Unidad de Control e Interfaz

J. M. Moreno, L. Medina, J. Brizuela, J. F. Cruza y C. Fritsch

Resumen— **Este trabajo describe la implementación de la Unidad de Control e Interfaz (UCI) para sistemas de adquisición y procesamiento distribuidos con tiempos de respuesta predecibles y errores del orden de 1 ns, independientemente de la ubicación del recurso accedido. La UCI desacopla la operación del sistema de un computador de control o *host*, al que libera de las tareas de control. A su vez, controla el sistema mediante solicitudes de lectura y escritura que son eventualmente atendidas y, en ciertos casos, ejecutadas simultáneamente en todo el sistema.**

Palabras clave— **Sistemas distribuidos, interfaz, mini-módulo, tiempo de respuesta.**

I. INTRODUCCIÓN

En muchas aplicaciones, la adquisición de señales y su procesamiento se realiza de forma distribuida en módulos. En ocasiones, la modularidad es un requisito debido a la complejidad del sistema. Además, a veces es necesario un control temporal estricto, con resoluciones en el entorno de 1 ns en aplicaciones tales como imagen ultrasónica, radar sintético, imagen médica nuclear, etc. y algo menos exigentes en otros ámbitos. Estas aplicaciones suelen desarrollarse con sistemas de control específicos, normalmente no transportables a otros. El objetivo principal de este trabajo es obtener una Unidad de Control e Interfaz (UCI) de propósito general para aplicaciones modulares con tiempos de respuesta predecibles.

El sistema se controla desde un puesto central desde donde se establecen las condiciones de adquisición de datos, los parámetros de procesamiento y recibe el flujo de resultados. El tiempo de respuesta del computador central o *host* no es lo suficientemente rápido como para realizar la gestión del sistema con los estrictos requerimientos temporales. Por esta razón el control del sistema debe realizarse de forma desacoplada y desatendida por el computador de operación.

Además la UCI se conecta al *host* a través de un enlace estándar (Ethernet, USB, etc.), cuyos tiempos de respuesta no son predecibles. Por otra parte, desde el punto de vista práctico, una vez establecidos los parámetros operativos, sólo interesa recibir la información suministrada por el sistema, bien para su visualización (imagen), almacenamiento o post-procesamiento.

Este trabajo presenta el diseño de una UCI para este tipo de aplicaciones, implementada en un mini-módulo de propósito general de diseño propio [1]. Proporciona una metodología para el desarrollo de la parte de control de sistemas complejos, aspecto raramente recogido por la literatura especializada. Los controladores se diseñan de forma específica para cada aplicación.

(1) Consejo Superior de Investigaciones Científicas
La Poveda (Arganda del Rey), 28500 Madrid
e-mail: jm.moreno@csic.es

La UCI controla el sistema con tiempos de ejecución predecibles y repetitivos, sin intervención del computador de operación o *host*. La UCI programa de forma autónoma parámetros de procesamiento, condiciones de adquisición, etc. y adquiere información del estado en el sistema distribuido. Asimismo recibe los resultados de la adquisición y procesamiento de señales, almacenándolos localmente para su envío a solicitud del computador de operación.

En realidad, la UCI actúa como un elemento con capacidad de decisión para realizar el control del sistema a alta velocidad mientras que el *host* se dedica exclusivamente a la recepción de resultados de la adquisición y procesamiento.

Aun cuando la UCI se ha diseñado como un elemento de propósito general, actualmente se está utilizando para controlar sistemas con centenares de canales de adquisición a alta velocidad (50 MS/s por canal) para formar imagen ultrasónica.

II. FUNCIONALIDAD

La UCI puede verse como un procesador de arquitectura no convencional, específicamente dirigido a operaciones de gestión y control de sistemas modulares con tiempos de respuesta cortos y predecibles. Este procesador ejecuta programas y accede a diversos recursos.

Como aspecto destacado, una vez programadas las tareas a realizar por el sistema en la UCI, ésta opera de forma autónoma e independiente del *host* y de las interfaces estándar utilizadas en las comunicaciones. La UCI tiene capacidad para modificar el estado y modos de operación del sistema sin intervención del *host*. Esta capacidad debe haber sido previamente programada por el *host* en la UCI durante la inicialización del sistema.

Actualmente, la UCI implementa en hardware las siguientes funciones:

- Programación desasistida de parámetros y recepción de resultados.
- Ejecución de comandos específicos (secuencias, ráfagas, disparo, bucles, etc.).
- Interfaz con el *host* mediante G-Ethernet y USB-2.0 (en fase de diseño USB 3.0).
- Gestión de comandos recibidos desde una interfaz estándar.
- Protocolo de comunicación con el sistema (estándar AMPLIA-3 y AMPLIA-4)
- Almacenamiento temporal de información recibida.
- Procesador Micro-Blaze, utilizado exclusivamente para las comunicaciones Ethernet.

En su aplicación, el *host* puede enviar comandos a la UCI o al sistema a través de alguna de las interfases estándar (USB, Ethernet). Estos comandos son ejecutados en el acto y, eventualmente, obtienen información de estado, que se almacena temporalmente para su envío a solicitud del *host*.

El *modo inmediato* se utiliza ocasionalmente (configuración, carga de memoria de comandos, depuración, etc.) pues no pueden garantizarse los tiempos de ejecución. Este es el único modo del que disponen los controladores convencionales.

Habitualmente se utiliza el modo de operación *programado*, donde los comandos se extraen de una memoria interna de la UCI, donde han sido previamente cargados. En este caso los tiempos de ejecución son predecibles al no estar sujetos a interrupciones o a retardos diferentes de los programados. Además, el *host* queda liberado de las tareas de control del sistema, ocupándose únicamente de recibir los resultados.

En ambos casos los comandos se interpretan como *solicitudes* de lectura, escritura o ejecución, acciones que se efectúan tras cierto tiempo de latencia. Esta latencia se requiere tanto para alcanzar un módulo alejado de la UCI (en un sentido *lógico* para una estructura encadenada), como para garantizar que ciertas operaciones se ejecutan *simultáneamente* en todos los módulos del sistema, con independencia de su ubicación o proximidad a la UCI.

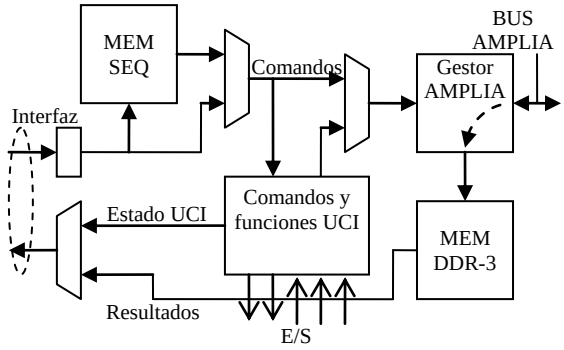

Fig. 1. Arquitectura general de la UCI

III. ARQUITECTURA

La Figura 1 muestra la arquitectura general de la UCI propuesta. La anchura de los datos internos es de 32 bits. Los elementos fundamentales y su función son:

1. Interfases estándar USB y Ethernet con el host. Por claridad se omiten detalles y los datos (recibidos o a enviar) se convierten al ancho interno de 32 bits.
2. Memoria de secuencias de comandos. Basada en BRAMs (16Kx32), almacena secuencias de comandos que permiten programar parámetros, ejecutar acciones y leer el estado en el sistema y, en la UCI ejecutar operaciones de control como saltos condicionales, bucles, temporizaciones, etc. Equivale a la "memoria de programa" de un procesador convencional.
3. Gestor de comandos y funciones UCI. La UCI cuenta con un juego de funciones propias que permiten realizar operaciones específicas de control, como saltos condicionales en la ejecución de secuencias, espera a eventos, configuración de FPGAs, bucles, enumeración de módulos e intercambios de información. Los comandos a la UCI son solicitudes de lectura o escritura en recursos propios, equivalentes a los comandos dirigidos a cualquier módulo en el sistema. Asimismo, esta unidad se relaciona con señales externas para sincronización, disparo, entradas/salidas de propósito general, etc.
4. Gestor del protocolo AMPLIA. El bus AMPLIA y su protocolo [2] se definió para realizar sistemas modulares distribuidos con temporización estricta. La versión actual (AMPLIA-3) cuenta con un bus de datos bidireccional de 16 bits y 8 líneas de control, con un reloj nominal de 40 MHz. Está en desarrollo la versión AMPLIA-4, que dispone en cada sentido 4/8 líneas para datos, una de reloj y otra de validación, todas ellas diferenciales LVDS, que pueden operar a cadencias de hasta 1 Gbit/s. En este caso, los datos son de 32 bits y se envían con un protocolo DDR a partir de un reloj base de 50 a 66 MHz.

IV. COMANDOS

Los comandos se interpretan como solicitudes de lectura/escritura en los recursos del sistema. Las posibles fuentes de comandos son el *host*, la memoria de secuencias y el gestor de funciones UCI. Todos los comandos son de 32 bits con la estructura mostrada en la Figura 2.

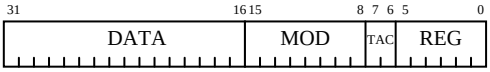

Fig. 2. Estructura de un comando (32 bits)

Los diferentes campos de un comando son:

DATA: 16 bits que indican el valor a escribir o el número de datos a leer en el recurso accedido (en este último caso se leen {DATA}+1).

MOD: Módulo accedido (8 bits). El módulo 0xFF es la UCI. El módulo más próximo a la UCI tiene, normalmente, la dirección 0x00.

TAC: Tipo de acceso, según:

00b: Escritura por *posición*: dirección indicada por campos MOD y REG. Se escribe {DATA} en un único recurso en el sistema.

01b: Escritura global: se accede a todos los módulos cuya dirección global coincida con la del campo MOD. La dirección global se programa en un registro específico de los módulos (se sugiere utilizar el registro 0). Se utiliza, por ejemplo, para programar el mismo valor de un parámetro en un grupo de módulos y su ejecución es simultánea.

10b: En la UCI hace referencia a una *función*. En los módulos, no se usa.

11b: Lectura por *posición*: dirección indicada por campos MOD y REG. Se accede a un único recurso en el sistema, realizándose {DATA}+1 lecturas en el mismo.

REG: Registro accedido en el módulo especificado. Cada módulo dispone de 64 direcciones para realizar sus funciones.

En todos los módulos, hay dos registros reservados (*registros AMPLIA*):

Registro 0: En lectura, devuelve 0xC51C (verificación de configuración) y, en escritura, contiene la dirección de acceso global.

Registro 1: es el registro de control del módulo en escritura y la palabra de estado en lectura.

La UCI también dispone de los dos registros AMPLIA, con operaciones reservadas en lectura y escritura. Los otros 62 registros son de uso libre (accedidos en escritura con TAC=00b y en lectura con TAC=11b). La escritura con TAC=10b se refiere a funciones específicas de la UCI.

V. FUNCIONES Y SECUENCIAS

Una secuencia típica contiene comandos y funciones de la UCI ({MOD} = 0xFF) o comandos al sistema ({MOD} ≠ 0xFF), de lectura ({TAC}=11b) o escritura ({TAC}≠ 11b). Las funciones de la UCI ({TAC}=10b) facilitan la realización de bucles, temporizaciones, disparos, sincronización y modificación de parámetros.

La UCI no interpreta los comandos enviados al sistema (MOD ≠ 0xFF), simplemente se entregan al gestor de protocolo AMPLIA. En cambio, los de lectura, detienen el envío de nuevos comandos hasta haber recibido todos los datos solicitados en un comando anterior.

Una función específica de la UCI (*BWC* o *Burst Write Control*) realiza escrituras en ráfagas sobre el recurso accedido. Esta función es particularmente útil para cargar tablas e inicializar contenidos de memorias, coeficientes de filtros, etc. La función lleva asociado en su campo DATA el número de palabras de 16 bits que siguen. Para direccionar el recurso en el que se va a escribir la ráfaga de datos, se realiza una escritura inicial con el primer valor de la ráfaga. A continuación se ejecuta la función BWC seguida de los datos a escribir en palabras de 32 bits. Si este número es impar, sólo son significativos los 16 LSB de la última palabra.

La ejecución deja de ser secuencial con funciones de bifurcación (salto condicional *JC cond, etiqueta* o arranque de secuencia *XSQ dir_mem*). La diferencia entre "*etiqueta*" y "*dir_mem*" es que, en el primer caso, la dirección de destino es el contenido del registro RLABEL[*etiqueta*], mientras que en el caso de "*dir_mem*", la dirección de salto es absoluta. La UCI dispone de 1024 registros de etiquetas denominados RLABEL[0] a RLABEL[1023]. La función *LBL num_label* guarda la dirección actual (+1) en el registro RLABEL[*num_label*].

Una primera forma de iniciar la ejecución de una secuencia es mediante la función *XSQ dir_mem* (ejecuta secuencia), donde el parámetro DATA=*dir_mem* indica la dirección absoluta de inicio de dicha secuencia. Esta función la emite normalmente el *host* pero, si se ejecuta desde una secuencia, tiene el mismo comportamiento que un comando de salto incondicional a la *dirección absoluta* indicada en el parámetro *dir_mem*.

Otra forma de iniciar una secuencia, cuando no se conoce la dirección absoluta, es creando una etiqueta (ejecutando la función *LBL num_label*) y realizando un salto incondicional a esa etiqueta (función *JC* 0, *num_label*). Esto permite guardar múltiples secuencias en memoria sin preocuparse de las posiciones que ocupan. Para ello, la función *LBL num_label* se envía desde el host inmediatamente antes de almacenar el primer comando de la secuencia, de forma que se ejecute con el valor actual del puntero a memoria. A continuación, se envía la secuencia de comandos para su almacenamiento, con llamadas a la función *WRSQ dato* (escritura en la memoria de secuencias).

La ejecución de una secuencia puede *suspenderse* transitoriamente de forma programada hasta la aparición de alguna condición (*SSQ condición*). Entre éstas están la espera a un evento de disparo o al final del procesamiento. Una vez producida esta condición, continúa la ejecución con el comando siguiente en la secuencia. Este mecanismo permite sincronizar el control del sistema a señales externas, a un periodo de repetición determinado, a que un *encoder* indique que se ha alcanzado una determinada posición, etc. La respuesta es inmediata con alguna latencia de algunos ciclos de reloj.

La ejecución de una secuencia se *interrumpe* únicamente mediante la programación a 1 del bit EXIT del registro de control. Esto no modifica el valor del puntero de ejecución, por lo que la secuencia se reanuda en el punto en que fue interrumpida poniendo EXIT:=0. Con esto y, tras verificar que no hay un procesamiento activo, puede modificarse desde el *host* el estado de los módulos y de la UCI.

Existen funciones que activan de forma simultánea la fase de adquisición y procesamiento en los módulos. Para ello, durante la fase inicial, se realiza un proceso de enumeración que asigna direcciones a los módulos y establece un retardo que ecualiza las latencias en el sistema [3]. Dado que los relojes están sincronizados mediante DLLs en cada FPGA, este mecanismo garantiza la simultaneidad dentro de un estrecho margen temporal en todo el sistema (<1 ns típico).

VI. LECTURA DE INFORMACIÓN

Durante la fase de adquisición y procesamiento el sistema envía resultados a la UCI, que son almacenados en la memoria DDR-3. Estos datos pueden ser leídos por el *host* en todo momento pero, para mejorar la eficiencia, se dispone de un mecanismo que evita accesos a los gestores de interfaz (*drivers* USB o Ethernet).

Para ello existen 4 modos de lectura no controlados por comandos, sino por 2 bits INFO del registro de control. En el *modo normal* (INFO = 00b), las lecturas realizadas desde el *host* acceden a la Memoria. En el *modo básico* (INFO = 01b), estas lecturas acceden a información básica (palabra de estado, número de datos y número de tramas completas disponibles en memoria. Los modos INFO= 1xb se reservan para ampliaciones.

El *modo básico* es particularmente interesante para que el *host* pueda conocer el estado de la UCI rápidamente. Antes de descargar datos de memoria (que se hará en el *modo normal*), el *host* debe conocer cuántos datos hay en memoria, para programar el número de transferencias y conocer el estado, operaciones que se realizan en el *modo básico*. En general, para mejorar la eficiencia, no se realizarán lecturas de memoria hasta que haya cierto volumen de datos disponibles.

Por otra parte, esta información se estructura en *tramas*, que son secuencias de datos enmarcadas entre una cabecera (4 palabras de 32 bits que incluyen la palabra de estado y la longitud de la trama actual) y el resto de información útil.

VII. GESTIÓN DE LA MEMORIA DDR-3

La memoria DDR-3 reparte su espacio de direcciones asignando el 75% al almacén de datos y resultados recogidos del sistema y el 25% restante a las necesidades del procesador y de las comunicaciones Ethernet. Esta distribución puede cambiarse para adaptarla a otras aplicaciones.

Para la parte de datos y resultados la memoria se comporta como una FIFO: los datos recibidos del sistema se almacenan secuencialmente en escritura y se leen a solicitud del *host*. Como en cualquier FIFO, lecturas y escrituras pueden ser "simultáneas", existiendo un arbitraje para acceder al único puerto físico de la memoria.

La Figura 3 muestra esquemáticamente el gestor de la memoria DDR-3. Dispone de 2 pequeños FIFOs, uno de entrada (FIFO_A) y otro de salida (FIFO_B) para acomodar las diferencias entre las tasas externas y las de acceso a memoria. Su tamaño es de 256x32 y, además, realizan la conversión de anchura de datos (sistemas de 16 bits en AMPLIA-3 y lecturas de 16 bits con USB-2.0).

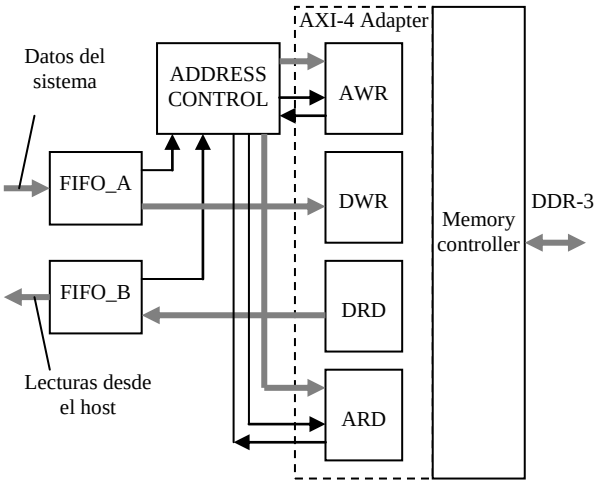

Fig. 3. Gestor de memoria DDR-3.

La FPGA utilizada, de la familia Spartan-6, dispone de controladores de memoria realizados en silicio, a los que se accede mediante IPs proporcionados por Xilinx que operan en el bus AXI-4.

El bloque *address control* realiza las operaciones de *handshake* del estándar AXI-4 y proporciona los parámetros para su funcionamiento. Entre otros, se han definido longitudes de ráfaga de 4, 8, 12 y 16 palabras de 32 bits para mejorar la eficiencia en las transferencias.

Los FIFOS auxiliares FIFO_A y FIFO_B entregan al bloque de control de direcciones los *flags* clásicos "*full*" y "*empty*", con los que se realiza el control. Así, FIFO_A se trata de mantener "vacío", esto es, cuando su contenido supera el tamaño de una ráfaga, se solicita al AXI-4 *Adapter* la escritura, proporcionando la dirección y los datos a los bloques AWR y DWR. El primero proporciona las señales de *handshake* para verificar la realización de la transferencia.

Por el contrario, FIFO_B se trata de mantener "lleno", de modo que las lecturas que realice el *host* puedan atenderse inmediatamente. Así, cuando dispone de espacio para albergar al menos una ráfaga, solicita al AXI-4 Adapter su lectura, proporcionando la dirección al bloque ARD y entregándose los datos por el bloque DRD. El bloque ARD gestiona las señales de *handshake* para verificar la ejecución de la lectura.

Para realizar estas operaciones es preciso conocer el volumen de datos almacenados en memoria en cada momento. Este dato se obtiene mediante un contador bidireccional que se incrementa con cada escritura y se decrementa con cada lectura. Como las lecturas y escrituras se realizan en ráfagas (mínimo 4 palabras de 32 bits), el reloj de accionamiento de este contador es relativamente lento.

VIII. ASPECTOS DE IMPLEMENTACIÓN

La UCI se ha implementado en el mini-módulo descrito en [1], sobre la FPGA Spartan-6 XC6SL100T de Xilinx. El mini-módulo dispone también de una memoria DDR-3 de 128 MB MT41J64M16LA de Micron, controlador de capa física Ethernet M88E1111 de Marvell, controlador USB-2.0 CY7C68013 de Cypress, memoria flash M25P64 para la configuración de la FPGA y memoria flash paralelo M29W128FH de ST Microelectronics. Este módulo tiene el tamaño de una tarjeta de crédito.

La memoria SDRAM se gestiona con el controlador de memoria dinámica de Xilinx, a través del bus AXI-4, con varios máster: UCI en lectura y escritura (descrito anteriormente), procesador Micro-Blaze y gestor DMA para la G-Ethernet.

La CPU Micro-Blaze se utiliza casi exclusivamente para gestionar las comunicaciones por el enlace G-Ethernet. No interviene para las comunicaciones USB, que se gestionan directamente desde la UCI y por medio del controlador CY7C68013 de Cypress.

La Fig. 4 muestra los recursos utilizados de la FPGA (aspectos más destacados). Se comprueba un bajo consumo de recursos lógicos y registros, ocupando apenas el 4% de los *slices* disponibles en esta FPGA.

La mayor utilización es de la memoria BRAM, que alcanza un 47% de los recursos disponibles. Esto se explica tanto por la memoria almacén de secuencias de comandos, como por los FIFOs asociados a la entrada y salida de datos.

Por otra parte, la actual implementación es preliminar de modo que la versión final aumentará ligeramente el consumo de recursos.

IX. DISCUSIÓN Y APLICACIÓN

Cabe preguntarse por qué no se utiliza el procesador Micro-Blaze como CPU para implementar la UCI. La respuesta es simple: por razones de rapidez.

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                1,080 out of 126,576    1%
    Number used as Flip Flops:                970
    Number used as Latches:                   110
    Number used as Latch-thrus:                 0
    Number used as AND/OR logics:               0
    Number of Slice LUTs:                   1,235 out of  63,288    1%
    Number used as logic:                   1,189 out of  63,288    1%
    Number using O6 output only:              735
    Number using O5 output only:               82
    Number using O5 and O6:                   372
    Number used as ROM:                         0
    Number used as Memory:                      0 out of  15,616    0%
    Number used exclusively as route-thrus:    46
    Number with same-slice register load:      40
    Number with same-slice carry load:          6
    Number with other load:                     0

Slice Logic Distribution:

  Number of occupied Slices:                  708 out of  15,822    4%
  Number of LUT Flip Flop pairs used:       1,562
  Number with an unused Flip Flop:          587 out of   1,562   37%
  Number with an unused LUT:                327 out of   1,562   20%
  Number of fully used LUT-FF pairs:        648 out of   1,562   41%
  Number of slice register sites lost
      to control set restrictions:              0 out of 126,576    0%

IO Utilization:

  Number of bonded IOBs:                       71 out of     296   23%
    Number of LOCed IOBs:                      45 out of      71   63%
    IOB Latches:                               17

Specific Feature Utilization:
  Number of RAMB16BWERs:                     126 out of     268   47%
  Number of RAMB8BWERs:                        0 out of     536    0%
  Number of BUFIO2/BUFIO2_2CLKs:               1 out of      32    3%
```

Fig. 4. Resumen de recursos utilizados.

En efecto, ciertas aplicaciones requieren cargar rápidamente los parámetros para una nueva adquisición de forma que se pueda operar con elevadas tasas de repetición del ciclo adquisición-procesamiento.

Así, en las aplicaciones de imagen ultrasónica, por ejemplo, es frecuente tener que cargar nuevos juegos de leyes focales (retardos de enfoque para emisión y recepción), parámetros de adquisición, apertura dinámica, etc. antes de realizar un nuevo ciclo.

Esto supone programar, en el caso más desfavorable, del orden de 2K parámetros de 16 bits por canal. La UCI puede hacer la programación a razón de 1 parámetro por ciclo de reloj (a 40 MHz, como en AMPLIA-3, 50 µs/canal y en AMPLIA-4, con transferencias a 1 GB/s, en 4 µs/canal). Un procesador convencional no puede operar a estas velocidades y, además, los tiempos no serían predecibles al estar sometido a interrupciones de diversa índole.

Por otro lado, al estar la UCI asociada a una cadena de procesamiento (de hecho es el último módulo de la cadena), pueden incorporarse múltiples funciones de procesamiento específico, aplicadas a los resultados que han proporcionado los módulos. Para estas funciones se requiere disponer de recursos libres.

Por ejemplo, en la aplicación actual de imagen ultrasónica, el último módulo realiza funciones de:

- Filtrado arbitrario de la señal para reducir ruido (filtros paso-banda de 64 coeficientes).
- Detección digital de la envolvente mediante filtros de Hilbert y extracción del módulo de la señal analítica.
- Filtrado de interferencias electromagnéticas (filtro EMI) y cancelación de ruido impulsivo (filtros no lineales).
- Promediado de adquisiciones.
- Compresores de datos y detectores de picos.
- Otras funciones auxiliares: seguimiento de codificadores de posición, disparo externo e interno periódico, generador de señales de test, etc.

CONCLUSIONES

Los sistemas de adquisición y procesamiento de datos que requieren una temporización estricta, deben ser controlados por un elemento independiente de un computador de operación. Dicho elemento o Unidad de Control e Interfaz (UCI) puede operar a alta velocidad en la programación de parámetros y recepción de resultados y estado del sistema.

Generalmente cada sistema se dirige a una aplicación particular y, convencionalmente, su control se realiza bien internamente mediante un SoC o lógica específica o

desde un procesador externo o host. Así, cada sistema requiere el diseño de su propio controlador, uno de los aspectos menos tratados en la literatura especializada.

En este trabajo se ha presentado un modelo de UCI de propósito general, cuya programación facilita la gestión de cualquier sistema adaptado al bus de salida. Opera en forma desacoplada del host que, una vez enviado el programa a la UCI, puede limitarse a obtener los resultados de la adquisición y procesamiento del sistema (una forma de control desatendido).

En su relación con el sistema, la UCI opera con solicitudes de lectura y escritura, una estructura que se adapta bien a arquitecturas segmentadas y modulares. Dispone de mecanismos de numeración de módulos y de compensación de latencias para la ejecución simultánea de comandos globales en distintos puntos o módulos del sistema.

Más que un diseño particular, este trabajo describe una metodología para intentar generalizar la realización de controladores de sistemas operando en tiempo real estricto.

AGRADECIMIENTOS

El presente trabajo ha sido financiado por el proyecto CICYT DPI 2010-17648.

REFERENCIAS

[1] Jose Miguel Moreno, Manuel Sánchez, J.F. Cruza y Luis Medina, Mini-Módulo HW de altas prestaciones, basado en Spartan6 con análisis de integridad de señal. *JCRA 2012*.

[2] C. Fritsch, T. Sánchez, D. Jiménez, O. Martínez, "AMPLIA: Una Arquitectura Modular para Procesamiento y Despliegue de Imagen Acústica", *Proc. Acústica 2000*, pp. 1-6, 2000.

[3] J. Camacho, O. Martínez, M. Parrilla, R. Mateos, C. Fritsch, "A Strict-time Distributed Architecture for Digital Beamforming of Ultrasound Signals", *Proc. IEEE Int'l Symp. Intelligent Signal Processing*, pp. 451-456, 2007.