Comunicación Hardware-Software Basada en Objetos para Sistemas Empotrados

Jesús Barba, Fernando Rincón, Julio D. Dondo,
David de la Fuente, Francisco Moya y Juan Carlos López
Escuela Superior de Informática
Univ. de Castilla-La Mancha
Ciudad Real
jesus.barba@uclm.es

Resumen

La programación de cualquier sistema electrónico que combine SW y HW es uno de los aspectos más críticos en el desarrollo de este tipo de sistemas. La mayor parte de los recursos y tiempo invertidos se destina a la codificación, verificación y revisión de lo que se ha denominado *Software dependiente del Hardware* (SdH)

Iniciar cuanto antes el desarrollo del SdH, clave para el éxito de cualquier proyecto, sólo es posible utilizando modelos de alto nivel que abstraen la interfaz entre el SW y el HW.

En este artículo, se describe la infraestructura desarrollada para sistemas empotrados (en particular y a modo de ejemplo, en los basados en µcLinux y MicroBlaze), la cual permite interactuar con el HW desde una perspectiva de orientación a objetos. La generación de dicha infraestructura se realiza de manera semi-automática a partir de una especificación de alto nivel de las diferentes interfaces[1].

1. Motivación

Desde hace tiempo, las compañías que realizan el desarrollo de sistemas electrónicos de última generación se encuentran en la obligación de reducir el tiempo de desarrollo de dichos sistemas.

Así, estudios en el sector de los sistemas empotrados, como los realizados anualmente por *Embedded System Design*, muestran una media de 4.4 meses de retraso sobre la planificación inicial, con más del 50% de los proyectos finalizados fuera de plazo (según los últimos datos publicados en el informe de 2009). Sin embargo, el interés mostrado por las compañías en la incorporación de técnicas de co-diseño o *Electronic System Level* (ESL) es limitado.

En su lugar, la utilización de metodologías de diseño, llamémoslas *tradicionales* (con un uso todavía muy extendido), representan una amenaza. La razón de ello reside principalmente en la dependencia de los desarrolladores del software empotrado de un prototipo de la plataforma hardware como condición necesaria para poder iniciar su trabajo. Plataformas virtuales o entornos de simulación representan una gran ayuda, pero son costosas y no siempre se encuentran disponibles.

Es ampliamente aceptado que para afrontar con éxito el desarrollo de los futuros sistemas, donde la programación del software es ya el cuello de botella, es necesario:

- Elevar el nivel de abstracción de los modelos utilizados en la especificación de la interfaz entre HW y SW.
- Utilizar herramientas que permitan la generación automática del SdH.

En un escenario ideal, los arquitectos hardware y los ingenieros encargados del software deberían utilizar la misma especificación del sistema durante todo el ciclo de vida del proyecto. Si esto fuera posible, ambos equipos podrían ser capaces de trabajar de manera simultánea [5], lo que se traduciría en tiempos óptimos de desarrollo al eliminar la citada dependencia de la plataforma física.

El trabajo presentado en este artículo ofrece una solución a la problemática de la interfaz entre HW y SW en sistemas heterogéneos desde la perspectiva de la orientación a objetos. Las bases de la infraestructura HW y SW que se describen en secciones posteriores pueden resumirse en:

[1] Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación a través del proyecto DAMA (TEC2008-06553/TEC) y por la Junta de Comunidades de Castilla-La Mancha a través del proyecto RGRID (PAI08-0234-8083).

- El sistema es concebido como una colección de objetos, con una interfaz perfectamente definida y conocida.
- Los objetos en el sistema se comunican mediante invocaciones a métodos remotos (RMI).
- RMI (*Remote Method Invocation*) ofrece los mecanismos suficientes para unificar las interfaces de los componentes HW y SW en el sistema.

Dependiendo de la aplicación, el concepto de objeto puede no ser el más apropiado, pero aporta muchas ventajas en áreas como computación reconfigurable [4] (p.ej. gestión del estado de los componentes). El concepto de objeto también se adapta perfectamente a modelos de programación que pueden ser utilizados en la abstracción de la interfaz entre HW y SW. Un ejemplo es Multiflex [8] que muestra las capacidades del modelo de objetos distribuido para aplicaciones multimedia y de red.

En nuestra propuesta, la utilización de objetos en el proceso de diseño de sistemas empotrados ofrece a los programadores un entorno de desarrollo más estable, así como una sólida base para llevar a cabo prácticas de diseño orientadas a la reutilización [9]. En otras soluciones, la interfaz que se ofrece a los programadores de los componentes HW de sistema es de bajo nivel (p.ej. utilizando registros mapeados en el espacio de direcciones de memoria). Estas interfaces son muy sensibles a cualquier cambio que se produzca en la implementación de dicho componente, incluso si éste es sustituido por otro equivalente funcionalmente (pero que no implementa el mismo juego de registros). En este caso los desarrolladores software pueden llegar a invertir la mayor parte de su tiempo y esfuerzo en reescribir sus rutinas, lo que supone trabajo no productivo.

Nuestro trabajo se completa con la generación automática de los drivers software y los adaptadores hardware. Dicho proceso de generación se basa en la utilización de plantillas, parecido al proceso que realizan la mayoría de los middleware comerciales para redes de computadores. De esta forma, se libera al diseñador de la tarea de (re)escribir y (re)definir las interfaces con el sistema en cada fase del diseño.

Por lo tanto, la reutilización de código y modelos, así como la automatización de la generación tanto de los *adaptadores* software como hardware, presente en nuestra solución, supone un incremento en la productividad de los equipos de desarrollo.

Este artículo se estructura de la siguiente manera. La sección 2 ofrece un resumen de los trabajos relaciones más importantes. Luego, en la sección 3, se ofrece una visión general del proceso de síntesis de la interfaz HW/SW. Las plataformas hardware y software que se han desarrollado, necesarias para la implementación de las ideas propuestas en este trabajo, están descritas en las secciones 4, 5 y 6 respectivamente. Para concluir, las secciones 7 y 8 muestran los resultados experimentales, conclusiones y trabajo futuro a desarrollar.

2. Trabajos relaciones

En los últimos años se han publicado importantes trabajos de investigación que abordan la problemática del modelado y generación de SdH en particular y la interfaz HW/SW en general. Por ejemplo, Mignolet [7] muestra en su trabajo un mecanismo de comunicación uniforme para HW y tareas SW en el marco del sistema operativo OS4RS. La solución descrita por Mignolet está limitada a aquellas aplicaciones que pueden ser modeladas mediante un conjunto de hilos, que implementan tareas, y que se ejecutan simultáneamente. El mismo modelo de aplicación es el que inspira TTL [12]. TTL es una interfaz a nivel de tarea que puede ser utilizada indistintamente para el modelado de aplicaciones multitarea y la integración de tareas HW y SW.

BORTH [11] es un sistema operativo que ofrece una interfaz basada en ficheros para procesos tanto software como *hardware*. Si bien es cierto que la utilización de una interfaz bien conocida como el API del núcleo del sistema operativo puede facilitar el desarrollo de las aplicaciones empotradas, no queda justificado en el trabajo que dicha interfaz favorezca la reutilización. La interfaz *IOREG* modela los accesos a los dispositivos como una memoria, lo que de hecho no eleva el nivel de abstracción de la interfaz.

Un modelo unificado de componente HW/SW, que da soporte a una arquitectura organizada en capas para la integración HW/SW se describe en [2]. Los diferentes niveles de abstracción de la especificación ofrecen

soluciones para cada uno de los pasos del proceso de diseño. La implementación de la interfaz está automatizada.

Otros trabajos como COSY [3] o ROSES [13] utilizan unas librerías de componentes para generar automáticamente los adaptadores HW y SW. En COSY, la comunicación entre componentes se modela mediante canales que posteriormente se implementan mediante FIFOs en el sistema físico. En la práctica, la interacción con los cores HW consiste en operaciones de lectura y escritura (bajo nivel de abstracción). Algo parecido es lo que ocurre en el caso de ROSES donde los servicios del API ofrecido a las tareas software son del tipo *read/write put/get*.

Schirner presenta en [10] un método para la generación automática de la interfaz HW/SW a partir de modelos TLM (*Transaction Level Model*). Los canales SpecC se modelan utilizando la semántica TLM y la comunicación de HW/SW sigue un modelo basado en capas tipo ISO/OSI. Otros trabajos como los llevados a cabo por Klingauf se apoyan igualmente en TLM para definir las HPC (*Hardware Procedure Calls)*. Las HPCs [6] (que conceptualmente son muy parecidas a las *Remote Procedure Calls* presentes desde los años 80´s en los sistemas distribuidos de computadores) suponen un verdadero cambio de filosofía respecto a trabajos previos al representar un salto en el nivel de abstracción de la interfaz de acceso a las funciones HW. Sin embargo, no se hacen referencias a la arquitectura que soporta su implementación ni datos relativos a su eficiencia.

3. Síntesis de la interfaz HW/SW

Tal y como se hizo mención en la introducción, este trabajo se basa en la semántica RMI como punto de partida para lograr la integración transparente y automática del SW y HW en un sistema empotrado. La implementación de todos los conceptos de RMI para Sistemas-en-un-Chip dio lugar a OOCE (*Object-oriented Communication Engine)* un middleware hardware/software. En [1], el lector puede encontrar una visión general de la arquitectura global de OOCE y los principales objetivos perseguidos en la implementación del middleware.

En este artículo nos centramos en el proceso de generación de la interfaz entre el HW y el SW tal y como es concebida en OOCE para una plataforma concreta basada. En particular y como ejemplo, utilizaremos la plataforma basada en el procesador MicroBlaze y el sistema operativo µcLinux. La plataforma sobre la que se ha realizado la implementación de toda la infraestructura que se describe a continuación es una placa Xilinx XUPV2P30.

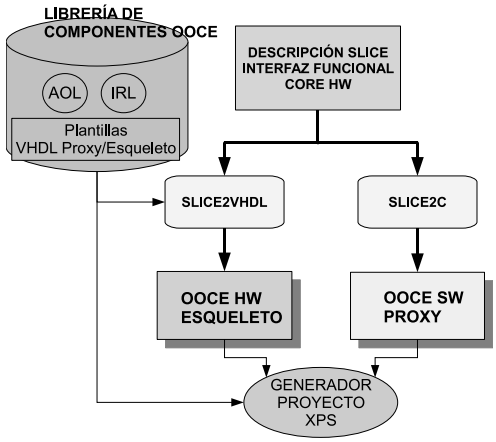

Figura 1. Flujo de síntesis de la interfaz HW/SW y generación de la plataforma.

3.1. Especificación de la interfaz funcional

La Figura 1 muestra de manera esquemática los pasos seguidos para la síntesis de la interfaz HW/SW así como las herramientas (compiladores de interfaces) necesarios en el proceso. Como entrada debe proporcionarse la especificación de la interfaz funcional del módulo IP (*Intellectual Property*) con el que debe interactuar el software. Dicha especificación puede ser realizada a mano o inferida automáticamente de la información de los distintos diagramas UML que se definen en la metodología OOCE para el desarrollo de sistemas empotrados (fuera del alcance de este artículo).

El lenguaje de descripción de interfaces utilizado en OOCE es SLICE (*Specification Language for ICE)*. Mediante SLICE se expresa la interfaz de los objetos presentes en el sistema, que es completamente ortogonal a la implementación de los mismos. El nivel de abstracción es suficiente para establecer un contrato entre el objeto que juega el papel del *cliente* (en este caso la aplicación software) y el objeto que hace el

papel de *servidor* (el IP que implementa la funcionalidad requerida). SLICE es un lenguaje puramente declarativo ya que sólo define tipos e interfaces.

```
module example {
["bus:protocol:OCP"]
 interface DES {
["OOCE:RMI:async"]
   void crypt(long key,long data,
              out long encrypted);
   void decrypt(long key, long data,
               out long decrypt);
 }
}
```

Figura 2. Especificación SLICE de un módulo criptográfico.

La Figura 2 muestra cómo sería en SLICE la descripción de la interfaz de un objeto hardware que implementa el algoritmo DES. A través de dicha interfaz el objeto hardware exporta al resto del sistema los servicios de encriptación y des-encriptación que implementa, y que pueden ser accedidos desde la aplicación que se ejecuta en el procesador. Mediante la utilización de *metadatos* (una característica presente en SLICE) se proporciona a los distintos generadores información específica de la plataforma o los mecanismos de comunicación del middleware que deben utilizar. Así, en nuestro ejemplo (ver Figura 2) vemos cómo se define el protocolo de bus al que se conectará el módulo hardware o el esquema de invocación OOCE que se utilizará (puede ser síncrona o asíncrona).

3.2. Los compiladores de interfaces

La generación tanto del código VHDL para los adaptadores hardware (denominados *esqueletos* en la terminología OOCE) y del código C para los drivers o adaptadores software (denominados *proxies* en OOCE) se han desarrollado sendos compiladores de interfaces: SLICE2VHDL y SLICE2C.

En el caso específico de la generación de la implementación VHDL, se ha optado por la utilización de plantillas, las cuales se especializan en base a la definición del método que se considere, así como los distintos parámetros de configuración proporcionados en la especificación SLICE. Una plantilla consiste en un conjunto de máquinas de estados finitas encargadas de:

- La serialización/recuperación de los paramétros/resultados de acuerdo a las reglas de codificación de datos definidas en OOCE: *marshalling*/*unmarshalling*.
- La conversión del protocolo punto a punto, definido entre el esqueleto y la interfaz OOCE de objeto hardware (*protocolo de invocación local*), al protocolo del bus.

3.3. La generación de la plataforma

El último paso consiste en la generación de toda la jerarquía de directorios y ficheros de un proyecto XPS (*Xilinx Platform Studio*) que recoge la configuración de la plataforma HW y SW. Llegados a este punto, el diseñador obtiene un primer prototipo de la plataforma listo para poder ser sintetizado, utilizando las herramientas del *Xilinx Embedded Development Kit*.

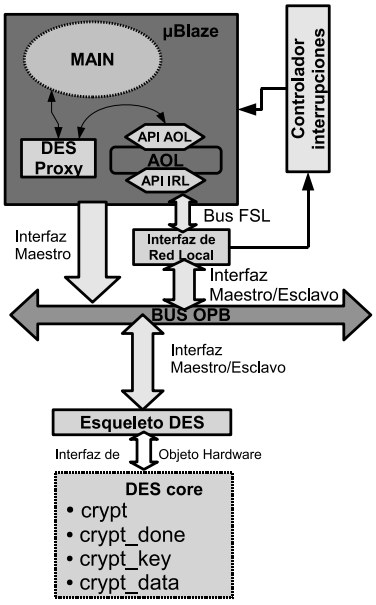

Figura 3. Plataforma HW-SW generada

En la Figura 3 se puede observar de manera simplificada la infraestructura HW y SW obtenida a partir de la especificación SLICE mostrada en la Figura 2. Las únicas acciones que se requieren por parte del usuario son:

- Conectar el módulo DES (representado con el contorno punteado). Dicho módulo debe seguir el estándar definido en OOCE para la ejecución de servicios según el modelo de objeto hardware.
- Escribir la aplicación principal, haciendo uso de los drivers OOCE generados.

A continuación, se describen en detalle todos los componentes y servicios de OOCE que toman parte en las comunicaciones entre HW y SW.

4. Esqueletos

El esqueleto es la versión hardware del adaptador OOCE cuya misión es la de proporcionar conectividad al IP que implementa la funcionalidad. El esqueleto monitoriza el bus de direcciones y activa la interfaz del módulo cuando una transacción se dirige a él. La lógica del esqueleto realiza la decodificación (*unmarsahlling*) del flujo de bytes antes de iniciar la ejecución del método hardware. La activación del método precede al paso de los argumentos. Se define para ello un sencillo protocolo de *handshaking* que permite la generación automática de las máquinas de estado que controlan dicho proceso. La finalización de la ejecución del método es notificada mediante la activación de una señal *done*, hecho que desencadena el proceso de codificación o *marshalling* del resultado y el envío de la respuesta por parte del esqueleto.

En OOCE se definen varias plantillas para los esqueletos que dan soporte a *invocaciones síncronas* o *asíncronas*.

5. La Interfaz de Red Local

La *Interfaz de Red Local* (o IRL) es el puente físico entre el procesador MicroBlaze y los componentes que implementan las distintas funciones que necesitan los clientes software. El IRL es un coprocesador que hace uso de dos enlaces *Fast Serial Link* (FSL) para las comunicaciones punto a punto con el procesador. El canal de control presente en este tipo de enlace es utilizado para sincronizar las operaciones entre el MicroBlaze y el IRL.

Una vez que desde el procesador se han depositado los datos del mensaje OOCE, que codifica la información necesaria para realizar una invocación a través del bus, el IRL solicita el acceso al bus. En la FIFO de salida presente en el enlace MicroBlaze→IRL se deposita primero la dirección física del esqueleto, seguido de los datos.

El otro enlace FSL se dedica a la comunicación IRL→Microblaze. Este enlace es necesario para encaminar hacia el procesador el tráfico de bus que codifica los mensajes OOCE de respuesta a invocaciones previas (si los métodos tienen parámetros de salida o generan un valor de retorno) que deben ser recibidos por objetos software en el procesador.

Para ello es necesaria una tabla de rutas (TR), que se implementa con una CAM, en la cual deben registrarse los objetos software (proporcionando un identificador que corresponde con los 12 bits más significativos de una dirección de acceso, tal y como si fueran un periférico más en el sistema). Las operaciones de actualización de la TR se codifican en mensajes especiales que reciben un tratamiento interno por parte del IRL.

La detección de una coincidencia por parte de la CAM hace que el IRL guarde en la cola de recepción los datos del bus (lo que sería el cuerpo del mensaje), y la dirección (que codifica la cabecera del mensaje en OOCE).

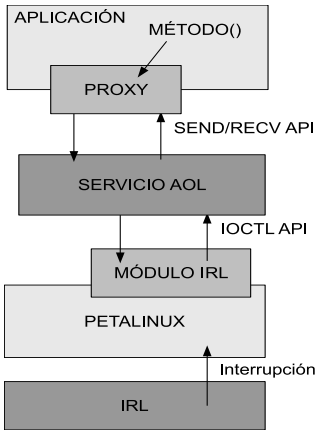

Figura 4. Arquitectura de la infraestructura SW.

6. Infraestructura software

Con el objetivo de abstraer a las aplicaciones de todos los detalles de implementación de bajo nivel (lo que incluye la implementación de la

comunicación entre la IRL-MicroBlaze y la implementación RMI para SoCs de OOCE) se han definido tres capas de servicios. La infraestructura software así definida permite a los programadores utilizar la infraestructura hardware de comunicación de una manera transparente, con un alto nivel de abstracción.

En la Figura 4 se encuentran representados los tres niveles mencionados. Seguidamente, ofrecemos una revisión de las principales características de cada uno y su implementación para poder entender cómo de manera conjunta ofrecen a los programadores esa visión del sistema como una colección de objetos.

6.1. KAOL

Sobre el sistema operativo, KAOL (Kernel Adaptador de Objetos Local) ofrece una interfaz para las operaciones de *envío* y *recepción* de mensajes a través del IRL así como para las operaciones de *gestión de la tabla de rutas*. KAOL está implementado como un módulo del núcleo del sistema operativo por lo que el nivel de abstracción ofrecido es bajo, basado en el uso de las primitivas *ioctl()*.

Sin embargo, las capas superiores evitan tener que utilizar directamente dicha interfaz para poder acceder al IRL.

KAOL implementa la rutina de gestión de la interrupción que proviene del IRL. Esta interrupción es activada cuando bien la cola de recepción o de envío en el enlace FSL contienen un nuevo mensaje. Dichos mensajes son temporalmente almacenados en un buffer interno para posteriormente ser recuperados por el *Servicio AOL*, el siguiente nivel en la jerarquía.

6.2. SAOL

El *Servicio de Adaptación de Objetos Local* (SAOL), implementado ya en espacio de usuario, hace uso directamente de la interfaz de bajo nivel de KAOL para la gestión del tráfico de mensajes desde y hacia las aplicaciones.

El SAOL es el único punto de acceso hacia los servicios del KAOL en el sistema. El SAOL actúa como un gestor centralizado del tráfico de mensajes desde y hacia las aplicaciones. En el caso del tráfico de salida, permite el uso compartido del IRL de forma segura implementando mecanismos de contención. Los mensajes entrantes son redirigidos a sus destinatarios, para ello el SAOL mantiene en memoria una lista de *respuestas pendientes* con toda la información necesaria.

Tal y como se mencionó anteriormente, el SAOL está implementado como un servicio en el sistema. El puerto y protocolo en el que SAOL se encuentra a la espera debe proporcionarse a los clientes (manualmente o a través de un fichero de configuración en el sistema). El acceso al SAOL se realiza, pues, desde las aplicaciones (a través de los proxies, nunca directamente, como veremos a continuación) utilizando una interfaz de paso de mensajes (*send* y *recv*).

Dos hilos en el SAOL se encargan, por separado, de las funciones de envío y recepción de mensajes. Ante una nueva petición de envío de mensaje por parte de un cliente, el SAOL crea un nuevo hilo para la gestión de dicha conexión.

```
void DES_crypt(tPrxy *prxy,long key,long data,
               long *encrypted) {
  tOOCE_msg msg;
  int sock, recv_code;

  msg.src = prxy->m_objid;  /* cabecera, ID origen */
  msg.dst = prxy->objid; /* ID destino */
  msg.rid = prxy->rid++; /* ID petición */
  msg.op = DES_CRYPT_MID;
  msg.type = OOCE_MSG_TWOWAY; /* se espera respuesta*/
  msg.size = 4; /* tamaño total cuerpo mensaje /
  *(long *)(msg.data+0) = key;  /*marshalling*/
  *(long *)(msg.data+8) = data;

  /* … Creación del socket de envío y conexión con el
     Servidor AOL. */
  send(sock, &msg, OOCE_HEADER + msg.size, 0);
  /* bloquea */
  recv_code = recv(sock, &msg, sizeof(tOOCE),0);

  /* unmarshalling */
  (*encrypted) = *(long *)(msg.data+0);

  return;
}
```

Figura 5. Código C generado para el proxy SW del método crypt, módulo DES.

6.3. Proxies software

Por último, encontramos las rutinas software generadas con SLICE2C, el compilador de interfaces para el lenguaje C. Estas rutinas son las que ofrecen a los programadores la ilusión de estar trabajando con objetos en lugar de tratar con los módulos hardware.

El proxy software es el encargado de rellenar las estructuras de datos que representan los mensajes OOCE. Este proceso incluye la codificación de los parámetros (marshalling). La Figura 5 muestra una versión simplificada del

código C generado por SLICE2C para el método *crypt*. Aunque C no es un lenguaje orientado a objetos, la utilización de punteros y estructuras permite implementar una pseudo-interfaz de objeto como se ve en la Figura 5.

Los programadores de aplicaciones sólo deben hacer uso de los proxies software generados y no necesitan conocer nada más de la infraestructura subyacente. La Figura 6 muestra cómo se realizaría la invocación del método crypt en el módulo hardware con nuestra infraestructura. El código es fácil de entender lo que permite su reutilización y facilita el mantenimiento del mismo.

```
#include <stdio.h>
#include "ooce.h"
#include "des.h"

int main(){
  tPrxy prxyDES;
  tLOA loa;
  long key,data,res;

  /* Obtener la referencia al SAOL.*/
  /* La cadena de conexión puede
     obtenerse automáticamente */
  LOA(&loa, "localhost:udp:123456");
  /* Creación del proxy al módulo DES */
  DES(&prxyDES,DESASYNC_BASEADDR);
  /* synchronous invocation */
  DES_crypt(&prxyDES,key,data,&res);
  /* Use the result */
  return 0;
}
```

Figura 6. Ejemplo de utilización de la infraestructura HW/SW generada.

Además, las aplicaciones son más robustas ante posibles cambios en la plataforma hardware que no se han previsto. Por ejemplo, si se produce una modificación o cambio en el IP que implementa DES, sólo se deben regenerar los esqueletos en el lado hardware; la aplicación no se ve afectada.

6.4. Resultados experimentales

La eficiencia en la comunicación entre el HW y el SW es crucial en todo sistema empotrado.

La plataforma utilizada para las pruebas y la realización del prototipo de nuestro sistema es una placa XUPV2P30, con un procesador *MicroBlaze v6.0b* y el bus de sistema funcionando a una velocidad de 100Mhz. La distribución de µcLinux utilizada es una *petalinux v0.3-rc1* y la versión del núcleo es una *2.4*.

Con el objetivo de evaluar la sobrecarga introducida por nuestra infraestructura, se han realizado dos tipos de pruebas:

- *Test 1*. Medición de la velocidad máxima de transferencia entre procesador y periférico.
- *Test 2*. Medición de los tiempos medios para una invocación SW a HW y estudio de su evolución al incrementar el número de aplicaciones tratando de acceder simultáneamente a la pila software.

Para el test nº1, se ha diseñado un periférico genérico que implementa los métodos *sink* y *generate*. Ambos métodos aceptan como uno de sus argumentos un entero que indica el número de elementos del vector que el módulo debe consumir o generar. Se han realizado mediciones del tiempo invertido en las escrituras y lecturas de vectores de distintos tamaños. Hemos comparado los resultados con una implementación equivalente utilizando la infraestructura IPIF que Xilinx ofrece en sus herramientas.

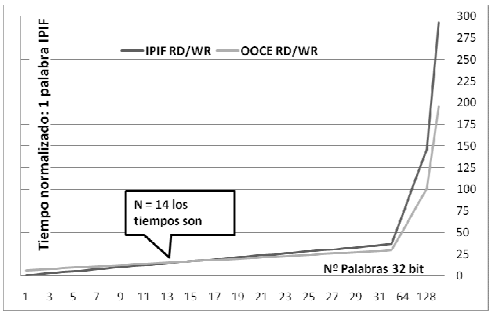

Figura 7. Tiempos de transferencia con IPIF y OOCE

La Figura 7 muestra, como era esperado, la penalización de la pila software (sistema operativo, KAOL y SAOL) pero sólo hasta que el número de palabras a transmitir en el mensaje de invocación alcanza el umbral de las 14 palabras. A partir de ese instante nuestra infraestructura tiene un mejor comportamiento debido principalmente a que las transferencias a través del bus se realizan mediante ráfagas y no palabra a palabra como ocurre si sólo se utiliza la interfaz maestra del procesador. Para mensajes de 128 palabras la mejora llega al 30%, llegando a un máximo de velocidad de transferencia de 5MB/s.

Se ha de remarcar que se trata de un prototipo inicial, y que la gran ventaja ofrecida por nuestro sistema radica en la transparencia ofrecida a los

programadores para utilizar la plataformas, así como la generación automática de la infraestructura.

El tiempo necesario para completar una invocación depende del tamaño de los parámetros que deben enviarse en el mensaje de invocación así como el tamaño de la respuesta. Debido a restricciones de espacio, no puede ser mostrada la gráfica con las mediciones realizadas para los mismos tamaños de transferencia que en la Figura 7. No obstante, el comportamiento del sistema es predecible y sigue una evolución lineal (aproximadamente de 4+0,25*N ms, siendo N el número total de palabras leídas y escritas).

También en el test nº2 se evaluó el comportamiento del sistema cuando el número de usuarios del SAOL se ve incrementado. Se quiere así tener una medida del impacto de los mecanismos de contención implementados en un escenario de estrés.

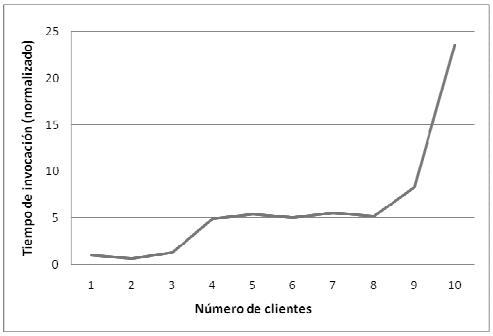

Figura 8. Evolución del tiempo medio de invocación. 20 invocaciones/cliente, método *crypt*.

La Figura 8 muestra una tendencia lineal en el incremento del tiempo medio de invocación hasta superar los 10 clientes, lo que puede considerarse el valor de saturación del sistema.

6.5. Conclusiones y trabajo futuro

La infraestructura que se ha presentado en este trabajo facilita el desarrollo del SdH gracias a la interfaz de objeto ofrecida a los programadores de la plataforma hardware, automáticamente obtenida de especificaciones textuales de alto nivel.

El siguiente paso en nuestro trabajo consiste fundamentalmente en la optimización de la arquitectura SW y HW de OOCE para la plataforma aquí descrita. También se está trabajando en mecanismos más eficientes para el movimiento de datos, en particular utilizando transferencias basadas en DMA.

Referencias

[1] Barba, J., et al., "OOCE: Object-Oriented Commu-nication Engine for SoC Design". *10th Euromicro Conference on DSD*, 2007.

[2] Bouchhima, A., et al., "A unified HW/SW interface model to remove discontinuities between HW and SW design". *EMSOFT'05,* 2005.

[3] Brunel, J., et al., "COSY communication IP's". *DAC, 2007.*

[4] Dondo, J. et al., "Dynamic reconfiguration management based on a distributed object model", *17th FPL International Conference*, 2007.

[5] Jerraya, A.A., "HW/SW Implementation from Abstract Architecture Models". *DATE*, 2007.

[6] Klingauf, W. et al.., "Embedded software development on top of transaction-level models". *CODES+ISSS '07*, 2007.

[7] Mignolet, J.-Y. et al., "Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip". *DATE,* 2003.

[8] Paulin, P.G. et al. "Parallel Programming Models for a Multiprocessor SoC Platform Applied to Networking and Multimedia", *IEEE Transactions on VLSI systems*, vol. 14, 17, July 2006.

[9] Rincón, F. et al., "Model reuse through hardware design paterns". *DATE*, 2005.

[10] Schirner, G., Gerstlauer, A., and Domer, R., "Automatic generation of hardware dependent software for MPSoCs from abstract system specifications". *ASPDAC,* 2008.

[11] So, H. K. and Brodersen, R., "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH". *Trans. on Embedded Computing Sys.* 7, 2, 1-28, Feb 2008.

[12] Van de Wolf, P., et al., "Design and Programming of Embedded Multiprocessors: An Interface-centric approach". *CODES+ISS'04*, 2004.

[13] Wagner, F. R., Cesário, W., and Jerraya, A.A., "Hardware/software IP integration using the ROSES design environment". *Trans. on Embedded Computing Sys.* 6, 3, July 2007.