o, t5_wb_cab_o, t5_wb_adr_o,
 t5_wb_sel_o, t5_wb_we_o, t5_wb_dat_o} = t5_out;
assign t5_in = {t5_wb_dat_i, t5_wb_ack_i, t5_wb_err_i};

//
// Group WB target 6 i/f inputs and outputs
//
assign {t6_wb_cyc_o, t6_wb_stb_o, t6_wb_cab_o, t6_wb_adr_o,
 t6_wb_sel_o, t6_wb_we_o, t6_wb_dat_o} = t6_out;
assign t6_in = {t6_wb_dat_i, t6_wb_ack_i, t6_wb_err_i};

//
// Group WB target 7 i/f inputs and outputs
//
assign {t7_wb_cyc_o, t7_wb_stb_o, t7_wb_cab_o, t7_wb_adr_o,
 t7_wb_sel_o, t7_wb_we_o, t7_wb_dat_o} = t7_out;
assign t7_in = {t7_wb_dat_i, t7_wb_ack_i, t7_wb_err_i};

//
// Assign to WB target i/f outputs
//
// Either inputs from the initiator are assigned or zeros.
//
assign t0_out = req_t[0] ? i0_in : {'TC_IIN_W{1'b0}};
assign t1_out = req_t[1] ? i0_in : {'TC_IIN_W{1'b0}};
assign t2_out = req_t[2] ? i0_in : {'TC_IIN_W{1'b0}};
assign t3_out = req_t[3] ? i0_in : {'TC_IIN_W{1'b0}};
assign t4_out = req_t[4] ? i0_in : {'TC_IIN_W{1'b0}};
assign t5_out = req_t[5] ? i0_in : {'TC_IIN_W{1'b0}};
assign t6_out = req_t[6] ? i0_in : {'TC_IIN_W{1'b0}};
assign t7_out = req_t[7] ? i0_in : {'TC_IIN_W{1'b0}};

//
// Assign to WB initiator i/f outputs
//
// Assign inputs from target to initiator outputs according to
// which target is accessed. If there is no request for a target,
// assign zeros.
//
assign i0_out = req_t[0] ? t0_in :

```



```

 req_t[1] ? t1_in :
 req_t[2] ? t2_in :
 req_t[3] ? t3_in :
 req_t[4] ? t4_in :
 req_t[5] ? t5_in :
 req_t[6] ? t6_in :
 req_t[7] ? t7_in : {'TC_TIN_W{1'b0}};

//
// Determine which target is being accessed.
//
assign req_t[0] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t0_addr_w]
== t0_addr);
assign req_t[1] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w]
== t1_addr);
assign req_t[2] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w]
== t2_addr);
assign req_t[3] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w]
== t3_addr);
assign req_t[4] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w]
== t4_addr);
assign req_t[5] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w]
== t5_addr);
assign req_t[6] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w]
== t6_addr);
assign req_t[7] = i0_wb_cyc_i & (i0_wb_adr_i[`TC_AW-1:`TC_AW-t17_addr_w]
== t7_addr);

endmodule

```