Ambiente de Desarrollo para el Diseño de Coprocesadores en Arquitectura Sparc V8-LEON2

Etcheverry L., Pérez Acle J., Oliver J.P.

Dpto. Electrónica, IIE, Facultad de Ingeniería, Universidad de la República, Uruguay
{letcheve, julio, jpo}@fing.edu.uy
http://iie.fing.edu.uy

Abstract. Se presenta un ambiente de desarrollo para el diseño de coprocesadores para un procesador de código abierto: el procesador LEON2. Se diseñó la Unidad de Ejecución del coprocesador en forma modular para permitir incorporar nuevas operaciones a través de una interfaz sencilla. Con la integración hardware resuelta, el usuario de este ambiente puede concentrarse en el diseño del hardware específico y en las modificaciones al software necesarias para hacer uso de llamadas al coprocesador. El desarrollo incluye toda la conexión al LEON2 y un banco de pruebas que permite simular a nivel RTL al procesador completo, haciendo uso de las instrucciones del coprocesador. El diseño fue probado y validado en hardware con aplicaciones reales. El ambiente de desarrollo fue utilizado por terceros con buenos resultados desde el punto de vista de la curva de aprendizaje.

1 Introducción

En el desarrollo de aplicaciones embebidas hay una permanente demanda por aumentar la velocidad de ejecución, bajo restricciones de costo y consumo. Existen múltiples métodos para aumentar el desempeño de aplicaciones ejecutando en un procesador de propósito general. Uno de ellos consiste en la utilización de bloques hardware especializados para realizar en forma rápida determinada operación o algoritmo. La utilización de FPGAs para alojar procesadores facilita enormemente el desarrollo de hardware a medida permitiendo adaptar la arquitectura de hardware al problema particular.

Sin pretender realizar una taxonomía exhaustiva, veremos varias alternativas para integrar a un procesador hardware diseñado a medida. Una primera opción consiste en conectar el bloque hardware específico a través del bus de acceso a memoria (o entrada/salida) y accederlo mediante las instrucciones estándar de transferencia de datos del procesador.

Algunas arquitecturas de procesadores tienen previstas interfaces dedicadas para la conexión de uno o más módulos coprocesadores. Hay múltiples ejemplos de estas interfaces como el coprocesador matemático en las arquitecturas x86-x87 de Intel y el soporte para coprocesadores en la arquitectura SPARC-V8 [1].

Por último, otra alternativa consiste en modificar internamente al procesador agregando bloques de hardware específicos para extender el repertorio de instrucciones que es capaz de ejecutar [2], [3], [4].

Cada una de las alternativas presentadas tiene ventajas y desventajas respecto a las otras. La primera de ellas claramente es la que permite la mejor utilización con diferentes procesadores y es aquella en que la integración a nivel de hardware es más sencilla. Tanto la primera como la segunda permiten procesamiento concurrente entre el procesador principal y el hardware dedicado, y además posibilitan la utilización de relojes independientes. La solución de extender el repertorio de instrucciones permite un acceso más rápido al banco de registros del procesador evitando cuellos de botella. Sin embargo se debe ser cuidadoso en el diseño del bloque de hardware en lo que respecta a los retardos del circuito para evitar degradar el rendimiento de todo el sistema.

Independientemente de cual sea la alternativa elegida, cuando un conjunto de operaciones es utilizado en forma frecuente, el desarrollo de los bloques a medida puede ser reutilizado por diferentes aplicaciones. Esto a menudo culmina en el desarrollo de bibliotecas o paquetes que incluyen los bloques hardware a medida, su integración al procesador y las versiones personalizadas de los compiladores y herramientas de desarrollo. La reutilización de estos paquetes disminuye dramáticamente el tiempo de desarrollo de nuevas aplicaciones. Por lo general estas extensiones están orientadas a un determinado dominio de aplicación, llegando en algunos casos a incorporarse en procesadores de propósito general como por ejemplo las extensiones MMX en procesadores Intel [5]. Otros ejemplos pueden encontrarse en criptografía [6], [7], [8], compresión de audio [9] y operaciones matemáticas [10].

Existen casos en los que la aplicación bajo desarrollo tiene requerimientos de velocidad de ejecución que obligan a incorporar hardware dedicado pero las operaciones involucradas no están cubiertas por paquetes disponibles. En ese caso se hace necesario el desarrollo de hardware a medida y su integración al procesador. De lo fácil o difícil que sea dicha integración depende muchas veces que una solución de este tipo sea viable o no.

Si se dispone de interfaces claras y bien documentadas, y de un ambiente de desarrollo que permita reutilizar todo lo que sea independiente de la aplicación, entonces el esfuerzo de diseño puede concentrarse en los aspectos específicos de la aplicación, disminuyendo de esta forma sensiblemente los tiempos y costos de desarrollo. La falta de este tipo de herramientas puede hacer poco atractiva la utilización de soluciones con hardware dedicado. Un ejemplo de esto fue la supresión en la arquitectura SPARC-V9 [11] del soporte para coprocesadores existente en arquitecturas previas, motivado por la falta de adopción por parte de la comunidad de usuarios[1].

Existen productos comerciales que permiten integrar en forma simple bloques de hardware a medida, como por ejemplo el ambiente de desarrollo para el procesador NIOS de Altera [2].

El objetivo del presente trabajo es alcanzar una solución que simplifique la incorporación de hardware específico para acelerar aplicaciones sobre un procesador de código abierto (Open Source). Con este fin se desarrolló un ambiente para el diseño de coprocesadores para el procesador LEON2 [12], que es una implementación de la arquitectura SPARC-V8 bajo licencia LGPL. Existen numerosos trabajos reportados que incorporan bloques de hardware diseñado a medida para extender el repertorio de instrucciones del procesador LEON2, al estilo de la última de las alternativas presentadas al inicio de esta introducción [3], [6], [7], [8]. La solución propuesta en este trabajo en

---

1 "The coprocessor opcodes were eliminated because they have not been used in SPARC-V7 and SPARC-V8, as witnessed by the lack of coprocessor implementations" cita textual del manual de SPARC-V9.

cambio, hace uso de la interfaz de coprocesador genérico del LEON2, al estilo de la segunda de las alternativas planteadas. Hasta donde es de conocimiento de los autores existen pocos trabajos reportados [9], [10], [13] utilizando la interfaz de coprocesador de LEON2, y se trata en estos casos de soluciones ya cerradas y no de un ambiente para el desarrollo de nuevos coprocesadores como es el presente trabajo.

El resto del artículo se organiza como sigue: en el capítulo 2 se presenta el soporte para coprocesadores definido a nivel de conjunto de instrucciones en la arquitectura SPARC-V8 y las previsiones para su implementación incluidas en el procesador LEON2 estándar. En el capítulo 3 se describe nuestra implementación y las pruebas de validación realizadas. Finalmente en el capítulo 4 se presentan las conclusiones y algunas líneas de trabajo futuro.

2 Interfaz de Coprocesador en la Arquitectura SPARC-LEON2

2.1 Coprocesador en Arquitectura SPARC-V8

La especificación de la arquitectura SPARC-V8 [1] define lógicamente al procesador como el conjunto formado por una Unidad de Enteros (IU), una Unidad de Punto Flotante y un Coprocesador (CP) opcional, cada uno con sus propios registros. Dado que SPARC es una arquitectura de conjunto de instrucciones (ISA) la especificación se limita a definir las instrucciones que interactúan con el coprocesador. La arquitectura de buses, conexiones y el diseño hardware propiamente dicho deben ser definidos por la implementación. Las instrucciones que dan soporte al coprocesador son las siguientes:

- Instrucciones *load* y *store* para mover datos entre memoria y registros del coprocesador.
- Instrucciones *cpop1* y *cpop2* para ejecutar una operación entre registros del coprocesador.

Se asume que el coprocesador cuenta con un conjunto de hasta 32 registros de 32 bits, que pueden ser referenciados tanto individualmente como en pares. El poder referirse a pares de registros, permite al coprocesador operar sobre datos de 64 bits.

Las instrucciones *cpop1* y *cpop2* difieren entre ellas en la forma en que afectan a las banderas (*coprocessor condition codes*). La sintaxis de las instrucciones en lenguaje ensamblador es la siguiente:

```
cpop1 opc, cregrs1, cregrs2, cregrd
cpop2 opc, cregrs1, cregrs2, cregrd
```

donde el primer parámetro *opc* indica cuál operación en particular debe ser realizada y los demás identifican a los registros origen (*cregs1* y *cregs2*) y destino de la operación. Estos son siempre registros del coprocesador, las instrucciones *cpop* no pueden acceder directamente a memoria ni a los registros del procesador, obligando a transferir los datos de entrada y los resultados finales mediante instrucciones *load* y *store*. El conjunto de operaciones posibles y su correspondencia con los valores del parámetro *opc* son definidas en cada implementación particular.

Dado que la micro-arquitectura y la organización hardware del procesador no está dictada por el estándar SPARC-V8, la interconexión depende de la implementación de LEON2 y se verá en el próximo apartado.

2.2 Interfaz genérica del coprocesador de LEON

El procesador LEON2, brindado en forma de código VHDL sintetizable, es un bloque flexible y ampliamente configurable. Una de sus opciones de configuración permite la interconexión con una unidad genérica de coprocesamiento, cuyo objetivo es el de realizar ciertas operaciones en un hardware especializado diseñado para ese fin. La interconexión del procesador principal con la unidad de coprocesamiento se logra a través de una interfaz compuesta por señales de datos y control que permite el envío de instrucciones al coprocesador, así como el intercambio de datos entre registros de éste último y memoria principal. Esta interfaz permite al coprocesador iniciar la ejecución de una nueva instrucción en cada ciclo de reloj y su operación en paralelo con el procesador principal siempre que no exista dependencia entre datos [12].

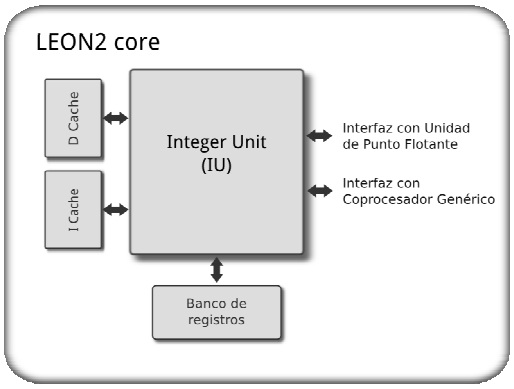

Fig. 1. Interfaces de coprocesador de LEON2

LEON2 presenta dos instancias idénticas de esta Interfaz de Coprocesador. La primera es utilizada para comunicar el procesador principal con la Unidad de Punto Flotante FPU si es que está presente. La segunda permite la incorporación de un coprocesador genérico como se mencionó anteriormente. El hecho de que ambas interfaces sean idénticas es importante, ya que permitió reutilizar gran parte del código de la implementación de la FPU como se describe en el capítulo siguiente.

El análisis de la arquitectura de la FPU es de interés, ya que es un ejemplo funcional de una unidad conectada a través de una *Interfaz de Coprocesador*. La FPU cuenta con un banco de registros de 32 bits, una Unidad de Ejecución y lógica de control. La lógica de control implementa un *pipeline* de cinco etapas; y su función es manejar el banco de registros así como despachar instrucciones y datos a la unidad de ejecución para realizar las distintas operaciones. La interconexión entre la lógica de control y la unidad de ejecución se lleva cabo a través de una *Interfaz de Unidad de Ejecución*, descrita brevemente en la documentación de LEON2[2]. Un análisis más detallado de la arquitectura de la FPU puede encontrarse en [14].

Esta interfaz se usa para darle a la unidad de ejecución un par de operandos de 64 bits cada uno, así como un código de operación (*opcode*) que indique la instrucción a ejecutar. La unidad de ejecución devuelve a través de esta misma interfaz, un resultado de 64 bits,

2 En el manual, esta interfaz está bajo la denominación *Generic Coprocessor Interface*.

así como códigos de condición y banderas de excepciones. Los códigos de condición pueden ser utilizados en el procesador principal para controlar el flujo del programa de acuerdo al resultado de las instrucciones ejecutadas por la FPU. Además, en caso en que la unidad de ejecución necesite un número variable de ciclos de reloj para completar la instrucción, la interfaz incluye también un par de señales de control (*handshaking*): *start* y *busy*.

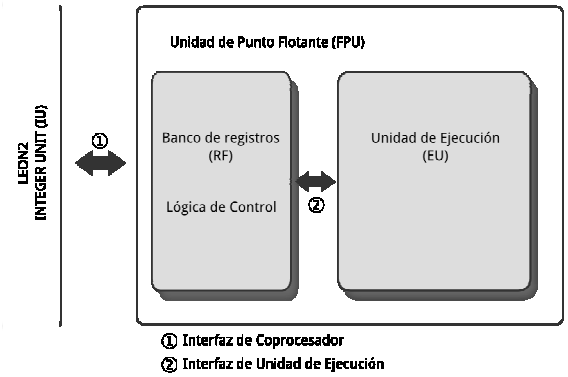

Fig. 2. Diagrama de bloques de la Unidad de Punto Flotante

Cuando la lógica de control de la FPU intenta despachar una instrucción a la unidad de ejecución, comienza por afirmar la señal *start* junto con el código de operación a ejecutar.

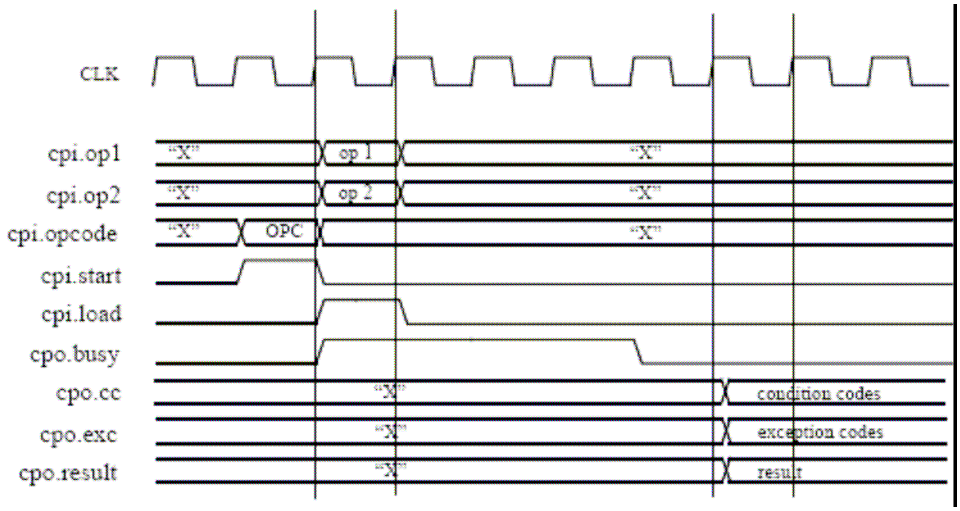

Fig. 3. Interfaz de Unidad de Ejecución (figura extraída del manual de LEON2)

Los operandos son aceptados por la unidad de ejecución en el próximo flanco de reloj, siempre que la señal *load* afirme la validez de los datos. Si la ejecución de la instrucción por la unidad de ejecución toma más de un ciclo, ésta debe mantener la señal *busy* activa hasta un ciclo antes a aquél en el cual el resultado y banderas son válidos. La Fig. 3 muestra

la evolución de las señales de la interfaz para el caso de una instrucción de varios ciclos de duración.

3 Arquitectura de la solución

La solución propuesta se basa en dos premisas:

1. Construir sobre el código existente, reutilizando en lo posible la lógica de control, manejo de registros y *pipeline* presente en la Unidad de Punto Flotante de LEON2
2. Diseñar una nueva unidad de ejecución que permita agregar, de forma simple y modular, nuevas operaciones al coprocesador.

Como describe el capítulo anterior, la FPU de LEON2 incluye una parte importante de la lógica necesaria para manejar los registros de la nueva unidad coprocesadora, así como el despacho de instrucciones a la unidad de ejecución. Además, cuenta con una interfaz de Unidad de Ejecución bien definida y especificada. El nuevo diseño (Fig. 4) mantiene casi toda la funcionalidad de la lógica de control de la FPU salvo por el cambio de los códigos de operación a los reservados para las instrucciones *cpop* y por algunos ajustes de resultados propios de las operaciones en punto flotante. En cambio, la unidad de ejecución se sustituye por una nueva entidad, a la que se conectan nuevos módulos, llamados Subunidades de Ejecución o ESUs encargados de implementar una operación determinada del coprocesador. Cada ESU se mapea directamente con un *opcode de coprocesador*, dado por el campo *opc* de las instrucciones *cpop* especificadas en SPARC-V8 descritas líneas arriba. Este nuevo bloque agrega Lógica de Interconexión, que permite seleccionar la ESU adecuada según la operación enviada a la unidad procesadora. Además, el bloque también implementa la *Interfaz de Unidad de Ejecución*, lo que permite una conexión directa con la lógica de control de la unidad coprocesadora.

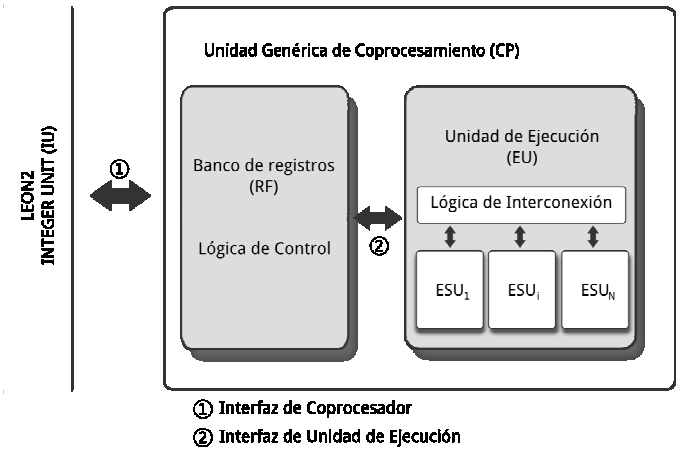

Fig. 4. Unidad Genérica de Coprocesamiento

En esta nueva unidad de ejecución, se implementó la funcionalidad necesaria para el *handshaking* en la comunicación con la lógica de control de la unidad coprocesadora. En esencia, la unidad de ejecución registra el código de operación de la instrucción y los operandos con la aserción de *start* y *load* respectivamente.

Mientras la unidad de ejecución no ha completado la operación, se mantiene *busy* en alto hasta un ciclo de reloj antes de que el resultado *result* sea válido. Una vez que el resultado es válido, se mantiene registrado hasta que se inicie la próxima instrucción.

El sistema completo está formado por fuentes VHDL de diferentes orígenes. La Unidad de Enteros y la Unidad de Punto Flotante son las suministradas con LEON2 sin modificaciones. El banco de registros y la lógica de control de la Unidad Genérica de Coprocesamiento fueron desarrollados partiendo de la Unidad de Punto Flotante de LEON2 con las modificaciones ya comentadas mientras que la Unidad de Ejecución fue desarrollada partiendo de cero. Finalmente, las Sub-unidades de Ejecución deben ser diseñadas por el usuario de este ambiente de desarrollo para implementar las nuevas operaciones a incorporar al coprocesador.

3.1 Interfaz de Sub-unidades de Ejecución

Cada ESU se comunica con la lógica de interconexión a través de una interfaz simple, que permite incorporar la sub-unidad a la unidad de ejecución con facilidad. Quien desee agregar una nueva instrucción al coprocesador, sólo debe implementar la funcionalidad deseada en un bloque que presente la siguiente interfaz:

```
entity my_sub_execution_unit is
port(
--datos
op1_vector : in  std_logic_vector(63 downto 0);
op2_vector : in  std_logic_vector(63 downto 0);
res_vector : out std_logic_vector(63 downto 0);
cc         : out std_logic_vector(1 downto 0);
--control
cs         : in  std_logic;
start      : in  std_logic;
done       : out std_logic
);
```

Esta es la interfaz que todas las subunidades de ejecución deben implementar.

El significado de las señales es:

- *op1_vector* y *op2_vector* - operandos
- *res_vector* - resultado de la operación
- *cc* - condition codes resultantes de la operación especificadas por SPARC-V8
- *cs* - chip select para esta subunidad
- *start* - señal que valida los operandos
- *done* - señal que valida el resultado

Una vez que la sub-unidad de ejecución implementa la interfaz común, sólo resta instanciarla en la unidad de ejecución del coprocesador, y asociarla al código de operación correspondiente.

3.2 Validación

Para probar la funcionalidad y validar el correcto desempeño de la plataforma con la interfaz de coprocesador ya implementada, se utilizaron algunas de las funciones tomadas de una implementación en C de un codificador de voz en el estándar G.729.

Las operaciones elegidas fueron *L_add*, *mult*, *L_mult*, *L_mac*. Estas implementan las operaciones de suma, multiplicación, multiplicación con corrimiento y multiplicación con acumulación, con la particularidad que todas estas operaciones saturan en un valor máximo prefijado. Su elección se realizó de acuerdo a los resultados obtenidos de un análisis del perfil de ejecución de la implementación original del codificador, donde estas operaciones son las que ocupan la mayor parte del tiempo de procesamiento y a su vez brindan una interfaz sencilla para ser implementadas en el coprocesador.

En el diseño de las operaciones elegidas se replicó su funcionamiento original basándose en el código C de referencia.

Con las funciones diseñadas e implementadas en el coprocesador, el flujo de trabajo para la realización de las pruebas se dividió en las siguientes partes

- Simulación de operaciones con ModelSim (HW). Validación de cada sub-unidad de ejecución en forma independiente.
- Integración a la plataforma (HW). Integrar las operaciones al coprocesador.
- Prueba individual de funciones (HW-SW). Se utilizaron pequeños programas en C cuya única funcionalidad era hacer uso de las operaciones mencionadas, se utilizó *inline assembler* para hacer las llamadas a las instrucciones del procesador. Inicialmente se probó con una simulación a nivel de hardware utilizando el banco de pruebas (*testbench*) suministrado por LEON para observar las instrucciones desensambladas (ver Fig. 5) y posteriormente con una ejecución real en hardware.
- Sustitución de código en codec (assembler). En esta etapa se modificó el código fuente del codificador del codec G.729, para que en el caso de las operaciones mencionadas, hiciera uso de la implementación hardware de las mismas integradas en el coprocesador. Las modificaciones consistieron en sustituir el código C original por el código assembler correspondiente de manera de hacer las llamadas al coprocesador.
- Prueba global. La validación del funcionamiento, se llevó a cabo comparando la secuencia de bits generada por la versión con coprocesador ejecutando en la placa con FPGA y la versión original del codificador software ejecutada en un computador personal.

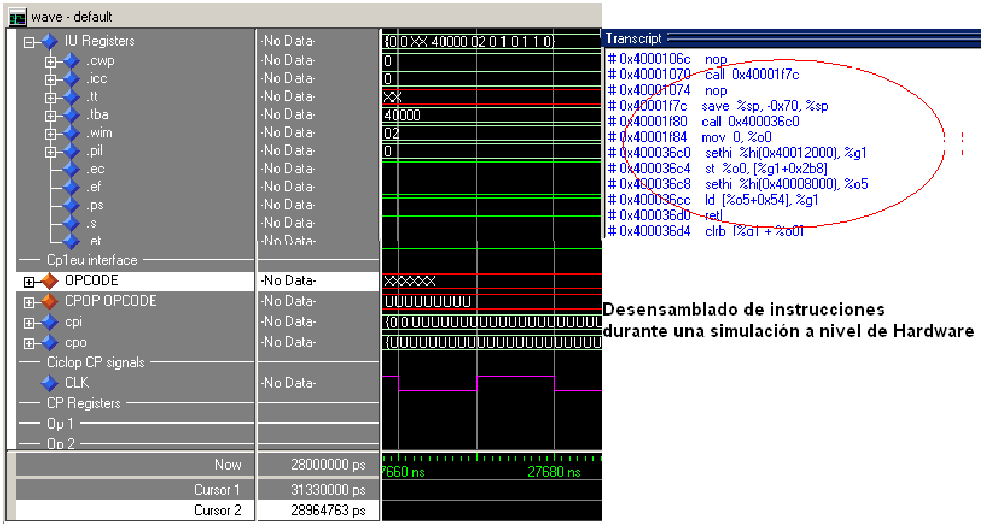

Fig. 5. Simulación combinada.

El sistema completo (LEON2 incluyendo controlador de SDRAM, coprocesador diseñado y unidad de depurado) fue sintetizado en una placa con un integrado de la familia Virtex-E de Xilinx (xcv1000e-7) y memoria SDRAM, funcionó con el reloj de dicha placa que es de 25MHz, y ocupó un total de 11409 LUTs (46%) y 62 bloques de ram (64%).

Por último, un grupo de estudiantes a partir de los archivos fuente y la documentación generada desarrolló un coprocesador para una aplicación específica totalmente independiente. El ejemplo desarrollado fue de procesamiento morfológico de imágenes. Esto valida no solo los bloques de circuito sino también el ambiente de desarrollo completo con una curva de aprendizaje rápida.

4 Conclusiones

El resultado principal es que se ha elaborado con éxito una herramienta que permite integrar en forma fácil un coprocesador dedicado a un LEON2. Este ambiente de desarrollo ha sido probado y validado en hardware con aplicaciones reales. Esto permite que usuarios de LEON2 puedan diseñar sus propios bloques de hardware e integrarlos de manera sencilla al procesador.

Desde el punto de vista del usuario el problema se reduce a trabajar en dos niveles de abstracción. En un primer nivel, se debe diseñar o modificar la aplicación software para hacer uso de llamadas al coprocesador. En segundo lugar, sólo resta implementar la operación deseada como un bloque de hardware utilizando HDL. Toda la comunicación de datos y control desde una llamada software al coprocesador, hasta el bloque de hardware diseñado por el usuario, ya está resuelta. La curva de aprendizaje del ambiente de desarrollo es muy buena, y como dato cuantitativo podemos decir que el tiempo promedio para integrar una nueva operación al coprocesador, una vez instalado todo el ambiente de desarrollo, fue de un día de trabajo.

Otro punto a destacar es la metodología que permite la simulación a nivel de hardware de toda la plataforma, ejecutando un determinado programa. El usuario puede desarrollar sus propias operaciones en hardware, integrarlas al coprocesador, y simular el comportamiento de todo el sistema, LEON2 y coprocesador, ejecutando instrucciones que hagan uso de estas nuevas operaciones hardware. La capacidad de simular todo el sistema, combinando hardware con software, es una herramienta extremadamente poderosa al momento de diseñar nuevas operaciones hardware para integrar al coprocesador. Este banco de pruebas permite una fácil y rápida integración de los nuevos bloques de hardware al coprocesador, ya que el diseñador puede visualizar la evolución de todas las señales del sistema e identificar cualquier problema que impida el buen funcionamiento del mismo.

Un próximo trabajo ya planificado será la migración del ambiente de desarrollo a la nueva versión del microprocesador LEON3 y difundirla dentro de la comunidad de usuarios.

Referencias

1. SPARC International, Inc.: The SPARC Architecture Manual Version 8. http://www.sparc.org/standards/V8.pdf. (1992).
2. Altera Corporation: Nios II Custom Instruction. User Guide (2008)
3. Tillich, S., Großsch¨adl, J. Leon2-CIS User's Manual. Instruction Set Extensions for Cryptography. Graz University of Technology. (2006)
4. P. Guironnet de Massas, P. A., Pétrot, F.: On SPARC LEON-2 ISA Extensions Experiments for MPEG Encoding Acceleration. Hindawi Publishing Corporation. VLSI Design. Vol. 2007, Article ID 28686, (2007) 1-10
5. Peleg, A., Weiser, U.: MMX technology extension to the Intel architecture. IEEE Micro, Vol. 16 (1996) 42-50.
6. Tillich, S. & Großschädl: Instruction Set Extensions for Efficient AES Implementation on 32-bit Processors. In: J. Goubin, L. & Matsui, M. (ed.) CHES 2006, LNCS 4249, Springer Verlag, (2006) 270-284.
7. Vejda, T.; Page, D. & Großschädl, J. Instruction Set Extensions for Pairing-Based Cryptography. In: T. Takagi et al. (Eds.): PAIRING 2007, LNCS 4575, Springer, (2007) 208-224.
8. Grabher, P.; Großschädl, J. & Page, D. Light-Weight Instruction Set Extensions for Bit-Sliced Cryptography. In: E. Oswald and P. Rohatgi (Eds.): CHES 2008, Springer Verlag LNCS 5154 (2008) 331-345.
9. Jacob Bower, Supervisor: Dr. Wayne Luk: A System-on-a-Chip for Audio Encoding. Imperial College London (2004).
10. Vladimirova, T., Eamey, D., Keller, S., Sweeting, M.: Floating-Point Mathematical Co-Processor for a Single-Chip On-Board Computer. In: Military and Aerospace Programmable Logic Device (MAPLD) International Conference (2003).
11. Weaver, D. L., Germond, T. E.: The SPARC Architecture Manual. Version 9 (1994).
12. Gaisler Research.: LEON2 Manual (v1.0.30) (2005).
13. Hodjat, A., Verbauwhede, I.: Interfacing a high speed crypto accelerator to an embedded CPU. In: Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, Vol.1. (2004) 488-492.
14. Etcheverry, L., Horta, J., Nan, S., Supervisores: Oliver, J.P., Pérez Acle, J.: Proyecto CICLOP. Documentación de proyecto de fin de carrera, IIE, FING, Universidad de la República (2007).