

Síntesis de Canales TLM para procesador Nios-II

Montón Macian M¹, Martínez Huerta B¹, Carrabina Bordoll, J¹

¹ Centre de Prototips i Solucions Hardware-Software (CEPHIS)
Escola Tècnica Superior d'Enginyeria. UAB Barcelona, España,
{marius.monton, borja.martinez, jordi.carrabina}@uab.cat
<http://cephis.uab.cat>

Abstract. En este artículo se detallan experiencias en el diseño y síntesis de canales de comunicación TLM. Estos canales se basan en FIFOs y su implementación permite la comunicación de módulos HW entre si como de módulos SW (ejecutado en un procesador soft-core Nios-II) con módulos HW o entre distintos módulos SW entre si. Se ha usado una herramienta de síntesis comportamental de SystemC para eliminar al máximo la intervención manual en todo el ciclo de diseño.

1 Introducción

TLM (Transaction Level Modeling) se está imponiendo como metodología de descripción de sistemas. Esta descripción permite simulaciones rápidas y eficientes de gran numero de ciclos en grandes sistemas complejos, modelando módulos sin la necesidad de especificar si serán sintetizados en HW o serán partes SW del sistema. Esta descripción de muy alto nivel se basa en abstraer la comunicación entre módulos, de manera que pueda ser descrita independientemente de todo lo demás.

En este trabajo se propone una implementación de un canal de comunicación estandar en TLM, de manera que sea posible sintetizar (o mapear a SW) los distintos módulos del sistema así como los canales de comunicación usados de forma rápida y automática.

Para ello se trabajará con el canal estándar TLM_FIFO (en el que se basan los demás canales propuestos por TLM) para hacerlo sintetizable directamente desde SystemC a HW y los *wrappers* necesarios para poderlo conectar a un procesador Nios-II para el caso de los módulos mapeados a SW.

La estructura del artículo es la que sigue: En el capítulo 2 se introducen brevemente conceptos de TLM tales como interfaces y canales. El capítulo 3 versa sobre el *soft-core* Nios-II de Altera con una pequeña introducción a las Custom Instructions. En el capítulo 4 se presenta la herramienta de síntesis comportamental para SystemC usada en este trabajo. El capítulo 5 se dedica a relatar cómo se ha enfocado la síntesis de los canales TLM tanto para módulos HW como módulos SW. En el capítulo 6 se detallan los experimentos realizados con dichos canales. Por último las conclusiones y trabajo futuro de este trabajo.

2 TLM SystemC

TLM (Transaction Level Modeling)[1][2] es una aproximación al modelado de sistemas, donde se ha separado la comunicación entre módulos de la implementación de los mismos. Los distintos mecanismos de comunicación se modelan como canales y se usan en los módulos usando interfaces de SystemC. Una transacción se genera llamando a funciones de interface en dichos canales, las cuales encapsulan los detalles de implementación de bajo nivel. De esta forma las transacciones guardan detalles de alto nivel, tales como quién genera los datos, a quién van dirigidos, los datos mismos, etc. De esta forma, se gana en velocidad de simulación, y permite a los diseñadores de sistemas probar distintas arquitecturas de comunicación de forma sencilla y rápida.

2.1 Interfaces TLM

Los interfaces de TLM se basan en las clases interfaces de SystemC (`sc_interface`), aprovechando las características de Orientación a Objetos. De esta forma, en el interface sólo está la definición de las funciones para acceder al canal (`put()`, `get()`, `peek()`, ...), mientras que la implementación puede estar en el canal o en el módulo destino [3].

En SystemC hay dos tipos de procesos: `SC_THREAD` y `SC_METHOD`. La diferencia principal es que es posible suspender un `SC_THREAD` usando la función `wait()`. Los procesos `SC_METHOD` sólo se pueden sincronizar haciéndolos sensibles a `sc_event` externos. Debido a esto, cada método de cada *interface* debe indicar si puede o contener llamadas a `wait()`, y por tanto sólo puede ser llamado desde un `SC_THREAD`, o se garantiza que no tendrá tal llamada, y por tanto puede ser llamado por un `SC_METHOD`. Al primer *interface* se le llama “bloqueante”, y “no_bloqueante” al segundo.

Con estas consideraciones, se definen los siguientes interfaces TLM:

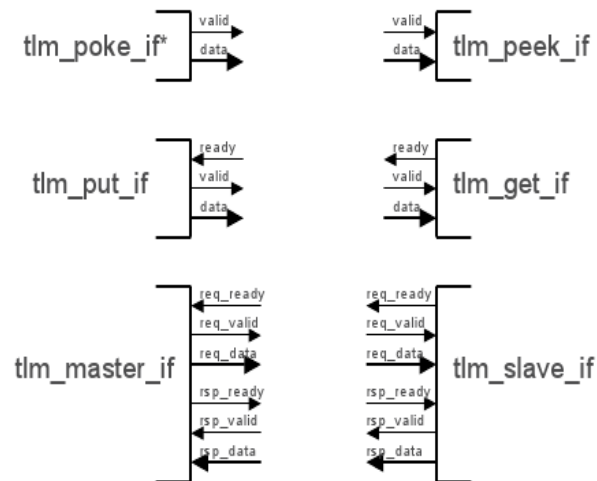


Fig. 1. Distintos interfaces definidos en TLM1.0 [3]

2.2 Canales TLM

Los interfaces presentados previamente pueden ser implementados en un canal específico, o directamente en el módulo destino. Así y todo, hay tres canales implementados y ampliamente usados incluidos en el estándar TLM.

tlm_fifo<T>

Este canal implementa todos los interfaces unidireccionales mencionados. Se basa en la implementación de una *sc_fifo*. Este canal implementa tanto métodos bloqueantes como no bloqueantes de acceso a los datos, permitiendo así el uso desde *SC_METHODS* y *SC_THREADS*.

tlm_req_rsp_channel<REQ,RSP>

Esta clase consisten en dos FIFOs, una para la petición desde el initiator hacia el target y otra para la respuesta desde el target al initiator. Estas FIFOs son de un tamaño cualquiera.

tlm_transport_channel<REQ,RSP>

Este canal se usa para modelar situaciones donde cada petición está ligada a una respuesta de forma que una petición no puede empezar hasta que acabó la petición anterior. Debido a esto, las FIFOs de petición y respuesta son de tamaño 1.

Estos canales TLM más complejos se basan en el canal *TLM_FIFO*, y es este el que se trabajará en este artículo.

3 Nios-II

El procesador Nios-II es un *soft-core* diseñado para las FPGAs de Altera [4]. Está basado en una arquitectura RISC estándar, y se permite la configuración de algunas de sus partes tales como: memoria cache de datos o instrucciones, etapas de *pipeline*, predicción de saltos, etc.

También es posible ampliar el repertorio de instrucciones, añadiendo módulos HW propios al *datapath* del procesador. Estas instrucciones (Custom Instructions o CI), se pueden usar desde lenguaje C por medio de macros generadas automáticamente. Existen tres tipos de CI, dependiendo del tiempo que tarden en devolver dato(s) al procesador: (i) combinational CI, que completan su función en un solo ciclo de reloj; (ii) Multi Cycle CI, que requieren más de un ciclo de reloj para terminar su ejecución; (iii) External Interface CI, que tienen acceso a lógica externa al procesador.

Estas External Interface CI son herramientas de gran potencia, ya que a través de este *interface* una instrucción del procesador puede acceder directamente a HW a medida.

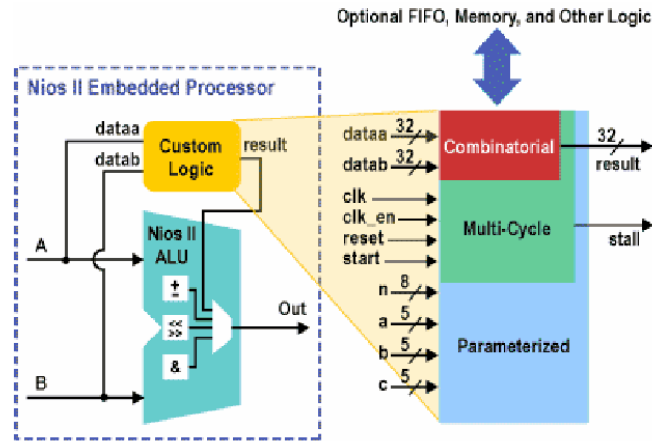


Fig. 2. Nios-II Custom Instructions: Combinational CI (Rojo), Multi-cycle CI (verde), External Interface CI (Azul)

4 Síntesis SystemC

En los últimos años han aparecido herramientas que permiten sintetizar descripciones de circuitos hechas en SystemC [5][6]. Estas herramientas de síntesis comportamental generan código RTL (Verilog o VHDL) a partir de una descripción SystemC de alto nivel.

Cynthesizer es la herramienta de Forte Design Systems para la síntesis comportamental de SystemC. Esta herramienta genera archivos Verilog a partir de código SystemC comportamental. Esta síntesis comportamental se basa en la extracción del algoritmo y sus dependencias temporales y la implementación a bajo nivel usando primitivas de cómputo. Además, a este código SystemC se le añaden directivas específicas para la herramienta (en forma de macros de C) para controlar ciertos pasos de la síntesis: *loop unrolling*, *array flatten*, etc. A partir de este código, la herramienta genera un SystemC nivel RTL para posteriormente generar el Verilog final. En todo momento, la herramienta genera *wrappers* y código para poder simular módulos en distintos niveles de descripción para poder validar el sistema completo.

Esta herramienta trata dos tipos distintos de código SystemC: código *cycle-accurate* (el cual está descrito a nivel de ciclo de reloj) y la herramienta implementará con dichas restricciones; y código *untimed* (donde no hay especificación de tiempo, sólo del algoritmo) y que la herramienta implementará según parámetros de más alto nivel (número de unidades funcionales, restricciones de tiempo total, etc). El código *cycle-accurate* se usa sobretodo para la entrada y salida del módulo, dejando el resto del código sin especificación de ciclos, dejando libertad a la herramienta para que implemente esa parte de código.

5 Implementación de canales TLM

5.1 TLM_FIFO HW

Para poder sintetizar el canal TLM_FIFO, se ha escrito una descripción a dos niveles SystemC, de forma que la herramienta synthesizer trabaje con la descripción TLM en las simulaciones de alto nivel, y una descripción RTL en las simulaciones de bajo nivel. Así también, la herramienta usará la descripción RTL para instanciar FIFOs IP de Altera cuando entre en la fase de síntesis.

La descripción TLM incluye todos los métodos estándar de acceso a la FIFO, que son heredados por la descripción de más bajo nivel (RTL) y sintetizados cuando procede. Estos métodos acaban transformados en *wrappers* de acceso a los bloques HW que permiten el acceso estandarizado a dichas FIFOs. De esta forma, podemos ver los bloques HW que acceden al canal FIFO en dos partes: la funcionalidad del módulo propiamente dicha, y un *wrapper* que le permite acceder a los datos de la FIFO, tanto en modo bloqueante como no-bloqueante.

5.2 TLM_FIFO Software

Cuando el canal TLM_FIFO es usado para comunicar un módulo HW con un módulo SW (esto es, un código SW ejecutado en un procesador), es necesario generar distintos *wrappers* para que el procesador pueda acceder a las FIFOs así como proporcionar las funciones de acceso a dicho canal. En nuestro caso, hemos usado Nios-II de Altera para las pruebas.

Para este procesador, existen dos posibles vías de implementación: tener un módulo de acceso a las FIFOs colgado del bus de sistema, o generar un instrucción de acceso a la FIFO dentro del *datapath* del procesador.

En este trabajo hemos implementado una CI correspondiente al acceso no bloqueante de la FIFO, para de esta manera implementar via SW los métodos de acceso bloqueantes.

De esta forma, un vez simulado el sistema descrito en SystemC a nivel TLM, es posible ejecutar un módulo en un procesador Nios-II usando dicho canal TLM, y sin necesidad de cambiar el código a excepción de las funciones y demás estructuras propias de SystemC que no serán necesarias en el código a ejecutar en el procesador.

6 Experimentos

Para comprobar la funcionalidad de lo anteriormente propuesto, se sintetizó un mismo sistema típico “Consumer-Producer” en los tres posibles tipos de implementación. Este sistema contiene dos canales TLM_FIFO, uno para los datos del *Consumer* hacia el *Producer*, y otro para la respuesta desde el *Producer* al *Consumer*.



Fig. 3. Sistema de prueba para todos los ejemplos. Los módulos *Consumer* y *Producer* pueden ser sintetizados o mapeados indistintamente a HW o a SW.

6.1 Sistema HW-HW

La primera prueba consistió en sintetizar los dos módulos a sendos módulos HW. Esto se hizo automáticamente con la herramienta Cynthesizer. Este sistema está completamente sintetizado en HW, y las FIFOs se generan a partir de instancias de los bloques IP FIFO propios de Altera.

Todo este proceso se sintetiza de forma automática sin intervención alguna del diseñador.



Fig. 4. Sistema con los dos módulos sintetizados a HW, juntamente con las FIFOs. Hemos representado fuera del módulo los *wrappers* de acceso a las FIFOs.

6.2 Sistema HW-SW

Este sistema de prueba ejecuta uno de los dos módulos en un Nios-II mientras que el otro módulo se sintetiza en HW. En este test hace falta la inclusión manual del *wrapper* para acceder a las FIFOs dentro del Nios-II y la pequeña librería asociada. Una vez hecho esto, el código SW que ejecuta el Nios-II es prácticamente el mismo que se tiene en la descripción original SystemC-TLM.

El módulo a implementar en HW no requiere de ninguna manipulación por parte del diseñador.

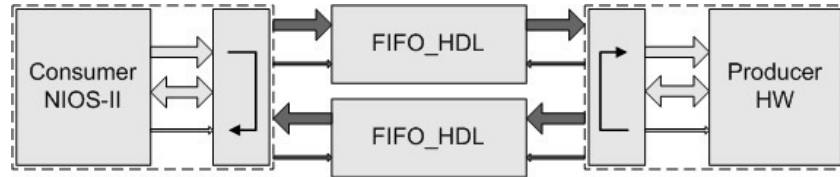


Fig. 5. Sistema con uno de dos módulos sintetizados a HW, juntamente con las FIFOs. El módulo “consumer” esta siendo ejecutado por el procesador Nios-II y accediendo a las FIFOs usando los mismos métodos.

6.3 Sistema SW-SW

Por último, se realiza el mismo sistema pero implementando los dos módulos en sendos procesadores Nios-II comunicados mediante el canal TLM_FIFO.

En este caso, hace falta la misma manipulación en ambos procesadores par incluir los *wrappers* de acceso a las FIFOs y la pequeña librería. También en este caso, el código de los dos módulos es muy similar al original SystemC-TLM.



Fig. 6. Los dos módulos están siendo ejecutados por sendos Nios-II usando FIFOs HW reales.

6.4 Resultados

En la tabla 1 resumimos la ocupación en elementos lógicos (LEs) de cada uno de los módulos sintetizados en HW, así como la ocupación de los *wrappers* de acceso a las FIFOs.

Módulo	LEs	Memory Bits
Consumer HW	149	0
Wrapper Nios-FIFO	38	0
FIFO (16x8bits)	131	128

Tabla 1. Resultados de síntesis de los distintos módulos i *wrappers* sobre una Cyclone 1C20 (20060 LEs)

Se observa una muy baja ocupación de los *wrappers* de acceso a las FIFOs debido a que las propias FIFOs implementan por si mismas buena parte de la lógica necesaria para emular los métodos TLM.

7 Conclusiones y Trabajo futuro

En este artículo se ha comentado el uso de TLM para modelar sistemas complejos, donde partes del mismo serán implementadas en HW y partes en SW. Para facilitar la tarea, se ha trabajado en un canal de comunicación estándar de TLM que sea sintetizable y que permita la comunicación de dos módulos estén implementados éstos en cualquiera de las dos formas posibles (HW o SW) de forma sencilla y casi-automática.

Además, debido al sencillo manejo de las FIFOs, el sobrecoste en ocupación de los *wrappers* de acceso al canal es mínimo, tanto en la implementación HW como en la implementación SW para el Nios-II.

Esta propuesta entendemos que es útil tan sólo para validaciones rápidas de sistemas, ya que una vez se tiene una descripción funcional del sistema, es posible sintetizarlo directamente. Puede ser útil también para poder obtener tiempos de ejecución de las partes SW o para obtener estimaciones de tiempos realistas en los módulos HW. Lo que nos parece claro es que difícilmente un sistema final va a usar estos mecanismos de comunicación, debido al coste en memoria de la FPGA que implica usar FIFOs, y que en caso de ser crítica la comunicación entre módulos, una solución ad-hoc puede representar serias ventajas o la necesidad de utilizar buses estándar que permita acceder a módulos como una memoria externa.

Como una primera mejora a este trabajo proponemos el estudio de la inclusión de dos o más módulos SW en el mismo procesador. Esto necesita de algún tipo de Sistema Operativo (o gestor de tareas) que permita esa multitarea en el procesador.

Con la aparición de TLM2, donde se especifican distintos y más complejos canales de comunicación, se presenta el reto de proporcionar algún mecanismo similar de síntesis rápida de alguno de los nuevos canales para poder implementar alguno de los futuros socket TLM directamente[7][8].

Referencias

1. Open SystemC Initiative: <http://www.systemc.org/>
2. Swan, S. Cadence Design: A Tutorial Introduction to the SystemC TLM Standard. 2006.
3. Rose A. Swan S., Pierce J., Fernandez JM: Transaction Level Modeling in SystemC. 2006
4. Altera Inc.: <http://www.altera.com/nios>
5. Coware <http://www.coware.com>
6. Forte Design Systems: www.forteds.com
7. TLM-2.0 standard. <http://www.systemc.org>
8. GreenSocs. <http://www.greensocs.com>