Plataforma Heterogénea Hw/Sw para Sistemas Multimedia Empotrados basada en OpenMAX

David de la Fuente[1] , Jesús Barba, Julio D. Dondo, Fernando Rincón y Juan C. López

Resumen— **En los sistemas multimedia empotrados, el rendimiento del sistema de interconexión es clave para alcanzar los requisitos del sistema, ya que éste debe cumplir unas fuertes restricciones temporales y manejar una gran cantidad de datos.**

Dado que los buses son una infraestructura de comunicación ampliamente utilizada en SoCs, este trabajo describe una plataforma multimedia Hw/Sw que ofrece unos mecanismos de comunicación eficientes y optimizados para sistemas basados en una arquitectura de bus, cuyos parámetros de comunicación pueden ser configurados dinámicamente para mejorar el rendimiento final del sistema. Estos mecanismos están inspirados en un estándar abierto para sistemas multimedia, el cuál ha sido adaptado y migrado a las necesidades específicas de los sistemas empotrados. Cualquier componente de un sistema multimedia, implementado sobre esta plataforma, se comunicará con el resto del sistema de manera transparente, homogénea y descentralizada, con una mínima sobrecarga añadida en la infraestructura de integración.

I. Introducción

En general, los sistemas multimedia tienen que hacer frente al procesado de grandes cantidades de datos con unas fuertes restricciones de tiempo. Con el fin de alcanzar estos requisitos, determinadas partes de la funcionalidad del sistema han sido tradicionalmente implementadas mediante circuitos electrónicos a medida. Esto ha supuesto una mejora del rendimiento y de la capacidad de procesamiento.

Las primeras aproximaciones que tratan con la aceleración hardware hacen uso de ASICs (Application Specific Integrated Circuit) o DSPs (Digital Signal Processors). Últimamente, la aparición de dispositivos reconfigurables de alta capacidad, como las FPGAs (Field Programmable Gate Array), ha favorecido el uso de esta tecnología por parte de los desarrolladores de sistemas multimedia empotrados. Algunas de las razones que justifican esta tendencia son: la flexibilidad y la capacidad para el desarrollo rápido de prototipos, la posibilidad de personalizar el hardware, el bajo coste con respecto otros dispositivos programables, etc.

Sin embargo, existe una falta de estandarización en muchas facetas del desarrollo de los sistemas multimedia empotrados como por ejemplo, los mecanismos de intercambio de datos entre los elementos Hw y Sw del sistema. Normalmente, en el ámbito de los sistemas empotrados, esto supone realizar implementaciones ad-hoc, mientras que si se utilizasen estándares obtendríamos diseños flexibles, reutilizables y portables.

En este trabajo se presenta una plataforma Hw/Sw para sistemas multimedia empotrados basada en la utilización del estándar OpenMAX [1]. El objetivo de dicha plataforma es proporcionar una infraestructura y mecanismos que permitan el desarrollo de sistemas completamente reutilizables y portables, en un tiempo razonable.

OpenMAX es un estándar abierto, promovido por el grupo Khronos, que persigue reducir el coste y la complejidad de migrar software multimedia a nuevas arquitecturas y procesadores. Debido a que la implementación de referencia de OpenMAX es software, ha sido necesario revisar y rediseñar todos los conceptos y protocolos del estándar para poder aplicarlos al entorno de los sistemas multimedia empotrados. OpenMAX proporciona una API que permite portar tanto componentes como aplicaciones completas a múltiples sistemas operativos y plataformas.

Como se ha mencionado previamente, la plataforma propuesta permite el desarrollo de sistemas multimedia Hw/Sw y por lo tanto, es necesario un mecanismo que gestione de manera transparente y eficiente la comunicación Hw/Sw. Además, para hacer frente tanto a los requisitos temporales como computaciones, es obligatorio que la sobrecarga introducida por la infraestructura de integración sea la mínima posible. Para conseguirlo, nuestra propuesta adopta las facilidades ofrecidas por OOCE (Object Oriented Communication Engine, [2]). OOCE es un middleware híbrido para SoCs basado en buses que proporciona los servicios básicos de gestión transparente de la comunicación entre las partes hardware y software de un sistema empotrado. Para la consecución de este trabajo, se han extendido las funcionalidades de OOCE para cumplir con las necesidades concretas de los sistemas multimedia y lograr los niveles de rendimiento requeridos.

El resto del documento se estructura de la siguiente manera: La sección 2 ofrece una visión general de los trabajos relacionados. En la sección 3 se realizará una breve descripción de OpenMAX y su aplicación en el campo de los sistemas empotrados. En la sección 4 se analizará nuestra propuesta y, en la sección 5 se explicará la especialización de los mecanismos de comunicación y sincronización, basados en una transferencia de datos descentralizada Hardware-to-Hardware, y determinadas optimizaciones aplicables a dichos mecanismos. Seguidamente, en la sección 6 se expondrán una serie de resultados experimentales. Por último, se enumerarán una serie de conclusiones en la sección 7.

[1]Dpto. de Tecnologías y Sistemas de Información, Univ. de Castilla la Mancha, e-mail: david.fuente@uclm.es

II. Trabajos relacionados

Las aplicaciones multimedia tienen sus propios requisitos que las diferencian del resto de aplicaciones de otros dominios. Algunos de estos requisitos son, por ejemplo, el intenso tráfico de datos, el uso intensivo del subsistema de memoria, un modelo característico de aplicación, restricciones de tiempo real, etc. Estas características particulares han llevado a la comunidad de investigadores a invertir un gran esfuerzo en proyectos que abordan los retos específicos impuestos por el desarrollo de los sistemas multimedia empotrados.

La solución más extendida para mejorar el rendimiento pasa por el desarrollo de plataformas multimedia con aceleración hardware. Esta solución es la mencionada en [3] y [4], que implementan, respectivamente, núcleos computaciones (similares a los coprocesadores) y motores reconfigurables heterogéneos.

El procesador Cell [5], con sus cuatro buses característicos en forma de anillo, combina una arquitectura PowerPC de propósito general con unidades de co-procesamiento que aceleran aplicaciones multimedia y de procesamiento vectorial.

Recientemente, las arquitecturas multi-core destinadas a la ejecución de aplicaciones gráficas (como las matrices de GPUs de NVidia), han adquirido una gran importancia debido principalmente por la aparición de CUDA [6], un marco abierto de desarrollo software basado en C.

PeaCE [7] es otra alternativa para desarrollar plataformas multimedia Hw/Sw. PeaCE proporciona un flujo de desarrollo (desde la simulación funcional hasta el proceso de síntesis) para aplicaciones multimedia con restricciones de tiempo real.

Una característica importante a tener en cuenta en estas plataformas es cómo están comunicados los componentes del sistema. Generalmente, los mecanismos de comunicación son fijos o con una capacidad limitada de parametrización. Ya que existen numerosos factores que pueden afectar al rendimiento de la comunicación, ofrecer la capacidad de reorganizar los canales de comunicación en tiempo de ejecución es esencial. Esto también es una herramienta muy útil para los desarrolladores ya que pueden realizar una rápida y sencilla exploración del espacio de diseño.

Por ejemplo, Na Ra Yang, entre otros, propone en [8] diferentes técnicas como doble buffer, memorias entrelazadas y *open row access* para conseguir una comunicación más eficiente. Otra alternativa para mejorar el rendimiento de la comunicación es la descrita en [9] que presenta una técnica novedosa que permite el uso multiplexado de memoria.

En relación con el análisis del rendimiento de la comunicación en plataformas multimedia dedicadas, cabe destacar el trabajo de [10] donde se comparan diversos mecanismos de intercambio de datos mediante memoria compartida.

III. Aplicaciones Multimedia en OpenMAX

Típicamente, una aplicación multimedia basada en OpenMAX esta formada por una cadena de elementos de procesamiento denominados Componentes OpenMAX (a partir de ahora Componente OMX) que procesan los datos de entrada, generando nuevos datos en su salida. Un Componente OMX implementa una o más funciones de procesamiento multimedia pertenecientes a uno de los cuatro dominios de aplicación (audio, vídeo, imagen y otro) tal y como define el estándar. Una de las capas del middleware OpenMAX es la capa de integración (OpenMAX Integration Layer (IL),[11]) que abstrae y unifica el acceso a la funcionalidad encapsulada en un Component OMX y proporciona diversas funciones: (a) establece la comunicación entre dos Componentes OMX; (b) envía comandos a los respectivos componentes; (c) configura los parámetros de los Componentes OMX y (d) obtiene los recursos necesarios.

La unidad mínima de transferencia entre dos Componentes OMX es el denominado buffer, una estructura que encapsula los datos multimedia. La comunicación entre dos componentes es gestionada a través de los puertos de comunicación. En función del rol que juegue el componente en la comunicación (productor o consumidor), un puerto puede clasificarse como puerto de entrada o de salida. OpenMAX describe tres tipos de comunicación:

- Comunicación Tunelada: La comunicación tiene lugar entre dos Componentes OMX de manera directa, sin intervención de ningún otro elemento del sistema. Con este fin, OpenMAX define un protocolo de intercambio de buffers.
- Comunicación No-Tuneleda: La comunicación entre dos componentes se produce a través de una entidad que implementa el control de la aplicación.
- Comunicación Propietaria: Este tipo de comunicación se produce entre dos componentes de manera directa pero el mecanismo de intercambio de buffers no está definido en el estándar.

La IL alberga algunas de las funciones más importantes de OpenMAX como la de inicialización y conexión de los Componentes OMX y la gestión de la sincronización. Es por ello que la implementación hardware de sus principales componentes contribuirá a aumentar el rendimiento de una aplicación multimedia basada en nuestra solución.

A. Implementación Hardware de OpenMAX

El principal esfuerzo para adaptar OpenMAX al entorno de los sistemas empotrados ha sido identificar las partes del estándar que deberían ser implementadas en Hw con el objetivo de conseguir la máxima eficiencia en las comunicaciones de datos, apoyándonos en las facilidades que ofrece OOCE.

La entidad más importante de OpenMAX IL, desarrollada en Hw ha sido el Componente OMX (OMX Hw) ya que encapsula la parte funcional del

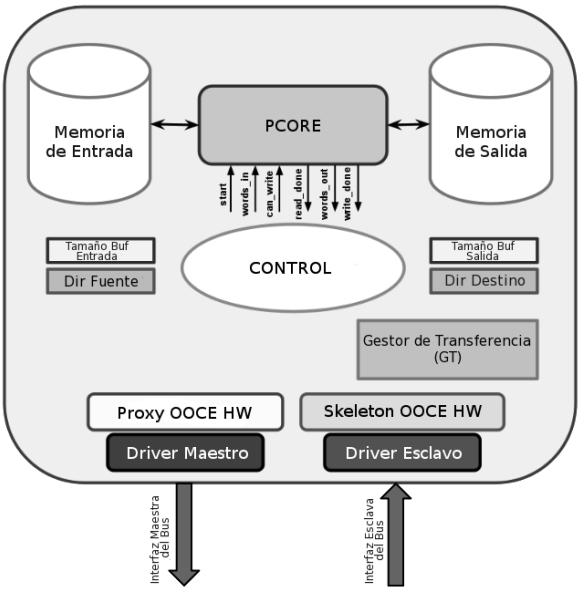

Fig. 1. Estructura de un Componente OMX Hw

sistema y de la aplicación. En el Componente OMX Hw la función de procesamiento multimedia está contenida en un módulo Hw denominado Proccesing Core (PCore) que interactúa con el resto de la lógica a través de una interfaz física pre-fijada, facilitando así su reutilización. La parte encargada de las comunicaciones con otros componentes se genera automáticamente y contiene una instancia de PCore y dos memorias locales donde se almacenarán tanto los *buffers* a procesar como los generados.

El responsable del intercambio de buffers entre Componentes OMX Hw y/o componentes de distinta naturaleza (Hw o Sw) es el Gestor de Transferencia (GT). El GT realiza la función de los puertos de comunicación descritos en el estándar OpenMAX y utiliza los registros *Fuente y Destino* para establecer la comunicación con los componentes anterior y posterior en la cadena de procesamiento. El GT también implementa los tres mecanismos de comunicación que se definen en OpenMax.

Aunque dos Componentes OMX Hw pueden comunicarse de tres manera diferentes (Tunelada, No-Tunelada y Propietaria), el mecanismo más óptimo para intercambiar buffers es el Tunelado porque la sincronización, el control y el tráfico de datos son manejados directamente por los dos componentes involucrados en el proceso de comunicación. Por este motivo, en este trabajo nos centraremos en la adaptación realizada de la comunicación Tuneleda. En nuestra implementación, esto significa que este proceso se lleva a cabo entre dos instancias de Componentes OMX Hw, sin la intervención de un controlador o cualquier tipo de proceso software. Ello supone un impacto positivo en el rendimiento, ya que el procesador no media en cada transferencia que tiene lugar en el sistema. OOCE hace posible este tipo de comunicación gracias a la semántica de las invocaciones Hardware-to-Hardware (basada en invocaciones a métodos remotos). Por otra parte, el uso de OOCE en nuestra propuesta enriquece la solución final al dotar a la plataforma de transparencia y homogeneidad desde el punto de vista de la comunicación dentro del chip.

Para mantener el correcto funcionamiento del Componente OMX Hw, se ha introducido la Unidad de Control que suministra los comandos OpenMAX a las entidades correspondientes y gestiona la comunicación interna del componente. El Componente OMX Hw recibe y envía dichos comandos a través de los adaptadores de OOCE (*proxies, skeletons y drivers*). Dichos adaptadores hacen al Componente OMX Hw independiente de la plataforma destino y del canal de comunicación. Además, OOCE establece la semántica para garantizar la transparencia y la interoperabilidad entre los Componentes OMX (Hw o Sw).

IV. La propuesta

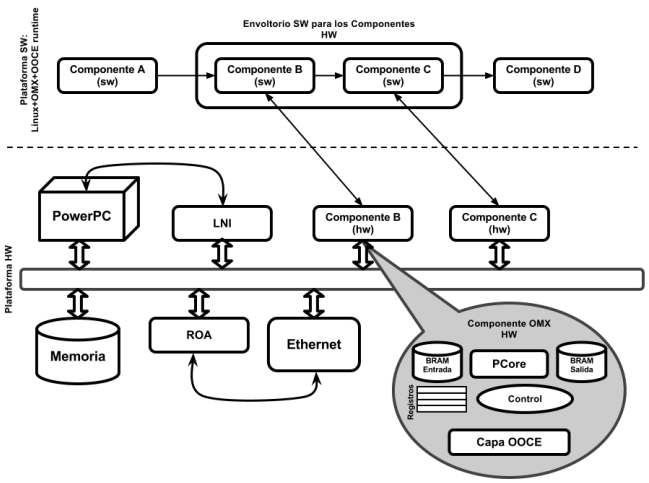

Fig. 2. Ejemplo de una cadena de procesamiento multimedia Hw/Sw con Componentes OMX Hw.

Nuestra propuesta está basada en el uso de las FPGAs y un estándar industrial para el diseño e implementación de sistemas multimedia. Esto permite una implementación flexible y portable de sistemas multimedia empotrados en un corto espacio de tiempo.

Para tener una visión global, en la figura 2 está representada una cadena de procesamiento basada en nuestra propuesta. En el ejemplo, los componentes B y C están asignados a una implementación Hw de un Componente OMX y pueden comunicarse utilizando cualquier modelo de comunicación descrito en OpenMAX. La comunicación entre dos Componentes OMX Sw no va a ser tratada en este trabajo y se basa en la implementación de referencia ofrecida por el grupo Khronos. Por lo tanto, nos centraremos en los escenarios donde estén presentes versiones Hw de los Componentes OMX.

Todo Componente OMX Hw necesita un homólogo implementado en Sw que se encargará de interactuar con el resto del middleware de OpenMAX y otros Componentes OMX, actuando como una fachada que redirige al Componente OMX Hw las interacciones

recibidas. Se consigue así transparencia respecto al resto de la aplicación que puede mantenerse intacta sin tener que hacer cambios.

Para ilustrar el significado de transparencia e integración de un Componente OMX Hw, se introduce un fragmento de código 1 de una aplicación OpenMAX de ejemplo, ejecutándose en una FGPA. La aplicación recibe un fotograma (a través de la interfaz Ethernet), transforma la imagen RGB a escala de grises, le aplica un algoritmo de detección de bordes y finalmente devuelve el resultado por la Ethernet. Los componentes *img_ethreader* y *image_ethsink* son los encargados de la adquisición de la imagen para ser procesada y el envío del resultado respectivamente. Estos componentes se ejecutan en Sw, mientras que los componentes *hw.img_RGB2BW* y *hw.img_sobel* están implementados en hardware. Sin embargo, no hay diferencia entre el uso de los componentes hardware y los software ya que ambos son compatibles con el estándar.

```
int main(int argc, char** argv) {
 /*Getting Components Handler*/
 OMX_GetHandle(&appPriv->ethreader, ``omx.
     ext.image.ethreader'', NULL,
                 &readercallbacks);
 OMX_GetHandle(&appPriv->hwrgb2bw, ``omx.hw
     .img_RBG2BW'', NULL,
                 &RGB2BWcallbacks);
 OMX_GetHandle(&appPriv->hwsobel, ``omx.hw.
     img_sobel'', NULL,
                 &hwsobelcallbacks);
 OMX_GetHandle(&appPriv->ethsink, ``omx.ext
     .image.ethsink'', NULL,
                 &sinkcallbacks);
 /*Set the size for img Hw OMX Components*/
     ;
 sSize.sWidth.nValue = 640;
 sSize.sHeigth.nValue = 480;
 OMX_SetParameters(&appPriv->hwrgb2bw,
     EXT_OMX_IndexConfigImgSize,
                 &sSize);
 OMX_SetParameters(&appPriv->hwsobel,
     EXT_OMX_IndexConfigImgSize,
                 &sSize);
 /*Setting up tunneled communication*/
 OMX_SetupTunnel(appPriv->hwrgb2bw, 1,
     appPriv->hwsobel,0);
 /* Change HW OMX Component state */
 OMX_SendCommand(appPriv->hwrgb2bw,
     OMX_CommandStateSet,
                 OMX_StateIdle, NULL);
 OMX_SendCommand(appPriv->hwsobel,
     OMX_CommandStateSet,
                 OMX_StateIdle, NULL);
 OMX_SendCommand(appPriv->hwrgb2bw,
     OMX_CommandStateSet,
                 OMX_StateExecuting, NULL)
                     ;
 OMX_SendCommand(appPriv->hwsobel,
     OMX_CommandStateSet,
                 OMX_StateExecuting, NULL)
                     ;
 ...
 OMX\_DeInit();
 return 0;
}
```

Listing 1
EJEMPLO DE UNA IMPLEMENTACIÓN DE UNA CADENA DE PROCESAMIENTO MULTIMEDIA HW/SW CON COMPONENTES OPENMAX HW.

En segundo plano, hay muchas cosas que suceden relacionadas con los componentes hardware. Por ejemplo, en la figura 3, se representa la secuencia de mensajes utilizados para configurar el Componente OMX Hw. El diagrama muestra como el Componente OMX Sw, usado como fachada, extiende el comportamiento de los mensajes estandarizados que recibe, generando invocaciones hacia el componente hardware. Tales mensajes son traducidos por OOCE en transacciones por el bus de comunicación, y llegan al Componente OMX Hw a través de los adaptadores de OOCE.

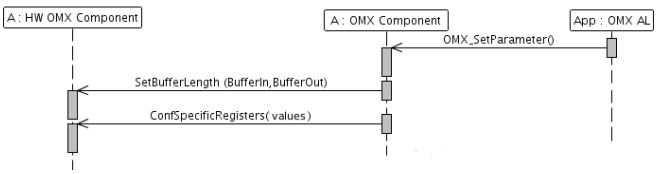

Fig. 3. Diagrama con la secuencia de mensajes de configuración de un Componente OMX Hw.

De igual forma, cuando un Componente OMX recibe una notificación para establecer una comunicación tuneleda o para comenzar su proceso de ejecución, extiende estas notificaciones al Componente OMX Hw. Una vez configurados y puestos en estado de ejecución, dos Componentes OMX Hw pueden comenzar a intercambiar buffers a través de la comunicación basada en la semántica de invocación Hardware-to-Hardware que se describe a continuación.

V. ESTRATEGIA DE INTERCAMBIO DE BUFFERS HARDWARE-TO-HARDWARE

La semántica de invocación Hardware-to-Hardware, en la cual basamos la comunicación (comunicación Hardware-to-Hardware), es un convenio que engloba el intercambio de buffers y un mecanismo de sincronización. Este convenio es una adaptación del modelo de comunicación Tunelado de OpenMAX para nuestra plataforma.

La comunicación cumple con el patrón productor/consumidor y los componentes involucrados en la comunicación intercambian mensajes (transacciones por el bus) del tipo "LlenaBuffer" y "VaciaBuffer".

El objetivo de la comunicación Hardware-to-Hardware es el de transferir, tan pronto como sea posible, el contenido de un buffer entre dos Componentes OMX Hw conectados a través de un bus, sin la intervención de ningún otro componente del sistema. La secuencia de pasos (ilustrada en la figura 4) para el intercambio de buffers con el modelo de comunicación Hardware-to-Hardware es la siguiente:

- El Componente OMX Hw (consumidor) solicita el llenado de su memoria de entrada (enviando un mensaje "LlenaBuffer") al componente que ejerza el rol de productor en la comunicación.
- El Componente OMX Hw productor transfiere el contenido de un buffer a la memoria de entrada del consumidor en estas condiciones: existe al

menos un buffer en su memoria de salida y ha recibido una petición "LlenaBuffer".

- El productor envía un un mensaje del tipo "VaciaBuffer" al consumidor para notificarle que dispone de nuevos datos en su memoria de entrada.
- El Componente OMX Hw consumidor comienza a leer los datos del nuevo buffer de su memoria de entrada y en cuanto está listo para recibir nuevos datos, vuelve a realizar la acción del primer paso.

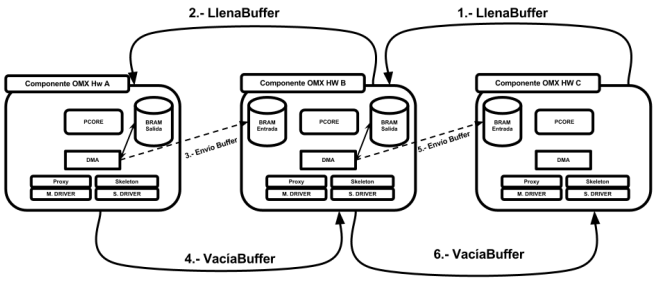

Fig. 4. Secuencia de mensajes en la comunicación Hw-to-Hw.

Sobre este modelo de comunicación genérico, podemos aplicar dos optimizaciones distintas que han sido desarrolladas para reducir la latencia de transmisión de los buffers por el bus y maximizar el ancho de banda disponible. Con las optimizaciones propuestas, el número de ciclos de espera perdidos ante una posible congestión del bus se reducen, redistribuyendo la carga del bus con técnicas que paralelizan e intercalan las transferencias entre varios Componentes OMX Hw. Esto se consigue aprovechando los tiempos muertos que se producen cuando un componente está procesando un buffer.

La primera optimización (denomina "No Esperar a Llenar el Buffer", figura 5) transfiere paquetes de N-palabras tan pronto como estén disponibles en su memoria de salida. La ventaja de usar esta técnica es doble, ya que:

- Se solapa el procesamiento, por parte del Componente OMX Hw, con la transferencia de su buffer de salida.
- Se pueden producir varias transferencias de buffers al mismo tiempo.

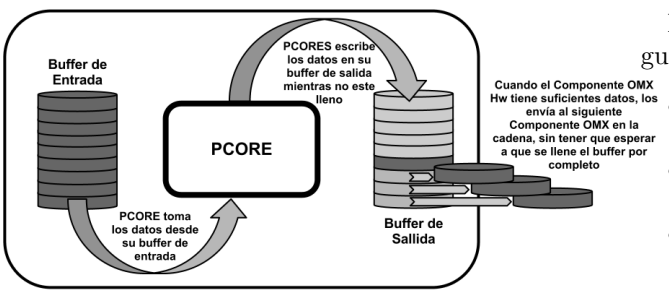

Fig. 5. Optimización "No Esperar a Llenar el Buffer".

La denominada "Multiples Buffers", figura 6, es la segunda de las optimizaciones y permite paralelizar: (1) la lectura del buffer de entrada actual con la escritura de un nuevo buffer de entrada; y (2) la transmisión del buffer actual de salida con la escritura de un nuevo buffer de salida. Para este fin, el espacio de direcciones físicas de las memorias locales está divido en N regiones distintas e independientes, y de este modo se consigue un paralelismo total.

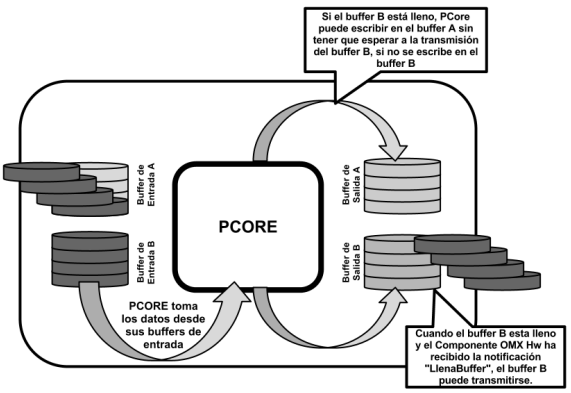

Fig. 6. Optimización "Multiples Buffers".

Ambas optimizaciones son compatibles y pueden usarse simultáneamente, pero dependiendo del escenario concreto su efectividad puede variar. No se va a entrar en la discusión sobre si es idóneo el uso de un bus cuando el ancho de banda de la aplicación excede de la capacidad del bus, y nos centraremos en los escenarios en los que la infraestructura de bus es capaz de absorber el tráfico generado por la aplicación multimedia.

Existen muchos parámetros de la comunicación que pueden influir en el rendimiento de la misma, por eso, a parte de las optimizaciones descritas, la plataforma permite para cada componente configurar: el tamaño de buffer, tamaño de las ráfagas, número de componentes conectados al bus, el tamaño de los datos transmitidos, etc.

VI. Resultados Experimentales

Para proporcionar al lector una idea de los beneficios y la eficiencia del enfoque propuesto, en esta sección vamos a detallar dos experimentos, los cuales se centran en analizar el comportamiento de las técnicas de "Multiples Buffers" y "No Esperar a Llenar el Buffer" y observar como pueden ayudar a aumentar el rendimiento del sistema.

Para ambos experimentos, se parte de una configuran base que consta de:

- Un reloj del sistema con una frecuencia de 100MHz.
- Un modelo SystemC del bus PLB proporcionado por GreenSoCs [12].
- Las memorias locales de todos los componentes son de 1K palabras.
- Un fichero de entrada con 1Mpalabras. Las simulaciones van a consistir en alimentar la cadena de procesamiento (variando el número de componentes) con los datos sintéticos contenidos en el fichero de entrada.

Se han realizado varias simulaciones para diferentes configuraciones de la cadena de procesamiento y valores de los parámetros de configuración. Merece la pena mencionar que en este caso se ha utilizado el modelo del bus PLB, pero se puede realizar nuevas estimaciones sobre otras plataformas, de manera casi directa, cambiando únicamente el modelo TLM del bus.

La primera imagen (figura 7) trata de ilustrar los beneficios obtenidos tras aplicar la técnica de "No Esperar a Llenar el Buffer" (descrito en la sección V). Como era de esperar, la ocupación de bus aumenta con el número de componentes en la cadena de procesamiento. Lo significativo es ver como nuestra propuesta escala linealmente. Los tiempos de transmisión se han reducido en torno al 20% para los seis casos en los que se compara con la estrategia de "Esperar al Llenado del Buffer".

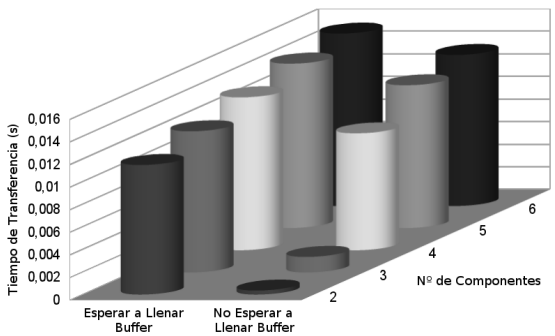

Fig. 7. Gráfica comparativa del tiempo base de transmisión, en base al número de componentes, aplicando la optimización de "No Esperar a Llenar el Buffer".

La segunda imagen (figura 8) representa cómo influye en el tiempo de transmisión total el tamaño de los buffers. Los beneficios de usar la técnica de "Multiples Buffers" se exponen en estos experimentos donde se delimita su aplicación a escenarios donde su uso tiene sentido.

Debido a que las memorias locales son de 1Kpalabras, el número de buffers que se pueden almacenar en ellas varia (por ejemplo, se pueden configurar dos buffers de 512 palabras cada uno). En esta gráfica, podemos observar cual es el mejor tamaño de buffer para una determinada configuración. Incrementar el número de buffers implica aumentar la lógica de control, por tanto es necesario hacer una estimación de coste/rendimiento para valorar el resultado.

De la figura se pueden extraer dos conclusiones: (1) en sistemas con una baja sobrecarga en base a los requisitos de ancho de banda del bus, cuanto mayor sea el tamaño del buffer se obtendrá un mayor rendimiento; mientras que (2) en el escenario contrario es preferible reducir el tamaño del buffer ya que se pueden intercalar las transmisiones de varios buffers, reduciendo los tiempos de espera.

Agradecimientos

Esta investigación ha sido financiada por el Ministerio Español de Ciencia e Innovación bajo el proyecto DREAMS (TEC2011-28666-C04-03), y por el Ministerio Español de Industria y el Centro para el Desarrollo Tecnológico Industrial bajo el proyecto ENERGOS (CEN-20091048).

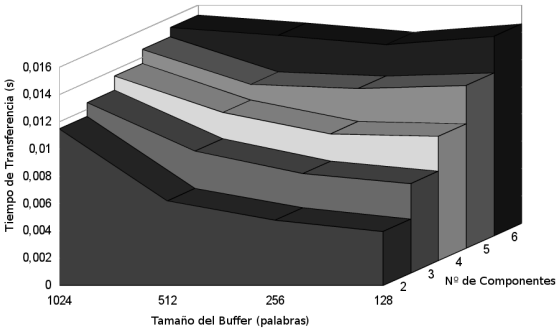

Fig. 8. Gráfica comparativa del tiempo de transferencia, en base al número de componentes, aplicando la optimización "Multiples Buffers".

Referencias

[1] The Khronos Group website, http://www.khronos.org/openmax/, Última visita en Abril de 2012.

[2] J. Barba, F. Rincón, F. Moya, J.D. Dondo, F.J. Villanueva, D. Villa, J.C. López. *OOCE: Object-Oriented Communication Engine for SoC Design*, DSD - Euromicro Conference on Digital System Design. Lübeck (Germany), 2007.

[3] Martijn J. Rutten, Jos T.J. van Eijndhoven, Evert-Jan D.Pol, Egbert G.T. Jaspers, Pieter Van der Wolf, Om Prakash Gangwal and Adwin Timmer. *Eclipse: A Heterogeneous Multiprocessor Architercture For Flexible Processing*, Philips Research Laboratories, 2002.

[4] Paul Brelet, Arnaud Grasset, Philippe Bonnot, Frank Ieromnimon and Dimitrios Kritharidis. *System Level Design for Embedded Reconfigurable Systems using MORPHEUS platform* , IEEE Annual Symposium on VLSI, 2010.

[5] Kahle J. A., Day M.N., Hofstee H. P., Johns C. R., Maeurer T. R. and Shippy D. *Introduction to the Cell Multiprocessor*, IBM Journal of Research and Development. 2005.

[6] Buck, Ian. *GPU computing with NVIDIA CUDA*, ACM SIGGRAPH'07. 2007. San Diego (California).

[7] Soonhoi Ha, Sungchan Kim, Youngming Yi, Seongnam Kwon and Young-pyo Joo. *PeaCE: A Hardware-Software Codesign Environment for Multimedia Embedded Systems.* ACM Transactions on Design Automation of Electronic Systems, August 2007.

[8] Na Ra Yang, Gilsang Yoon, Jeonghwan Lee, Intae Hwang, Cheol Hong Kim, Sung Woo Chung, Jong Myon Kim. *Improving the System-on-a-Chip Performance for Mobile Systems by Using Efficient Bus Interface* cmc, vol. 2, pp.606-608, 2009 WRI International Conference on Communications and Mobile Computing, 2009.

[9] C. Brunelli, F. Garzia, C. Giliberto, and J. Nurmi. *A dedicated DMA logic addressing a time multiplexed memory to reduce the effects of the system bus bottleneck*, Proc. FPL, 2008, pp.487-490.

[10] Lahiri, Kanishka and Raghunathan, Anand and Dey Sujit. *Fast performance analysis of bus-based system-on-chip communication architectures*, Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design (ICCAD '99).

[11] The Khronos Group Inc. *OpenMAX Integration Layer Application Programming Interface Specification*, Version 1.1.1. 2007.

[12] The GreenSoCs website, http://www.greensocs.com/ Última visita en Abril de 2012.