

Implementación en VHDL de un CORE compatible con el microprocesador ARM, usando una arquitectura *pipelined*.

F. J. Jurado Carmona, J. N. Tombs, M. A. Aguirre y A. Torralba

Departamento de Tecnología Electrónica, Escuela Superior de Ingenieros, Universidad de Sevilla.

frasko1978@yahoo.es, jon@gte.esi.us.es

Resumen—En el presente artículo se describe la implementación de un CORE compatible con el juego de instrucciones de los procesadores ARM8/9. La arquitectura utilizada es similar a la ARMv4 y el diseño ha sido descrito usando VHDL. El corazón del core es una estructura completamente *pipeline* de 5 etapas, lo cual aumenta notablemente la velocidad. Se realizó una validación exhaustiva del diseño combinando la herramienta VHDL-Simili [1] con otras aplicaciones creadas exclusivamente para tal fin, generando así un entorno de simulación apropiado para un diseño de estas características, y que además resultó ser bastante flexible. Tras finalizar con éxito la fase de simulaciones, se realizó un prototipo de pruebas utilizando la FPGA Virtex XC800 de Xilinx, junto con un periférico especialmente concebido para poder desarrollar el test de campo. Por último, el diseño fue sintetizado usando la librería para standard cell AMS 0.35u, y se creó el layout de la mega-celda.

Palabras clave—VHDL, ARM, pipeline, FPGA.

I. INTRODUCCIÓN

En los últimos años han proliferado los diseños microelectrónicos que utilizan tecnologías basadas en FPGAs o ASICs. La tendencia ahora es describir el hardware utilizando lenguajes de alto nivel tales como VHDL o Verilog, de modo que el diseñador tiene que concebir primero la arquitectura, para posteriormente poder describir su comportamiento. El proceso de diseño a alto nivel ha generado nuevos mercados de IPs, o los orientados a la venta de celdas complejas ya diseñadas y verificadas, que pueden ser incorporadas dentro de la jerarquía del diseño como celdas de librería [2].

Por otro lado, la familia de procesadores ARM [3] ha experimentado en los últimos años una vertiginosa evolución, no sólo tecnológica, sino también comercial.

Estos factores, unidos a experiencias anteriores [4] con este tipo de dispositivos han motivado el desarrollo del proyecto que nos disponemos a presentar.

II. PROCESADOR ARM CON ARQUITECTURA ARMv4

Las siglas ARM hacen referencia a un microprocesador de la familia RISC avanzado

(Advanced RISC Machine). Los procesadores basados en la arquitectura ARMv4 [6] implementan el ISA ARM estándar, y cuentan con los siguientes elementos: banco de 37 registros de 32 bits; ALU con operandos y resultado en 32 bits, con desplazamiento de operando en serie; multiplicador de 32x32 bits y resultado en 64 bits; modo *user* y 6 modos privilegiados de funcionamiento (uno de ellos para el usuario); excepciones atendidas en virtud de su prioridad: Reset, Abort (fallo de memoria), FIQ (Fast Interrupt reQuest), IRQ (Intererupt ReQuest), SWI (interrupción software) y *undefined*. Existe la posibilidad de elegir el formato de numeración de memoria entre big-endian o little-endian mediante un pin externo, y también la opción de introducir un número variable de ciclos de espera (entre 0 y 3) en el acceso a memoria simplemente configurando dos pines externos.

III. ESTRUCTURA PIPELINE

El principio básico en una arquitectura de este tipo radica en el hecho de que no se espera a que una instrucción haya escrito el resultado de su operación en el registro destino para cargar una nueva instrucción en el CORE, sino que en un instante dado, existirán tantas instrucciones activas como etapas tenga la estructura *pipeline*. De este modo, en régimen permanente, una instrucción es cargada en el procesador y otra finaliza su procesamiento, para cada ciclo de reloj. Así, en una *pipeline* de d etapas, una instrucción requerirá d ciclos para completar su ejecución, pero existirán d instrucciones procesándose concurrentemente, con lo que a efectos prácticos, se ejecuta una instrucción en cada ciclo de reloj (en cada ciclo de reloj, la instrucción que ha alcanzado la última etapa de la *pipeline* escribirá el resultado de su procesamiento en el registro correspondiente)

El comportamiento descrito está directamente relacionado con el parámetro *speedup*, que nos da idea de la mejora en la velocidad de ejecución de un conjunto de instrucciones en un procesador *pipelined*, con respecto al tiempo que tardarían en ejecutarse en un procesador de estructura simple. Si ambos tipos de procesador utilizan la misma tecnología para su implementación, el valor del *speedup* se puede definir como:

$$speedup = \frac{CPI_{no\ pipelined} \times t_{cyc}^{no\ pipelined}}{CPI_{pipelined} \times t_{cyc}^{pipelined}} = \frac{CPI_{no\ pipelined}}{CPI_{pipelined}}$$

Suponiendo una situación ideal, un conjunto de n instrucciones en un procesador *no pipelined* requerirá de un total de $n \cdot d$ ciclos, mientras que en un procesador completamente *pipelined* necesitará únicamente $n+d$ ciclos de reloj. Normalmente, $n \gg d$, de modo que podemos expresar el parámetro *speedup* como:

$$speedup = \frac{n \cdot d}{n + d} \approx d$$

En la práctica, la situación no es tal ideal, pues se producen fenómenos tales como el *interlocking* entre registros. En la Figura 1 se ilustra este hecho: en el diagrama están representadas las cinco etapas que componen la estructura *pipeline*, IF: Instruction fetch, DE: decode, EX: execute, ME: memory, Wr: write. En el quinto ciclo la instrucción i_3 se encuentra en la etapa EX, pero su resultado no será escrito en el registro destino hasta alcanzar la etapa Wr; imaginemos ahora que la instrucción que le sigue, i_4 , requiere como operando al registro destino de la instrucción i_3 , que no será actualizado hasta que transcurran dos ciclos de reloj. Se ha producido lo que se denomina *interlocking* de registros en el sistema, con lo que la instrucción i_4 quedará estancada en la etapa DE hasta que i_3 finalice su ejecución, introduciéndose en las etapas sucesivas ciclos de no operación, denominados *bubbles* hasta que i_3 abandone la estructura.

Ciclo	IF	DE	EX	ME	Wr
1	i_1				
2	i_2	i_1			
3	i_3	i_2	i_1		
4	i_4	i_3	i_2	i_1	
5	i_5	i_4	i_3	i_2	i_1
6	i_5	i_4	●	i_3	i_2
7	i_5	i_4	●	●	i_2
8	i_6	i_5	i_4	●	●

Fig. 1. Interlocking entre registros

Otra situación que reduce el *speedup* de un procesador *pipelined* es la ejecución de una instrucción de salto. La ejecución de este tipo de instrucciones es condicional, de modo que éstas no pueden ser detectadas hasta la etapa EX; una vez que se confirma su ejecución, las etapas IF y DE deben ser “vacías”. En la Figura 2 se ilustra este hecho: en el 4º ciclo de reloj, i_2 alcanza la etapa EX y se confirma que se trata de una instrucción de salto que debe ser ejecutada. La instrucción que debe seguir a i_2 es la dirección objetivo del salto (en el ejemplo, i_a), y no i_3 . Este es el motivo de que las etapas

IF y DE sean inicializadas y se introduzcan *bubbles* en las que corresponda. En el caso de las excepciones el comportamiento es bastante similar, siendo el diagrama temporal idéntico a la Figura 2.

Ciclo	IF	DE	EX	ME	Wr
1	i_1				
2	i_2	i_1			
3	i_3	i_2	i_1		
4	i_4	i_3	i_2	i_1	
5	i_a	●	●	i_2	i_1
6	i_b	i_a	●	●	i_2
7	i_c	i_b	i_a	●	●
8	i_d	i_c	i_b	i_a	●

Fig. 2. Ejecución de la instrucción de salto

Todas las situaciones anteriores reducen notablemente el valor efectivo del parámetro *speedup*. Además, existen otros factores tales como la introducción de ciclos de espera adicionales en el acceso a memoria o las instrucciones que requieren más de un ciclo de reloj por etapa, que reducen aún más el rendimiento.

IV. IMPLEMENTACIÓN

El alcance de este proyecto abarca el diseño de un CORE compatible con el procesador ARM, utilizando una arquitectura del tipo ARMv4. Esto implica que deberá ser capaz de decodificar el ISA ARM estándar, a excepción de dos instrucciones que fueron excluidas de la implementación por diversos motivos: en primer lugar, la instrucción Block Data Transfer, no fue implementada porque requiere un número variable de ciclos para su ejecución (esto rompe la naturaleza RISC del juego de instrucciones ARM, y por tanto no es fácil de implementar en nuestra estructura *pipeline*); en segundo lugar, las instrucciones de coprocesador fueron desechadas por no disponer de la implementación de un coprocesador adecuado. Además, ambas instrucciones pueden ser emuladas vía software a través de la rutina de atención a la excepción *undefined*.

Desde el principio se optó por un diseño moderno y eficiente, por lo que la estructura del CORE es completamente Harvard (nótese que hasta el procesador ARM7 se venía utilizando la estructura Von Neumann). En lo que se refiere a la arquitectura interna, como se trata de diseñar una máquina RISC, y teniendo en cuenta las características del ISA ARM, la arquitectura *pipeline* surge como la más adecuada para este tipo de máquinas (junto con otras estructuras paralelo). En lo que se refiere al número de etapas, se optó por utilizar una *pipeline* de 5 etapas, en lugar de las tradicionales estructuras de 3 ó 4 etapas. Esta decisión fue motivada por dos factores, fundamentalmente: en primer lugar,

resultados experimentales indican que el número de etapas que maximiza la frecuencia obtenida en la implementación en una FPGA de una estructura de estas características viene dada por la expresión $\log_2(\text{ancho del bus})$ [7]; en nuestro caso todos los buses son de 32 bits, y por tanto el óptimo de etapas es 5. Por otro lado, está el hecho de que los procesadores comerciales ARM8/9 utilizan estructuras *pipeline* de 5 etapas.

En lo que se refiere al compromiso velocidad-área, se opta por sacrificar área en pro de una mejora del *speedup*. Nótese que esta restricción no es tan fuerte como puede parecer, pues difícilmente un diseño de semejantes características (buses de 32 bits) podrá alcanzar una implementación óptima sobre una FPGA. En cambio, ciertas mejores estructurales pueden lograr un incremento notable en la velocidad.

Antes de comenzar la implementación, es necesario desarrollar una estructura base capaz de soportar el juego de instrucciones ARM; con esto nos estamos refiriendo a los buses y registros de 32 bits necesarios para cursar y almacenar datos e instrucciones, lógica de control de acceso a registros, contador de programa (PC), etc. En segundo lugar, será necesario especificar la arquitectura de una ALU que permita realizar todas las operaciones requeridas por cualquiera de las instrucciones que componen el ISA ARM. Esta ALU es bastante compleja, pues requiere de 3 buses de entrada de 32 bits (A, B y C), y además incluye un *shifter* que actúa sobre el operando B, de modo que el desplazamiento se realiza en serie con la operación aritmética o lógica. Existen un total de 16 operaciones distintas que se pueden combinar con cualquier tipo de rotación sobre el operando B. La ALU también contiene un multiplicador capaz de realizar multiplicaciones con acarreo usando operandos de 32 bits, y resultado en 64. El hecho de que la lectura del resultado de este tipo de multiplicaciones requiera de 2 ciclos de reloj, complica la estructura *pipeline* obligándonos a introducir una instrucción “fantasma” en la etapa *execute* tras la ejecución de una instrucción de multiplicación de 64 bits.

El multiplicador del diseño fue implementado en un archivo separado, permitiendo así el uso de multiplicadores precompilados, mega-celdas que contengan multiplicadores, o simplemente ofreciendo la posibilidad de deshabilitar esta capacidad.

Llegados a este punto, ya es posible implementar las cinco etapas que componen la estructura *pipeline*. En la Figura 3 se muestra un diagrama simplificado de la estructura utilizada. Una etapa no es más que un conjunto de unidades funcionales, y cada unidad funcional deberá realizar su parte correspondiente en el procesamiento de una instrucción. Cada etapa ha sido diseñada intentando reducir al máximo la complejidad de la siguiente. Para ello en la etapa de decodificación de la instrucción se genera una “configuración” de 32 bits que se propagará de una etapa a otra a través de la *pipeline*, y cuando se encuentra en una etapa concreta, activa/desactiva las unidades funcionales necesarias para procesar la instrucción. De este modo, todo el proceso de decodificación se realiza en la etapa adecuada (DE: decode), en la cuál se decide qué unidad funcional será necesario activar en cada etapa, generando así el patrón de configuración correcto. Una vez que el ISA ARM es descompuesto en funcionalidades, y las unidades funcionales correspondientes han sido implementadas, añadir instrucciones no es más que incorporar nuevos patrones de configuración a la etapa *decode*.

Para finalizar el diseño, es necesario implementar una estructura de control que permita atender las excepciones que puedan producirse, y también un interfaz de memoria lo suficientemente flexible para poder adaptar al core a cualquier sistema de estructura desconocida. Las especificaciones de esta interfaz son bastante claras para que el futuro diseñador pueda adecuar su sistema de memoria a nuestra interfaz de forma sencilla.

Una descripción más detallada de la arquitectura completa del diseño puede encontrarse en [5].

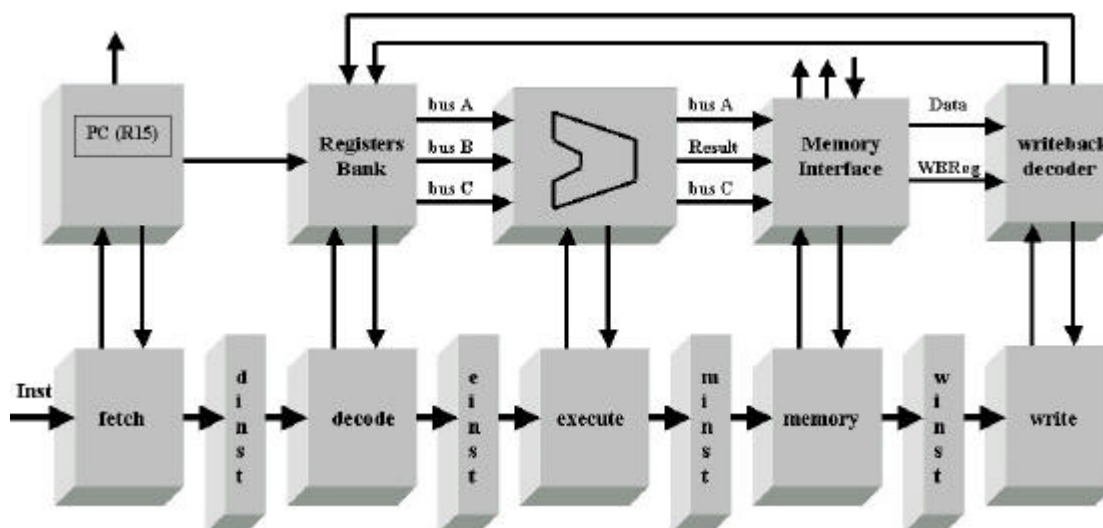


Fig. 3. Diagrama de la arquitectura implementada

V. SIMULACIONES

La simulación del diseño es quizá la fase más importante en un proyecto de esta envergadura, y al mismo tiempo es la más crítica y compleja. Obviamente, se trata de un diseño difícil de monitorizar, y tratar de reducir el espacio de errores a su mínima expresión es una tarea ardua, por lo que hay que encontrar un método que permita automatizar el proceso de verificación.

La solución adoptada se basa en la herramienta de libre distribución VHDL-Simili. Esta herramienta es capaz de realizar una compilación de la descripción VHDL y una posterior ejecución del diseño compilado en un tiempo bastante reducido. Los resultados no serán tan fidedignos como los obtenidos con una simulación a nivel de puertas, pero tiene la ventaja de no requerir una síntesis previa del diseño, lo cual se traduce en velocidad y flexibilidad en las simulaciones.

Existe además una librería que permite utilizar una serie de sentencias que no son sintetizables, pero que nos permiten manejar ficheros en disco; de este modo, se puede incorporar al diseño un banco de pruebas descrito en VHDL, y escribir el resultado de la simulación en ficheros.

El proceso de simulación es el siguiente: se escribe la secuencia de instrucciones que se desea verificar, codificadas en hexadecimal, en un fichero de texto. Mediante la herramienta *Txt2vhd.exe*, creada exclusivamente para este proyecto, se convierte el fichero de texto anterior en un modelo VHDL de una ROM. Utilizando un proceso por lotes, la ROM es compilada junto con la descripción completa del diseño y las librerías correspondientes, y posteriormente es ejecutada, todo ello utilizando VHDL-Simili. El banco de pruebas incorporado al diseño será el encargado de gobernar el reloj del sistema y de almacenar los valores de todos los registros, buses internos y señales de control en tres ficheros, denominados “traza.dat”, “pipeline.dat” y “memory.dat”.

En la Figura 4 se muestra el flujo completo de simulación. Los ficheros de salida recogen, ciclo a ciclo, toda la información necesaria para verificar la correcta ejecución del código de prueba contenido en la ROM, junto con información detallada del estado de cada etapa de la estructura *pipeline*.

Una de las mayores ventajas de nuestra estrategia de diseño es que desde que se implementa la primera instrucción, la estructura *pipeline* es totalmente verificable, de modo que cada vez que una instrucción es implementada, puede depurarse el diseño solventando errores conceptuales y evaluando las situaciones críticas (ej. interrupciones que llegan durante un proceso de *interlocking*, o cualquier combinación de circunstancias extremas con la introducción de ciclos de espera adicionales en el acceso a memoria, etc). Esta particularidad, unida al hecho de que la adición de nuevas unidades funcionales no altera como norma generar el comportamiento de las ya existentes, facilita notablemente la depuración del diseño.

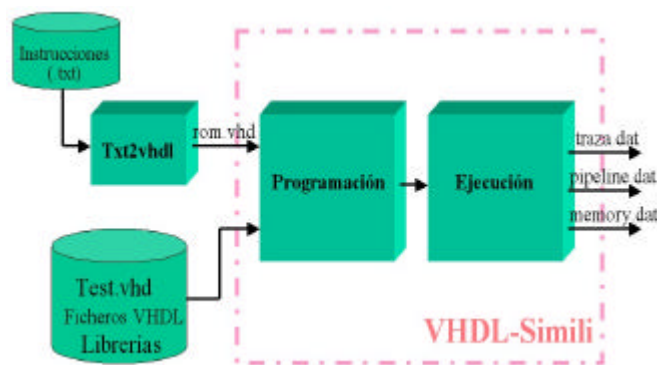


Fig. 4. Flujo de Simulación

VI. LAYOUT DE UNA MEGA-CELDA DEL PROCESADOR. OPTIMIZACIONES

El proyecto estaba originalmente orientado al diseño de un CORE implementado en una FPGA, pero debido a que VHDL es independiente de la tecnología, resulta sencillo generar una descripción del diseño en standard-cell (ASIC). Por ello, el diseño fue sintetizado a standard-cell utilizando SYNOPSIS, y el layout fue generado usando Cadence Silicon Ensemble.

El diseño fue implementado utilizando la tecnología AMS 0.35u de 3 metales, obteniendo una mega-celda de 2.4x2.4 mm² (sin multiplicador), capaz de operar a una frecuencia de 30 MHz.

Tras un análisis de los resultados obtenidos, se pudo observar que la frecuencia máxima del diseño estaba limitada por el retardo aportado por la ALU; en particular, por aquellas instrucciones que hacen uso del *shifter* en serie con una operación de suma. Se puede alcanzar una mejora de la frecuencia máxima de funcionamiento realizando una modificación estructural en el diseño: como el operando B está ya disponible en la etapa *decode*, existe la posibilidad de realizar el desplazamiento en dicha fase. Es decir, se puede extraer el *shifter* de la ALU e introducirlo en la etapa *decode*, de modo que en el bus B se cargue el operando ya desplazado. El incremento obtenido en la frecuencia máxima de funcionamiento tras realizar estos cambios estuvo en torno al 15%.

VII. PROTOTIPO DE PRUEBAS

Para poder concluir el ciclo de diseño, y teniendo en cuenta las dimensiones del mismo, es necesario elaborar un prototipo de pruebas que posibilite la realización de un test de campo adecuado, capaz de respaldar los resultados aportados por las simulaciones.

Como el CORE generado no dispone de ningún dispositivo de entrada/salida, se diseñó un periférico mapeado en memoria consistente en una pantalla compuesta por cuatro displays de 7 segmentos, controlada por un bus de 12 bits. Los cuatro bits más significativos se utilizaron para seleccionar el display activo en un instante determinado, y el resto es capaz de activar o desactivar individualmente cualquiera de los 7

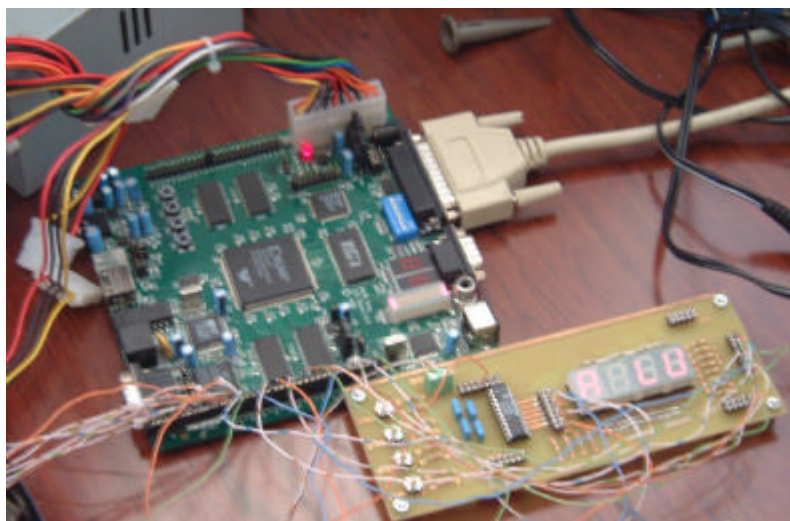


Fig. 5. Prototipo en Funcionamiento

segmentos que componen dicho display más el punto decimal. Se escribió un programa en código máquina de ARM para refrescar cada elemento de la pantalla con una frecuencia elevada, de modo que un mensaje deslizante aparecía desplazándose lentamente de izquierda a derecha, dando así la sensación de ser un texto circular.

En la Figura 5 se muestra el sistema completo en funcionamiento. El diseño fue implementado utilizando el 50% de la FPGA Virtex XC800 de Xilinx en la placa de evaluación Xess. El programa que fue ejecutado por el procesador fue concebido para forzar la utilización del mayor número de instrucciones diferentes, y así poder verificarlas en un entorno real. Los resultados fueron completamente satisfactorios.

VIII. CONCLUSIONES Y TRABAJOS FUTUROS

Se ha conseguido diseñar con éxito un CORE compatible con el microprocesador ARM, utilizando para ello una estructura completamente *pipelined*. El diseño ha sido verificado y se ha demostrado su funcionamiento mediante la creación de un prototipo de pruebas.

Este diseño [5] estará disponible bajo licencia pública de libre distribución, en la siguiente URL: <http://www.gte.us.es/~jon/PFCS>

En el futuro, se realizarán trabajos que incluyan un estudio detallado del retardo máximo, aportando propuestas que permitan una mejora de la frecuencia máxima de funcionamiento.

Además, se implementará un coprocesador, y se dotará al ARM de una interfaz con el mismo y de las instrucciones correspondientes, así como también se buscará una forma eficiente de implementar la instrucción BDT.

IX. REFERENCIAS

- [1] SymphonyEDA, VHDL Simili V1.0 Documentation <http://symphonyeda.com/doc/simulimanual.pdf>
- [2] W. Hobbs "Model Availability, Portability and Accuracy, An IC Vendor's Perspective Efforts and ision for the Future" Workshop on Libraries, Component Modeling and quality Assurance, Nantes, April, 1995
- [3] Advanced RISC Machines Ltd. <http://www.arm.com>
- [4] J.A. Zafra Costa. "Implementación en VHDL de los microcontroladores PIC-16/17". Proyecto Fin de Carrera, Escuela Superior de Ingenieros de Sevilla. Mayo 2000
- [5] F.J. Jurado Carmona "Implementación en VHDL del microprocesador ARM9" . Proyecto Fin de Carrera, Escuela Superior de Ingenieros de Sevilla. Abril 2002.
- [6] D. Jagger "ARM Architecture Reference Manual", Morgan Kaufmann Publishers, 2nd edition, ISBN: 0201737191, 2000
- [7] Xilinx, "X-Files – News from the Leading Provider of FPGA and Desktop ASIC" 2000