Un DSP de coma fija integrado en FPGA aplicado en Electrónica de Potencia

Santiago de Pablo, Jesús Cebrián, Luis C. Herrero
Departamento de Tecnología Electrónica
Universidad de Valladolid
Valladolid (España)
sanpab@eis.uva.es

Alexis B. Rey
Dpto. Electrónica, Tecn. de Computadoras y Proyectos
Universidad Politécnica de Cartagena
Cartagena, Murcia (España)
alexis.rey@upct.es

***Abstract*—Este artículo presenta el procesador "DSPuva16", desarrollado específicamente para aplicaciones de Electrónica de Potencia. Como indica su nombre, este procesador está orientado al cálculo: opera en coma fija de 16 bits extendiendo su precisión hasta 24 bits. Ejecuta regularmente operaciones MAC entre registros internos (rD = rD ± rS * rT) alcanzando 10 MIPS a 40 MHz. Este procesador ha sido probado físicamente sobre un sistema de desarrollo para FPGA.**

La aplicación de este procesador en Electrónica de Potencia (sistemas de microgeneración fotovoltaica, filtros activos, control de motores de alterna, etc.) resulta bastante natural porque todas las magnitudes empleadas están limitadas en rango. Además, es posible utilizar diversos procesadores, con ciclos de trabajo entre 5 y 100 microsegundos, para ejecutar los diversos bucles de regulación y control de cada equipo.

Keywords: Procesador empotrado, DSP, Circuitos de Control.

I. INTRODUCCIÓN

Los primeros dispositivos empleados para controlar convertidores electrónicos de potencia eran circuitos analógicos basados en amplificadores operacionales [7]. Cuando aparecieron los primeros microprocesadores y éstos tuvieron suficiente potencia de cálculo, fueron inmediatamente implantados en los circuitos de control [5] sustituyendo a sus equivalentes analógicos por sus inherentes ventajas de estabilidad, inmunidad al ruido y facilidad de ajuste. Posteriormente se desarrollaron los procesadores digitales de señal (DSP), con una capacidad de cálculo muy superior, lo que permitió un control más preciso y alcanzar unas prestaciones notablemente superiores [2].

Actualmente estamos en un punto en el que es patente que no es suficiente controlar todo el sistema con un único procesador: se requiere controlar múltiples bucles de regulación, algunos de los cuales se han de ejecutar a muy altas frecuencias (cercanas al microsegundo) [6], y además se ha de atender al usuario (teclado, display, modem, servidor web para monitorización y configuración remota) con un comportamiento cada vez más exigente.

La solución más sencilla a este problema consiste en usar varios procesadores para atender las diversas tareas, pues no sólo resulta más barato y fácil de controlar que un único procesador de muy altas prestaciones, sino que además es la única manera de garantizar que el control se realiza en tiempo real.

Pero, ¿qué ocurre si se evalúa la posibilidad de conectar varios procesadores en una tarjeta de circuito impreso? Pues que inmediatamente nos encontramos con nuevos problemas: se requiere un elevado número de pistas para el flujo de datos entre procesadores, pues los canales serie estándar no siempre son suficientes; se necesitan elementos intermedios adicionales para acomodar las distintas velocidades de trabajo; los componentes empleados se vuelven rápidamente obsoletos, lo que obliga a rediseñar equipos completos. Todo esto conduce inevitablemente a unos costes elevados de ingeniería no recuperables.

El panorama es completamente distinto si se utilizan procesadores empotrados (*embedded*). Posiblemente sus prestaciones no sean tan buenas como las de sus homólogos dedicados, pero al estar todos integrados en un único chip (FPGA o ASIC) desaparecen inmediatamente todos los problemas anteriores: la comunicación entre procesadores es flexible e inmediata empleando memorias de doble puerto [4], su uso dentro de estos componentes apenas afecta al coste del conjunto [1], y la obsolescencia es nula ya que el diseño realizado siempre puede ser recompilado sobre otro dispositivo más moderno que, además de tener mayor capacidad, podrá funcionar a una frecuencia igual o mayor que la especificada para el componente anterior.

La principal limitación que nos encontramos, sobre todo en el ámbito de las FPGA, es la imposibilidad de usar unidades de cálculo en coma flotante, pues resultan prohibitivas mientras no se integren a través de módulos cableados. Sin embargo, y aunque resulta más cómodo el uso de la coma flotante en aplicaciones de potencia, también es posible y no resulta excesivamente complejo el uso de aritmética en coma fija: todas la magnitudes (tensiones, corrientes, ganancias de reguladores) están limitadas, de forma natural como ocurre con

las tensiones, o por imposición del algoritmo de control que no puede permitir sobrecargas en el sistema [6].

Por tanto, en este tipo de aplicaciones es normal el uso de varios procesadores de cálculo en coma fija y la incorporación de alguno de propósito general, para atender al usuario. La comunicación entre ellos en un dispositivo FPGA o ASIC no resulta costosa, pues apenas existen restricciones en el número de líneas, ya que son internas, ni en el ancho de banda ni en el tipo de recurso intermedio, pues las memorias de doble puerto apenas ocupan espacio.

En este artículo se presenta un DSP en coma fija de suficientes prestaciones y pequeño tamaño, preparado para manejar un conjunto reducido de variables. Se ha diseñado con la idea de que es mejor emplear varios procesadores que resuelvan tareas distintas, pero acopladas, que un único procesador que haga todo el trabajo, lo que obligaría a emplear una frecuencia de reloj considerablemente mayor y un complejo esquema de interrupciones. En cambio, varios procesadores pueden ejecutar, a diferentes ritmos, cada uno de los distintos lazos de regulación del sistema, y su acoplamiento resulta fácil como se ve en la sección VII y se demuestra en [1].

II. Características Generales

El DSPuva16 es un procesador de cálculo que trabaja en coma fija de 16 bits con precisión extendida de hasta 24 bits. Su arquitectura externa es tipo Harvard, pues accede por una parte a su memoria de programa (cuyo tamaño varía entre 256x16 y 4Kx16, según el modelo de procesador empleado) y por otros buses distintos a los datos (256 puertos síncronos de 16 bits).

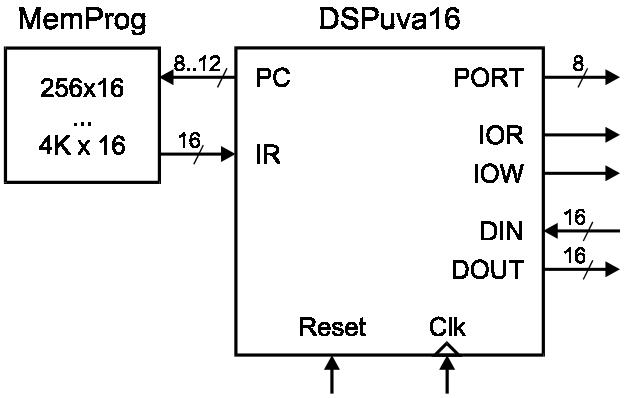

Figura 1. Conexiones externas del DSPuva16.

Su operación básica es la multiplicación con acumulación, tanto positiva como negativa ("rD = rD ± rS * rT"), que ejecuta de forma regular en un único ciclo de instrucción. Éste se compone siempre de cuatro ciclos de reloj, como corresponde a su arquitectura RISC.

Dispone de 16 registros de 24 bits, denominados de 'r0' a 'r15', que pueden ser empleados en cualquier operación salvo 'r0', que no puede ser usado como operando. Precisamente esta característica permite ampliar considerablemente las posibilidades del código de instrucciones sin añadir excesiva complejidad al diseño interno del procesador: cuando se encuentra la codificación equivalente al registro 'r0' en la ubicación del registro 'rS', se anula el valor del operando, y cuando se encuentra en la posición del registro 'rT', se sustituye su valor por una constante de 16 bits que se toma de la propia memoria de programa, sin retrasar la ejecución de la instrucción. Esto permite, por ejemplo, que una misma instrucción básica del procesador como es la suma (rD = rS + rT) permita operar con una constante inmediata (rD = rS + K) y realizar asignaciones directas a un registro (rD = 0 + rT; rD = 0 + K). Esta propiedad es ampliamente utilizada en la programación de filtros digitales, cuya estructura general es:

$$y(t) = c_0 \cdot x(t) + c_1 \cdot x(t-T) + \ldots + d_1 \cdot y(t-T) + d_2 \cdot y(t-2T) + \ldots$$

donde todos los coeficientes son constantes.

Los accesos externos se realizan a través de 256 puertos síncronos (cada acceso se completa en un único ciclo de reloj) con direccionamiento directo: "pN = rS; rD = pN"; donde 'N' es cualquier valor entero entre 0 y 255. Puede parecer que este tipo de direccionamiento es menos flexible que el indirecto, pero en el campo de aplicación de este procesador esto no supone ninguna limitación: es habitual que los puertos se correspondan con posiciones de memoria que amplían las capacidades de almacenamiento del procesador, o que conecten con dispositivos físicos que permitan leer o generar señales analógicas. En todo caso, el direccionamiento indirecto siempre es posible empleando un puntero externo al procesador, pero integrado en el mismo chip, pues debemos recordar que estamos trabajando dentro de una FPGA o un ASIC.

El control de las subrutinas es similar al de otros procesadores, pero el usuario debe dedicar un registro para guardar y retener la dirección de retorno; luego se puede emplear ese mismo registro para efectuar el retorno al punto de llamada. Si se desea llamar a una subrutina dentro de otra, siempre se puede guardar en una pila LIFO externa la dirección de retorno anterior, o bien usar otro registro para gestionar la nueva subrutina.

III. Conjunto de Instrucciones

El conjunto de instrucciones del DSPuva16, como se muestra en la tabla I, es muy sencillo. Consta únicamente de 17 instrucciones distintas, pero alcanza una notable flexibilidad gracias a cómo utiliza la codificación correspondiente a 'r0', tal como se ha explicado en el apartado anterior.

Tabla I. Instrucciones del DSPuva16.

Código de operación	Mnemonico	Operación
0000 AAAA AAAA DDDD	call (rD) addr	Salto absoluto
0001 0000 0000 SSSS	ret (rS)	Retorno (pc = rS)
0001 1FFF AAAA AAAA	jpFlag addr	Salto condicional
0010 DDDD NNNN NNNN	rD = pN	Lectura externa
0011 SSSS NNNN NNNN	pN = rS	Escritura externa
0100 DDDD SSSS TTTT	rD = rS * rT	Producto normaliz.
0101 DDDD SSSS TTTT	rD = rS x rT	Producto/desplaz
0110 DDDD SSSS TTTT	rD += rS * rT	MAC positiva

0111 DDDD SSSS TTTT	rD –= rS * rT	MAC negativa
1000 0FFF DDDD TTTT	ifFlag rD = rT	Asign. condicional
1001 0FFF DDDD TTTT	ifFlag rD = –rT	Asign. condicional
1010 DDDD SSSS TTTT	rD = rS + rT	Suma
1011 DDDD SSSS TTTT	rD = rS – rT	Resta
1100 DDDD SSSS TTTT	rD = rS and rT	AND lógico
1101 DDDD SSSS TTTT	rD = rS or rT	OR lógico
1110 DDDD SSSS TTTT	rD = rS nor rT	NOR lógico
1111 DDDD SSSS TTTT	rD = rS xor rT	XOR lógico

Siempre que se llama a una subrutina se guarda la dirección de retorno (PC + 1) en un registro, que suele ser 'r0' ya que no puede ser usado como operando. Sólo se dedican ocho bits para indicar la dirección de destino, lo que limita la longitud de los programas a tan sólo 256 instrucciones. Aunque no parezca mucho, esto es suficiente en muchos casos, pues como se ha dicho se prefiere emplear varios procesadores ejecutando cada uno de ellos un bucle específico de regulación. Sin embargo, se ha ampliado el código ejecutable del procesador utilizando un mecanismo sencillo y potente: se admiten hasta cinco modelos distintos de procesador ('A', 'B', 'C', 'D' y 'E'), con longitudes de programa de hasta 256, 512, 1K, 2K y 4K instrucciones. Cuando se efectúa un salto absoluto (con la instrucción 'call'), sólo se puede saltar a las posiciones pares cuando se emplea el modelo 'B', sólo a una de cada cuatro si se emplea el modelo 'C', y así sucesivamente. Eso significa que todas las subrutinas deben estar correctamente "alineadas" en la memoria de programa, para lo cual se ha creado la directiva del ensamblador "#align", y se indica qué modelo se está utilizando a través de la directiva "#model {A,B,C,D,E}".

Los saltos condicionales son relativos, precisamente para evitar el problema que causaría el alineamiento de todas las direcciones de salto. Como muchos otros procesadores, el salto se debe realizar a una posición cercana (en un entorno de unas ±128 instrucciones), y se pueden usar las condiciones típicas dependientes del resultado anterior: 'eq', 'ne', 'gt', 'ge', 'lt', 'le', 'v' y 'nv'. Estas mismas condiciones pueden ser usadas para realizar asignaciones condicionales, como ocurre en "rN = rN; iflt rN = -rN", que calcula el valor absoluto de un número guardado en un registro.

Las lecturas (rD = pN) y escrituras (pN = rS) sobre puertos externos utilizan únicamente el modo de direccionamiento directo. Esto es adecuado en este procesador, pues está orientado a manejar un conjunto reducido de variables, aplica siempre y de forma regular el mismo algoritmo sobre los mismos datos, y emite siempre el mismo número de resultados. Los accesos son síncronos y se ejecutan en un único ciclo de reloj. Esto tampoco supone ninguna limitación, pues los recursos con los que se tiene que comunicar están implementados con la misma tecnología, al estar en el mismo chip, y por tanto pueden trabajar a la misma frecuencia de reloj.

Las instrucciones que realizan productos, sumas y restas, y las operaciones lógicas, tienen una estructura regular de dos operandos y un registro de destino. Como ya se ha dicho, se amplían las posibilidades del conjunto de instrucciones permitiendo la anulación del primer operando ('rS') y la sustitución del segundo ('rT') por una constante de 16 bits ('K'). De esta manera, se pueden aplicar máscaras lógicas inmediatas (rD = rS and/or/xor K), se puede negar el contenido de un registro (rD = 0 nor rT) y se puede inicializar cualquier registro con una constante (rD = 0 + K).

La operación más importante de este procesador, y su razón de ser, es el producto en coma fija, con o sin acumulación. En general, se toman dos operandos de 16 bits en formato normalizado <1.15> (un bit entero que contiene el signo y 15 bits fraccionarios, lo que permite representar números entre –1.0 y +0.99996948), entregando un número de 24 bits también normalizado (un bit entero de signo y 23 bits fraccionarios, lo que se denota como <1.23>, y que permite representar números entre –1.0 y +0.99999988). Cuando al producto le sigue una acumulación, tanto positiva como negativa, esta última operación se hace con operandos de 24 bits, para no perder precisión.

También se ha incorporado un tipo de producto (rD = rS x rT) que permite realizar desplazamientos. El segundo operando se interpreta en formato <8.8>, es decir, ocho bits de parte entera y otros ocho de parte fraccionaria, con lo que se puede representar, entre otros, números como 1/128, 1/64, ..., 1/4, 1/2, 2, 4, ..., 32 y 64, además de otros valores intermedios. El resultado se produce igual que antes en formato normalizado <1.23>.

Otras instrucciones típicas se han implementado como "macros" reconocidas por el lenguaje ensamblador. La macroinstrucción 'nop', que no debe hacer nada, se sustituye por "r1 = r1 or r1" y la macroinstrucción 'break', que permite introducir puntos de parada en el simulador, se reemplaza por "r1 = r1 and r1". El emulador no dispone de puntos de parada.

IV. Ciclo de Instrucción

Como hemos indicado, la arquitectura interna de este procesador es RISC. Eso significa que ejecuta todas sus instrucciones de forma regular, en particular en cuatro ciclos de reloj. Aunque, desde el punto de vista del usuario, todas las instrucciones se ejecutan en cuatro ciclos, realmente existe cierto solapamiento entre instrucciones, y de hecho muchas instrucciones se terminan cuando ya se está ejecutando la siguiente. En cualquier caso, sólo se produce un único efecto de "latencia" que será explicado más tarde.

Todas las instrucciones comienzan leyendo de la memoria de programa el código de la operación que se debe realizar, dedicando a esta función los dos primeros ciclos de reloj. Después de emplean otros dos ciclos para leer los operandos, 'rS' en primer lugar y después 'rT'. Este último es sustituido por una constante leída de la memoria de programa si se hace referencia a 'r0'. Por último, en el primer ciclo de la siguiente instrucción, se realiza la operación solicitada y se almacena el resultado en 'rD'.

1) Envía PC a la memoria de programa.
2) Recoge el código de la instrucción sobre IR.
3) Lee rS sobre un acumulador intermedio ACC.
4) Lee rT sobre RegT y lleva el valor de ACC a RegS.

1') Guarda en ‘rD’ la operación entre RegS y RegT.

El contador de programa PC se incrementa durante la fase '2' y, si se lee una constante de la memoria, también durante la fase '4'. Si la operación requiere un salto (‘call’, ‘ret’, ‘jpFlag’), no se guarda ningún resultado de ninguna operación, sino que únicamente se modifica el contador de programa durante la fase '4'.

Con este esquema tan sencillo es posible ejecutar todas las instrucciones salvo las de producto. Como veremos enseguida, el multiplicador de este procesador requiere cuatro ciclos de reloj para completar su operación, pues así puede tener un tamaño mucho menor[1] y no deteriora la frecuencia de trabajo del conjunto. Para efectuar las multiplicaciones, con o sin acumulación, se emplean los cuatro ciclos de la siguiente instrucción y otros dos de la posterior:

1) Envía PC a la memoria de programa.
2) Recoge el código de la instrucción sobre IR.
3) Lee rS sobre el registro intermedio ACC.
4) Lee rT sobre RegT y lleva el valor de ACC a RegS.
1') Primera fase del MAC usando RegS y RegT.
2') Segunda fase del MAC usando RegS y RegT.
3') Tercera fase del MAC usando RegS y RegT.
4') Cuarta fase del MAC usando RegS y RegT.
1") Ciclo adicional por la segmentación del multiplicador.
2") Lee ‘rD’ y opera con el resultado del MAC.

De esta manera se introduce una latencia que habrá de tener en cuenta el programador: el resultado de un producto no está disponible para su uso general en la instrucción siguiente, sino en la inmediatamente posterior. En cambio, cuando se utiliza en posición acumuladora se evita la latencia, pues la acumulación también se lleva a cabo un ciclo tarde. Esto se ilustra en el siguiente ejemplo:

```
r1 = 0.27          // Se asigna un valor al registro r1
r2 = r1 + 0.32     // Correcto, pues ya se dispone de r1
r3 = r1 * r2       // Se multiplican los valores de r1 y r2
r3 = r3 + r2 * r2  // El uso acumulativo de r3 es correcto
nop                // Se espera a que termine el cálculo de r3
p4 = r3            // Ahora, y no antes, r3 vale r1*r2+r2*r2
```

Por tanto, cuando se quiere usar el resultado de un producto, salvo cuando se hace en posición acumuladora, hay que añadir un ‘nop’ después de la multiplicación para dar tiempo a que se calcule el resultado. Ésta es la única latencia que introduce la arquitectura segmentada de este procesador.

V. Arquitectura Interna

El DSPuva16 ha sido descrito usando Verilog y se basa, como muestra esquemáticamente la figura 2, en una arquitectura RISC de dos buses de 24 bits, uno para los operandos y el otro para los resultados.

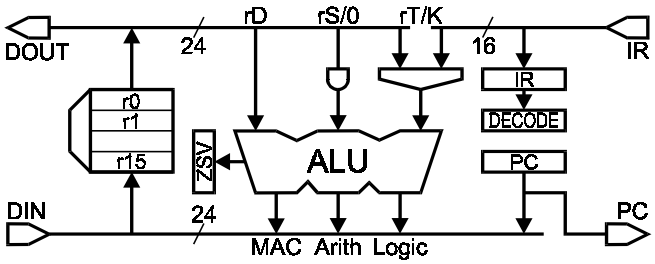

Figura 2. Arquitectura interna del DSPuva16.

El contador de programa (‘PC’) tiene entre 8 y 12 bits, según el modelo de procesador implementado. El código de instrucción que se recibe a través de ‘IR’ tiene siempre 16 bits. Empleando este valor, un decodificador de instrucciones segmentado va activando cada parte del circuito hasta completar cada operación.

El banco de registros, que retiene los valores de ‘r0’ a ‘r15’, es una memoria con escrituras síncronas y lecturas asíncronas de tamaño 16x24. Ocupa sólo 24 celdas básicas en algunos dispositivos FPGA, lo que equivale al 5% de todo el procesador.

La ALU, por su parte, se compone de tres unidades que realizan operaciones lógicas, aritméticas o multiplicativas. Después de cada instrucción se actualizan los *flags* de “cero” (‘Z’), “signo” (‘S’) y “desbordamiento” (‘V’) que permiten efectuar las asignaciones y los saltos condicionales correspondientes. No se dispone de acarreo (‘C’) pues no es necesario en los programas que se han de ejecutar.

Cuando se intercambian datos con elementos externos se utilizan dos buses distintos, uno para enviar datos y otro para recibirlos, lo que evita algunos *buffers* triestado que, por otra parte, son innecesarios dentro de un chip.

A. *Arquitectura de dos buses*

Muchos procesadores RISC tienen actualmente tres o más buses, lo que permite incluso realizar todas las operaciones relativas a cada instrucción en un único ciclo de reloj.

Ya que este procesador necesita cuatro ciclos de reloj para completar cada multiplicación, y que el uso de un tercer bus nos conduciría a emplear memorias de doble puerto para el banco de registros[2], se ha optado por dedicar un único bus para los operandos. Por este bus circulan los valores de ‘rD’ durante los dos primeros ciclos de cada instrucción y los valores de ‘rS’ y ‘rT’ durante los dos últimos.

El bus de resultados se dedica a recoger las salidas de las diferentes unidades de cálculo y de otras fuentes, como son la entrada de datos externos cuando se lee de un puerto y el valor del contador de programa cuando se llama a una subrutina, para guardar la dirección de retorno en un registro.

B. *Multiplicador de cuatro etapas*

La operación central de este procesador es la multiplicación. Opera sobre dos valores de 16 bits y emite un

[1] Al fraccionar el cálculo de la multiplicación en cuatro etapas se consigue reducir el tamaño del multiplicador a una cuarta parte, aproximadamente. Aún así, el multiplicador necesita unas 250 celdas básicas, que es la mitad de lo que ocupa el procesador.

[2] Estas memorias permiten dos lecturas simultáneas y una escritura síncrona al terminar el ciclo, pero ocupan el doble de espacio que las memorias usadas por este procesador.

resultado de 32, de los cuales sólo están disponibles finalmente 24. En general los operandos están en formato <1.15> y el resultado en formato <1.23>.

Para construir el multiplicador se ha decidido dividir la operación en cuatro etapas[3], multiplicando en cada paso un operando de 16 bits por otro de sólo cuatro bits, y emitiendo un resultado intermedio de 20 bits. Esta operación se puede realizar con cuatro sumadores y con un registro intermedio de segmentación. Si toda la operación se realizara en un único paso se necesitarían 15 sumadores ocupando una superficie cuatro veces superior, y sólo se conseguiría mantener la frecuencia de trabajo introduciendo registros intermedios, que no reducirían el tamaño del conjunto.

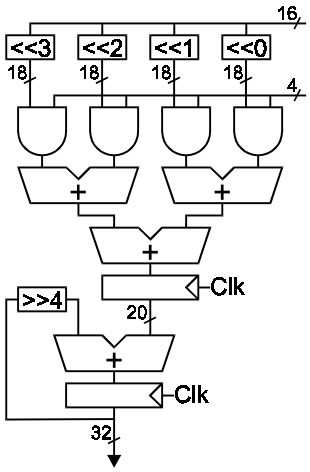

Figura 3. Estructura segmentada del multiplicador en coma fija.

VI. Entorno de Desarrollo

El entorno de desarrollo integrado de este procesador, denominado "IDEuva16", se corresponde con las necesidades que se encuentran cuando se aplica al control de convertidores electrónicos de potencia. Los programas típicos que tiene que ejecutar este procesador suelen tener entre cincuenta y mil instrucciones, que se corresponden con los 5 a 100 microsegundos típicos de sus ciclos de trabajo. Por tanto, la programación en lenguaje ensamblador es suficiente. En todo caso, la sintaxis empleada por las instrucciones es bastante cómoda, como se ha podido apreciar hasta ahora.

El programa IDEuva16 se ejecuta en modo gráfico[4] e incorpora un editor sencillo, un ensamblador con preprocesador, un simulador y un enlace con el procesador físico para controlarlo en modo emulador, a través del puerto paralelo del ordenador o por conexión USB. En la figura 4 se muestra el resultado de una simulación en la que se calcula "r2 = sin(r1); r3 = cos(r1)" con un ángulo "r1 = 0.4·π". Todo el proceso se ejecuta en 14 microsegundos.

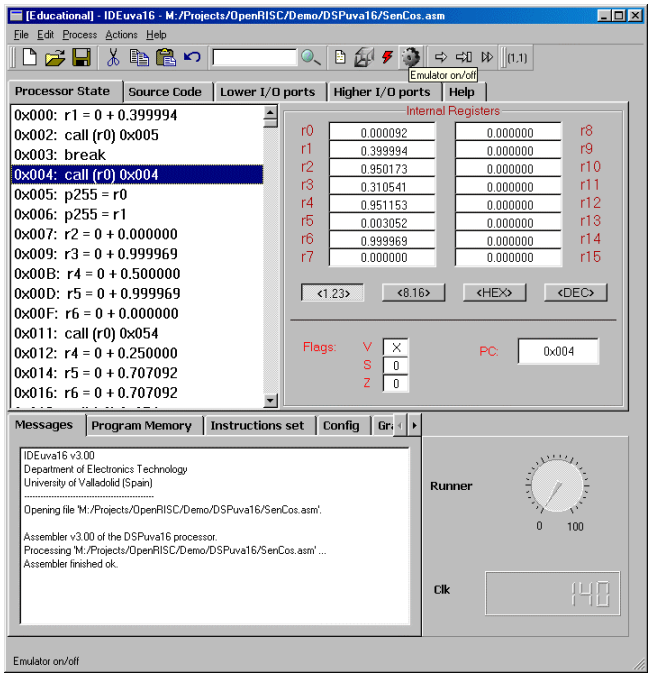

Figura 4. Entorno de desarrollo IDEuva16.

Después de editar el código fuente ensamblador, donde se dispone de las típicas directivas "#include", "#define", "#ifdef", etc., se realiza el ensamblado simplemente pulsando un botón. El proceso tarda apenas unos segundos. Después se puede simular paso a paso o hasta una instrucción 'break'; también se pueden ejecutar miles o cientos de miles de instrucciones en un solo paso, pues los tiempos que se necesita simular son en general de varios milisegundos. El proceso completo de simulación suele tardar unos minutos empleando un Pentium 4. La ventaja que tiene este modo es que se ve completamente cómo se comporta el procesador, incluso gráficamente a través de una ventana sobre la que se puede ir volcando los resultados (usando los puertos 'p0' a 'p15').

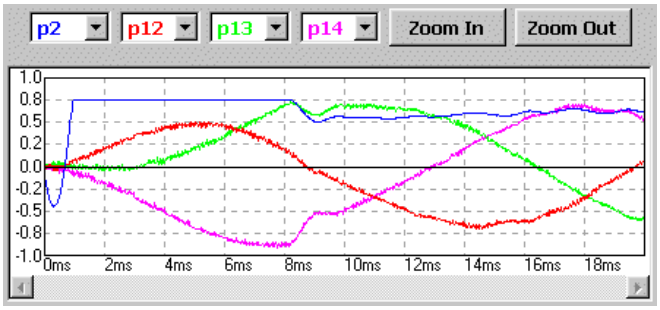

Figura 5. Salida gráfica de una simulación en el entorno de desarrollo.

Cuando los algoritmos son suficientemente estables se puede pasar al "modo emulador": se sintetiza e implementa un DSPuva16 sobre una FPGA, que se conecta al ordenador a través de un interfaz adecuado. El propio entorno de desarrollo transfiere a la memoria de programa del DSP el producto del ensamblado, y luego se controla su estado de *reset-run*. Cuando el usuario detiene el procesador se capturan los resultados intermedios o finales emitidos por éste a través de algunos de sus puertos, visualizándolos en la pantalla del ordenador.

[3] No todos los dispositivos FPGA disponen de multiplicadores embebidos.

[4] De momento sólo bajo Windows 98/XP, tanto en inglés como en español.

La diferencia entre la simulación y la emulación es que la primera se hace completamente en el ordenador, mientras que la segunda se ejecuta en tiempo real sobre el equipo físico, implantado de momento en una FPGA[5].

VII. Sistemas Multiprocesador

La principal ventaja de este DSP es su reducido tamaño: apenas ocupa el 12% de una FPGA de 200K puertas. Esto significa que, aunque un único DSP no pueda realizar toda la tarea de control, no es demasiado costoso añadir otros hasta alcanzar las prestaciones deseadas. En una aplicación de potencia que se está desarrollando, y en la que se está empleando este DSP como elemento de control, es necesario sincronizar un equipo fotovoltaico con la red eléctrica y regular las potencias activa y reactiva que se intercambian; esta tarea la realiza un primer DSP con un ciclo de trabajo de 25 microsegundos. También es necesario controlar las corrientes del equipo y la frecuencia de conmutación de los polos de potencia, y esto se ha de realizar al menos cada 5 microsegundos; esta tarea la realiza un segundo DSP. Actualmente se ha emulado el comportamiento del circuito eléctrico cada microsegundo con un tercer DSP que opera en 60 microsegundos[6], y todo dentro de una única FPGA empleando en total unas 100K puertas equivalentes.

Para la comunicación entre procesadores en este ámbito se han usado memorias de doble puerto[4], aunque no está limitado a este procedimiento: un DSP lee y escribe por un puerto de la memoria, mientras otro DSP lee, pero no puede escribir. Para cada DSP los intercambios son accesos normales a puertos (rD = pN; pN = rS). Empleando este recurso, que ocupa muy poco espacio en una FPGA, no es necesario preocuparse de ningún tipo de conflicto en la compartición de recursos: ambos procesadores pueden usar la memoria común en cualquier momento. Tampoco es necesario señalizar o sincronizar las transferencias, pues la frecuencia natural de las señales compartidas es muy inferior al ciclo de trabajo de los procesadores, por lo que apenas importa que los valores leídos por el procesador de destino se correspondan con los valores actuales o los anteriores, emitidos por el otro DSP.

VIII. Conclusiones

En este artículo se ha mostrado cómo es el procesador DSPuva16 y cómo, a pesar de sus limitadas características, puede resolver fácilmente problemas complejos de control en tiempo real, simplemente añadiendo tantos procesadores como se necesite, pero siempre dentro de un chip.

Su principal limitación es que opera con pocos datos y en coma fija, pero como hemos visto eso no supone ningún problema en el control de convertidores electrónicos de potencia, pues todas las magnitudes físicas son fáciles de normalizar y no es difícil garantizar que se van a mantener dentro de un determinado rango, salvo situaciones de avería que provocarían la parada del equipo.

El DSP completo, descrito en Verilog, ocupa unas 24.000 puertas equivalentes.

Reconocimientos

Nuestro agradecimiento a Carmen Cascón [3] y a Juan del Barrio [1], que han contribuido notablemente en el desarrollo del IDE de este procesador y en la primera aplicación de este DSP en un sistema fotovoltaico de generación distribuida.

También queremos agradecer a Le Duc Hung, de la University of Natural Sciences en Ho Chi Minh City (Vietnam) sus continuos comentarios y mejoras sobre este procesador y sus herramientas.

Referencias

[1] J. del Barrio, Desarrollo sobre FPGA de un emulador de una planta de microgeneración eléctrica. Proyecto Fin de Carrera en la ETSII. Universidad de Valladolid, España, 2004.

[2] B.K. Bose, Power Electronics and AC drives. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[3] C. Cascón, Diseño de un entorno de desarrollo para un DSP en coma fija de 16/24 bits integrado en FPGA. Proyecto Fin de Carrera en la ETSII. Universidad de Valladolid, España, 2003.

[4] S.K. Knapp, "XC4000 Series Edge-Triggered and Dual-Port RAM Capability", Xilinx XAPP065, 1996.

[5] B. Norris, Electronic Power Control and Digital Techniques (Texas Instruments Electronics Series). McGraw-Hill, 1976.

[6] A.B. Rey, Control digital vectorial con sliding en fuente de corriente para convertidores CC/CA trifásicos conectados a red. Tesis Doctoral, Universidad de Valladolid, España, 2000.

[7] J. Schaefer, Rectifier Circuits: Theory and Design. John Wiley & Sons, Inc., Library of Congress Catalog Card Number 65-12703, 1965.

[5] Los resultados de este artículo han sido probados sobre una XC2S200-PQ208 de Xilinx trabajando a 40 MHz.

[6] Antes de empezar con las pruebas de potencia sobre el convertidor real es conveniente probar físicamente los DSPs de control sobre una planta eléctrica "emulada". Esta tarea se lleva a cabo empleando un DSP adicional, que calcula cómo se comportaría el circuito eléctrico. Sin embargo, su ciclo de trabajo es habitualmente mucho mayor que el de los otros procesadores, lo que obliga a añadir pausas adicionales en los demás para que le esperen, perdiendo en sentido estricto las características de tiempo real. En todo caso, la emulación en FPGA realiza en unos pocos segundos lo que la simulación sobre un Pentium-4 requiere varios minutos.