Escuela Técnica Superior de Ingeniería

Análisis y diseño de un procesador RISC simple para adquisición y proceso de datos

Memoria del proyecto final de carrera correspondiente a la titulación de Ingeniería Superior Informática realizado por Javier Gimbert Moreno y dirigido por Joan Oliver Malagelada.

Bellaterra, 14 de junio de 2007

El firmante, Joan Oliver Malagelada, profesor del Departamento de Microelectrónica y Sistemas Electrónicos de la Universidad Autónoma de Barcelona

CERTIFICA:

Que la presente memoria ha sido realitzada bajo su dirección por Javier Gimbert Moreno

Bellaterra, 14 de junio de 2007

Joan Oliver Malagelada

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Existen dos clases principales de microprocesadores: los microprocesadores computacionales de propósito general utilizados en servidores y ordenadores personales, y los microprocesadores especializados para sistemas empotrados y aplicaciones system-on-chip (SoC). Los primeros pueden contener uno o varios núcleos homogéneos utilizando memoria principal externa. Los últimos, en cambio, normalmente combinan uno o varios núcleos heterogéneos utilizando memoria en el interior del propio chip.

Aunque la mayoría de investigaciones actuales se centran en los procesadores computacionales de propósito general, éstos representan solo una pequeña parte del total de procesadores fabricados cada año. De hecho, los procesadores especializados para aplicaciones empotradas son los más abundantes hoy en día y representan la mayor parte de procesadores fabricados anualmente[16].

En algunos sectores del mercado de SoC, el costoso proceso de personalización de procesadores de alto rendimiento es difícil de justificar, por lo que muchas compañías prefieren diseñar y utilizar su propio procesador. Además, como los diseñadores a menudo conocen el tamaño necesario para almacenar los programas y los datos en tiempo de diseño, pueden incluir las estructuras de memoria necesarias, ajustándose al máximo.

Con múltiples procesadores (algunos especializados, otros genéricos), múltiples estructuras de memoria y controladores especializados, uno de los grandes objetivos de los SoC es diseñar una estructura de buses robusta que pueda cumplir con los requisitos de comunicaciones y permita una integración y verificación de sistemas complejos.

En la actualidad, algunas empresas suelen comprar procesadores genéricos como propiedades intelectuales (IP) para integrarlos dentro de sus diseños. Uno de los procesadores más populares en este sentido es el ARM 7 TDMA [17]. Este procesador bajo licencia es de 32 bits y de arquitectura RISC (*Reduced Instruction Set Computer*). Su éxito viene provocado en parte por su diseño compacto y en parte por la gran cantidad de componentes IP disponibles en el mercado que utilizan el mismo bus AMBA que el

procesador.

La filosofía de diseño RISC ya hace tiempo que se ha establecido como un concepto importante en el diseño de procesadores de propósito general [27]. Las principales características de la arquitectura RISC son la separación de las instrucciones de acceso a memoria de las operaciones aritméticas, además de una ejecución de instrucciones en pipeline. Estas características hacen posible que los procesadores RISC puedan trabajar a altas frecuencias de reloj y por lo tanto que consigan un gran rendimiento. Esta filosofía de diseño se usa como concepto básico en el diseño de microcontroladores y procesadores de propósito general.

En general, las arquitecturas RISC utilizan la técnica del pipeline todo lo posible para intentar paralelizar el máximo de tareas posibles, utilizando así los recursos hardware del procesador eficientemente. Esto permite trabajar con frecuencias de reloj más altas y por lo tanto proporcionar un mayor rendimiento. En los procesadores utilizados para servidores y ordenadores personales podemos observar arquitecturas de pipeline de diez, veinte y de incluso, más de treinta etapas. Pero el uso de un pipeline de muchas etapas puede tener algún inconveniente. Por ejemplo cuando se ejecuta una instrucción de salto, el pipeline se rompe y debe ser rellenado de nuevo con la instrucción a la que apuntaba la instrucción de salto. Cuantas más etapas tenga el pipeline más ciclos de reloj serán necesarios para volver a llenar el pipeline. En los sistemas empotrados, los bucles en los programas suelen ser relativamente cortos y suelen ser procesados más veces que en aplicaciones ofimáticas. Algunas investigaciones han demostrado que las instrucciones de salto pueden llegar al 30 % del total de instrucciones ejecutadas [15].

Este proyecto tiene como objetivo el análisis de prestaciones de procesadores RISC de bajo coste y el diseño de un procesador RISC simple para aplicaciones de propósito general relacionadas con la adquisición y el proceso simple de datos. El procesador propuesto debe poder ser ampliado en el futuro añadiéndole un pipeline, que siguiendo con la misma filosofía de simplicidad, tendrá que ser de pocas etapas. Además también se ha de poder acoplar fácilmente a sistemas más complejos mediante el cumplimiento de algún estándar de buses para SoC.

A continuación se muestran las características más importantes que el procesador ha de cumplir:

- Tiene que ser simple. Sus principales aplicaciones serán las relacionadas con la adquisición y el proceso simple de datos.

- El conjunto de instrucciones, a su vez, debe ser también simple y eficiente.

- Ha de poseer una interfaz compatible con algún estándar de buses.
- Se simulará y sintetizará mediante el lenguaje de descripción de hardware VHDL en una FPGA (*Field Programmable Gate Array*). Al ser un soft-core podrá utilizarse para aplicaciones reales en SoC.
- Se ha de acompañar con un simulador para poder utilizarlo como plataforma educativa [28].[1]

Los capítulos 2 y 3 proporcionan una visión del *state-of-the-art* sobre procesadores RISC simples y arquitecturas de buses para sistemas SoC actuales. Dado este *state-of-the-art* los siguientes capítulos ya se centran propiamente en el proyecto. De esta manera, en el capítulo 4 se realiza un análisis de sus prestaciones, además de comentar la viabilidad y costes del proyecto. En el capítulo 5 se describe el procesador implementado, y para finalizar, en los capítulos 6 y 7 se realiza un estudio de los resultados obtenidos y una conclusión, respectivamente. Al final del documento también se pueden encontrar varios Apéndices a los que se les irá realizando referencia a lo largo de este documento.

[1] Actualmente existen muchos procesadores descritos mediante lenguajes de descripción de hardware (normalmente Verilog o VHDL) sintetizables mediante FPGAs (se puede ver una extensa lista en [18] o en [9]). Sin embargo son muy pocos los procesadores que están orientados a ámbitos educativos, pero menos aún los que pueden utilizarse tanto para aplicaciones reales en SoC como para el ámbito educativo.

Capítulo 2

Análisis de procesadores simples

En este capítulo se lleva a cabo una breve descripción de los procesadores RISC simples más populares que existen actualmente y se realiza una comparación entre ellos a fin de ver sus principales ventajas y caréncias.

2.1. El Procesador DLX

El procesador DLX es un procesador teórico basado en arquitectura RISC diseñado por Hennessy y Patterson e introducido por primera vez en [20]. Cabe destacar que sus autores fueron a su vez los diseñadores principales de las arquitecturas MIPS y Berkeley RISC respectivamente.

El DLX es básicamente un procesador MIPS revisado y simplificado con una arquitectura simple de carga/almacenamiento (*load/store*) de 32 bits. Pensado principalmente para propósitos educativos, se utiliza ampliamente en cursos de nivel universitario sobre arquitectura de computadores.

Sus diseñadores se basaron en las observaciones sobre las primitivas más frecuentemente utilizadas en los programas para realizar su arquitectura. Al igual que la mayoría de procesadores de carga/almacenamiento el DLX hace enfasis en:

- Un sencillo repertorio de instrucciones de carga/almacenamiento.
- Diseño de segmentación (*pipelining*) eficiente.
- Un repertorio de instrucciones fácilmente decodificables.
- Eficiencia como objeto del compilador.

El procesador tiene 32 registros de propósito general (GPR) de 32 bits; el valor de R0 siempre es 0. Adicionalmente, hay un conjunto de registros de punto flotante (FPR),

que pueden utilizarse como 32 registros de simple precisión (32 bits), o como parejas par-impar que contienen valores de doble precisión. Se pueden realizar operaciones en simple y doble precisión. También existe un conjunto de registros especiales utilizados para acceder a la información sobre su estado, algunos de los cuales se pueden transfereir a y desde registros enteros.

La memoria es direccionable por bytes en el modo «Big Endian» con una dirección de 32 bits. Todas las referencias a memoria se realizan a través de cargas o almacenamientos entre memoria y los GPR o FPR. Todos los accesos a memoria deben estar alineados. También hay instrucciones para transferencia entre un FPR y un GPR.

Todas las instrucciones son de 32 bits y deben estar alineadas. Hay cuatro clases de instrucciones:

- Cargas y almacenamientos. Cualquiera de los registros de propósito general o de punto flotante se pueden cargar o almacenar, excepto cargar R0 que no tiene efecto. Hay un modo único de direccionamiento, registro base + desplazamiento de 16 bits con signo. Las cargas de media palabra y de byte ubican el objeto cargado en la parte inferior del registro.

- Operaciones de ALU. Todas las instrucciones de la ALU son instrucciones de registro-registro. Las operaciones incluyen operaciones aritméticas sencillas y operaciones lógicas: suma, resta, AND, OR, XOR, y desplazamientos (*shifts*). Se proporcionan las formas inmediatas de todas estas instrucciones, con un valor inmediato con signo-extendido a 16 bits. También hay instrucciones de comparación, que comparan dos registros. Si la condición es cierta, estas instrucciones colocan un 1 en el registro destino, en otro caso colocan el valor 0.

- Saltos y bifurcaciones. El control se realiza mediante un conjunto de bifurcaciones y saltos. Las tres instrucciones de bifurcación están diferenciadas por las dos formas de especificar la dirección destino y por si existe o no existe enlace. Dos bifurcaciones utilizan un desplazamiento (*offset*) con signo de 26 bits sumado al contador de programa; las otras dos instrucciones de bifurcación especifican un registro que contiene la dirección destino. Hay dos tipos de bifurcación: bifurcación simple, y bifurcación y enlace (utilizada para llamadas a procedimientos). La última coloca la dirección de retorno en el registro R31. Todos los saltos son condicionales. La condición de salto se especifica en la instrucción.

- Operaciones en punto flotante. Las instrucciones de punto flotante manipulan los regsitros de punto flotante e indican si la operación a realizqar es en simple o en doble precisión. Las operaciones de punto flotante son suma, resta, multiplicación y división. El formato de punto flotante es el del IEEE 754.

Formato	Bits																			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Tipo I	código op						rs1					rd					inmediato														
Tipo R	código op						rs1					rs2					rd					func									
Tipo J	código op						desplazamiento añadido al PC																								

Cuadro 2.1: Formato instrucciones DLX

Todas estas instrucciones se pueden separar en tres tipos de instrucciones (en el Cuadro 2.1 se puede observar sus formatos):

- Clase I *(inmediato)*. Estas instrucciones especifica un registro fuente, un registro destino y un valor inmediato de 16 bits.
- Clase R *(registro)*. Estas instrucciones especifican tres registros (dos registros fuente donde se encuentran los operandos y un registro destino para el resultado de la operación).
- Clase J *(salto)*. Estas instrucciones únicamente especifican un valor inmediato de 26 bits que es usado para calcular la dirección de destino.

Los códigos de instrucción contenidos en las instrucciones son de 6 bits, lo que hace un total de 64 posibles instrucciones básicas. Hacen falta 5 bits para seleccionar cada uno de los 32 registros de propósito general que incorpora el procesador.

El DLX, al igual que el MIPS, basa su rendimiento en el uso del pipeline. En el caso del DLX el pipeline se divide en cinco etapas (Figura 2.1):

- IF - Unidad de obtención de instrucción.
- ID - Unidad de decodificación de instrucción. Esta unidad toma la instrucción del IF, y extrae el código de operación y los operandos.
- EX - Unidad de ejecución. Ejecuta la instrucción, normalmente referido como ALU (Unidad Aritmético Lógica).
- MEM - Unidad de acceso a memoria. Obtiene datos de la memoria, controlada desde el ID y el EX.
- WB - Unidad de WriteBack o de almacenamiento.

Este procesador ha sido utilizado por otros autores en sus libros para explicar conceptos relacionados con la arquitectura de computadores o el diseño de sistemas digitales, como en el caso de Ashenden, que en su libro [13] describe, mediante lenguaje VHDL, el procesador DLX.

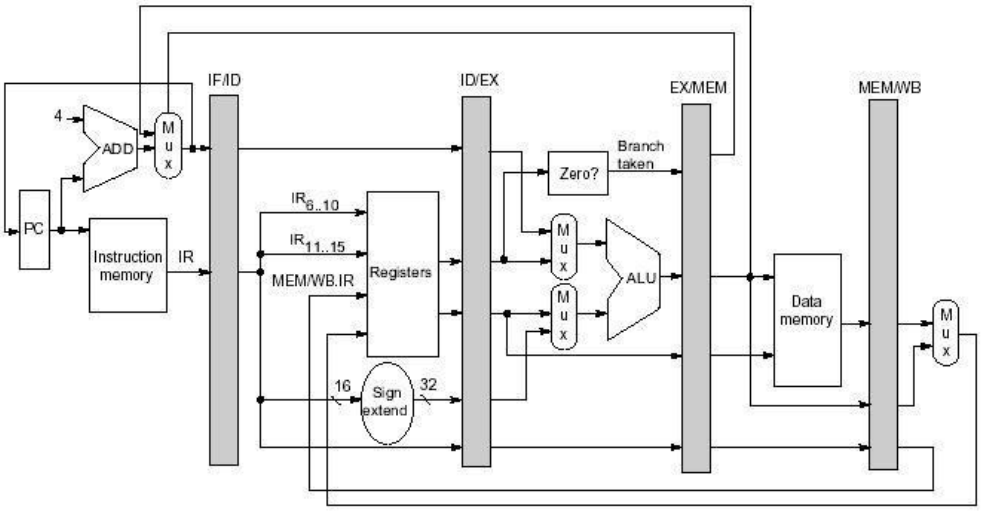

Figura 2.1: Arquitectura del procesador DLX

Existe una aproximación más sencilla de este procesador denominada DLXS (la S proviene de Stuttgart) diseñada por Gumm [19] y utilizada para propósitos educativos en la universidad de Stuttgart [14].[1] La principal diferencia del DLXS respecto al DLX original es que no incorpora pipeline, simplificando enormemente su arquitectura y funcionamiento. Esta basado en una arquitectura de tres buses y la mayoría de instrucciones se ejecutan en cinco ciclos.

2.2. El Procesador MicroBlaze™

El procesador MicroBlaze™[8] es un procesador soft-core de arquitectura RISC de 32 bits diseñado por Xillinx® y optimizado para funcionar en sus FPGAs. Se puede observar un esquema de su arquitectura en la Figura 2.2.

Este procesador permite un cierto grado de configuración dependiendo de las necesidades. Las características fijas que incluye el procesador son las siguientes:

- 32 registros de propósito general de 32 bits.
- Instrucciones de 32 bits con tres operandos y dos tipos de direccionamiento.
- Bus de direcciones de 32 bits.
- Pipeline de 3 o 5 etapas.

[1] Además de utilizarse para proposito educativo, el procesador DLXS se ha llegado a sintetizar en un ASIC [12].

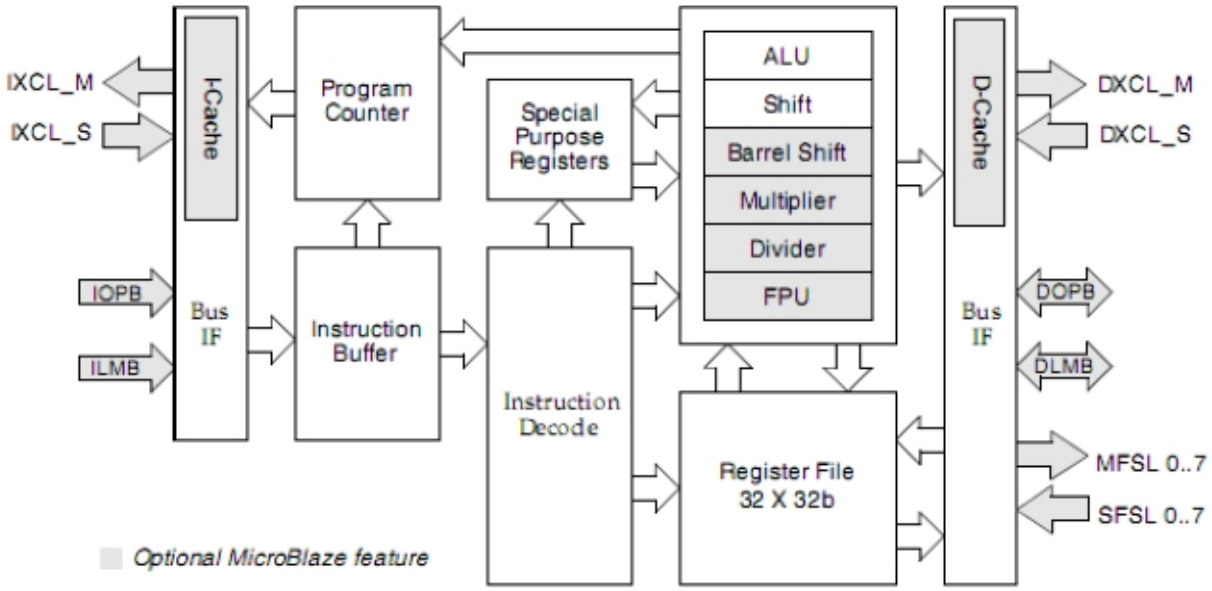

Figura 2.2: Arquitectura del procesador MicroBlaze™

Además de las características fijas, el procesador MicroBlaze se puede configurar para habilitar nuevas funcionalidades. Las versiones antiguas de este procesador solo soportan algunas de estas características adicionales, en cambio, la última versión de este procesador soporta todas las características adicionales. Por eso, Xillinx recomienda el uso de la última versión para la realización de nuevos proyectos.

MicroBlaze utiliza el formato de representación de datos «Big-endian». Soporta trabajar con datos de 8, 16 y 32 bits.

Las instrucciones de este procesador son de 32 bits y se pueden clasificar en dos tipos (en el Cuadro 2.2 se puede observar sus formatos):

- Instrucciones de tipo A. Estas instrucciones especifican dos registros fuente y un registro destino.

- Instrucciones de tipo B. Estas instrucciones especifican un registro fuente y un operando inmediato de 16 bits, el cual se puede extender a 32 bits precediendo a la instrucción con una instrucción especial llamada IMM). Las instrucciones de tipo B también especifican un registro destino.

	Bits				
Formato	0 5	6 10	11 15	16 20	21 31
Tipo A	código op	rd	rs1	rs2	func
Tipo B	código op	rd	rs1	inmediato	

Cuadro 2.2: Formato instrucciones MicroBlaze™

Las instrucciones, a su vez, se pueden clasificar según su funcionalidad en las siguientes categorías:

- Instrucciones aritméticas.
- Instrucciones lógicas.
- Instrucciones de salto.
- Instrucciones de carga/almacenamiento.
- Instrucciones especiales.

El procesador MicroBlaze incorpora 32 registros de propósito general de 32 bits y hasta 18 registros de propósito especial, dependiendo de las opciones de configuración elegidas. el valor del registro R0 siempre es 0. El registro R14 se utiliza para almacenar la dirección de retorno tras tratar una interrupción. Si el procesador se configura de forma que éste soporte las excepciones hardware, en el registro R17 se almacena la dirección de la instrucción siguiente a la que ha causado la excepción. El registro especial MSR (*Machine Status Register*) contiene bits de control y estado del procesador.

La ejecución de las instrucciones del procesador se realiza a través de un pipeline. Cada etapa del pipeline tarda un ciclo de reloj en ejecutarse para la mayoría de instrucciones. Por lo tanto, el número de ciclos necesarios para completar cada instrucción es igual al número de etapas del pipeline. En definitiva, en cada ciclo de reloj se completa una instrucción. Únicamente unas cuantas instrucciones requieren varios ciclos de reloj para completar alguna etapa del pipeline. Para solucionar este problema el pipeline se paraliza.

Cuando se utiliza una memoria lenta, la búsqueda de instrucciones puede requerir varios ciclos. Esta latencia adicional afecta directamente a la efectividad del pipeline. El procesador MicroBlaze implementa un buffer de instrucciones buscadas que reduce el impacto producido por esta latencia. De esta forma, cuando el pipeline requiere más de un ciclo para realizar la etapa de ejecución, el buffer de instrucciones se continúa llenando secuencialmente. Cuando el pipeline acaba la etapa de ejecución, la etapa de búsqueda puede cargar una nueva instrucción directamente del buffer sin verse afectado por la latencia de la memoria.

Cuando se habilita la optimización de área, el pipeline se divide en tres etapas para minimizar el coste de hardware. En cambio cuando se deshabilita, el pipeline se divide en cinco etapas para optimizar el rendimiento.

Este procesador esta implementado con una arquitectura de memoria tipo Harvard, lo que significa que los accesos a instrucciones y a datos se realizan en espacios de direccionamiento diferentes. Cada espacio de direccionamiento es de 32 bits. Los datos deben estar alineados en memoria, aunque el procesador está configurado para soportar

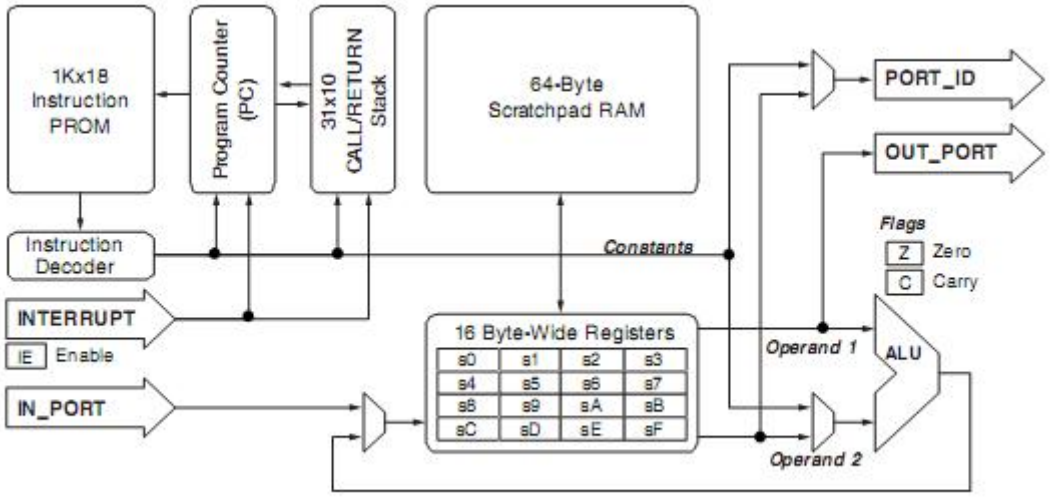

Figura 2.3: Arquitectura del procesador PicoBlaze™

excepciones de no alineamiento. El procesador no separa los accesos a datos de memoria de los accesos a datos de E/S (Entrada/Salida), es decir, el procesador utiliza E/S mapeada en memoria.

Xillinx también posee otro procesador mucho más sencillo y menos potente llamado PicoBlaze™.

El procesador PicoBlaze [5], es un microcontrolador compacto basado en arquitectura RISC de 8 bits, diseñado y optimizado para ser sintetizado en las FPGAs de la propia empresa. El procesador sin embargo es de código abierto y se proporciona en lenguaje VHDL sintetizable, para su uso en FGPAs Xillinx.

Xillinx también proporciona un ensamblador, un entorno gráfico de desarrollo integrado, y un simulador gráfico. PicoBlaze consigue un rendimiento de entre 44 y 100 MIPS (Millones de instrucciones por segundo) dependiendo de la familia de FPGAs en las que se sintetice.

El procesador incluye 16 registros de 8 bits cada uno, puede ejecutar programas de hasta 1024 instrucciones y posee sistema de interrupciones. En la Figura 2.3 se puede observar su arquitectura.

2.3. El Procesador Nios®II

El procesador Nios®II [10] es un procesador de arquitectura RISC de 32 bits y softcore, desarrollado por el fabricante de FPGAs Altera® y optimizado para funcionar en sus FPGAs. En la Figura 2.4 se puede ver un esquema de su arquitectura.

Un sistema Nios II es equivalente a un microcontrolador o a un «ordenador on-

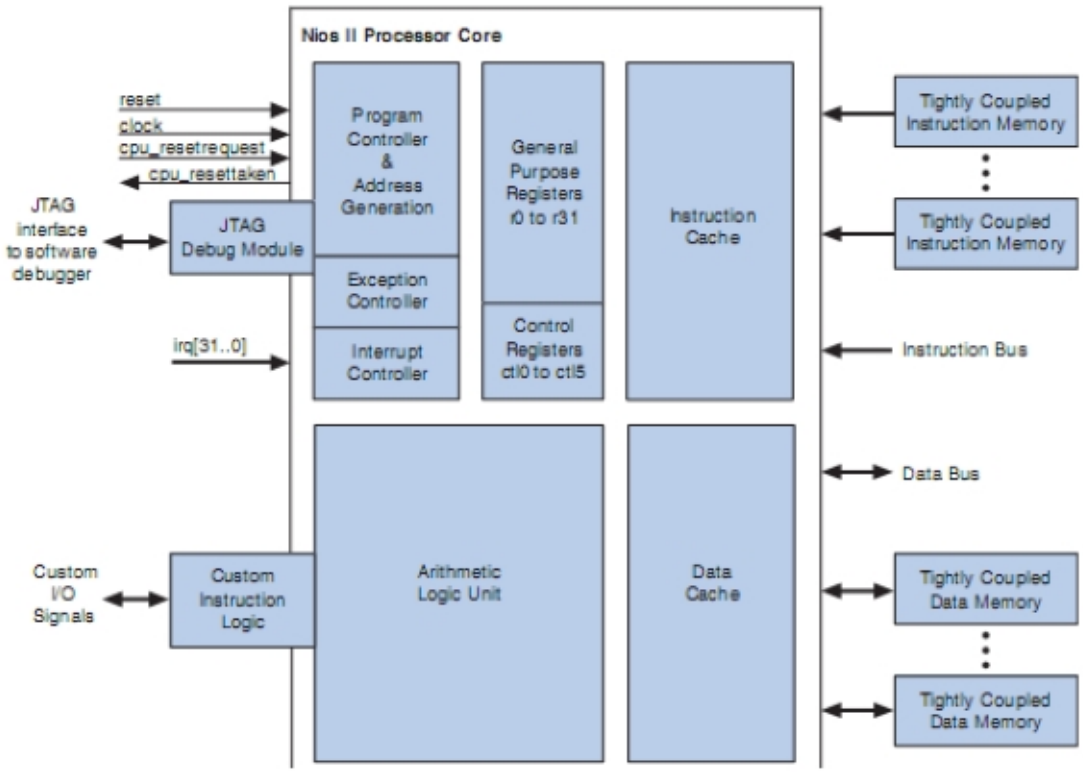

Figura 2.4: Arquitectura del procesador Nios®II

chip», que incluye un procesador y una combinación de periféricos y memoria en un único chip.

Las principales características del procesador son las siguientes:

- Arquitectura RISC, conjunto de instrucciones y espacio de direcciones de 32 bits.
- 32 Registros de propósito general de 32 bits.
- Soporte para 32 fuentes externas de interrupción.
- Instrucciones de multiplicación y división de 32 bits.
- Instrucciones dedicadas para computo de productos de 64 y 128 bits de números en punto flotante.
- Incorpora un barrel shifter.
- Proporciona acceso a múltiples periféricos dentro del propio chip e interfaces para memorias y periféricos externos.
- Rendimiento de hasta 250 DMIPS (*Dhrystone Million Instructions Per Second*).

El procesador Nios II es un procesador soft-core configurable. Esto significa que se le pueden añadir funcionalidades aumentando su rendimiento o quitárselas disminuyendo

su área, según las necesidades del sistema.

Existen tres configuraciones básicas del procesador que a su vez se pueden personalizar:

- **Nios II/e**. El Nios II/"*economy*" esta diseñado para ocupar el mínimo espacio posible. Como resultado este procesador tiene unas funcionalidades limitadas y algunas configuraciones no están disponibles para este procesador.

- **Nios II/s**. El Nios II/"*standard*" esta diseñado para ocupar un espacio reducido pero manteniendo el rendimiento.

- **Nios II/f**. El Nios II/"*fast*" esta diseñado para obtener el máximo rendimiento. Como resultado este procesador presenta un elevado rango de opciones de personalización.

El procesador incorpora 32 registros de propósito general de 32 bits, además de seis registros de 32 bits de control. La arquitectura está preparada para albergar registros de punto flotante en futuras versiones del procesador.

La arquitectura de este procesador soporta buses de instrucciones y de datos separados, clasificándose asi como una arquitectura de tipo Harvard. Ambos buses se implementan como puertos maestros del bus Avalon (ver Capítulo 3). El puerto de datos se conecta a componentes de memoria y a periféricos, mientras que el puerto de instrucciones solo se conecta a componentes de memoria. El Nios II utiliza E/S mapeada en memoria y el formato de representación de datos es «Little-endian». La arquitectura soporta cinco tipos de direccionamiento diferentes.

La ALU de este procesador opera con los datos almacenados en los registros de propósito general. Las operaciones pueden requerir uno o dos operandos y almacenan el resultado en un registro. La ALU de este procesador soporta operaciones aritméticas, de comparación, lógicas y de desplazamiento y rotaciones. Las operaciones más complejas se realizan mediante software utilizando combinaciones de las anteriores. El procesador además soporta instrucciones definidas por el propio usuario, incorporando la lógica necesaria para implementarlas en hardware, a la ALU. Las operaciones en punto flotante están soportadas según la especificación IEEE 754.

Las instrucciones se pueden clasificar de manera similar al procesador DLX, aunque con alguna diferencia (en el Cuadro 2.3 se puede observar sus formatos):

- Clase I *(inmediato)*. Estas instrucciones especifican un registro fuente, un registro destino y un valor inmediato de 16 bits.

- Clase R *(registro)*. Estas instrucciones especifican tres registros (dos registros fuente donde se encuentran los operandos y un registro destino para el resultado de la operación), además de un campo de extensión de operaciones.

- Clase J *(salto)*. Estas instrucciones únicamente especifican un valor inmediato de 26 bits que es usado para calcular la dirección de destino.

Formato	Bits																															
	0					5	6										16	17					21	22					26	27				31
Tipo I	código op						inmediato																	A						B			
Tipo R	código op						OPX											C						B						A			
Tipo J	código op						inmediato																											

Cuadro 2.3: Formato instrucciones Nios®II

La configuración más sencilla de este procesador (Nios II/e) no incorpora pipeline obteniendo una tasa de 6 ciclos por instrucción en la mayoría de estas. En cambio, tanto la configuración media (Nios II/s), como la configuración más potente (Nios II/f) de este procesador incorporan un pipeline de 5 y 6 etapas, respectivamente. Además estas dos últimas configuraciones también incorporan un predictor de saltos.

Tal como su nombre indica, el Nios II tiene un predecesor: el Nios [4]. Este procesador soporta dos tipos de arquitecturas: de 16 o de 32 bits, aunque ambas trabajan con instrucciones de 16 bits. Incorpora un banco de registros en ventana, disponiendo de un máximo de 512 registros de propósito general. En cada ventana son visibles 24 registros más 8 globales fijos. Esta técnica permite llamadas a subrutinas muy rápidas y eficientes. También incorpora un pipeline de 5 etapas con memorias de instrucciones y datos separadas (Arquitectura Harvard). Al igual que el Nios II también trabaja con el bus Avalon.

2.4. El Procesador AVR®32

El procesador AVR®32 [6] pertenece a la compañia Atmel®. Posee una arquitectura innovadora y un diseño completamente síncrono y sintetizable. Se puede integrar en SoC fácilmente debido a que Atmel lo proporciona como IP (*Intellectual Property*). Se ha diseñado para intentar conseguir densidades de código óptimas. Además de bajar los requisitos de la memoria, un código compacto también contribuye a disminuir el consumo.

La arquitectura AVR32 define varias microarquitecturas para intentar así llegar a un rango más amplio de aplicaciones. Cada microarquitectura está especializada en un

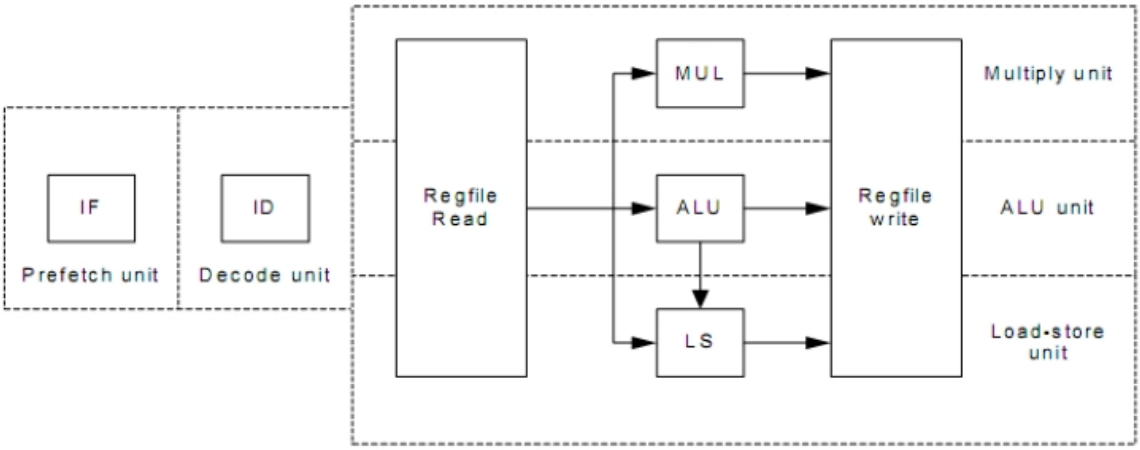

Figura 2.5: Pipeline del procesador AVR®32

tipo de aplicación final diferente, dando la opción al diseñador de elegir la que más se adecue a su sistema.

El procesador dispone de 15 registros de propósito general, además del PC (*Program Counter*), el registro de enlace, y el registro SP (*Stack Pointer*). Adicionalmente dispone de un registro donde se almacena el valor de retorno de una función que es usado implícitamente por algunas instrucciones.

Las operaciones de carga y almacenamiento se pueden realizar para tamaños de 8, 16, 32 y 64 bits. Los datos se suelen guardar en formato «Big-endian» y todas las instrucciones son interpretadas así. De todas maneras, para soportar transferencias de datos en formato «Little-endian» el procesador dispone de instrucciones de carga y almacenamiento especiales, capaces de modificar el formato de Big a Little-endian y viceversa. El sistema de memoria soporta accesos no alineados para algunas instrucciones de carga y almacenamiento. Cualquier otro acceso no alineado provoca una excepción de dirección. La E/S es mapeada en memoria.

El conjunto de instrucciones del AVR32 incluye dos tipos de instrucciones: compactas y extendidas. Las instrucciones compactas tienen una longitud de 16 bits mientras que las extendidas tienen una longitud de 32 bits. De esta manera, el procesador posee un conjunto de instrucciones con formatos muy diferentes dependiendo de la instrucción. Además para conseguir una reducción de código, algunas instrucciones tienen múltiples modos de direccionamiento, siendo el compilador el que elige el necesario en cada momento.

La arquitectura AVR32 define un acelerador hardware de Java® como opción, en forma de máquina virtual de Java hardware.

Los últimos procesadores AVR32 disponen de un pipeline de 3 etapas. En la primera etapa, se busca y almacena una instrucción de 32 bits o dos de 16 bits en el buffer de

instrucciones, cada ciclo de reloj. La segunda etapa decodifica la instrucción y genera las señales necesarias para la ejecución de la misma. Por último, la tercera etapa ejecuta la instrucción mediante una de sus tres unidades: ALU, unidad de multiplicación y unidad de carga/almacenamiento. No existen problemas de dependencias de datos en este pipeline, por lo que los registros pueden ser actualizados en el mismo ciclo en que la instrucción se está ejecutando. En la Figura 2.5 se puede ver un esquema del pipeline del AVR32.

Atmel también dispone de procesadores AVR de 8 bits mucho más sencillos que los de la familia AVR32. Estos procesadores de 8 bits trabajan con instrucciones de 16 bits y disponen de 32 registros de propósito general de 8 bits. Al igual que los AVR32, la E/S es mapeada en memoria y pueden direccionar hasta 26 bits de memoria. Disponen de un pipeline sencillo de 2 etapas, pudiendo ejecutar la mayoría de instrucciones en un ciclo de reloj. Al igual que la arquitectura AVR32, esta también dispone de diversas versiones, con rendimientos y características diferentes dependiendo de las aplicaciones en las que se vayan a utilizar.

2.5. El Procesador LatticeMico32™

El procesador LatticeMico32™ [7] es un procesador soft-core configurable para las FPGAs de la propia empresa Lattice®. Combina una arquitectura RISC de 32 bits con 32 registros de propósito general, proporcionando un rendimiento y flexibilidad adecuados para una gran variedad de aplicaciones.

Con buses de instrucciones y de datos separados, este procesador de arquitectura Harvard es capaz de ejecutar una instrucción por ciclo de reloj. Proporciona un conjunto de instrucciones simple, pero eficiente. En la Figura 2.6 se puede ver un esquema de su arquitectura.

Algunas de las características principales de este procesador son las siguientes:

- Arquitectura RISC de 32 bits.
- Instrucciones de 32 bits.
- 32 registros de propósito general.
- Hasta 32 interrupciones externas.
- Caché de instrucciones y de datos opcionales.
- Interfaz de memoria dual (instrucciones y datos) Wishbone (Ver Capítulo 3).

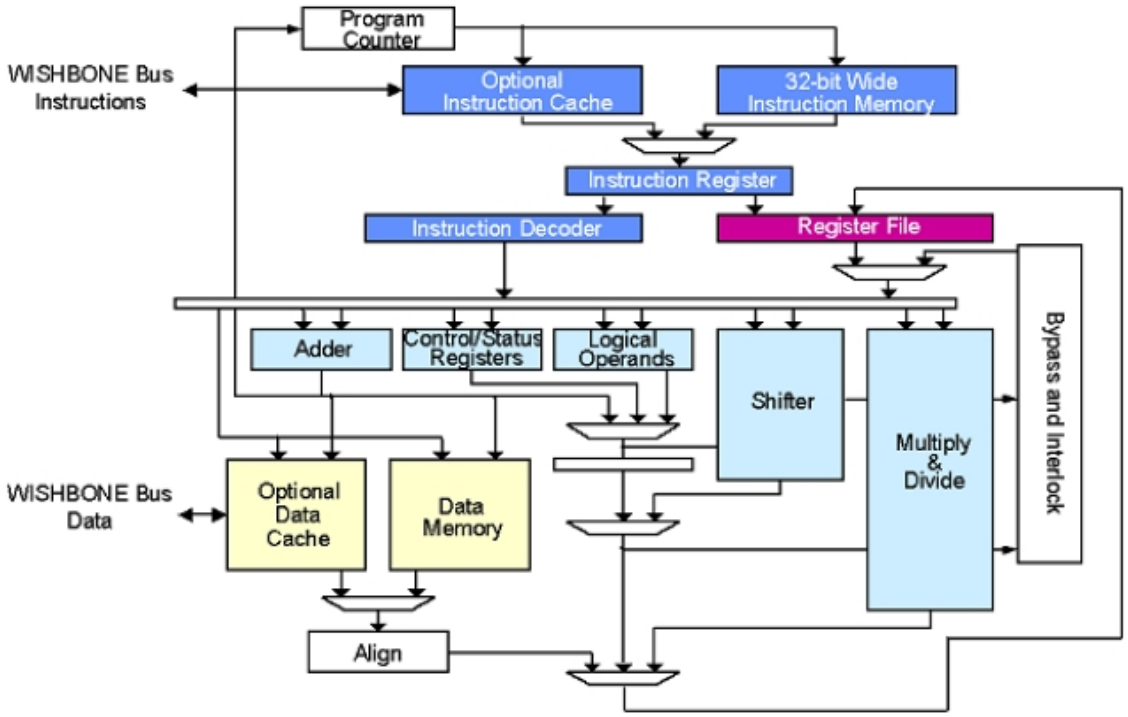

Figura 2.6: Arquitectura del procesador LatticeMico32^{TM}

Este procesador utiliza un pipeline de 6 etapas. Incorpora un detector de dependencias *read-after-write* que permite una compactación del código al eliminar las instrucciones nop.

Como ya se ha comentado, el procesador incorpora 32 registros de propósito general de 32 bits cada uno. Al igual que en otros procesadores, el registro R0 siempre contiene el valor 0, pero al principio de cada programa el usuario tiene que cargar manualmente este valor en el registro ya que éste no se resetea a 0 automáticamente. El registro R29 lo utiliza la instrucción de llamada a función para guardar el valor de retorno. El registro R30 se utiliza para almacenar el valor del PC (*Program Counter*) en caso de que ocurra una excepción y el registro R31 se utiliza para lo mismo, pero en caso de que ocurra una excepción de breakpoint. Además, el procesador incluye otros registros de control y estado.

El procesador tiene un espacio de direcciones de 32 bits direccionable por bytes. La E/S es mapeada en memoria y en caso de utilizar una memoria caché, se mapea en la parte no cacheable del espacio de direcciones. El procesador utiliza el formato «Big-endian».

El conjunto de instrucciones del LatticeMico32 se divide en las siguientes categorías:

- Instrucciones aritméticas. Operaciones de 32 bits aritméticas estándar. Las instrucciones de multiplicación y de división son opcionales.

- Instrucciones lógicas. Operaciones lógicas de 32 bits estándar.
- Instrucciones de comparación. Operaciones de comparación básicas entre registros, entre registro y valor inmediato, y comparaciones con signo y sin signo.
- Instrucciones de desplazamiento. El número de bits a desplazar se indican mediante un registro o mediante un valor inmediato de 5 bits.
- Instrucciones de transferencia de datos. Las transferencias de memoria pueden ser de 8, 16 o 32 bits entre memoria y registros. Las direcciones de memoria son relativas y se consiguen mediante la suma de un registro y un valor inmediato con signo de 16 bits. También existen instrucciones de transferencia de datos entre registros.
- Instrucciones de control de programa. Incluyen instrucciones de salto, llamadas a funciones y excepciones, y retornos.

No todas las instrucciones están disponibles en todas las configuraciones del procesador. Todas las instrucciones tienen alguno de los cuatro formatos que se pueden ver en el Cuadro 2.4.

Formato	Bits					
	31	30 26	25 21	20 16	15 11	10 0
Tipo RI	0	código op	Reg 0	Reg 1	inmediato	
Tipo RR	1	código op	Reg 0	Reg 1	Reg 2	00x00
Tipo CR	1	código op	CSR	Reg	x0000	
Tipo I	1	código op	inmediato			

Cuadro 2.4: Formato instrucciones LatticeMico32™

2.6. Simple RISC Computer

El procesador SRC (*Simple RISC Computer*), al igual que el DLX, es un procesador teórico introducido por Heuring y Jordan en [21]. Este procesador, como su propio nombre indica, se basa en una arquitectura RISC de 32 bits. Los autores proporcionan la descripción RTN (*Register Transfer Notation*) del procesador y el diseño de circuitos lógicos correspondientes a componentes del procesador.

El procesador incorpora 32 registros de propósito general de 32 bits cada uno, además de un registro contador de programa y un registro de instrucción.

La memoria principal está organizada como un vector de 8 bits y únicamente se puede trabajar con datos de 32 bits. El procesador puede llegar a direccionar hasta 4 Gb de memoria gracias a sus 32 bits. Los valores almacenados en la memoria

únicamente pueden ser accedidos mediante la utilización de instrucciones de carga y almacenamiento. La E/S es mapeada en memoria.

El código de operación de cada instrucción es de 5 bits y de tamaño fijo, pudiendo llegar a tener, por lo tanto, hasta 32 instrucciones diferentes. Sin embargo, el procesador original introducido en [21] únicamente utiliza 28 códigos de operación diferentes, de los cuales cinco son utilizados para dar soporte al sistema de interrupciones. El conjunto de instrucciones del SRC se divide en las siguientes categorías:

- Instrucciones de carga/almacenamiento. Existen cuatro instrucciones de carga y dos de almacenamiento.
- Instrucciones de salto. Existen dos instrucciones de salto que permiten realizar saltos condicionales e incondicionales a posiciones de memoria almacenadas en registros. Una de ellas permite realizar saltos almacenando el contenido del registro PC (*Program Counter*) en el registro indicado.
- Instrucciones aritméticas. Existen cuatro instrucciones aritméticas, algunas de ellas con sus variantes de un operando como valor inmediato.
- Instrucciones lógicas y de desplazamiento. Existen nueve instrucciones de este tipo. En las operaciones de desplazamiento, el desplazamiento en sí se puede encontrar en un registro o en un valor inmediato.
- Instrucciones varias. Operaciones sin operandos.

El formato de las instrucciones del SRC se puede ver en el Cuadro 2.5.

Este procesador no dispone de ninguna unidad ni soporte para números en coma flotante, por lo que su diseño se simplifica bastante. Si dispone de control de interrupciones basado en un único par de líneas.

	Bits						
Formato	31 27	26 22	21 17	16 12	11 5	4 3	2 0
Tipo 1	código op	ra	rb	c2			
Tipo 2	código op	ra	c1				
Tipo 3	código op	ra	sin uso	rc	sin uso		
Tipo 4	código op	sin uso	rb	rc	sin uso		cond
Tipo 5	código op	ra	rb	rc	sin uso		cond
Tipo 6	código op	ra	rb	rc	sin uso		
Tipo 7 a	código op	ra	rb	sin uso		count	
Tipo 7 b	código op	ra	rb	rc	sin uso	count	
Tipo 8	código op	sin uso					

Cuadro 2.5: Formato instrucciones SRC

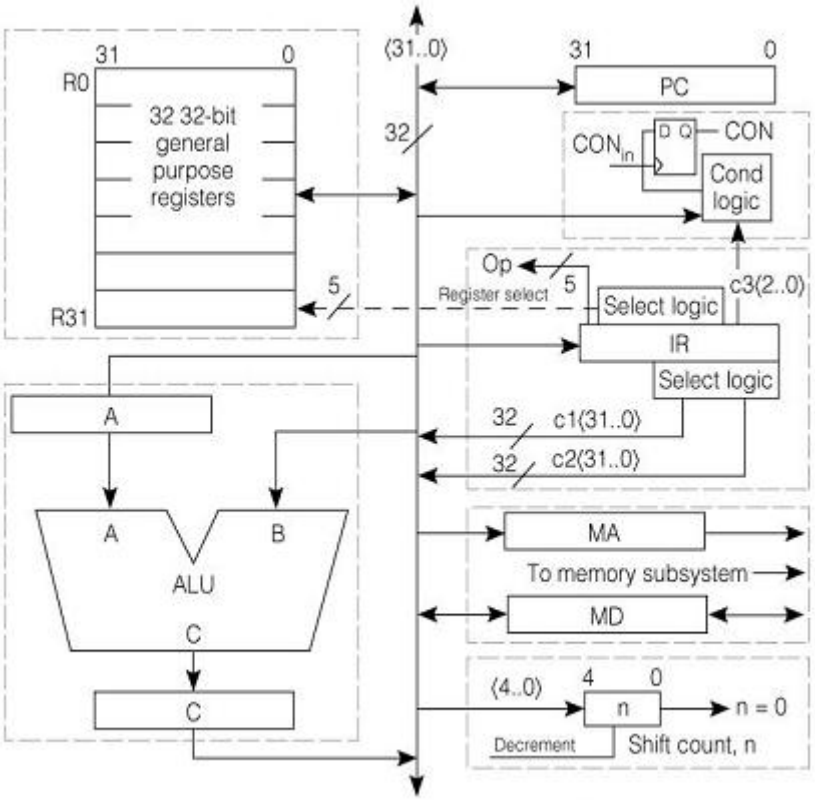

Figura 2.7: Arquitectura de 1 Bus del procesador SRC

En [22] se describe el procesador SRC en su versión de un único bus interno (Figura 2.7), aunque también se muestran brevemente las arquitecturas de dos y tres buses. Además, en el mismo texto también se introduce una versión con pipeline de cinco etapas.

El procesador SRC, en su variante de un bus único se ha llegado a sintetizar en un ASIC [23].

2.7. Comparativa de procesadores

En el Cuadro 2.6 se puede ver un resumen de las características principales de los procesadores analizados en este capítulo. Básicamente las características mostradas hacen referencia a sus arquitecturas, conjunto de instrucciones y rendimientos. También se muestra el año de aparición de cada procesador.

Los datos referentes al pipeline únicamente se muestran si el procesador o alguna de sus variantes posee pipeline.

Los valores de rendimiento mostrados son el CPI (Ciclos por instrucción) y los MIPS (Millones de Instrucciones Por Segundo). Estos valores, en su mayoría, se han

obtenido de los propios fabricantes o diseñadores. Existe alguna excepción, como en el caso del procesador DLX y su variante DLXS, o el SRC. En concreto, en el caso del DLX no se han conseguido valores sobre su rendimiento en MIPS, al no encontrar ninguna implementación de este procesador. En el caso del DLXS, los datos relativos al CPI y MIPS se han obtenido de terceras partes [12, 14], al igual que en el caso de los MIPS del SRC [23].

Para el cálculo del CPI de las diferentes versiones del procesador SRC (1 bus, 2 buses y 3 buses) es necesario conocer los ciclos de reloj necesarios para ejecutar cada instrucción, y la mezcla típica de instrucciones. El cálculo de la mezcla típica de instrucciones del procesador es una tarea muy complicada y muy variable respecto al código del programa a ejecutar. Por ello se han tomado unos valores aproximados, que aunque no sirvan para calcular un CPI exacto, si pueden ser de utilidad a la hora de realizar comparaciones:

`alu`	`50%`
`salto`	`20%`
`carga`	`20%`
`almacenamiento`	`10%`

Características	Procesadores									
	DLX	*DLXS*	*MicroBlaze*	*PicoBlaze*	*Nios II*	*Nios*	*AVR32*	*AVR*	*LatticeMico32*	*SRC*
Bits	32	32	32	8	32	32 - 16	32	8	32	32
Arquitectura	Pipeline	1 Bus	Pipeline	No Pipeline	No Pipeline / Pipeline	Pipeline	Pipeline	Pipeline	Pipeline	1, 2 o 3 Buses / Pipeline
Pipeline	5 etapas		3–5 etapas		5–6 etapas	5 etapas	3 etapas	2 etapas	6 etapas	5 etapas
Tamaño instrucción	32 bits	32 bits	32 bits	18 bits	32 bits	16 bits	32 y 16 bits	16 bits	32 bits	32 bits
Instrucciones	83	52	55	37	83	75-63	220	138	67	50
Tipos de direccionamiento	1	2	2	2	5	5	6	13	1	5
Registros	32 de 32b 32 de 64b	32 de 32b	32 de 32b	16 de 8b	32 de 32b	hasta 512 de 16b	15 de 32b	32 de 8b	32 de 32b	32 de 32b
Bus de direcciones	32 bits	32 bits	32 bits	8 bits	32 bits	32-16 bits	32 bits	16 bits	32 bits	32 bits
Tipo de E/S	Mapeada memoria	Mapeada memoria	Mapeada memoria	512 puertos	Mapeada memoria	Mapeada memoria	Mapeada memoria	Mapeada memoria	Mapeada memoria	Mapeada memoria
Soporte números flotantes	SI IEEE 754	NO	SI (opcional) IEEE 754	NO	SI	NO	NO	NO	NO	NO
Operaciones complejas	mult div barrel shifter	barrel shifter	mult div barrel shifter	mult - sw div - sw	mult div barrel shifter	mult barrel shifter	mult div - sw barrel shifter	mult barrel shifter	mult div barrel shifter	NO
CPI	1	5	1	2	6 / 1	1	1	1	1	6,5 / 5,7 / 4,7 / 1
MIPS		10 – 6.6	240 – 94 [1]	100 – 44	218 – 31 [1]	80 – 30	210 [1]	20	116 – 81	20 – 3
Fecha anunciada	1990	1995	2002	2004	2004	2001	2006	1996	2006	1997

[1] DMIPS (Dhrystone Million Instructions Per Second). Cantidad resultante de dividir la puntuación obtenida en el Dhrystone Benchmark entre 1,575 (número de DPS - Dhrystones Per Second obtenidos por el computador VAX 11/780, normalizado como computador de 1 MIPS).

Cuadro 2.6: Resumen procesadores analizados

Capítulo 3

Análisis de buses

En este capítulo se lleva a cabo una breve descripción de tres tipos de arquitecturas de buses diferentes. Estas tres arquitecturas de buses son de las más populares actualmente. El bus AMBA™ se utiliza en prácticamente todos los sistemas con procesadores ARM. El bus Avalon se utiliza en los sistemas con procesadores soft-core Nios II de Altera. Por último el bus Wishbone se utiliza en bastantes diseños, sobretodo, dentro del proyecto Opencores [9].

Las tres arquitecturas de buses analizadas en este capítulo persiguen el mismo objetivo: conectar componentes IP. Los tres tipos de buses proporcionan un proceso de negociación y buses de datos de tamaños variables. Ninguno de ellos especifica una frecuencia de reloj determinada, ya que esto supondría un serio problema a la hora de conectar componentes de diferentes diseñadores.

3.1. Bus AMBA™

El bus AMBA™ (*Advanced Microcontroller Bus Architecture*) se introdujo en 1996. Se utiliza sobretodo en SoC basados en procesadores ARM, aunque es independiente de estos. Los primeros buses AMBA fueron el ASB (*Advanced System Bus*) y el APB (*Advance Peripheral Bus*). En la segunda generación se incluyó el AHB (*Advanced High Performance Bus*) [1].

AMBA define un sistema de buses multinivel, formado por un bus de sistema y un bus de bajo nivel para la comunicación con los periféricos. Existen dos buses de sistema diferentes: el AHB y el ASB. En la Figura 3.1 se puede observar un sistema típico AMBA.

Los buses de sistema soportan 32, 64 y 128 bits de bus de datos con un espacio de direccionamiento de 32 bits. Estos buses son síncronos, no multiplexados y soportan

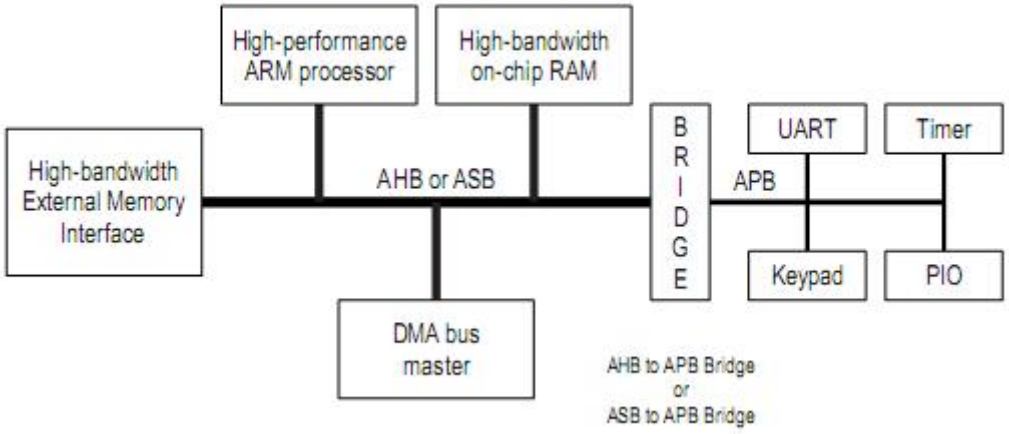

Figura 3.1: Sístema típico AMBA™

pipeline y bursting. El ASB se utiliza en sistemas simples. En cambio, para sistemas más sofisticados es necesario el uso del bus AHB. Actualmente el procesador ARM trabaja con el bus AHB.

El sistema de buses AMBA define una jerarquía de buses formada, como ya se ha comentado, por un bus de sistema y un bus de interconexión de periféricos. Estos dos buses se comunican mediante un bridge que funciona como un master para los componentes esclavos conectados al bus de los periféricos.

En una configuración típica el bus de sistema conecta el procesador o procesadores con el controlador de memoria, el controlador DMA y con cualquier otro componente de alto rendimiento. Los componentes periféricos lentos se conectan mediante el bus más sencillo de periféricos APB. El bus de sistema y el bus de periféricos pueden funcionar a diferentes frecuencias de reloj.

Actualmente existe una tercera generación de AMBA que incluye un nuevo bus: el AXI (*Advanced eXtensible Interface*).

Las características de los diferentes buses AMBA son las siguientes:

AHB (Advanced High Performance Bus)

Este bus esta diseñado para ser utilizado en la comunicación de dispositivos de alto rendimiento, tales como CPUs, DMAs y DSPs.

- Bus de altas prestaciones.
- Multi Master.
- Transferencias a trozos.
- Selección de master en un ciclo.
- 32 a 128 bits de bus de datos.
- Incorpora un mecanismo de protección de acceso.
- Espacio de direcciones limitado a 32 bits.

- Proporciona sistema de retención de datos para los dispositivos más lentos.
- Soporta arbitraje, REQ, GNT y LOCK.
- Soporta transferencias de bytes, words, etc...

ASB (Advanced System Bus)

Bus de propósito general. Se utiliza para la interconexión de microcontroladores y periféricos.

- Bus de sistema de primera generación.
- Multi Master.
- Transferencias Pipeline.
- 32 a 128 bits de bus de datos.
- Incorpora un mecanismo de protección de acceso.
- Bus de datos bidireccional.
- Espaccio de direcciones limitado a 32 bits.
- Proporciona sistema de retención de datos para los dispositivos más lentos.
- Soporta arbitraje, REQ, GNT y LOCK.
- Soporta transferencias de bytes, words, etc...

APB (Advance Peripheral Bus)

Bus de interconexión de periféricos. Este bus se centra en un mínimo consumo y en la facilidad de uso.

- Bus de periféricos de baja potencia y bajo rendimiento.
- Single Master.
- Únicamente 4 señales de control (mas clock y reset).
- Espacio de direcciones limitado a 32 bits.
- Hasta 32 bits de bus de datos.
- Bus de datos de lectura y escritura separados.

Tanto los protocolos de comunicación como las diferentes señales empleadas en las interfaces de los componentes pueden varíar dependiendo de la versión de bus (AHB, ASB, APB o AXI) empleada.

3.2. Bus Avalon

Avalon [2] es una arquitectura de bus simple diseñada para interconectar procesadores integrados y periféricos dentro de un SOPC (*Sistem-on-a-programmable chip*). Diseñado por Altera para utilizarlo junto a su procesador, soft-core, Nios II. Avalon

es un interfaz que especifica los pines de conexión entre los componentes maestros y esclavos, además de los tiempos requeridos para su comunicación.

La transacción básica de este bus es capaz de transferir de 8 a 32 bits entre un componente maestro y un periférico esclavo. Después de completarse una transferencia, el bus queda inmediatamente libre para que en el siguiente ciclo de reloj se pueda producir otra transferencia, bien entre los mismos componentes o bien entre otros distintos. El bus Avalon también soporta otros modos de transferencia más avanzados que logran varias transferencias, entre distintos componentes, simultáneas.

Avalon soporta una técnica de arbitraje que permite conectar varios componentes maestros con un mismo componente esclavo, de manera que el árbitro decide en cada momento que componente maestro realizará una transferencia con el periférico esclavo.

Las características principales del bus Avalon son las siguientes:

- Arquitectura multi master.
- Espacio de direcciones de 32 bits donde mapear los distintos componentes de memoria y periféricos.
- Todas las señales del bus están sincronizadas con el reloj.
- Bus de direcciones y de datos separados.
- El bus Avalon genera automáticamente las señales de Chip Select para todos los periféricos.

Todo sistema que utilice el bus Avalon debe incorporar un módulo específico, el cual contiene todas las señales de control, de datos y de direcciones, además de la lógica de arbitraje necesaria para conectar todos los componentes del sistema entre sí. Este modulo, denominado módulo de bus Avalon, implementa una arquitectura de bus configurable que puede variar para adaptarse a los requerimientos de interconexión solicitados por el diseñador. El diseñador del sistema no tiene que conectar manualmente todos los componentes, ya que es la propia herramienta SOPC Builder la que construye el módulo dependiendo de los componentes a conectar. La visión del diseñador del bus se limita a los puertos específicos de cada componente. Se puede observar un ejemplo de sistema con bus Avalon en la Figura 3.2.

Un periférico conectado al bus Avalon es un componente lógico que puede estar implementado dentro del mismo chip o fuera de él. Cada periférico puede tener una tarea diferente y pueden ser añadidos o eliminados del sistema (y por lo tanto su conexión al módulo del bus Avalon) en tiempo de diseño, dependiendo de los requerimientos.

Los periféricos conectados al bus Avalon pueden ser maestros o esclavos. Un periférico maestro puede iniciar una transferencia mediante el bus Avalon y al menos tiene

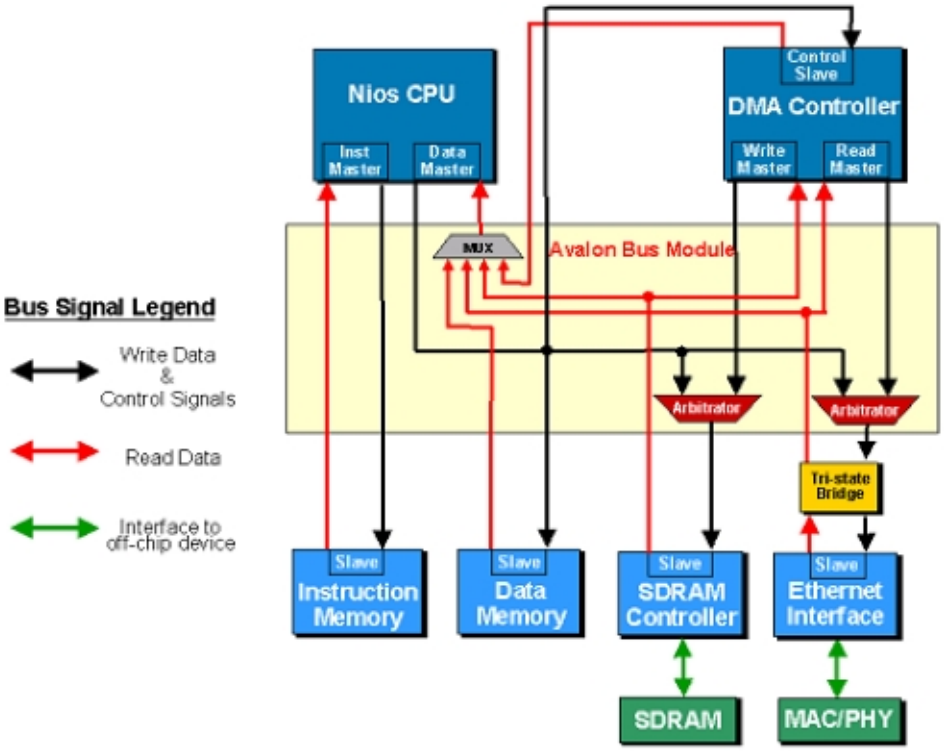

Figura 3.2: Sístema típico Avalon

un puerto maestro, que se conecta al módulo del bus Avalon. Un periférico maestro también tiene un puerto esclavo que le permite recibir transferencias mediante el bus, iniciadas por otros componentes maestros. En cambio, un periférico esclavo únicamente acepta transferencias del bus Avalon y no puede iniciar él mismo las transferencias. Estos últimos, que suelen ser memorias u otros periféricos, normalmente tienen un puerto esclavo que se conecta al módulo del bus Avalon.

Las especificaciones del bus Avalon definen las señales y los requerimientos temporales requeridos para la correcta transferencia de datos entre un puerto maestro y un puerto esclavo, vía el módulo del bus Avalon. Las señales que componen la interfaz entre dicho módulo y el periférico pueden ser diferentes, dependiendo del tipo de transferencia.

Avalon es un bus síncrono, dirigido por el reloj del bus. Todas las transferencias ocurren de forma síncrona con dicho reloj y se inician en el flanco de subida del reloj.

3.3. Bus Wishbone

Wishbone [3] es un bus hardware de código abierto utilizado para comunicar los diferentes componentes de un circuito integrado. Wishbone intenta ser un bus lógico, no

especificando información eléctrica o de topología. Por eso es útil para comunicar diferentes componentes diseñados en lenguajes de descripción de hardware, ya sea Verilog, VHDL u otros.

La arquitectura de bus Wishbone es bastante simple. Únicamente existe un solo bus. Un sistema con varios componentes puede incluir dos buses diferentes: uno para componentes de alto rendimiento y otro para componentes de más bajo rendimiento. En la Figura 3.3 se puede observar una conexión punto a punto.

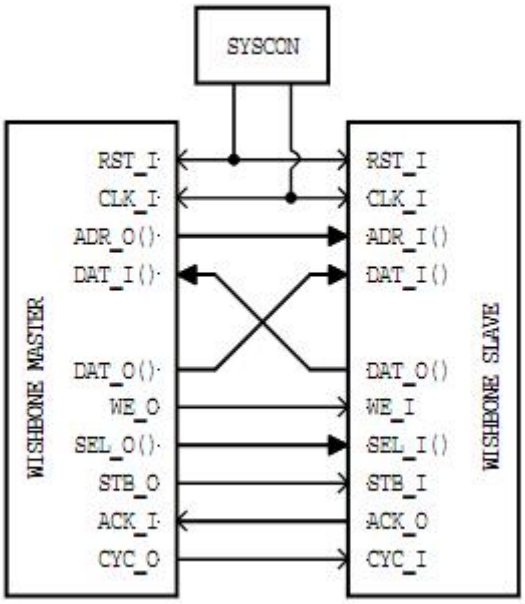

Figura 3.3: Interconexión Punto a Punto Wishbone

A continuación se muestran algunos detalles técnicos:

- Arquitectura de un bus para todas las aplicaciones.
- Arquitectura simple.
- Multi master.
- Espacio de direcciones de 64 bits.
- Bus de datos de 8 a 64 bits.
- Basado en protocolos estándares de transferencia de datos.
- Soporta varios tipos de interconexiones.
- Protocolo de handshake para regular la velocidad de transferencia de datos.
- Soporta varias terminaciones de ciclos.
- TAGs definidos por el usuario para identificar el tipo de transferencia de datos.
- Basado en arquitectura Maestro/Esclavo.

En la especificación Wishbone se detallan características, clasificadas en los siguientes términos:

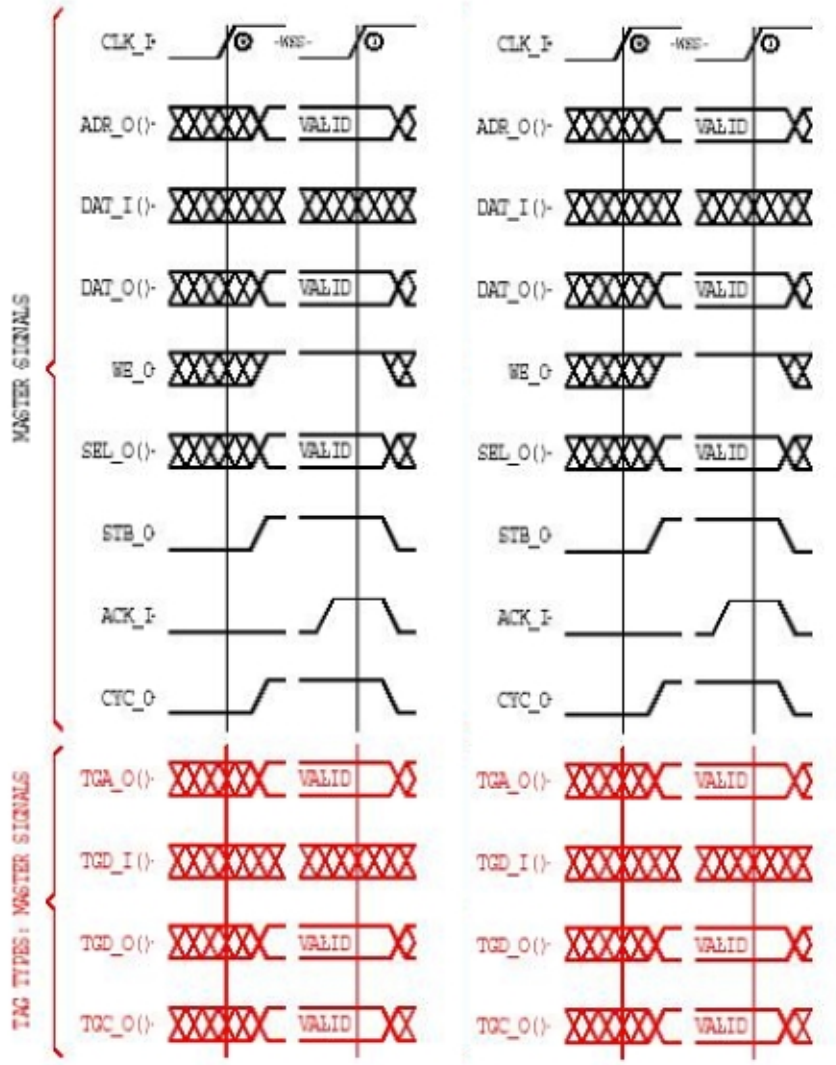

Figura 3.4: Ciclos Read/Write Simples

- Regla. Todas deben ser seguidas para asegurar una compatibilidad entre interfaces.
- Recomendaciones. Cuando aparece una recomendación, se aconseja seguirla. El no adoptarla puede implicar una pérdida de rendimiento.
- Sugerencia. Es un consejo que puede ser considerado por el diseñador. Puede ser útil, pero no vital para el funcionamiento de la interfaz.
- Permiso. Cuando se indica que algo puede hacerse o no, la decisión de implementarlo puede quedar a cargo del diseñador.
- Observación. Usualmente son textos que clarifican alguna situación, pero no aportan nada más que eso.

Las señales permiten tres ciclos básicos: Read, Write y RMW. Pero no hay obligación de soportarlos todos. Además permiten un handshake para adecuar la velocidad

de transferencia de datos, e indicar errores y reintentos.

Las señales no son bidireccionales, siempre son entradas o salidas. Esto es así debido a que muchas veces el diseño puede llegar a querer implementarse en hardware que no soporta internamente señales bidireccioneles, como por ejemplo las FPGAs de Altera. A todas las señales se les agrega la coletilla "_I" o "_O" al final de las mismas para indicar si son salidas o entradas al componente.

Las interfaces Wishbone deben de inicializarse con el primer flanco de subida del reloj en el que el reset esté activo. Una vez inicializadas ya se pueden realizar ciclos de lectura y escritura. En la Figura 3.4 se puede ver un diagrama de tiempos de los ciclos de lectura y escritura simples.

Wishbone es totalmente gratuito y es utilizado en muchos componentes diseñados dentro del proyecto OpenCores [9]. El usuario también tiene la posibilidad de crear su propio substandard de Wishbone, especificando el orden de los datos (little / big endian) y el significado de los TAGs.

Capítulo 4

Análisis de prestaciones. Tecnología de diseño. Planificación

Este capítulo está dividido en tres secciones. En la primera se desvela el procesador elegido para su descripción mediante el lenguaje de descripción de hardware VHDL. En la segunda sección se comenta la tecnología empleada para la realización de su prototipo y por último, en la tercera sección se explica brevemente la planificación del proyecto.

Análisis de prestaciones

Como se ha comentado con anterioridad, el procesador propuesto se utilizará en aplicaciones de adquisición y proceso simple de datos, por lo que es necesario que el procesador posea una arquitectura sencilla y un conjunto de instrucciones simple. Por lo tanto el procesador no necesita soporte para números en coma flotante ni soporte hardware para operaciones complejas. Además, tal como se comenta en la introducción de este texto, es preferible un procesador sin pipeline y con una arquitectura de buses bien definida para poder ser utilizado en propositos educativos (aunque es recomendable que el procesador posea alguna versión con pipeline para futuras modificaciones). Por todo esto el procesador elegido es el Simple RISC Computer en su variante de 3 buses.

El SRC, en su variante de 3 buses, consigue unas buenas prestaciones con una arquitectura simple de 3 buses obteniendo un buen compromiso entre rendimiento y sencillez. Además dispone de una Unidad Aritmético Lógica sencilla, sin operaciones complejas. De todas maneras en este procesador se aprecian algunas limitaciones a intentar superar, tales como la capacidad de trabajar con un único dispositivo interruptor, o un CPI algo elevado (alrededor de 5 ciclos por instrucción).

La estructura de buses de la unidad de proceso tiene un importante impacto en el rendimiento del procesador. Por un lado, un incremento del número de buses significa

reducir los ciclos por instrucción (CPI) necesarios para ejecutar cada instrucción [22]. Por otro lado aumentamos los retardos provocados por la propagación de las distintas señales por los buses, lo que significa aumentar el período mínimo de reloj y disminuir con ello la frecuencia del procesador.

Con una arquitectura de tres buses podemos obtener una mejora de más de un 80 % respecto de una arquitectura de un solo bus. Aun teniendo en cuenta los retardos introducidos por los buses en el período mínimo de reloj, la mejora sigue siendo importante, estando alrededor del 50 %. La mejora provocada en el CPI por la introducción de un cuarto bus se vería contrarrestada por el aumento del período mínimo de reloj. A partir de aquí la incorporación de más buses solo provocaría un aumento del periodo mínimo de reloj bajando el rendimiento del procesador en picado.

El período de reloj mínimo para una transferencia de registro dada es el tiempo requerido para propagar datos enteramente alrededor del circuito, de modo que el nuevo valor esté en el registro de destinación y listo para otro ciclo. El período mínimo de reloj para esta transferencia de registro en particular se calcula de la siguiente forma:

$$t_{min} = t_g + t_{bp} + t_{comb} + t_l$$

t_{min}: Período mínimo de reloj.

t_g: Tiempo de propagación de puerta lógica.

t_{bp}: Tiempo de propagación de bus.

t_{comb}: Retardo lógico (ALU).

t_l: Tiempo de propagación de Flip-Flop.

Como se puede observar el término t_{comb} representa los retardos lógicos introducidos por la ALU, por lo que simplificar la ALU puede provocar la disminución del período mínimo de reloj o lo que es lo mismo: aumentar la frecuencia de reloj del procesador. La Unidad Aritmético Lógica es, por lo tanto, otro punto clave en el diseño del procesador.

Todos los procesadores estudiados en el capítulo 2 a excepción del SRC incorporan un barrel shifter en la Unidad Aritmético Lógica. Este componente utilizado para realizar desplazamientos de varias posiciones en el mismo ciclo de reloj supone un incremento sustancial de la lógica requerida para la síntesis del procesador, además de incrementar notablemente el período mínimo de reloj.

Un shifter de un único bit por ciclo de reloj provoca, en cambio, que el número de instrucciones del programa a ejecutar crezca y por lo tanto que se tengan que realizar más accesos a memoria [26], bajando el rendimiento del procesador.

Teniendo en cuenta lo anterior y que los desplazamientos de más de dos posiciones no son muy comunes en los programas, existe la posibilidad de utilizar una técnica que estaría a medio camino entre las dos anteriores. Se trata de equipar a la ALU

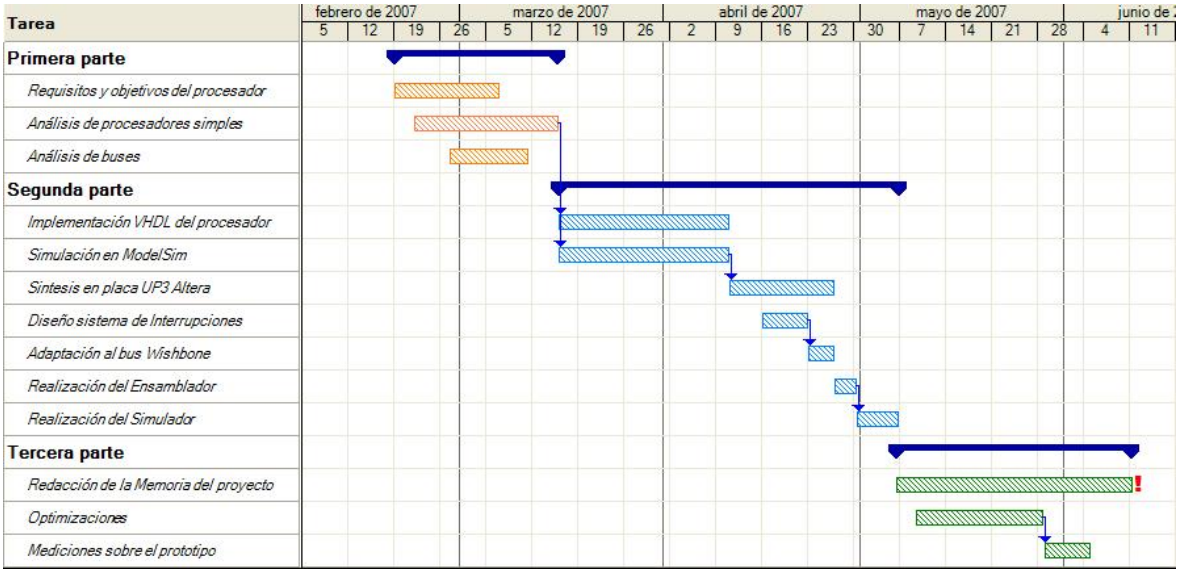

Figura 4.1: Diagrama de Gantt que muestra la planificación del proyecto

de únicamente desplazamientos de un bit, pero permitir que el procesador decodifique instrucciones de desplazamiento de más bits. Para conseguir este propósito es necesario utilizar un contador para llevar el control del número de desplazamientos de un bit necesarios para completar el desplazamiento de n bits indicado por la instrucción. Con esta técnica se simplifica enormemente la ALU y no se realizan más accesos a memoria. Por el contrario se incrementa el CPI, aunque de forma lineal y únicamente en el caso de desplazamientos de más de un bit. Esta técnica es la empleada en el procesador SRC.

En relación a los tres tipos de buses analizados en el Capítulo 3, el que más se adapta a la filosofía de simplicidad del procesador elegido es el Wishbone. Además de ser el más simple de los tres buses es muy flexible y el único totalmente libre de derechos.

Tecnología de diseño

En el Capítulo 1 se comenta que el procesador se ha de simular y sintetizar mediante el lenguaje de descripción de hardware VHDL en una FPGA.

Según estudios recientes [24], los circuitos implementados en FPGAs, respecto a los implementados en Standard Cells (ASICs), ocupan aproximadamente 35 veces más, son entre 3,4 y 4,6 veces más lentos de media y consumen 14 veces más. El uso de memorias y multiplicadores dedicados por parte de las FPGAs más modernas producen una considerable reducción del área requerida y del consumo del circuito, en cambio estos bloques no tienen efectos importantes en cuanto a velocidad.

Aun teniendo en cuenta estos valores, se ha decidido apostar por una implementación del procesador en una FPGA debido al menor tiempo de diseño necesario y

sobretodo por los excesivos costes asociados a un diseño en Standard Cells y su posterior fabricación. El diseño del procesador en Standard Cells únicamente sería rentable para tiradas largas debido a los altos costes NRE (*Non-Recurring engineering*). Además los fabricantes de FPGAs ofrecen unas herramientas de diseño con un interfaz sencillo y amigable, lo que permite acortar notablemente la curva de aprendizaje. Por esto una implementación en Standard Cells se hace inviable, sobre todo para la realización del prototipo del procesador.

Planificación

Hasta ahora se han introducido los objetivos del proyecto, se han analizado distintas alternativas de procesadores y de arquitecturas de buses, y se ha realizado un análisis de sus prestaciones. Todo esto lo podríamos englobar en la primera parte de las tres en las que se puede dividir el proyecto. En la Figura 4.1 se puede ver un diagrama de Gantt representando la planificación seguida en el proyecto. La planificación inicial, que no era tan precisa, únicamente difería de ésta en que la primera etapa del proyecto estaba planificada para realizarla durante los meses de octubre y diciembre.

En el diagrama se pueden observar las tres grandes partes en las que se divide el proyecto y las tareas que forman parte de ellas. La segunda parte del proyecto es la más costosa debido principalmente a la implementación, simulación y prototipado del procesador, sin menospreciar el tiempo dedicado a la realización de las herramientas software (ensamblador y sobretodo, simulador).

Capítulo 5

El procesador SR3C

En este capítulo se describe a fondo el procesador SR3C (Simple RISC 3-buses Computer) implementado en este proyecto. Este procesador, tal como se puede observar por su nombre, es una evolución del procesador SRC introducido en el Capítulo 2.

En el primer apartado de este capítulo se introduce el procesador SR3C clasificándolo según su conjunto de registros y su conjunto de instrucciones. Después se describe su arquitectura, el sistema de interrupciones diseñado, algunas modificaciones realizadas sobre él y por último se expone el ensamblador y simulador realizados.

5.1. Introducción

El procesador SR3C, aún basandose en el procesador SRC, incorpora bastantes modificaciones importantes. Básicamente las modificaciones introducidas son las siguientes:

- Modificaciones en la arquitectura para disminuir los ciclos por instrucción y aumentar el rendimiento del procesador.
- Especificación de un controlador de interrupciones.
- Introducción de instrucciones adicionales en el repertorio del procesador.
- Adaptación a la interfaz estándar de buses Wishbone.
- Versión con multiplexores para la eliminación de estados en alta impedancia, evitando así consumos innecesarios.

Antes de explicar la arquitectura del procesador, es interesante situar el procesador, según sus registros y el tipo de instrucciones que posee, dentro de las posibilidades

existentes.

Clasificación del SR3C según el tipo de registros incorporados

Respecto a los registros de un procesador, éstos se pueden clasificar dentro de una de las siguientes categorías:

- Registros acumuladores. El procesador necesita una dirección de memoria en cada instrucción aritmética para poder operar. La ventaja es que disminuye el tamaño de memoria requerida, pero por contra los procesadores que únicamente disponen de este tipo de registros son muy limitados para el cálculo de expresiones con muchos términos y factores.
- Pila. No se hace necesaria ninguna dirección de memoria en las instrucciones aritméticas. Siempre se opera con los elementos más arriba de la pila. El número de instrucciones requeridas para realizar una instrucción aritmética crece notablemente.
- Registros de propósito general. El coste viene dado por identificar dichos registros en la instrucción (numero de bits según número de registros). Este problema se disuelve con procesadores de 32 bits.

El procesador SR3C únicamente dispone de un conjunto de 32 registros de propósito general de 32 bits cada uno.

Clasificación del SR3C según el tipo de instrucciones

Respecto al tipo de instrucciones que posee el procesador, hay que considerar el número de direcciones de memoria que incluye en las instrucciones.

En una instrucción aritmética de 2 operandos se necesitan los siguientes datos:

- Operación a realizar.
- Localización del primer operando.
- Localización del segundo operando.
- Lugar para almacenar el resultado.
- Localización de la siguiente instrucción a ejecutar.

Los procesadores se pueden clasificar de la siguiente forma según el número de instrucciones de memoria que utilicen para indicar los datos anteriores:

- Procesadores de 4 direcciones. Las instrucciones son demasiado largas y se deben realizar demasiados accesos a memoria. Únicamente en el caso de que los accesos

a memoria fuesen más rápidos que los tiempos de ALU, tendrían que tenerse en cuenta como alternativa de diseño.

- Procesadores de 3 direcciones. Se incluye un registro contador de programa (PC) para indicar la siguiente instrucción. Las instrucciones ocupan menos y el tiempo de acceso disminuye. La unidad de control es la encargada de gobernar el PC.

- Procesadores de 2 direcciones. El resultado de la instrucción se guarda en la misma dirección que uno de los operandos. Se reduce el tamaño de la instrucción pero no el número de accesos a memoria. Muy similares a los procesadores de 3 direcciones.

- Procesadores de 1 dirección. Se utiliza un registro acumulador, que al ser único no hace falta mencionar en la dirección. Se disminuye el tamaño de la instrucción pero se necesitan más instrucciones para cargar y guardar los valores del acumulador.

- Procesadores de 0 direcciones. Los operandos se encuentran en las dos últimas posiciones de una pila. Es necesario instrucciones de push y de pop que si que requieren un campo de dirección. No se han implementado muchas máquinas de este tipo debido a que las maquinas con registros de propósito general son mas dadas a técnicas de speedup como por ejemplo el pipeline.

- Procesadores de 1-$\frac{1}{2}$ direcciones. Este tipo de procesadores se basan en que se requiere menos espacio en la instrucción para direccionar registros que direcciones de memoria. Estos procesadores suelen tener una dirección de memoria y un registro en las instrucciones (de ahí el nombre 1-$\frac{1}{2}$). De esta forma se reduce el tamaño de las instrucciones.

El procesador SR3C, al igual que la mayoría de procesadores RISC es de tipo carga/almacenamiento, siendo las instrucciones de carga y almacenamiento las únicas que utilizan una dirección de memoria. Dichas instrucciones se pueden clasificar en el tipo de instrucciones de 1-$\frac{1}{2}$ direcciones. En cambio el resto de instrucciones no utilizan ninguna dirección de memoria ya que los operandos se indican mediante registros.

5.2. Arquitectura

El procesador SR3C, al igual que la mayoría de procesadores simples se compone de una UP (Unidad de proceso) y de una UC (Unidad de control). La UP contiene los registros y unidades de proceso mientras que la UC es la encargada de interpretar las

instrucciones y generar las señales de control necesarias para ejecutarlas.

Unidad de Proceso

La Unidad de Proceso (UP) del SR3C está compuesta por los siguientes registros y unidades funcionales:

- PC (Contador de Programa). Registro de 32 bits donde se almacena la siguiente instrucción a ejecutar. Incluye una señal (*pc_new*) que la activa la UC para incrementar en 1 el PC [1].
- IR (Registro de Instrucción). Registro de 32 bits donde se almacena la actual instrucción a ejecutar. Este registro decodifica la instrucción enviando el código de operación de la instrucción en curso a la Unidad de Control del procesador. Además selecciona los registros activos en dicha instrucción y la condición de salto (para instrucciones de salto). Puede volcar en el Bus A la instrucción entera, el valor inmediato c1, o el valor inmediato c2 (Ver apartado Instrucciones del SR3C).
- MA (Registro Dirección de Memoria). Registro de 32 bits donde se almacena la dirección de memoria a acceder. Está conectado al bus de dirección de memoria.
- MD (Registro Dato de Memoria). Registro de 32 bits donde se almacena el dato a escribir en memoria o donde se recibe el dato leído de memoria.
- REGS (Registros de propósito general). Banco de 32 registros de propósito general de 32 bits cada uno. El registro r0 siempre contiene el valor 0, por lo que cualquier instrucción que desee modificar dicho registro será descartada.
- ALU (Unidad Aritmético Lógica). Puede realizar las siguientes operaciones identificadas por la señal *alu_op* de 4 bits:
  - CeqA_op: Transparencia Bus A.
  - NAND_op: Operación lógica nand.
  - NOR_op: Operación lógica nor.
  - ADD_op: Operación aritmética de suma.
  - SUB_op: Operación aritmética de resta.
  - NEG_op: Operación aritmética de negación.
  - NOT_op: Operación lógica not.

[1] El procesador se ha utilizado junto a una memoria de 32 bits de ancho de palabra. En caso de utilizar una memoria de 8 o 16 bits, el PC se tendrá que incrementar de 4 en 4 o de 2 en 2 respectivamente.

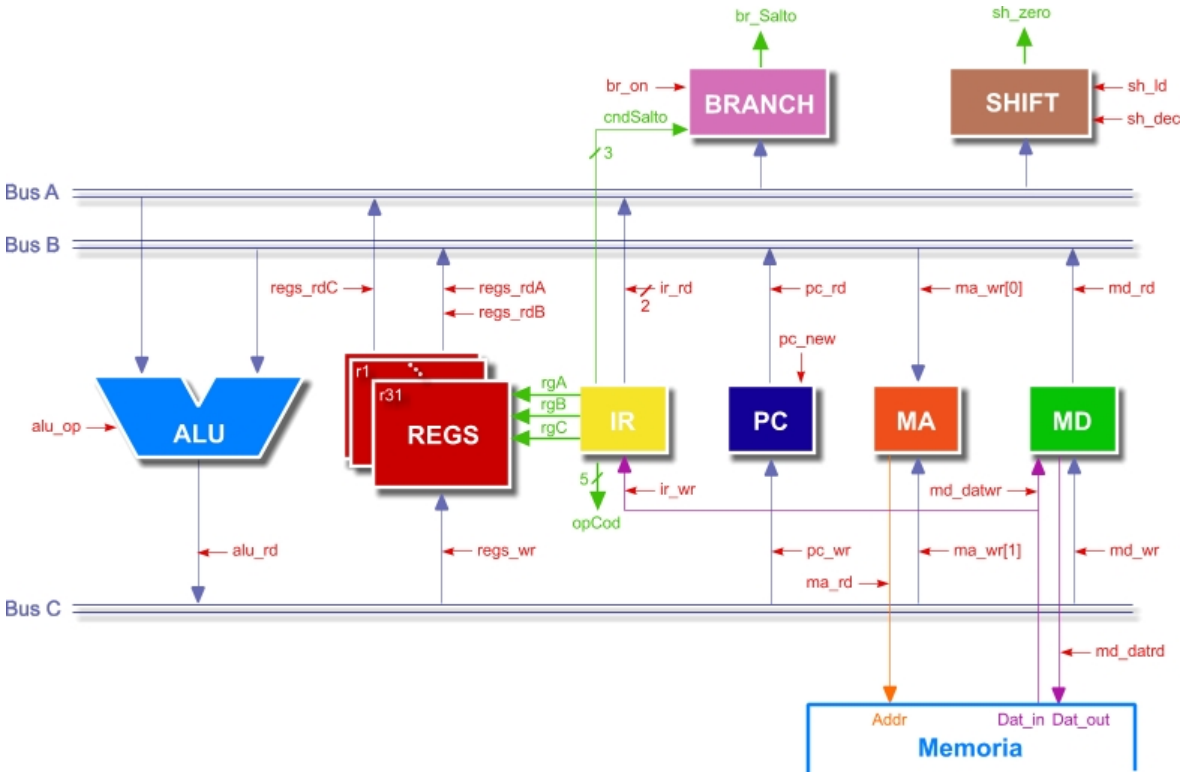

Figura 5.1: Arquitectura de la Unidad de Proceso del Procesador SR3C

- SHR_op: Operación lógica de desplazamiento de 1 bit a la derecha.
- SHL_op: Operación lógica de desplazamiento de 1 bit a la izquierda.
- SHRA_op: Operación aritmética de desplazamiento de 1 bit a la derecha.
- SHC_op: Operación lógica de rotación de 1 bit a la izquierda.
- AND_op: Operación lógica and.
- OR_op: Operación lógica or.
- XOR_op: Operación lógica xor.
- NXOR_op: Operación lógica nxor.
- CeqB_op: Transparencia Bus B.

- SHIFT (Contador de desplazamientos). Unidad que incorpora un contador de 5 bits para implementar instrucciones de desplazamiento y rotación de más de un bit.

- BRANCH (Unidad de Salto). Unidad que indica si hay que efectuar un salto o no dependiendo de la condición de salto de la instrucción.

En la Figura 5.1 se puede ver la arquitectura de la UP. Como se puede observar, la arquitectura tiene 3 buses (Bus A, B y C). Además de los componentes que la forman (registros y unidades funcionales) también se pueden ver las señales de control

que actúan sobre dichos componentes (en rojo) procedentes de la Unidad de control. Las señales que salen de los componentes de la UP (en verde) van, o bien a otros componentes (indicadores de registros y condición de salto), o bien a la Unidad de control. A continuación se puede observar la interfaz de la UP en lenguaje VHDL:

```
entity up is
  port( nR:         in std_logic;                        -- Reset
        ck:         in std_logic;                        -- Reloj
        alu_op:     in opALU;                            -- Operación ALU
        alu_rd:     in std_logic;                        -- ALU -> Bus C
        regs_rdA:   in std_logic;                        -- R[ra] -> Bus B
        regs_rdB:   in std_logic;                        -- R[rb] -> Bus B
        regs_rdC:   in std_logic;                        -- R[rc] -> Bus A
        regs_wr:    in std_logic;                        -- Bus C -> R[ra]
        ir_rd:      in std_logic_vector (1 downto 0);    -- IR -> Bus A
        ir_wr:      in std_logic;                        -- Bus C -> IR
        pc_rd:      in std_logic;                        -- PC -> Bus B
        pc_new:     in std_logic;                        -- Incremento PC
        pc_wr:      in std_logic;                        -- Bus C -> PC
        ma_rd:      in std_logic;                        -- MA -> Addr
        ma_wr:      in std_logic_vector (1 downto 0);    -- Bus A o C -> MA
        md_rd:      in std_logic;                        -- MD -> Bus B
        md_wr:      in std_logic;                        -- Bus C -> MD
        md_datrd:   in std_logic;                        -- MD -> Dat_out
        md_datwr:   in std_logic;                        -- Dat_in -> MD
        br_on:      in std_logic;                        -- BRANCH on
        sh_ld:      in std_logic;                        -- IR -> SHIFT
        sh_dec:     in std_logic;                        -- SHIFT dec
        opCod:      out instruccion;                     -- Código op
        br_Salto:   out std_logic;                       -- Salto
        sh_zero:    out std_logic;                       -- SHIFT a 0
        Addr:       out std_logic_vector (31 downto 0); -- Addr
        Dat_out:    out std_logic_vector (31 downto 0); -- Dat_out
        Dat_in:     in std_logic_vector (31 downto 0)); -- Dat_in
end;
```

Unidad de Control

Existen dos aproximaciones a la hora de diseñar una Unidad de Control. Por un lado existen las UC cableadas. Estas UC trabajan con señales de control cableadas

directamente entre la propia UC y la UP. En cambio las unidades microprogramas disponen de un circuito lógico sencillo que permite realizar la ejecución en secuencia de un conjunto de instrucciones muy elementales llamadas microinstrucciones. Cada microinstrucción contiene las señales de control que se envían a la UP, junto a una serie de bits que permitan seguir el secuenciamiento. A continuación se describen algunas características de estas técnicas:

- Velocidad: Las UC cableadas tienen una latencia provocada por los retardos de unas pocas puertas lógicas. En cambio las microprogramadas necesitan realizar una búsqueda en memoria para cada microinstrucción.
- Facilidad de prototipado: Las UC microprogramadas son sencillas de prototipar debido a que únicamente hay que reprogramar una memoria. En cambio en las UC cableadas hay que rediseñar la lógica.
- Flexibilidad de uso: Las UC microprogramadas son bastante flexibles a la hora de añadir o eliminar instrucciones del repertorio del procesador. En las UC cableadas hay que añadir lógica de control y conexiones.

El procesador SR3C posee una UC cableada para evitar latencias provocadas por accesos a memoria y mejorar así su rendimiento. El diseño no se ha complicado por este hecho, debido a que el procesador tiene una arquitectura bien definida. Además al estar implementado en VHDL, la UC se describe como una máquina de estados finitos. A continuación se puede observar su interfaz en lenguaje VHDL:

```
entity uc is
  port( nR:       in std_logic;                          -- Reset
        ck:       in std_logic;                          -- Reloj
        opCod:    in instruccion;                        -- Operación ALU
        br_Salto: in std_logic;                          -- Salto
        sh_zero:  in std_logic;                          -- SHIFT a 0
        alu_op:   out opALU;                             -- Operación ALU
        alu_rd:   out std_logic;                         -- ALU -> Bus C
        regs_rdA: out std_logic;                         -- R[ra] -> Bus B
        regs_rdB: out std_logic;                         -- R[rb] -> Bus B
        regs_rdC: out std_logic;                         -- R[rc] -> Bus A
        regs_wr:  out std_logic;                         -- Bus C -> R[ra]
        ir_rd:    out std_logic_vector (1 downto 0);     -- IR -> Bus A
        ir_wr:    out std_logic;                         -- Bus C -> IR
        pc_rd:    out std_logic;                         -- PC -> Bus B
        pc_new:   out std_logic;                         -- Incremento PC
```

```
        pc_wr:    out std_logic;                          -- Bus C -> PC
        ma_rd:    out std_logic;                          -- MA -> Addr
        ma_wr:    out std_logic_vector (1 downto 0);      -- Bus A o C -> MA
        md_rd:    out std_logic;                          -- MD -> Bus B
        md_wr:    out std_logic;                          -- Bus C -> MD
        md_datrd: out std_logic;                          -- MD -> Dat_out
        md_datwr: out std_logic;                          -- Dat_in -> MD
        br_on:    out std_logic;                          -- BRANCH on
        sh_ld:    out std_logic;                          -- IR -> SHIFT
        sh_dec:   out std_logic;                          -- SHIFT dec
        nCE:      out std_logic;                          -- Chip Enable
        nOE:      out std_logic;                          -- Output Enable
        nWE:      out std_logic);                         -- Write Enable
end;
```

Instrucciones del SR3C

El procesador SR3C dispone de 32 códigos de instrucción diferentes (5 de ellas para el tratamiento de las interrupciones), aprovechando así al máximo los 5 bits dedicados a indicar el código de operación dentro de cada instrucción. Todas las instrucciones tienen alguno de los ocho formatos diferentes que se muestran en el Cuadro 2.5 del capítulo 2. El conjunto de instrucciones del SR3C está formado por las siguientes instrucciones, divididas en siete categorías distintas dependiendo de su funcionalidad:

- Instrucciones de Carga

```
ld  ra, c2      ;Carga directa: R[ra]=M[c2]
ld  ra, c2(rb)  ;Carga indexada(rb!=0): R[ra]=M[c2+R[rb]]
la  ra, c2      ;Carga dir. desplazamiento directo: R[ra]=c2
la  ra, c2(rb)  ;Carga dir. desplazamiento indexado: R[ra]=c2+R[rb]
ldr ra, c1      ;Carga relativa: R[ra]=M[PC+c1]
lar ra, c1      ;Carga dirección relativa: R[ra]=PC+c1
```

- Instrucciones de Almacenamiento

```
st  ra, c2      ;Almacenamiento directo: M[c2]=R[ra]
st  ra, c2(rb)  ;Almacenamiento indexado(rb!=0): M[c2+R[rb]]=R[ra]
str ra, c1      ;Almacenamiento relativo: M[PC+c1]=R[ra]
```

- Instrucciones Aritméticas

```
add  ra, rb, rc ;Suma en Complemento a 2: R[ra]=R[rb]+R[rc]
addi ra, rb, c2 ;Suma inmediata en C2: R[ra]=R[rb]+c2
sub  ra, rb, rc ;Resta en Complemento a 2: R[ra]=R[rb]-R[rc]
neg  ra, rc     ;Negación: R[ra]=-R[rc]
and  ra, rb, rc ;AND lógica: R[ra]=R[rb] and R[rc]
andi ra, rb, c2 ;AND lógica inmediata: R[ra]=R[rb] and c2
or   ra, rb, rc ;OR lógica: R[ra]=R[rb] or R[rc]
ori  ra, rb, c2 ;OR lógica inmediata: R[ra]=R[rb] or c2
not  ra, rc     ;NOT: R[ra]=not(R[rc])
xor  ra, rb, rc ;XOR lógica: R[ra]=R[rb] xor R[rc]
nxor ra, rb, rc ;NXOR lógica: R[ra]=R[rb] nxor R[rc]
nand ra, rb, rc ;NAND lógica: R[ra]=R[rb] nand R[rc]
nor  ra, rb, rc ;NOR lógica: R[ra]=R[rb] nor R[rc]
```

- Instrucciones de Salto

```
br  rb, rc      ;Salto a R[rb] si R[rc] cumple condición
brl ra, rb, rc  ;Salto con link a R[rb] si R[rc] cumple cond. R[ra]=PC

Condiciones:
  brnv, brlnv   ;Nunca
  br, brl       ;Siempre
  brzr, brlzr   ;Si R[rc]=0
  brnz, brlnz   ;Si R[rc]!=0
  brpl, brlpl   ;Si R[rc]<31>=0 (R[rc]>=0)
  brmi, brlmi   ;Si R[rc]<31>=1 (R[rc] negativo)
```

- Instrucciones de Desplazamiento

```
shr  ra, rb, rc    ;Desplaza R[rb] a derecha en R[ra], R[rc] bits
shr  ra, rb, count ;Desplaza R[rb] a derecha en R[ra], count bits
shra ra, rb, rc    ;Desplaza Arit. R[rb] a der. en R[ra], R[rc] bits
shra ra, rb, count ;Desplaza Arit. R[rb] a der. en R[ra], count bits
shl  ra, rb, rc    ;Desplaza R[rb] a izquierda en R[ra], R[rc] bits
shl  ra, rb, count ;Desplaza R[rb] a izquierda en R[ra], count bits
shc  ra, rb, rc    ;Desplaza R[rb] circularmente en R[ra], R[rc] bits
```

```
shc  ra, rb, count ;Desplaza R[rb] circularmente en R[ra], count bits
```

- Otras Instrucciones

```
nop      ;No operación
stop     ;Parar ejecución
```

- Instrucciones de Interrupción. Explicadas en la siguiente sección (5.3. Sistema de Interrupciones)

En el Apéndice A se muestran las microinstrucciones que forman cada instrucción, además de las señales de control activadas por la UC para llevar a cabo dichas microinstrucciones.

Como se puede comprobar en la breve descripción de las diferentes instrucciones, el procesador dispone de varios tipos de direccionamiento. En concreto el procesador SR3C dispone de cinco tipos de direccionamiento diferentes:

- Inmediato
- Directo
- Relativo
- Desplazado
- Registro directo

Variaciones en el SR3C

Como se puede observar en el Cuadro A.1 ubicado en el Apéndice A, la búsqueda de instrucciones requiere de dos ciclos de reloj para completarse. En estos dos ciclos se han de realizar dos tareas: almacenar en el registro IR el contenido de la dirección de memoria existente en el registro PC e incrementar el PC en 1. Los pasos serían los siguientes:

```
MA<-PC
IR<-M[MA] ; PC<-PC+1
```

Se ha realizado una variación del procesador SR3C en el que la búsqueda de instrucciones se realiza en un solo ciclo de reloj. Para ello es necesario incluir un multiplexor y una nueva señal de control en la Unidad de Proceso. En la Figura 5.2 se pueden observar los cambios introducidos en la UP. El único paso a realizar sería el siguiente:

```
IR<-M[PC]
```

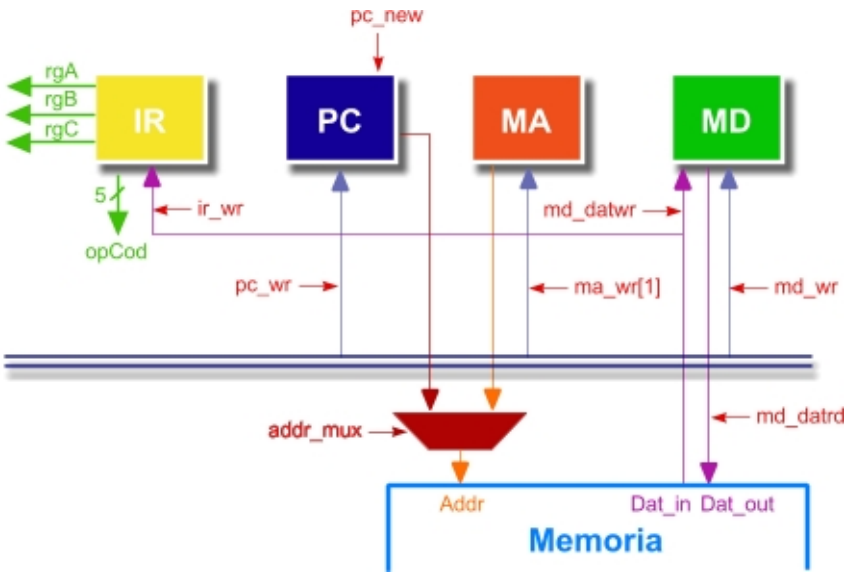

Figura 5.2: UP modificada para la búsqueda de instrucciones en un ciclo de reloj

El incremento del registro PC se llevaría a cabo en el siguiente ciclo, durante la ejecución de la propia instrucción.

Esta última versión del procesador SR3C de un único ciclo de búsqueda de instrucción se puede utilizar con memorias asíncronas, las cuales posicionan el dato en el bus de datos en cuanto se le indica la dirección de memoria buscada mediante el bus de direcciones.

También se ha desarrollado una versión del procesador SR3C con un bus de datos bidireccional, debido a que algunas memorias incorporan un bus de datos bidireccional en vez de un bus de datos de entrada y otro de salida.

5.3. Sistema de interrupciones

Debido a que se prevee el uso del procesador en aplicaciones de adquisición de datos, se hace necesario, que éste posea un sistema de interrupciones adecuado. El sistema de interrupciones permite mejorar el procesado en este tipo de aplicaciones frente al uso común de técnicas de *polling*.

En un sistema de interrupciones el procesador ha de poder identificar el dispositivo que ha realizado la interrupción. Existen dos posibilidades para llevar a cabo esa identificación:

- Vector de Interrupción. El dispositivo interruptor indica al procesador la dirección de memoria donde se localiza la rutina de interrupción adecuada.
- Registro de Información. Existe una única rutina de interrupción estándar para

todos los dispositivos interruptores. Con la ayuda de un registro de información rellenado por el dispositivo interruptor en el momento de la interrupción, se ejecuta la parte de la rutina adecuada a dicho dispositivo.

El procesador SR3C implementa una solución mixta a las dos aproximaciones anteriores. De esta forma se añaden al procesador nuevos registros y señales de control:

- ireq (*Interrupt Request*). Señal que indica al procesador que un dispositivo ha realizado una interrupción.
- iack (*Interrupt Acknoledge*). Señal que indica al dispositivo interruptor que el procesador va a tratar dicha interrupción.
- eoi (*End Of Interrupt*). Señal que indica que el procesador ha acabado de procesar la interrupción.
- IE (*Interrupt Enable*). Registro de 1 bit que indica si las interrupciones estan habilitadas o no.
- IPC (*Interrupted Program Counter*). Registro de 32 bits donde se almacena el contenido del registro PC al tratar una interrupción para poder seguir ejecutando el programa cuando el tratamiento de la interrupción finalice.
- II (*Interrupt Information*). Registro de 32 bits donde el dispositivo interruptor puede escribir información sobre la interrupción.

Además también se añaden cinco nuevas instrucciones relacionadas con el sistema de interrupciones. Las instrucciones para el tratamiento de las interrupciones son las siguientes (en el Cuadro A.8 del Apéndice A se pueden observar las instrucciones con sus microinstrucciones y señales de control):

```
svi ra, rb    ;Guarda los reg. II e IPC en Ra y Rb respectivamente.
ri  ra, rb    ;Vuelca los reg. ra y rb en II e IPC respectivamente.
een           ;Habilita las interrupciones. IE=1
edi           ;Deshabilita las interrupciones. IE=0
rfi           ;Fin de rutina de interrupción. PC=IPC e IE=1
```

Por último también se ha tenido que modificar la Unidad de Control del procesador, añadiendo el control para las instrucciones anteriores. Además al iniciar el primer ciclo de ejecución de cada instrucción se ha de mirar si se ha recibido alguna interrupción para poder tratarla.

La interfaz en lenguaje VHDL del procesador SR3C con sistema de interrupciones es la siguiente:

```
entity src is
  port( nR:           in std_logic;                          -- Reset
        ck:           in std_logic;                          -- Reloj
        ma_addbus:    out word;                              -- Addr
        md_indatBus:  in word;                               -- Dat_in
        md_outdatBus: out word;                              -- Dat_out
        nCE:          out std_logic;                         -- Chip Enable
        nOE:          out std_logic;                         -- Output Enable
        nWE:          out std_logic;                         -- Write Enable
        Isrc_vect:    in std_logic_vector (7 downto 0);      -- Vector Int.
        Isrc_info:    in std_logic_vector (15 downto 0);     -- Inform. Int.
        ireq:         in std_logic;                          -- Int. Request
        iack:         out std_logic;                         -- Int. Ack
        eoi:          out std_logic);                        -- End Of Int.
end;
```

Las señales Isrc_vect e Isrc_info corresponden a los vectores de rutina de interrupción y de información de interrupción respectivamente. Estos vectores proporcionados por el dispositivo interruptor sirven para ejecutar la rutina de interrupción correcta. El vector Isrc_info se introduce en el registro II y el vector Isrc_vect se introduce en el registro PC desplazado 4 posiciones a la izquierda, es decir: PC=x00000 Isrc_vect x0.

El funcionamiento del proceso de interrupción sería el siguiente:

- El procesador recibe la señal ireq de un dispositivo interruptor.
- En el ciclo de búsqueda de instrucción se detecta la interrupción y se genera la señal iack.
- Se deshabilitan las interrupciones. IE=0.
- Se guarda el registro PC en el registro IPC.
- Se recibe en el registro II el vector de información del dispositivo interruptor.
- Se actualiza el registro PC con la dirección de la rutina de interrupción indicada por el dispositivo interruptor mediante el vector de rutina de interrupción.
- se habilitan las interrupciones. IE=1.
- Se comienza a ejecutar la rutina de interrupción.
- Al acabar la rutina de interrupción se restaura el registro PC almacenado en IPC, mediante la instrucción rfi.
- Se activa la señal eoi.

Con todas las modificaciones hasta ahora indicadas, el procesador SR3C dispone de un sistema de interrupciones sencillo, pero eficaz. Únicamente existe un problema: solo hay una señal de *ireq* y otra de *iack* por lo que solo se puede conectar al procesador un dispositivo interruptor. Por esta razón también se ha diseñado un controlador de

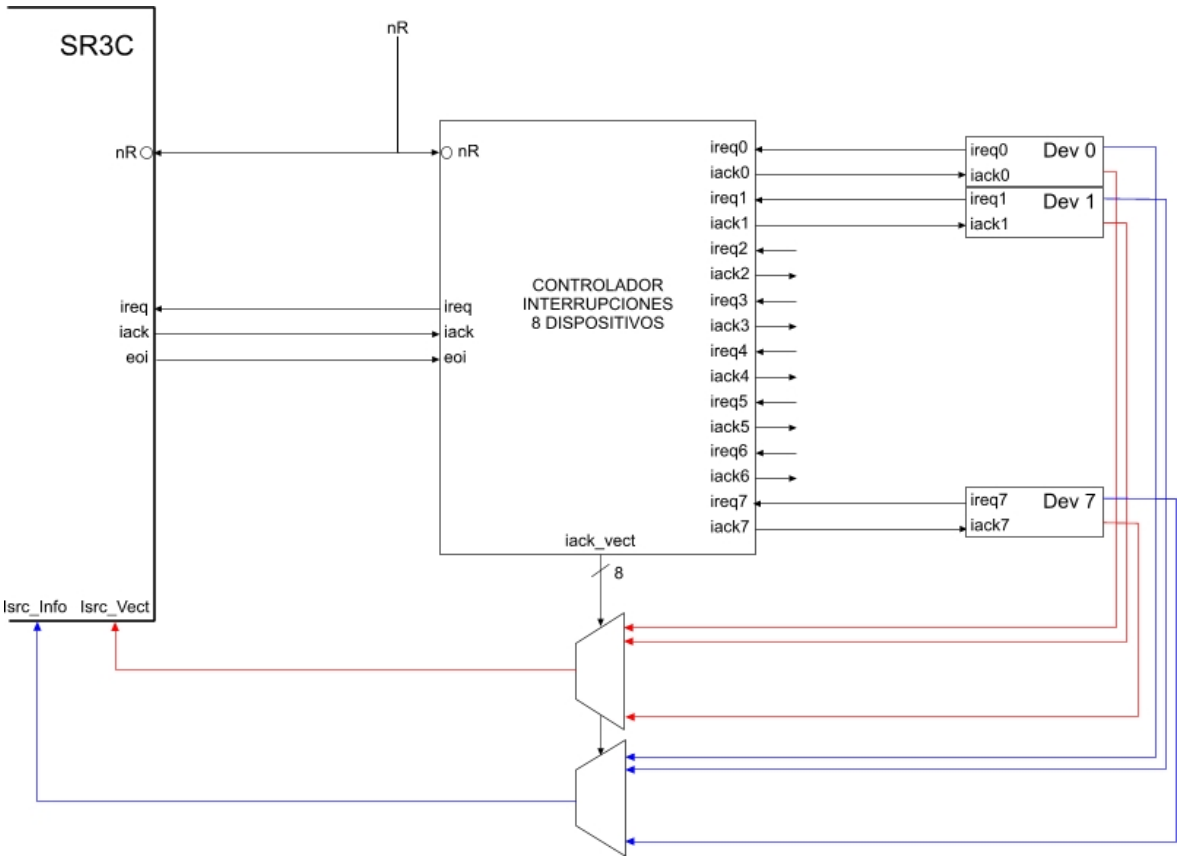

Figura 5.3: Esquema de conexiones del controlador de interrupciones

interrupciones que solucione dicho problema.

Controlador de Interrupciones

El controlador de interrupciones permite conectar y gestionar hasta ocho periféricos interruptores. Cada uno de los ochos periféricos conectados tiene una prioridad diferente (de 0 a 7). En la Figura 5.3 se puede observar su esquema de conexiones. El controlador se basa en dos conceptos:

- Prioridad. Si varios dispositivos realizan sus interrupciones al mismo tiempo, únicamente la interrupción más prioritaria será contestada por el procesador.
- Jerarquia. Una vez que se está procesando una interrupción, no se aceptan otras menos prioritarias que ésta. Para conseguir esto, se utiliza un registro que enmascara las interrupciones menos prioritarias que la que se está llevando a cabo en ese momento.

El controlador se puede conectar a otros controladores, no limitando de esta forma el número de periféricos interruptores conectables al procesador. Por ejemplo se pueden conectar ocho controladores como dispositivos interruptores a un controlador, que

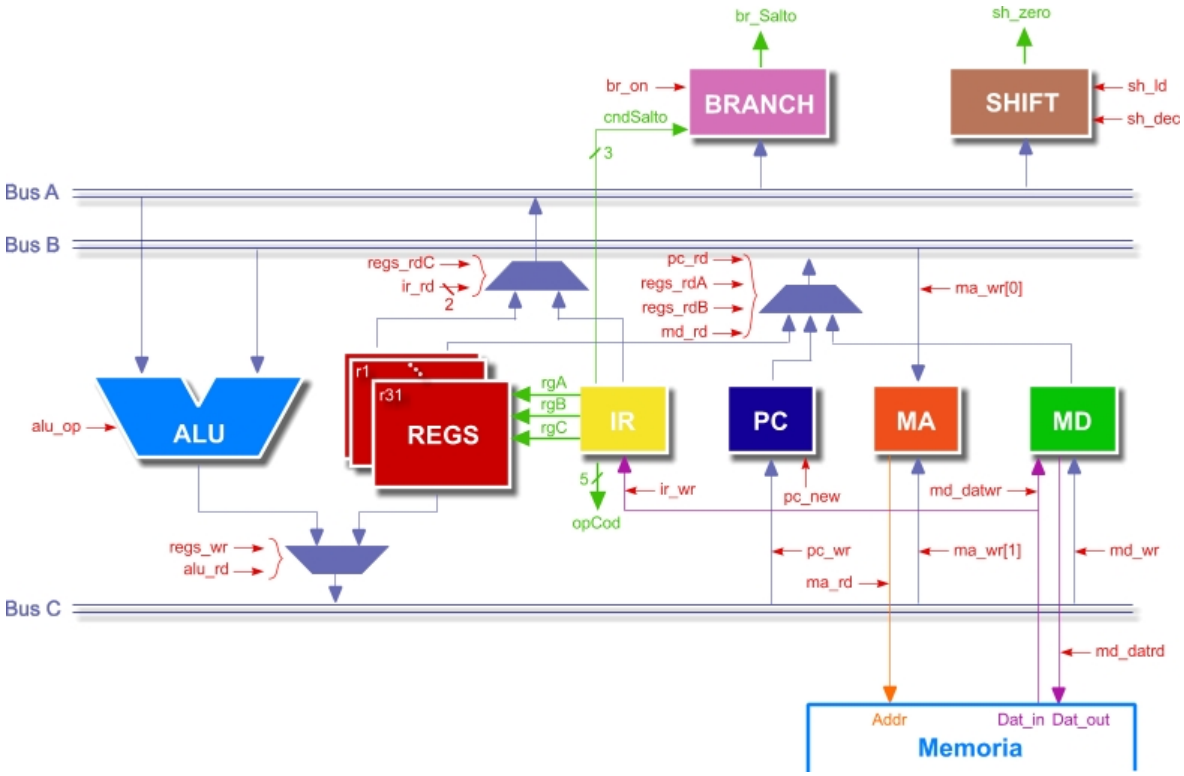

Figura 5.4: Arquitectura de la UP del Procesador SR3C con Multiplexores

será el que se conectará al procesador. De esta forma se podrían conectar hasta 64 dispositivos interruptores al procesador.

5.4. SR3C con Multiplexores

Como ya se ha comentado con anterioridad, el procesador SR3C tiene una arquitectura de 3 buses. En estos tres buses escriben diferentes registros y unidades funcionales. Por ello, y para evitar problemas, todos los elementos que pueden modificar el estado de cualquier bus deben acceder a éstos mediante un buffer triestado. De esta forma cuando dichos elementos no escriben nada, el bus permanece en alta impedancia.

Otra posibilidad es la de incluir un multiplexor en cada bus que controle el acceso al bus por parte de los elementos que modifican su estado. El multiplexor además puede dejar el bus en el estado lógico 0, cuando ningún elemento modifique su estado, eliminando así el estado eléctrico de alta impedancia del circuito. Por esto se ha desarrollado otra versión del procesador SR3C incluyendo dichos multiplexores. En la Figura 5.4 se puede observar la arquitectura de la Unidad de Proceso con los multiplexores.

5.5. SR3C compatible con Bus Wishbone

Uno de los requisitos adicionales, presentados en la introducción de este documento, era que el procesador se ha de poder acoplar fácilmente a sistemas más complejos mediante el cumplimiento de algún estándar de buses para SoC. Por eso se ha desarrollado una versión del procesador SR3C compatible con el estándar de buses Wishbone.

Los cambios introducidos han sido mínimos. Básicamente se ha modificado la interfaz del procesador, quedando de la siguiente manera:

```
entity src is
  port( RST_I:        in std_logic;                        -- Reset
        CLK_I:        in std_logic;                        -- Clock
        ADR_O:        out std_logic_vector(31 downto 0);   -- Addr
        DAT_I:        in std_logic_vector(31 downto 0);    -- Dat_in
        DAT_O:        out std_logic_vector(31 downto 0);   -- Dat_out
        SEL_O:        out std_logic;                       -- Selec. Byte
        STB_O:        out std_logic;                       -- Strobe
        CYC_O:        out std_logic;                       -- Ciclo Activo
        WE_O:         out std_logic;                       -- Write Enable
        ACK_I:        in std_logic;                        -- Acknowledge
        ISRC_VECT_I:  in std_logic_vector (7 downto 0);    -- Vector Int.
        ISRC_INFO_I:  in std_logic_vector (15 downto 0);   -- Inform. Int.
        IREQ_I:       in std_logic;                        -- Int. Request
        IACK_O:       out std_logic;                       -- Int. Ack
        EOI_O:        out std_logic);                      -- End Of Int.
end;
```

Para comprobar el correcto funcionamiento de la interfaz Wishbone del procesador se ha desarrollado una memoria RAM FASM *(FPGA and ASIC Subset Model)* síncrona. Esta memoria está descrita en el propio manual del estándar Wishbone [3]. En la Figura 5.5 se puede ver un diagrama de conexiones, un diagrama de tiempo para la lectura y un diagrama de tiempo para la escritura.

Durante los ciclos de escritura, la memoria RAM FASM almacena el dato de entrada en la dirección indicada cuando: la señal WE (*Write Enable*) está activa y llega un flanco de subida de reloj.

Durante los ciclos de lectura, la memoria RAM FASM funciona como una memoria asíncrona. Los datos son buscados en la dirección de memoria indicada por el bus de direcciones, y aparece en bus de datos de salida. La señal de reloj es ignorada.

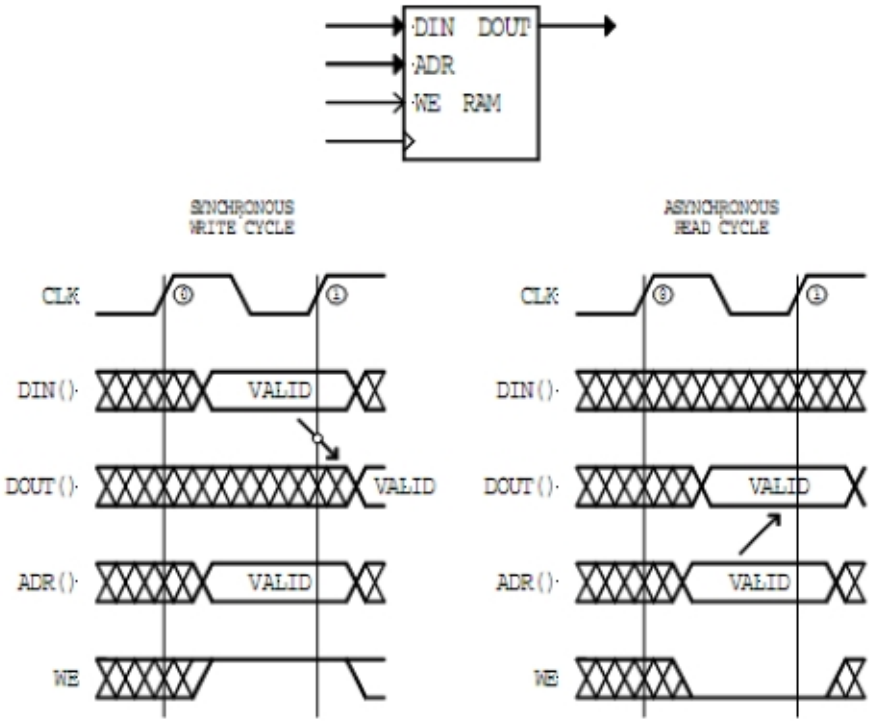

Figura 5.5: Diagrama de Conexiones y de tiempos de la memoria síncrona FASM RAM

Por lo tanto, durante los ciclos de escritura, el bus de datos de salida se actualiza inmediatamente.

5.6. Ensamblador

Para facilitar la escritura de programas para el procesador SR3C, se ha desarrollado un ensamblador. El ensamblador lee un fichero escrito en lenguaje ensamblador del SR3C y sustituye cada uno de los códigos mnemotécnicos que aparecen por su código de operación correspondiente en sistema binario.

El ensamblador se ha desarrollado mediante el lenguaje de programación de código abierto Python. El ensamblador puede ejecutarse en diversas plataformas gracias a que Python es un lenguaje de programación interpretado. Aunque también se puede compilar en algunas plataformas para no necesitar ninguna consola virtual.

En el Apéndice B se encuentra un sencillo manual del ensamblador desarrollado en este proyecto para el procesador SR3C.

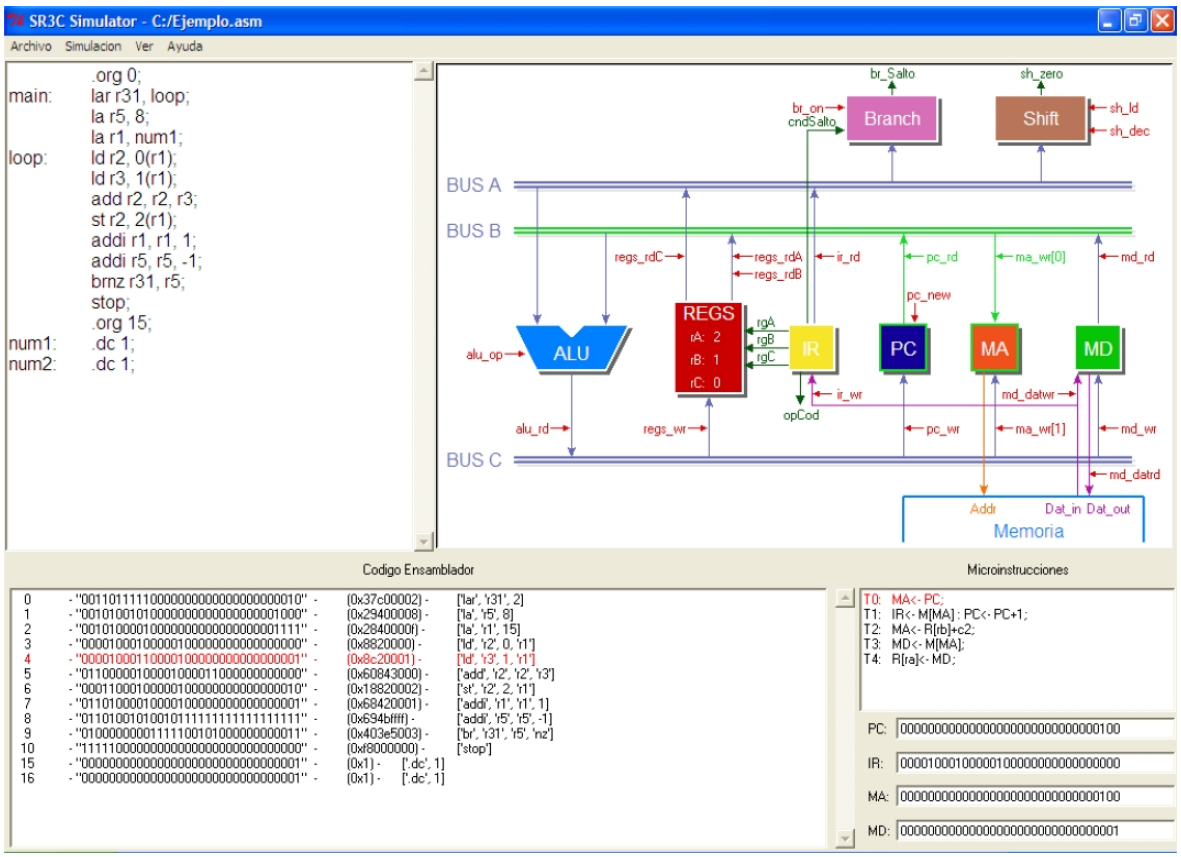

Figura 5.6: Captura de pantalla del Simulador del procesador SR3C

5.7. Simulador

El último de los requisitos del procesador, presentados en la introducción de este documento, hacía referencia a la realización de un simulador de cara a utilizar el procesador como plataforma educativa.

Al igual que el ensamblador presentado en la sección anterior, el simulador también está desarrollado en Python. Para la realización de la interfaz gráfica se ha recurrido a la biblioteca de Python: Tkinter.

El simulador permite introducir código ensamblador, ensamblarlo y simular su ejecución paso a paso. Mediante un esquema del procesador se pueden apreciar tanto las señales de control activadas por la UC, como los caminos de datos activos en cada instrucción del programa. El contenido de la memoria y de los registros del procesador se puede consultar en cualquier momento, visualizandolo a través de dos ventanas auxiliares. Además permite guardar y abrir archivos con programas en lenguaje ensamblador del procesador SR3C.

El simulador se divide en 5 bloques: código en ensamblador, código binario ensamblado, esquema gráfico del procesador, microinstrucciones de cada instrucción y contenido de los registros PC, IR, MA y MD. Además de las dos ventanas auxiliares

que muestran el contenido de la memoria y de los registros de propósito general. En la Figura 5.6 se puede ver una captura de pantalla del simulador.

En el Apéndice C se encuentra el manual del simulador desarrollado en este proyecto.

Capítulo 6

Resultados

El procesador SR3C, introducido en el anterior capítulo, se ha descrito mediante el lenguaje de descripción de hardware VHDL. Gracias a esto se ha podido sintetizar en una FPGA y probar su correcto funcionamiento.

Mediante el uso del ensamblador se han escrito unos algoritmos para testear el buen funcionamiento del procesador. Estos algoritmos son básicamente los siguientes:

- Serie de Fibonacci.
- Multiplicación mediante desplazamientos.
- División entera mediante restas.

También se ha escrito una versión del algoritmo de multiplicación con la característica de incorporar una rutina de interrupción para probar la versión del procesador con soporte para interrupciones. Con estos algoritmos se han probado todas las versiones, explicadas en el capítulo anterior, del procesador SR3C.

Con la ayuda del programa ModelSim®, de la compañía Mentor Graphics®, se ha realizado una simulación funcional. Primero de todos los componentes que forman el procesador por separado, y posteriormente del procesador SR3C al completo. Los algoritmos antes mencionados se han ejecutado correctamente en el proceso de simulación mediante el uso de una memoria de 32 bits, descrita también mediante VHDL.

Por ejemplo, el código en ensamblador del procesador SR3C del algoritmo de Fibonacci es el siguiente:

```
      .org 0;
main: lar r31, loop;
      la r5, 8;
      la r1, num1;
```

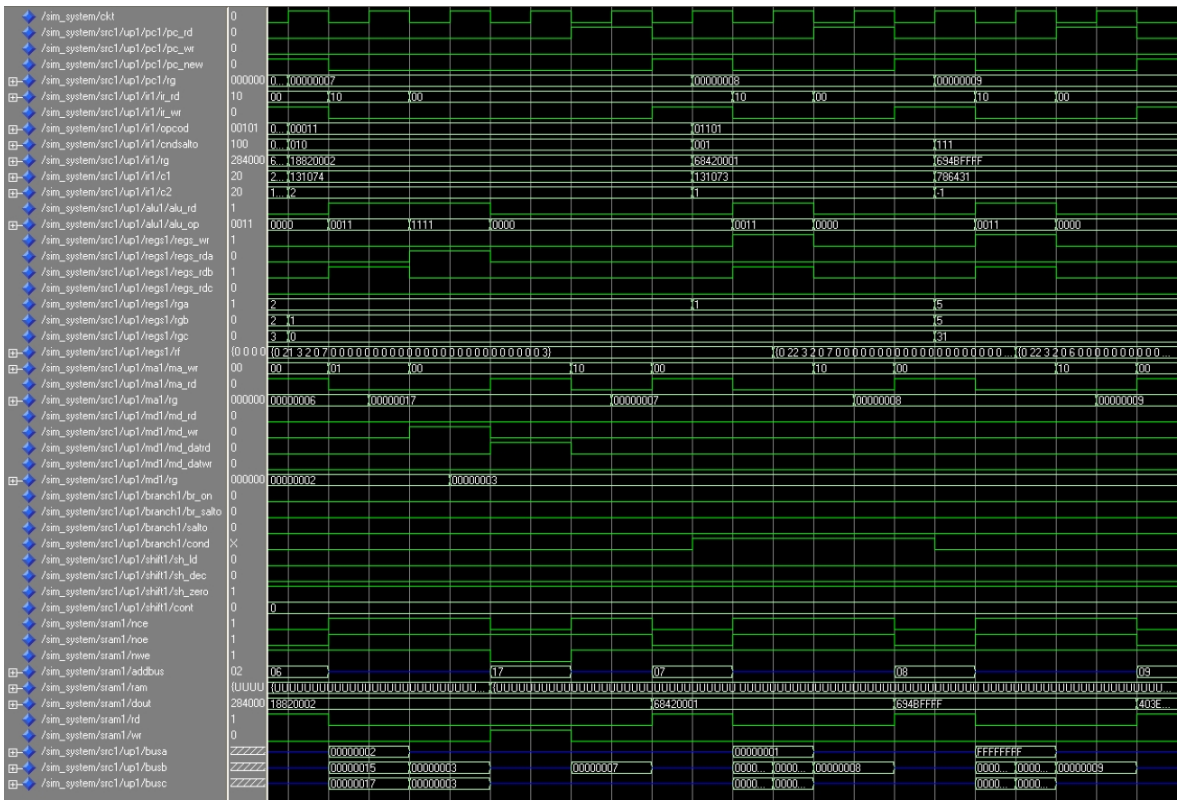

Figura 6.1: Diagrama temporal de la simulación algoritmo de Fibonacci

```
loop: ld r2, 0(r1);
      ld r3, 1(r1);
      add r2, r2, r3;
      st r2, 2(r1);
      addi r1, r1, 1;
      addi r5, r5, -1;
      brnz r31, r5;
      stop;
      .org 15;
num1: .dc 1;
num2: .dc 1;
```

El código anterior calcula la sucesión de Fibonacci hasta n=10, obteniendo como resultado el valor 55. Este código se ha introducido ya ensamblado en la memoria del sistema para poder realizar la simulación. En la Figura 6.1 se puede ver un diagrama temporal de las señales del procesador, durante la simulación del procesador ejecutando el anterior algoritmo. En dicha Figura se puede observar, entre otras cosas, como se va incrementado el Contador de Programa, el contenido de los registros de propósito general, o los buses del procesador.

Otro ejemplo interesante es el provocado por la aparición de una interrupción.

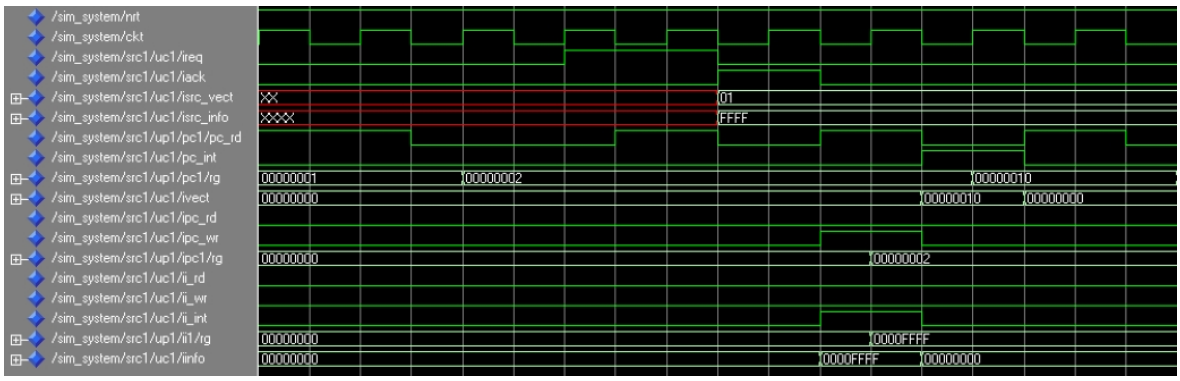

Figura 6.2: Diagrama temporal de la simulación de una interrupción

En la Figura 6.2 se puede observar un diagrama temporal de la simulación de una interrupción. Se puede ver como al llegar la señal *ireq*, el procesador responde con la activación de la señal *iack*, provocando que se actualice:

- El registro PC con la dirección de la rutina de interrupción indicada por el dispositivo interruptor mediante el vector de interrupción *Isrc_vect*.
- El registro II con la información proporcionada por el dispositivo interruptor mediante el vector de información *Isrc_info*.
- El registro IPC con la dirección que contiene el registro PC antes de modificarse con la dirección de la rutina de interrupción. De esta forma se podrá volver al punto donde se dejó la ejecución.

Posteriormente a la simulación se ha sintetizado el procesador en una FPGA Cyclone EP1C12Q240 incorporada en la placa de prototipado UP3-1C12 de Altera. Para sintetizar el procesador dentro de la FPGA se ha utilizado el software, también de Altera, Quartus® II.

Para la realización de la síntesis del procesador se ha utilizado la versión realizada con multiplexores y sin buses bidireccionales debido a que las FPGAs de Altera no pueden implementar buffers triestados, ni buses bidireccionales, a no ser que estos últimos salgan hacia el exterior de la FPGA. Además no se ha podido utilizar el procesador que realiza las búsquedas de instrucciones en un solo ciclo debido a que las FPGAs de Altera únicamente sintetizan memorias en bloques de memoria (no consumiendo así recursos lógicos) si estas tienen todas sus entradas y salidas registradas. En [11] se muestran los estilos de codificación de VHDL correctos para síntetizar en FPGAs Altera.

Para comprobar el correcto funcionamiento del procesador sintetizado, éste se ha incluido dentro de un sistema compuesto, básicamente, por los siguientes elementos:

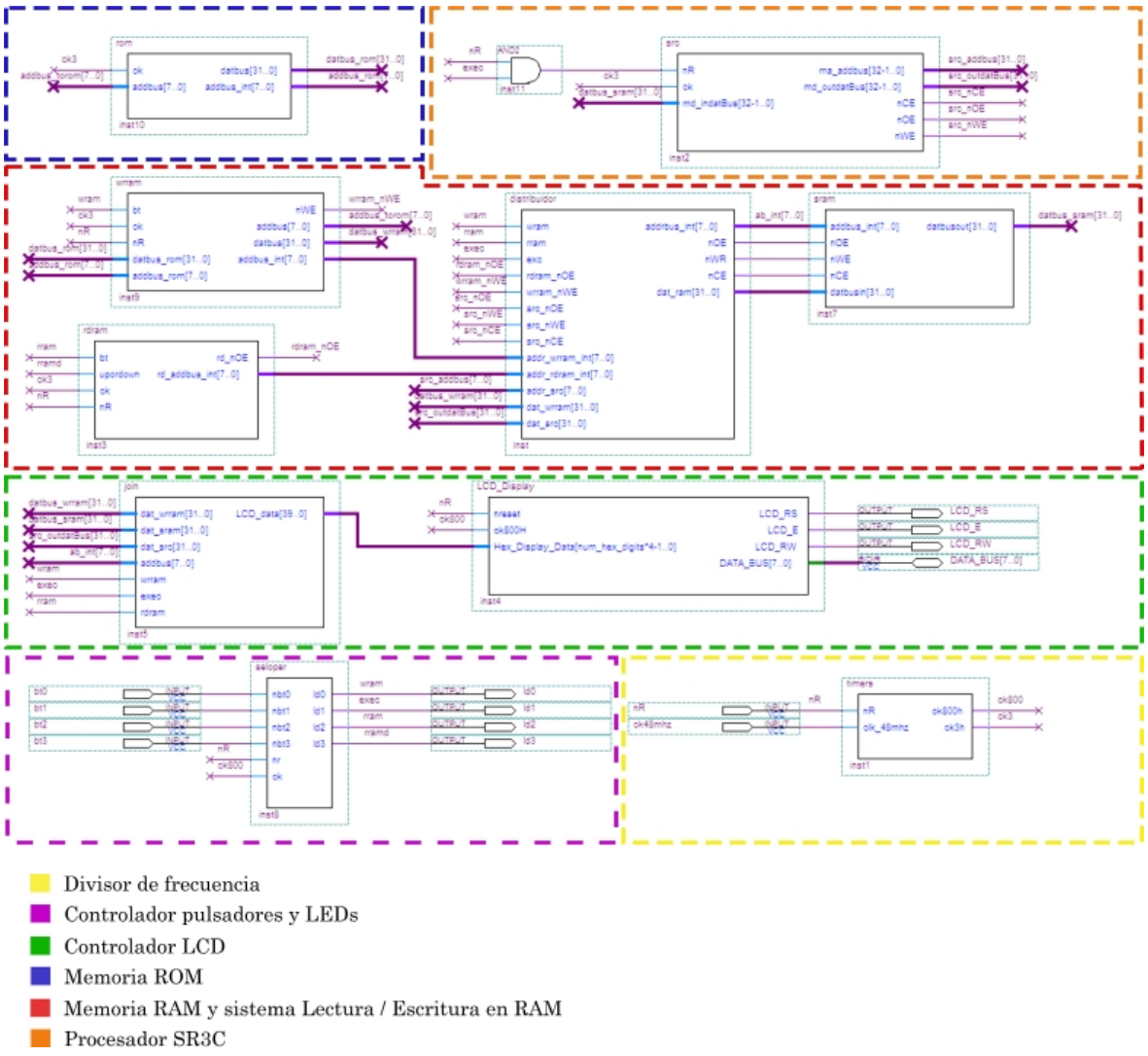

Figura 6.3: Diagrama del Sistema sintetizado en la FPGA

- Divisor de Frecuencia. El reloj de entrada del sistema funciona a 48Mhz. Con esta señal de reloj se generan dos señales de reloj: una a 800Mhz para el controlador del LCD (comentado a continuación) y otra a 3hz para el resto del sistema.
- Controlador Pulsadores y LEDs. El sistema incorpora pulsadores y LEDs (*Light-Emitting Diodes*) para interactuar con el usuario.
- Controlador LCD. El sistema incorpora un controlador para trabajar con el LCD (*Liquid Crystal Display*) que incorpora la placa de prototipado.
- ROM. Memoria donde se alberga el código del programa a ejecutar por el procesador.
- RAM. Memoria donde se ejecuta el programa y se almacenan los datos.
- Procesador SR3C.

FPGA	Lógica requerida	Frecuencia de trabajo
Altera Cyclone	2100 LEs	20 Mhz
Altera Stratix	2100 LEs	27 Mhz
Xillinx Spartan 3	30.000 gates	16 Mhz
Xillinx Virtex 2	30.000 gates	32 Mhz

Cuadro 6.1: Resultados de síntesis en FPGA obtenidos

En la Figura 6.3 se puede observar un diagrama del sistema sintetizado en la FPGA.

El sistema, una vez sintetizado en la FPGA de la placa de prototipado, funciona de la siguiente manera:

- Al pulsar el primer botón, el código de programa almacenado en la ROM se descarga en la memoria RAM. Mientras se lleva a cabo este proceso se visualizan las instrucciones (y sus direcciones) copiadas por la pantalla LCD, en formato hexadecimal.
- Al pulsar el segundo botón, se ejecuta el código almacenado en la RAM mediante el procesador SR3C. Cada instrucción ejecutada por el procesador se visualiza por el LCD.
- Al pulsar el tercer botón, se puede ver el contenido de la memoria RAM a través del LCD. En cada ciclo de reloj, es decir, cada 0,33 segundos la instrucción y dirección de memoria mostradas en el LCD se sustituyen por la instrucción de memoria contenida en la siguiente dirección de memoria, en sentido ascendente. El sentido de visualización del contenido de la memoria (ascendente o descendente) se puede modificar mediante un switch en la propia placa.

Tanto las direcciones de memoria como las instrucciones, se pueden visualizar en formato hexadecimal a través del LCD gracias a que el sistema funciona a 3hz. De todas maneras este parámetro se puede ajustar fácilmente.

En el Cuadro 6.1 se pueden observar los resultados de sintesis obtenidos en varios modelos de FPGA, tanto de la compañía Altera, como de la compañía Xillinx. En el caso de las FPGAs Xillinx, se ha utilizado el programa ISE 8.1 para la síntesis del procesador.

Rendimiento del procesador SR3C

En el Cuadro 6.2 se pueden observar los ciclos de reloj necesarios para la ejecución de cada instrucción del repertorio de instrucciones del procesador SR3C. Los cálculos se han realizado teniendo en cuenta que la búsqueda de instrucción se realiza en dos ciclos de reloj, tal como se puede observar en el Cuadro A.1 del Apéndice A.

Instrucción	Ciclos
ld	5
ldr	5
la	3
lar	3
st	4
str	4
add	3
addi	3
sub	3
neg	3
and	3
andi	3
or	3
ori	3
not	3
xor	3

Instrucción	Ciclos
nxor	3
nand	3
nor	3
br	4
brl	5
shr	4+n
shra	4+n
shl	4+n
shc	4+n
nop	3
stop	3
svi	4
ri	4
een	3
edi	3
rfi	3

Cuadro 6.2: Ciclos requeridos para la ejecución de las instrucciones del SR3C

Teniendo en cuenta los datos mostrados en dicho Cuadro y la mezcla típica de instrucciones utilizada para el cálculo del CPI de las versiones del procesador SRC (ver capítulo 2), se puede concluir que el procesador SR3C tiene un CPI aproximado de 3,7 ciclos por instrucción. Es decir, se ha reducido en un ciclo por instrucción el CPI respecto a la versión de 3 buses del procesador SRC introducida en [22]. Además si se tiene en cuenta la versión del procesador SR3C en el que la búsqueda de instrucciones se realiza en un solo ciclo de reloj, el CPI obtenido es entonces de tan solo 2,7 ciclos de reloj.

El procesador SR3C por lo tanto obtiene una cifra cercana a los 12 MIPS, gracias a su CPI de 2,7 ciclos de reloj y a su frecuencia de trabajo de 32 Mhz conseguida en la FPGA Xillinx Virtex 2. Si además se tienen en cuenta lo explicado en el capítulo 4 referente a las tecnologías de diseño actuales (las FPGAs son alrededor de 4 veces más lentas que los ASICs), el procesador SR3C podría conseguir cifras cercanas a los 50 MIPS trabajando a una frecuencia de reloj de 130 Mhz en un ASIC.

Capítulo 7

Conclusiones

El objetivo de este proyecto, tal como indica su nombre, era el análisis y diseño de un procesador RISC simple para adquisición y proceso de datos. Las características principales que se le requerían a dicho procesador eran las siguientes:

- El procesador debía de ser simple.
- El conjunto de instrucciones del procesador, a su vez, también debía de ser simple.
- El procesador tenía que poseer una interfaz compatible con algún estándar de buses.
- El procesador se tenía que simular y sintetizar mediante VHDL.
- Se había de desarrollar un simulador de dicho procesador.

El procesador obtenido, el SR3C, cumple con todos estos requisitos gracias, en parte, a su arquitectura RISC sencilla y su rendimiento.

Gracias a la versión del procesador compatible con el estándar de buses Wishbone, el SR3C se puede integrar fácilmente en aplicaciones reales basadas en FPGAs o SoCs.

Las herramientas software desarrolladas en este proyecto: el ensamblador y el simulador gráfico ofrecen la posibilidad de utilizar este procesador como plataforma educativa. Incluso se puede utilizar para este fin el sistema realizado para su síntesis en la placa de prototipado UP3 de Altera. De esta forma, el procesador SR3C puede ser de utilidad en la realización de las siguientes asignaturas de la titulación de ingeniería informática:

- Fundamentos de computadores. La arquitectura simple del procesador SR3C, junto al ensamblador y simulador pueden ser de utilidad en la realización de esta asignatura, sobretodo en las sesiones prácticas.

- Diseño de circuitos integrados. Tanto el procesador SR3C, como el sistema de prototipado para la placa UP3 de Altera pueden ser útiles en la parte práctica de esta asignatura. Se pueden realizar modificaciones sobre el procesador, introducir mejoras o diseñar componentes periféricos.

- Compiladores. Al disponer de una arquitectura simple y un repertorio de instrucciones reducido, el procesador SR3C puede ser una buena plataforma para el desarrollo de un compilador de un lenguaje de programación simplificado. Incluso, el código compilado por el compilador se puede simular o ejecutar directamente en el procesador.

En comparación con el procesador SRC documentado en [22], el SR3C ha introducido las siguientes mejoras:

- Arquitectura mejorada para disminuir los ciclos por instrucción y aumentar el rendimiento del procesador.

- Especificación de un controlador de interrupciones.

- Mejora y ampliación del repertorio de instrucciones.

- Adaptación a la interfaz estándar de buses Wishbone.

- Versión del procesador con multiplexores para la eliminación de estados en alta impedancia.

Debido a la gran cantidad de carga de trabajo de este proyecto y al tiempo limitado para llevarlo a cabo, no se ha podido trabajar en algunos aspectos del procesador o relacionados con éste. Por lo tanto quedan pendientes algunas modificaciones y ampliaciones que quizá serían interesantes. A continuación se muestran algunas que podrían mejorar el procesador considerablemente:

- Optimizaciones de espacio y rendimiento. Seguramente estudiando las recomendaciones sobre el diseño en lenguajes de descripción de hardware de los fabricantes de FPGAs, y realizando algunas pruebas, se podrían conseguir mejoras en cuanto al espacio ocupado y el rendimiento del procesador SR3C.

- Diseño de algunos controladores y periféricos. Sería interesante dotar al procesador SR3C de un conjunto de controladores y periféricos tales como UARTs, timers, controladores de interfaces (I2C, SPI, Canbus,...) para convertirlo en un microcontrolador.

- Realización de un Compilador. Si se quieren escribir y ejecutar en el procesador programas más complejos se hace casi indispensable el disponer de un compilador eficiente.
- Diseño de una versión con pipeline. Una versión con pipeline del procesador mejoraría sustancialmente su rendimiento, y con ello, el rango de aplicaciones en las que sería útil. Además sería interesante comparar la versión con pipeline con la desarrollada en este proyecto.

Apéndice A

Conjunto de instrucciones del SR3C

En este apéndice se puede ver todo el conjunto de instrucciones del procesador SR3C. De cada instrucción se señalan sus microinstrucciones y las señales de control que se activan en cada caso[1].

Las instrucciones están divididas en 7 grupos. Las microinstrucciones correspondientes a la búsqueda de la instrucción (comunes a todas ellas) se han separado en otro Cuadro para evitar repeticiones. Cada grupo de instrucciones está en un Cuadro separado. Los grupos son los siguientes:

- Busqueda de Instrucciones (Cuadro A.1)
- Instrucciones de Carga (Cuadro A.2)
- Instrucciones de Almacenamiento (Cuadro A.3)
- Instrucciones Aritméticas (Cuadro A.4)
- Instrucciones de Salto (Cuadro A.5)
- Instrucciones de Desplazamiento (Cuadro A.6)
- Otras Instrucciones (Cuadro A.7)
- Instrucciones de Interrupción (Cuadro A.8)

[1]Las señales de control pueden tomar el valor 1 (activadas) o 0 (desactivadas). Cuando no tienen ningún valor asociado significa que están a 0 (desactivadas).

Instrucción	next	pc_rd	pc_wr	pc_new	ir_rd(2)	ir_wr	regs_wr	regs_rdA	regs_rdB	regs_rdC	alu_op	alu_rd	br_on	ma_wr(2)	ma_rd	md_rd	md_wr	md_datrd	md_datwr	sh_ld	sh_dec
Búsqueda de instrucción																					
Fetch (genérico)	**2 primeras μ-instrucciones, comunes a todas las instrucciones**																				
T0 MA←PC;	**T1**	1												10							
T1 IR←M[MA] ; PC←PC+1;	**T2**			1		1									1						

Cuadro A.1: Busqueda de Instrucciones

Instrucciones de Carga																			
Instrucción	next	pc_rd	pc_wr	pc_new	ir_rd(2)	ir_wr	regs_wr	regs_rdA	regs_rdB	regs_rdC	alu_op	alu_rd	br_on	ma_wr(2)	ma_rd	md_rd	md_wr	md_datrd	md_datwr	sh_ld	sh_dec
ld (opCod=00001)	**Carga directa/indexada: R[ra]←M[c2] / R[ra]←M[c2+R[rb]]**																				
T2 MA←R[rb]+c2;	**T3**				**10**				**1**		**0011**	**1**		**01**							
T3 MD← M[MA];	**T4**														**1**				**1**		
T4 R[ra]←MD;	**T0**						**1**				**1111**	**1**				**1**					
ldr (opCod=00010)	**Carga relativa: R[ra]←M[PC+c1]**																				
T2 MA←PC+c1;	**T3**	**1**			**01**						**0011**	**1**		**01**							
T3 MD← M[MA];	**T4**														**1**				**1**		
T4 R[ra]←MD;	**T0**						**1**				**1111**	**1**				**1**					
la (opCod=00101)	**Carga dirección desplazamiento indexado/indirecto: R[ra]←c2 / R[ra]←c2+R[rb]**																				
T2 R[ra]←R[rb]+c2;	**T0**				**10**		**1**		**1**		**0011**	**1**									
lar (opCod=00110)	**Carga dirección relativa: R[ra]←PC+c1**																				
T2 R[ra]←PC+c1;	**T0**	**1**			**01**		**1**				**0011**	**1**									

Cuadro A.2: Instrucciones de Carga

Instrucciones de Almacenamiento																					
Instrucción	next	pc_rd	pc_wr	pc_new	ir_rd(2)	ir_wr	regs_wr	regs_rdA	regs_rdB	regs_rdC	alu_op	alu_rd	br_on	ma_wr(2)	ma_rd	md_rd	md_wr	md_datrd	md_datwr	sh_ld	sh_dec
st (opCod=00011)	Almacenamiento directo / indexado: M[c2]←R[ra] / M[c2+R[rb]]←R[ra]																				
T2 MA←R[rb]+c2;	T3				10				1		0011	1		01							
T3 MD←R[ra];	T4							1			1111	1					1				
T4 M[MA]←MD;	T0														1			1			
str (opCod=00100)	Almacenamiento relativo: M[PC+c1] ←R[ra]																				
T2 MA← PC+c1;	T3		1		01						0011	1		01							
T3 MD←R[ra];	T4							1			1111	1					1				
T4 M[MA]←MD;	T0														1			1			

Cuadro A.3: Instrucciones de Almacenamiento

Instrucciones Aritméticas																				
Instrucción	next	pc_rd	pc_wr	pc_new	ir_rd(2)	ir_wr	regs_wr	regs_rdA	regs_rdB	regs_rdC	alu_op	alu_rd	br_on	ma_wr(2)	ma_rd	md_rd	md_wr	md_datrd	md_datwr	sh_ld	sh_dec
add (opCod=01100)	Suma: R[ra]←R[rb] + R[rc]																				
T2 R[ra]←R[rb] + R[rc];	T0						1		1	1	0011	1									
addi (opCod=01101)	Suma inmediata: R[ra]←R[rb] + c2<16..0>																				
T2 R[ra]←R[rb] + c2;	T0				10		1		1		0011	1									
sub (opCod=01110)	Resta: R[ra]←R[rb] - R[rc]																				
T2 R[ra]←R[rb] - R[rc];	T0						1		1	1	0100	1									
neg (opCod=01111)	Negación: R[ra]← -R[rc] → C(2)																				
T2 R[ra]←neg(R[rc]);	T0						1			1	0101	1									
and (opCod=10100)	And: R[ra]←R[rb] ∧ R[rc]																				
T2 R[ra]←R[rb] ∧ R[rc];	T0						1		1	1	1011	1									
andi (opCod=10101)	And immediata: R[ra]←R[rb] ∧ c2<16..0>																				
T2 R[ra]←R[rb] ∧ c2;	T0				10		1		1		1011	1									
or (opCod=10110)	Or: R[ra]←R[rb] ∨ R[rc]																				
T2 R[ra]←R[rb] ∨ R[rc];	T0						1		1	1	1100	1									
ori (opCod=10111)	Or immediata: R[ra]←R[rb] ∨ c2<16..0>																				
T2 R[ra]←R[rb] ∨ c2;	T0				10		1		1		1100	1									
not (opCod=11000)	Not : R[ra]←not(R[rc]) → C(1)																				
T2 R[ra]←not(R[rc]);	T0						1			1	0110	1									
xor (opCod=11001)	Xor: R[ra]←R[rb] xor R[rc]																				
T2 R[ra]←R[rb] xor R[rc];	T0						1		1	1	1101	1									
nxor (opCod=10011)	Nxor: R[ra]←R[rb] nxor R[rc]																				
T2 R[ra]←R[rb] nxor R[rc];	T0						1		1	1	1110	1									
nand (opCod=00111)	Nand: R[ra]←R[rb] nand R[rc]																				
T2 R[ra]←R[rb] nand R[rc];	T0						1		1	1	0001	1									
nor (opCod=10010)	Nor: R[ra]←R[rb] nor R[rc]																				
T2 R[ra]←R[rb] nor R[rc];	T0						1		1	1	0010	1									

Cuadro A.4: Instrucciones Aritméticas

Instrucciones de Salto																					
Instrucción	next	pc_rd	pc_wr	pc_new	ir_rd(2)	ir_wr	regs_wr	regs_rdA	regs_rdB	regs_rdC	alu_op	alu_rd	br_on	ma_wr(2)	ma_rd	md_rd	md_wr	md_datrd	md_datwr	sh_ld	sh_dec
br (opCod=01000)	Salto (según condición) : ?PC←R[rb]																				
T2 BRANCH←R[rc];	**T3**									1			1								
T3 ?br_Salto=1→ PC←R[rb]; ?br_Salto=0	**T0** **T0**		1						1		1111	1									
br1 (opCod=01001)	Salto con link - almacena dirección de retorno - (según condición): ?PC←R[rb]																				
T2 BRANCH←R[rc];	**T3**									1			1								
T3 ?br_Salto=1→ R[ra]←PC; ?br_Salto=0	**T4** **T0**	1						1			1111	1									
T4 PC←R[rb];	**T0**		1						1		1111	1									

Cuadro A.5: Instrucciones de Salto

Instrucciones de Desplazamiento																				
Instrucción	next	pc_rd	pc_wr	pc_new	ir_rd(2)	ir_wr	regs_wr	regs_rdA	regs_rdB	regs_rdC	alu_op	alu_rd	br_on	ma_wr(2)	ma_rd	md_rd	md_wr	md_datrd	md_datwr	sh_ld	sh_dec
shr (opCod=11010)	Desplazamiento derecha: R[ra]<31..0>←(n@0)#R[rb]<31..0>																				
T2 SHIFT←IR<4..0>;	T3				11															1	
T3 ?sh_zero=1→ SHIFT←R[rc]<4..0>; R[ra]←R[rb]	T4						1		1	1	1111	1								1	
T4 ?sh_zero=0→ R[ra]←shd(R[ra]); ?sh_zero=1	T4 T0						1		1		0111	1									
shra (opCod=11011)	Desplazamiento derecha artimético: R[ra]<31..0>←(n@R[rb]<31>)#R[rb]<31..0>																				
T2 SHIFT←IR<4..0>;	T3				11															1	
T3 ?sh_zero=1→ SHIFT←R[rc]<4..0>; R[ra]←R[rb]	T4						1		1	1	1111	1								1	
T4 ?sh_zero=0→ R[ra]←shda(R[ra]) ?sh_zero=1	T4 T0						1		1		1001	1									
shl (opCod=11100)	Desplazamiento izquierda: R[ra]<31..0>←R[rb]<31..0>#(n@0)																				
T2 SHIFT←IR<4..0>;	T3				11															1	
T3 ?sh_zero=1→ SHIFT←R[rc]<4..0>; R[ra]←R[rb]	T4						1		1	1	1111	1								1	
T4 ?sh_zero=0→ R[ra]←shi(R[ra]) ?sh_zero=1	T4 T0						1		1		1000	1									
shc (opCod=11101)	Desplazamiento circular izquierda: R[ra]<31..0>←R[rb]<31..0>#(n@R[rb]<31>)																				
T2 SHIFT←IR<4..0>;	T3				11															1	
T3 ?sh_zero=1→ SHIFT←R[rc]<4..0>; R[ra]←R[rb]	T4						1		1	1	1111	1								1	
T4 ?sh_zero=0→ R[ra]←shi(R[ra]) ?sh_zero=1	T4 T0						1		1		1010	1									

Cuadro A.6: Instrucciones de Desplazamiento

Otras Instrucciones																																
Instrucción	next	pc_rd	pc_wr	pc_new	ir_rd(2)	ir_wr	regs_wr	regs_rdA	regs_rdB	regs_rdC	alu_op	alu_rd	br_on	ma_wr(2)	ma_rd	md_rd	md_wr	md_datrd	md_datwr	sh_ld	sh_dec
nop (opCod=00000)	No operación																					
T2	T0																					
stop (opCod=11111)	Parar																					
T2	Tf																					

Cuadro A.7: Otras Instrucciones

Instrucciones de Interrupción																																
Instrucción		next	pc_rd	pc_wr	pc_new	pc_int	ir_rd(2)	ir_wr	regs_wr	regs_rdA	regs_rdB	regs_rdC	alu_op	alu_rd	br_on	ma_wr(2)	ma_rd	md_rd	md_wr	md_datrd	md_datwr	sh_ld	sh_dec	ipc_rd	ipc_wr	ii_rd	ii_wr	ii_int
svi (opCod=10000)	Almacenar II e IPC: R[ra]←II; R[rb]←IPC;																															
T2	R[ra]←II;	T3							1																	1						
T3	R[rb]←IPC;	T0							1															1								
ri (opCod=10001)	Recuperar II e IPC: II←R[ra]; IPC←R[rb];																															
T2	II←R[ra];	T3								1																	1					
T3	IPC←R[rb];	T0									1														1						
een (opCod=01010)	Habilitar Interrupciones: IE←1																															
T2	IE←1;	T0																														
edi (opCod=01011)	Deshabilitar Interrupciones: IE←0																															
T2	IE←0;	T0																														
rfi (opCod=11110)	Fin de interrupción: PC←IPC; IE←1																															
T2	PC←IPC; IE←1	T0		1																					1							

Cuadro A.8: Instrucciones de Interrupción

Apéndice B

Manual del ensamblador

El ensamblador del procesador SR3C está realizado en Python. Además del uso de las instrucciones válidas para el procesador SR3C, también permite el uso de ciertas directivas (también llamadas pseudo operaciones).

Al ejectuar el ensamblador, éste solicita introducir el nombre del archivo fuente donde se encuentra el programa escrito en lenguaje ensamblador SR3C. Cada línea de dicho archivo ha de tener la siguiente estructura:

```
Etiqueta:       código operación       operandos       ;comentarios
```

La etiqueta y los dos puntos que la siguen pueden no aparecer, ya que son opcionales. El campo código operación contiene un código mnemotécnico de unos pocos caracteres o una pseudo operación, la cual empieza con un punto.

Los operandos pueden ser registros o constantes dependiendo de la instrucción. Los operandos se listan separados mediante comas, en el orden en que aparecen de izquierda a derecha en la instrucción en código máquina. Los direccionamientos indexados (registro + constante) se expresan de la siguiente forma: constante(registro). Ej.: 12(r3). Este tipo de direccionamiento únicamente ocurre en las instrucciones de carga y almacenamiento.

El registro r0 siempre contiene el valor 0, por lo que cualquier instrucción que desee modificar dicho valor será descarta.

Los valores por defecto son decimales. Para escribir valores en binario, dicho valor ha de ir precedido por una b. En caso de querer escribir valores en hexadecimal, los valores han de ir precedidos por una h.

Instrucciones

A continuación se muestran las instrucciones permitidas en el lenguaje ensamblador

SR3C:

```
ld    ra, c2         ;Carga directa: R[ra]=M[c2]
ld    ra, c2(rb)     ;Carga indexada(rb!=0): R[ra]=M[c2+R[rb]]
la    ra, c2         ;Carga dir. desplazamiento directo: R[ra]=c2
la    ra, c2(rb)     ;Carga dir. desplazamiento indexado: R[ra]=c2+R[rb]
ldr   ra, c1         ;Carga relativa: R[ra]=M[PC+c1]
lar   ra, c1         ;Carga dirección relativa: R[ra]=PC+c1
st    ra, c2         ;Almacenamiento directo: M[c2]=R[ra]
st    ra, c2(rb)     ;Almacenamiento indexado(rb!=0): M[c2+R[rb]]=R[ra]
str   ra, c1         ;Almacenamiento relativo: M[PC+c1]=R[ra]
add   ra, rb, rc     ;Suma en Complemento a 2: R[ra]=R[rb]+R[rc]
addi  ra, rb, c2     ;Suma inmediata en C2: R[ra]=R[rb]+c2
sub   ra, rb, rc     ;Resta en Complemento a 2: R[ra]=R[rb]-R[rc]
neg   ra, rc         ;Negación: R[ra]=-R[rc]
and   ra, rb, rc     ;AND lógica: R[ra]=R[rb] and R[rc]
andi  ra, rb, c2     ;AND lógica inmediata: R[ra]=R[rb] and c2
or    ra, rb, rc     ;OR lógica: R[ra]=R[rb] or R[rc]
ori   ra, rb, c2     ;OR lógica inmediata: R[ra]=R[rb] or c2
not   ra, rc         ;NOT: R[ra]=not(R[rc])
xor   ra, rb, rc     ;XOR lógica: R[ra]=R[rb] xor R[rc]
nxor  ra, rb, rc     ;NXOR lógica: R[ra]=R[rb] nxor R[rc]
nand  ra, rb, rc     ;NAND lógica: R[ra]=R[rb] nand R[rc]
nor   ra, rb, rc     ;NOR lógica: R[ra]=R[rb] nor R[rc]
br    rb             ;Salto incondicional a R[rb]
brl   ra, rb         ;Salto incondicional con link a R[rb]. R[ra]=PC
brlnv ra, rb, rc     ;No se salta pero se salva PC. R[ra]=PC
brzr  rb, rc         ;Salto a R[rb] si R[rc] es cero
brlzr ra, rb, rc     ;Salto con link a R[rb] si R[rc] es cero. R[ra]=PC
brnz  rb, rc         ;Salto a R[rb] si R[rc] no es cero
brlnz ra, rb, rc     ;Salto con link a R[rb] si R[rc] no es cero. R[ra]=PC
brpl  rb, rc         ;Salto a R[rb] si R[rc] es mayor o igual a 0
brlpl ra, rb, rc     ;Salto con link a R[rb] si R[rc]>=0. R[ra]=PC
brmi  rb, rc         ;Salto a R[rb] si R[rc] es negativo
brlmi ra, rb, rc     ;Salto con link a R[rb] si R[rc]<0. R[ra]=PC
shr   ra, rb, rc     ;Desplaza R[rb] a derecha en R[ra], R[rc] bits
shr   ra, rb, count  ;Desplaza R[rb] a derecha en R[ra], count bits
shra  ra, rb, rc     ;Desplaza Arit. R[rb] a der. en R[ra], R[rc] bits
shra  ra, rb, count  ;Desplaza Arit. R[rb] a der. en R[ra], count bits
```

```
shl   ra, rb, rc    ;Desplaza R[rb] a izquierda en R[ra], R[rc] bits
shl   ra, rb, count ;Desplaza R[rb] a izquierda en R[ra], count bits
shc   ra, rb, rc    ;Desplaza R[rb] circularmente en R[ra], R[rc] bits
shc   ra, rb, count ;Desplaza R[rb] circularmente en R[ra], count bits
nop                 ;No operación
stop                ;Parar ejecución
svi   ra, rb        ;Guarda los reg. II e IPC en Ra y Rb respectivamente.
ri    ra, rb        ;Vuelca los reg. ra y rb en II e IPC respectivamente.
een                 ;Habilita las interrupciones. IE=1
edi                 ;Deshabilita las interrupciones. IE=0
rfi                 ;Fin de rutina de interrupción. PC=IPC e IE=1
```

Directivas

A continuación se muestran las pseudo operaciones permitidas en el lenguaje ensamblador SR3C:

- Definición de constantes.

```
.equ valor     Define constantes
               ej.:   cnt: .equ  8
```

  Con el uso de esta directiva se pueden declarar constantes asociadas a etiquetas que posteriormente el ensamblador se ocupará de sustituir por sus valores.

- Dirección de inicio.

```
.org valor     Dirección de inicio
               ej.:        .org  0
```

  Esta directiva se utiliza para indicar la posición de memoria a partir de la cual se definirán variables, se reservará memoria o se almacenará el código del programa. Si no se utiliza, el valor por defecto es 0. El código del programa debe empezar en la posición 0 de memoria. Por eso, si se utiliza esta directiva con otro valor para definir variables o reservar espacio de memoria, se debe volver a utilizarla con el valor 0 justo antes de escribir el código del programa.

- Definición de variable.

```
.dc  valor     Fijar valor a variable de 32 bits
               ej.:   seq: .dc   1
```

Esta directiva almacena el valor indicado en memoria y asocia a una etiqueta la dirección de dicha posición de memoria.

- Reserva de memoria.

```
.dw  valor     Reserva memoria para cont palabras de 32 bits
               ej.:   arr: .dw   8
```

Esta directiva reserva el número de posiciones indicadas y asocia la posición de la primera dirección de memoria reservada a la etiqueta introducida.

Apéndice C

Manual del simulador

El simulador del procesador SR3C está realizado con Python y ofrece una interfaz gráfica amigable y sencilla. El simulador permite simular paso a paso el código escrito en ensamblador, después de ensamblarlo convirtiéndolo en código máquina.

El simulador se divide en 5 bloques (En la Figura 5.6 se puede ver una captura de pantalla del simulador):

- Formulario código en ensamblador.
- Forumlario código binario ensamblado.
- Esquema gráfico del procesador
- Formulario microinstrucciones.
- Contenido de los registros PC, IR, MA y MD.

El menú del programa se encuentra dividido en cuatro bloques: Archivo, Simulación, Ver y Ayuda. Dentro de estos bloques podemos encontrar las siguientes acciones:

- Archivo
    - Nuevo. Deja en blanco el formulario donde se escribe el código ensamblador. Si había código escrito sin guardar, da la opción de guardar dicho código.
    - Abrir. Permite abrir un archivo .asm con código ensamblador y muestra su contenido en el formulario del código ensamblador. Al igual que la acción de Nuevo, da la opción de guardar los cambios, si no se habían guardado.
    - Guardar. Permite guardar el contenido del formulario del código ensamblador en un archivo .asm, indicando el nombre y la ubicación de éste si todavía no se había indicado.

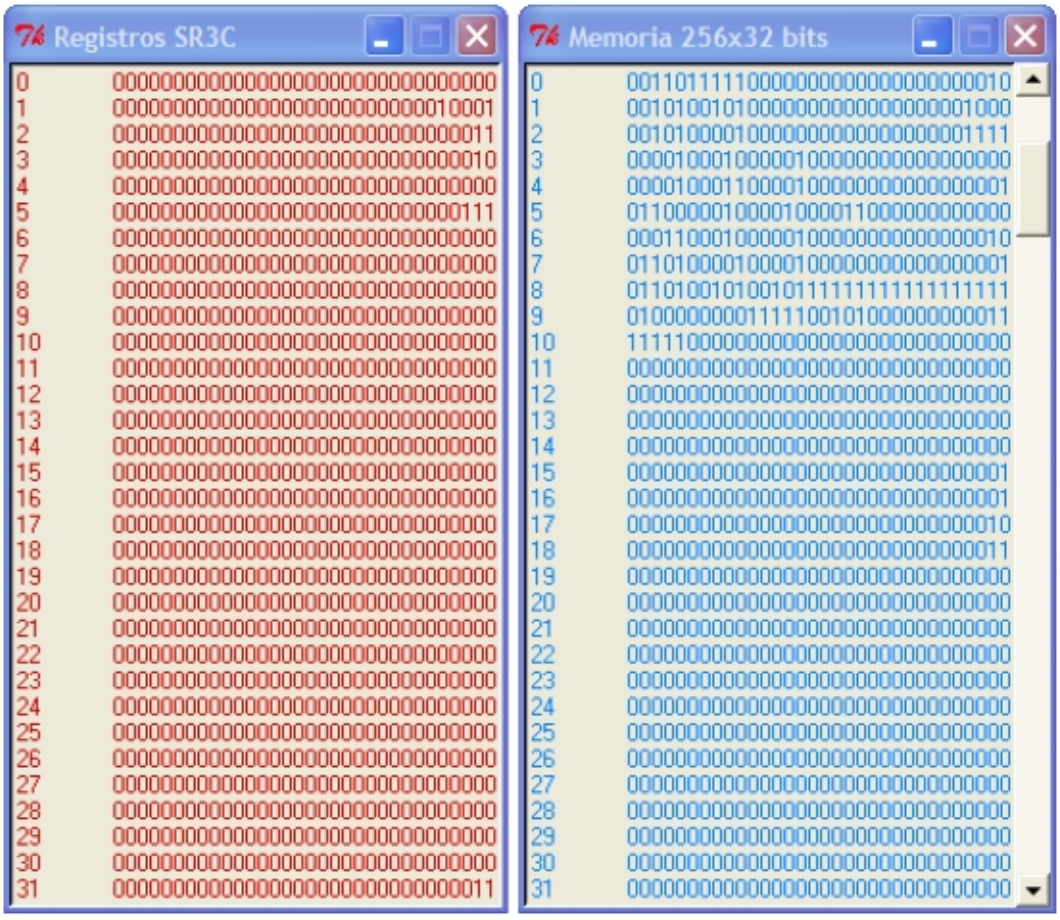

Figura C.1: Ventanas que muestran el contenido de la memoria y de los registros

- Guardar como. Permite guardar el contenido del formulario del código ensamblador en un archivo .asm, indicando el nombre y la ubicación de éste aunque ya tuviese nombre y ubicación.
- Salir. Cierra el simulador.

- Simulación
  - Simular. Simula la ejecución del programa entero.
  - Simular un paso. Simula un nuevo ciclo de reloj, en el que se ejecuta una nueva microinstrucción. La microsintrucción que se está ejecutando en ese momento se muestra resaltada en rojo dentro del formulario de microinstrucciones. Además se activan (se ponen en color verde) las señales de control y los caminos de datos necesarios para ejectuar dicha microinstrucción, en el esquema gráfico del procesador. Es necesario ensamblar el programa antes de simularlo.
  - Ensamblar. Convierte el código ensamblador en código máquina y lo muestra en el formulario de código ensamblado.

- Ver
  - Memoria. Abre la ventana que muestra el contenido de la memoria de 256 posiciones de 32 bits cada una.
  - Registros. Abre la ventana que muestra el contenido de los 32 registros de 32 bits de propósito general del procesador SR3C.

    En la Figura C.1 se puede ver una captura de pantalla de las ventanas que muestran el contenido de la memoria y el de los registros del procesador.
- Ayuda
  - Acerca de. Información acerca del simulador del SR3C y de su autor.

El código en ensamblador introducido ha de seguir las normas indicadas en el manual del ensamblador (Apéndice B).

Bibliografía

[1] *AMBA Specification (Revision 2.0).* ARM, Inc, Abril 1999. http://www.arm.com/products/solutions/AMBA_Spec.html.

[2] *Avalon Bus Specification Reference Manual.* Altera, corp, Julio 2002. www.altera.com.cn/literature/manual/mnl_avalon_bus.pdf.

[3] *Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores (Revision B.3).* OpenCores.org, 7 de Septiembre 2002.

[4] *Nios 3.0 CPU Data Sheet.* Altera, corp, Noviembre 2004. http://www.altera.com/literature/ds/ds_nios_cpu.pdf.

[5] *PicoBlaze 8-bit Embedded Microcontroller User Guide.* Xillinx, Inc, Noviembre 2005. http://www.xilinx.com/bvdocs/userguides/ug129.pdf.

[6] *AVR32 Architecture Document.* Atmel, Febrero 2006. http://www.atmel.com/dyn/resources/prod_documents/doc32000.pdf.

[7] *LatticeMico32 Processor Reference Manual.* Lattice Semiconductor Corporation, Septiembre 2006.

[8] *MicroBlaze Processor Reference Guide.* Xillinx, Inc, Septiembre 2006. http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf.

[9] Microprocessors on opencores.org. 2007(9 de Mayo), 8 de Mayo 2007. http://www.opencores.org/browse.cgi/filter/category_microprocessor.

[10] *Nios II Processor Reference Handbook.* Altera, corp, Mayo 2007. http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf.

[11] *Recommended HDL Coding Styles.* Altera, corp, Mayo 2007. http://www.altera.com/literature/hb/qts/qts_qii51007.pdf.

[12] I. H. Al-Mohandes, A. M. Rashed, H. F. Ragaie, and M. K. Elsaid. Synthesis and physical design of dlx risc processor. *Radio Science Conference, 1999. NRSC '99. Proceedings of the Sixteenth National*, pages C26–1–C26–8, 23-25 Febrero 1999.

[13] P. J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, San Francisco, second edition, 2002.

[14] M. Buhler and U. G. Baitinger. Vhdl-based development of a 32-bit pipelined risc processor for educational purposes. *Electrotechnical Conference, 1998. MELECON 98., 9th Mediterranean*, 1:138–142, 18-20 Mayo 1998.

[15] M. Dolle, S. Jhand, W. Lehner, O. Muller, and M. Schlett. A 32-b risc/dsp microprocessor with reduced complexity. *Solid-State Circuits, IEEE Journal of*, 32(7):1056–1066, Julio 1997.

[16] M. J. Flynn and P. Hung. Microprocessor design issues: thoughts on the road ahead. *Micro, IEEE*, 25(3):16–31, Mayo-Junio 2005.

[17] S. Furber. *ARM: System-on-Chip Architecture*. Addison-Wesley, Harlow, second edition, 2000.

[18] S. Guccione. List of fpga-based computing machines. 2007(9 de Mayo), 21 de Agosto 2000. http://www.io.com/∼guccione/HW_list.html.

[19] M. Gumm. *VLSI Design Course Notes: VHDL-Modeling and Synthesis of The DLXS RISC Processor*. University of Stuttgart, Germany, Diciembre 1995.

[20] J. L. Hennessy and D. Patterson. *Computer Architecture: A quantitative approach*. Morgan Kaufmann, San Mateo, 1990.

[21] V. P. Heuring, H. F. Jordan, and M. Murdocca. *Computer Systems Design and Architecture*. Addison Wesley Longman, Menlo Park, CA, 1997.

[22] V. P. Heuring, H. F. Jordan, and M. Murdocca. *Computer Systems Design and Architecture*. Pearson Prentice Hall, Upper Saddle River, second edition, 2004.

[23] A. Kim, G. R. Weistroffer, D. M. Grammer, and R. H. Klenke. The vcu src ii: a full-custom vlsi 32-bit risc processor. *University/Government/Industry Microelectronics Symposium, 2001. Proceedings of the Fourteenth Biennial*, pages 201–204, 17-20 Junio 2001.

[24] I. Kuon and J. Rose. Measuring the gap between fpgas and asics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):203–215, Febrero 2007.

[25] I. Mezei and V. Malbasa. Formal specification of an fpga based educational microprocessor. *Microelectronics, 2002. MIEL 2002. 23rd International Conference on*, 2:667–670, 12-15 Mayo 2002.

[26] V. Milutinovic, M. Bettinger, and W. Helbig. Multiplier/shifter design tradeoffs in a 32-bit microprocessor. *Computers, IEEE Transactions on*, 38(6):874–880, Junio 1989.

[27] D. Patterson. Reduced instruction set computers. *Commun. ACM*, 28(1):8–21, Junio 1985.

[28] L. Ribas and D. Rodríguez. jsyasp: Un simulador de un procesador educativo en javascript. 2005.

Resum

Aquest projecte té com objectiu l'anàlisi de prestacions de processadors RISC de baix cost i el disseny d'un processador RISC simple per a aplicacions de propòsit general relacionades amb l'adquisició i el procés simple de dades. Com resultat es presenta el processador SR3C de 32 bits i arquitectura RISC. Aquest processador s'ha descrit i simulat mitjançant el llenguatge de descripció de hardware VHDL i s'ha sintetitzat en una FPGA. El processador està preparat per a poder utilitzar-se en SoCs reals gràcies al compliment de l'estàndard de busos Wishbone. A més també es pot utilitzar com plataforma educativa gràcies a l'essamblador i simulador desenvolupats.

Resumen

Este proyecto tiene como objetivo el análisis de prestaciones de procesadores RISC de bajo coste y el diseño de un procesador RISC simple para aplicaciones de propósito general relacionadas con la adquisición y el proceso simple de datos. Como resultado se presenta el procesador SR3C de 32 bits y arquitectura RISC. Dicho procesador se ha descrito y simulado mediante el lenguaje de descripción de hardware VHDL y se ha sintetizado en una FPGA. El procesador está preparado para poder utilizarse en SoCs reales gracias al cumplimiento del estándar de buses Wishbone. Además también se puede utilizar como plataforma educativa gracias al ensamblador y simulador desarrollados.

Abstract

The objective of this project is to analyze the efficiency of the low cost RISC processors, and realize the design of a simple RISC processor which is related with the general adquisition and the simple data process. This project presents the RISC architecture of the 32 bits SR3C processor. This processor has been described, simulated and synthesizd in a FPGA by VHDL language. Thanks to the compliment of the Wishbone bus standard, this processor can be used in real SoCs. In addition it also can be used like education platform thanks to the developed software of assembler and simulator.