Ingeniería. Revista de la Universidad de Costa Rica
ISSN: 1409-2441
marcela.quiros@ucr.ac.cr
Universidad de Costa Rica
Costa Rica

Yu Lo, Lucky Lochi
Generación Automática de Modelos a Nivel de Transferencia de Sistemas Incrustados para aplicaciones multimedios
Ingeniería. Revista de la Universidad de Costa Rica, vol. 24, núm. 1, febrero-agosto, 2014, pp. 57-68
Universidad de Costa Rica
Ciudad Universitaria Rodrigo Facio, Costa Rica

Ingeniería 24 (1): 57-68, ISSN: 1409-2441; 2014. San José, Costa Rica

GENERACIÓN AUTOMÁTICA DE MODELOS A NIVEL DE TRANSFERENCIA DE SISTEMAS INCRUSTADOS PARA APLICACIONES MULTIMEDIOS

Lucky Lochi Yu Lo

Resumen

Con el incremento en complejidad y miniaturización de los sistemas computacionales actuales, su diseño se ha vuelto cada vez más difícil y lento. Para ello se necesitan modelos óptimos, y desarrollados tempranamente para probar los componentes. Para poder crear modelos automáticamente, se necesitan estructuras de datos y herramientas adecuadas. Si el modelo está bien declarado, con reglas y estructuras definidas, es posible crear un algoritmo que lo genere automáticamente. Para lograr la generación, se definió un estilo de modelaje y una estructura de datos que capturara todas sus características. Luego se implementaron algoritmos en C++ en una herramienta para generar modelos usando como entrada una aplicación en C y especificaciones de la arquitectura del sistema. La salida consistió en un modelo ejecutable del sistema en SystemC. Se seleccionaron dos aplicaciones industriales como prueba: un decodificador MP3 y un decodificador H264. El aporte de la investigación consistió en la definición de modelos adecuados para generar algoritmos que logran producir modelos ejecutables a nivel de transferencia de forma automática y sin necesidad de codificación en otro lenguaje, en una fracción del tiempo requerido. También, se desarrolló una herramienta que genera automáticamente modelos a nivel de transferencia de sistemas computacionales incrustados, útiles para simulación de alto nivel de abstracción para decisiones de arquitectura y desempeño.

Palabras clave: Generación automática; modelos a nivel de sistemas; SystemC; sistemas incrustados, ingeniería eléctrica..

Abstract

The design of current embedded computer systems has evolved into a lengthy and expensive process, which needs accurate and early models in order to test its components. In order to generate these models automatically, we need strictly defined data structures and adecuate tools. If the model is well declared, with clearly defined rules and structures, it is possible to create an algorithm that can automatically generate models. The main benefit of automatization is that the designer can move into a higher abstraction plane where he can focus on more important design decisions: architecture, performance, improving his productivity. To achieve automatic generation, we defined a modeling style and data structure that would capture all important characteristics of a system. Then, the generation algorithms were implenented in C++ in a tool that could take a C application, system specifications and would generate an executable SystemC model of the embedded system. We chose two industrial applications in order to test the algorithms: a MP3 decoder and a H264 decoder. The tool generated quickly good quality models and were sucessfully simulated alongside the original C model.

Keywords: Automatic generation; system level model, SystemC, embedded system.

Recibido: 26 de setiembre de 2012 • **Aprobado:** 16 de setiembre de 2013

1. INTRODUCCIÓN

Gordon Moore predijo en 1965 que el número de transistores en un circuito integrado se duplicaría cada 2 años. Esta predicción se ha mantenido cierta por casi medio siglo y continuará por al menos una década más. Debido a esto, casi todas las propiedades de los circuitos integrados han sido afectados por él: velocidad de procesamiento, capacidad de memoria, tamaño

y costo de los circuitos. Esto ha provocado que los circuitos integrados sean producidos masivamente, su costo haya disminuido y que la sociedad esté inmersa en el mundo digital.

Casi cualquier aspecto de nuestras vidas involucra algún tipo de interacción con circuitos integrados, y no solamente con computadoras de escritorio, sino con dispositivos computacionales que están en el carro, en la radio, celular, cámaras digitales, etc. Estos dispositivos se denominan sistemas incrustados, pues están incrustados en el dispositivo que deseamos usar (carro, elevador, radio). Estos sistemas incrustados han evolucionado a ser cada vez más complejos. Ejemplo de ello es la comparación del primer teléfono celular (Motorola TAC) con un teléfono de última generación (iPhone): la capacidad y utilidad han aumentado increíblemente. Esta evolución ha creado que el público demande cada vez más funcionalidad a estos dispositivos (cámaras, teléfonos, reproductores de música, etc).

Mientras los consumidores demandan más los ingenieros deben afrontar tiempos de diseño más cortos, más complejidad y menos consumo de poder, menos costo.

Actualmente se diseñan circuitos integrados con microprocesadores, buses de comunicación, memorias y hardware especializado en una misma pieza de silicio. Esta gran integración es posible gracias a los grandes avances de la última década. Estos sistemas se llaman Sistema-en-un-Chip o SoC. Adicionalmente, se requieren múltiples procesadores para las aplicaciones actuales, entonces estos sistemas se les denominan SoC de Múltiples Procesadores o MPSoC. El diseño de estos sistemas se ha vuelto cada vez más difícil y lento por los retos mencionados arriba.

En el proceso de diseñar un MPSoC complejo para una aplicación incrustada, los ingenieros de software y hardware necesitan trabajar juntos para delinear los requerimientos del sistema, las necesidades y limitaciones para poder empezar temprano y lograr las metas en el tiempo. Para lograr estas metas, se necesitan modelos óptimos desarrollados tempranamente para probar los componentes. La definición de estos modelos no es una meta trivial.

1.1 Diseño a nivel de sistemas

Para diseñar sistemas complejos, se divide el diseño en niveles de abstracción. Un nivel de abstracción le oculta sus detalles a niveles superiores e inferiores, mientras se basa en el nivel inferior para dar funcionamiento al nivel superior. Por ejemplo el nivel de abstracción más bajo son los transistores. En este nivel, los ingenieros diseñan los transistores que serán creados con silicio. Ya creados, el siguiente nivel de abstracción son las compuertas lógicas, las cuales están compuestas por múltiples transistores. Se utilizan las compuertas para crear módulos de función para que el siguiente nivel use para crear unidades procesamiento, y así sucesivamente. Un ingeniero de compuertas lógicas no necesita saber sobre la composición de los transistores, sino simplemente los utiliza, al igual que el arquitecto de unidades de procesamiento no necesita saber los detalles de cómo funciona internamente las compuertas. Así, mientras se sube en la jerarquía de diseño, los niveles superiores ocultan más detalle y manejan menos elementos que los inferiores (en el superior se manejan varios procesadores, mientras que en el inferior se manejan miles de compuertas o millones de transistores).

Una forma de abordar los retos expuestos anteriormente es elevar el nivel de abstracción del modelaje en el diseño (haciendo los modelos más abstractos), reduciendo el número de elementos y acelerando las decisiones de diseño. Los Modelos a Nivel de Transferencia (Transaction Level Models - TLM) han emergido como una nueva alternativa para diseñar Sistemas de MultiProcesadores en un Chip (MPSoC). Los TLM expresan la comunicación entre los módulos a nivel de una transacción, sin simular las decenas de señales de comunicación en un bus de comunicación. No obstante, hay algunas desventajas u obstáculos: los TLM son rápidos para simular, pero su exactitud es baja, tienen que ser codificados manualmente, usando un lenguaje de descripción a nivel de sistemas (System Level Design Language - SLDL), y no hay un estilo establecido con herramientas de soporte y además no hay forma de verificarlos.

Este artículo pretende mostrar una forma de sobrepasar estos obstáculos, definiendo un estilo

estricto de modelaje de sistemas incrustados para poder diseñar algoritmos de generación automática de modelos.

2. TRABAJOS RELACIONADOS

2.1 Modelos a nivel de transferencia

Los TLMs han subido en su uso desde que fue introducido (Grotker, 2002) como parte de la iniciativa de modelaje a alto nivel de SystemC. Se han desarrollado varios flujos de diseño y modelaje (Ghenassia, 2005), centrados alrededor de TLM.

También se han presentado un conjunto de modelos TLM (Baghdadi, 2002) que describen el flujo de diseño a través de los niveles de abstracción de TLMs. Identifican cinco niveles de abstracción que van desde procesos paralelos sin detalles de implementación hasta un modelo exacto en cuanto a tiempo.

Otros presentan (Cai, 2003) una semántica de diferentes modelos de transferencia basados en la granularidad de la temporización. Distinguen los TLMs por la exactitud en el tiempo de comunicación y tiempo de computación. Pueden ser sin tiempo, con temporización aproximada o temporización exacta.

Se han propuesto también optimizaciones de diseño y evaluación en TLMs (Cai, 2003). Mientras tanto, otros investigadores (Ogawa, 2003) proponen una generación automática de arquitecturas de buses, y usan transacciones de escritura y lectura para modelar un bus de TLM. El diseñador necesita especificar el mapeo de memoria, la topología de comunicación y esquema de arbitraje. Estos abordajes no separan claramente la computación de la comunicación.

2.2 Generación Automática de modelos

Han habido varios intentos de automáticamente generar modelos ejecutables en SystemC, a partir de descripciones abstractas. La principal desventaja es que todos requieren el uso de otro lenguaje para modelar el sistema, a diferencia del trabajo expuesto aquí.

Uno de los abordajes es de utilizar Modelado Específico de Dominios, para definir la semántica de un lenguaje de modelaje, y utilizarla para generar automáticamente el código del modelo (Gaither, 2013).

Se propone la generación desde código UML (Bruschi, 2003 y Kangas, 2006). Pero se propone todo un complejo de trabajo que cubre las fases de diseño desde modelaje de sistemas hasta el prototipo en FPGA.

Otros autores automáticamente generan modelos ejectuables de SoC (Sarmento, 2004), tomando como punto de partida especificaciones abstractas de arquitectura. Estas especificaciones son también escritas en SystemC pero no son ejecutables.

También se ha utilizado diagramas de modelaje como StateCharts (Moreno-Díaz, 2013), para refinar modelos desde nivel de transacción a niveles exactos a nivel de ciclos. Otro abordaje ha sido la generación automática del software a diferentes niveles de abstracción para la cosimulación idónea de hardware-software (Wu, 2010), no obstante, este abordaje toma como entrada código en ensamblador que decompilan a código en C.

3. GENERACIÓN AUTOMÁTICA

El concepto de generación automática involucra más que lo que se espera inicialmente. Si bien con un proceso automatizado el factor humano deja de ser un factor limitante (el proceso se realiza más rápidamente), la importancia no es solamente el ahorro en tiempo. La principal consecuencia es que el diseñador deja de codificar manualmente el código y pasa a tomar decisiones más interesantes e importantes de diseño: como el modelo puede ser generado automáticamente en pocos segundos, puede decidir cambiar la arquitectura y rápidamente ver los resultados del desempeño del sistema, y explorar nuevas posibilidades en poco tiempo. Se puede probar múltiples diseños en el mismo tiempo en que el diseñador tardaría en codificar manualmente un solo diseño. Así que el aumento en la productividad es gigantesco.

Uso de librerías y generación: se puede proceder a construir el sistema usando componentes tomados de una librería predefinida,

pero este abordaje limita la flexibilidad en que el modelo pueda cambiar, o daría muy pocas opciones de modificación. La única opción para minimizar este efecto sería construir una librería muy extensa, que cubra todas las posibles opciones, y esta misma librería se debería generar automáticamente, por lo cual, simplemente se estaría moviendo la complejidad de una fase del diseño al otro. Considerando que el tiempo de generación es mucho menos que el de simulación, es más factible generarlos al momento de usarlos, así que en este caso, todo el modelo es generado, sin tomar elementos predefinidos.

3.1 Definición del modelo y proceso de diseño.

Antes de tener la capacidad de generar automáticamente un modelo, es necesario definirlo formalmente. Nuestros modelos estarán compuestos por elementos físicos que representan el hardware (Elementos de procesamiento (Processing Elements - PEs), Transductores (TX), Buses de comunicación (UBC-Canal de bus universal), y elementos de comunicación en software (API). Dentro de cada PE existirán procesos (código en C corriendo en procesadores de uso específico o general).

Estos elementos son definidos por el usuario por medio de una herramienta con interfaz gráfica escrita en Python. Se compone el sistema incrustado a base de CPUs o procesadores (PEs) y módulos de comunicación (UBC y TX). Posteriormente se mapean los procesos de un programa (escrito en C). Eventualmente estos programas podrán ser implementados como software en un procesador de uso general o implementados en hardware como una unidad de uso específico (Application Specific Integrated Circuit – ASIC) o comprado a otra empresa (Intelectual Property – IP).

La necesidad de tener transductores radica en que algunos IPs solamente están diseñados para un protocolo específico de comunicación de buses. Si se dispone de diferentes IPs conectados a diferentes buses, la única forma en que se comuniquen, es por medio de un transductor TX.

El siguiente paso luego de colocar los elementos es asignar canales y rutas de comunicación entre los procesos, que pasan por los elementos de comunicación designados.

Este modelo es grabado automáticamente en un formato XML (es transparente: el usuario no requiere realizar ninguna codificación en XML) el cual detalla todas las características de cada elemento. A este archivo se le nombra una extensión .eds y se le refiere como archivo EDS. Junto con el código fuente en C, se utilizará para generar automáticamente el modelo ejecutable en SystemC. Al ejecutarse este código, se estará simulando todo el sistema y su comunicación.

La generación ocurre por etapas: generación de PEs, generación del API, generación del UBC, generación del TX, y generación del módulo madre.

Todo este proceso puede tener la duración de minutos (exceptuando la codificación del programa en C, que se asume, ya fue hecha anteriormente como código de legado o se nos lo dió). La generación per se dura mucho menos tiempo, como se verá en los resultados experimentales.

3.2 Generación del PE

El Algoritmo 1se muestra a continuación.

ALGORITMO 1

Generación de pes

1. PE = Conjunto de elementos de procesamiento en un diseño
2. for all pe ϵ PE do
3. //generar módulo PE
4. gen: “class P_” + pe.nombre + “:public sc_module{“
5. Generar_constructor()
6. for all proc ϵ pe do
7. Instanciar_modulos_proceso()
8. Conectar_puerto_buses()
9. end for
10. gen:”}”
11. end for

Se toma cada objeto pe del XML y se generará la declaración de la clase PE y cada objeto proc perteneciente al pe y se genera los módulos que albergan a los procesos. La generación de los procesos se muestra en el Algoritmo 2

ALGORITMO 2

Generación de módulos de procesos
1. PE = Conjunto de elementos de procesamiento en un diseño
2. for all pe ϵ PE do
3. for all proceso ϵ pe do
4. Generar_constructor()
5. gen:"SC_THREAD(main_"+proceso.nombre+")"
6. Declarar_puertos_buses()
7. //Generación función main()
8. gen:"void main_"+proceso.nombre+"(void){"
9. gen:" "+proceso.nombre+"();"
10. gen:"}"
11. end for
12. gen:"}"
13. end for

Aquí se instancia cada hilo de ejecución y se declara la función main() que llama la función en C brindada por el usuario.

3.3 Generación del API de comunicación

El API no es solamente el puente entre los PEs y los elementos de comunicación, sino entre los procesos en C corriendo la aplicación y el código SystemC generado. En otras palabras, es el puente entre el código de legado, y el código generado por la herramienta.

El Algoritmo 3 se muestra mas adelante. Para cada canal mapeado, existe una ruta asignada. Para cada ruta de proceso a procesos, se genera una función send y una función receive. Existen dos tipos de funciones, las locales y las que abarcan más de dos buses. Para cada ruta de proceso a memoria, se genera una función read o una función write. Adicionalmente, se revisa si es una función que utiliza un FIFO o no.

ALGORITMO 3

Generación del API de comunicación
1. R = conjunto de rutas en el diseño
2. PR = conjunto de procesos en el diseño
3. for all route rt ϵ R do
4. chrt = canal emparejado a rt
5. srcrt = proceso-memoria origen en rt
6. destrt = procso-memoria destino en rt
7. if chrt == canal proceso-proceso then
8. txrt == conjunto de transductores en rt
9. if |txrt| > 1 then
10. //transacciones multisalto
11. Gen_multiTXsend(srcrt, chrt)
12. Gen_mutliTXrecv(destrt, chrt)
13. else
14. //transacciones del bus local
15. Gen_send(srcrt, chrt)
16. Gen_recv(destrt, chrt)
17 endif
18. else if chrt == canal de memoria then
19. if srcrt ϵ PR then
20. Gen_escribir(srcrt, chrt)
21. else
22. Gen_leer(destrt, chrt)
23. endif
24. else
25 //es una red de procesos
26. Gen_red_procesos(chrt)
27. end if
28. end for

3.4 Generación de UBC

El UBC se modela según el canal SystemC definido en la sección anterior. Para cada bus en la plataforma, se genera una implementación única del UBC. Cada UBC tendrá un conjunto de funciones particulares que consisten de:
1. Tabla de sincronización
2. Árbitro
3. Funciones read-write
4. Funciones send-receive
5. Función memoryService
6. Función de estimación de retardo

Para generar la función de sincronización, se utiliza la tabla de sincronización del EDS, que muestra de una forma sencilla, todos los canales de comunicación entre cada par comunicante.

Básicamente se toma cada elemento de la tabla correspondiente al bus que se va a generar y se crear la función IF-ELSE en la función syncronize() del bus. La generación no es compleja, debido a lo explícito como se construye la tabla.

La generación de las funciones de Send-receive se construyen apartir de estos datos. (ver Algoritmo 4)

ALGORITMO 4

Generación de la función Send del UBC

```
1:  //Generar Árbitro y variables
2:  Generate Arbiter()
3:  gen:“int BusAddr; sc event AddrSet;”
4:  //Generar send()
5:  gen:“if (MyMode==initiator){”
6:  for all ex ∈ Ebus do
7:  gen:“ if (MyID==i”+ex .initiator+“ && PartnerID==”+ex .resetter+“){”
8:  gen:“ while(BusAddr!=”+ex.chname+”_ADDR){“
9:  gen:“ wait(AddrSet);”
10: gen:“ }”
11: gen:“ }”
12: end for
13: Copiar punteros
14: Notificar eventos de transacción
15: Esperar eventos de transacción
16: gen:“}”
17: gen:“else if (MyMode==”+ex .resetter+“){”
18: Solicitar Árbitro
19: for all ex ∈ Ebus do
20: gen:“ if (MyID==”+ex .resetter+“ && PartnerID==”+ex .initiator+“){”
21: gen:“ BusAddr=”+ex .chname+“ ADDR;”
22: gen:“ }”
23: gen:“}”
24: end for
25: Copiar punteros
26: Notificar evento AddrSet
27: Esperar por evento de transacción1
28: Resetear BusAddr
29: Notificar evento de transacción2
30: Soltar árbitro
31: gen:“}”
```

3.5 Generación del Transductor

La generación del transductor es la más compleja de todas, pues debe manejar todas las rutas de comunicación posibles entre los PEs. El transductor consta de 3 elementos básicos: un Request buffer que es la unidad que recoge el comando del PE para procesar un mensaje, un FIFO que es la estructura donde se almacena temporalmente el mensaje, y una unidad de entrada y salida (IO) que se encarga de enviar/recibir el mensaje del PE y leer/escribir al FIFO.

La generación ocurre por etapas: Request Buffer, IOs y luego FIFOs. Existe un set de Request Buffers y IOs para cada Bus conectado al TX, y un sólo FIFO para todo el TX. Cada Request Buffer consta de varias unidades de RB, o RBUs. El transductor contiene RBUs para cada canal de comunicación, y cada canal difiere en estas características:

- Escritura o lectura
- Acceso remoto o local
- Canal con saltos múltiples (multihop) o saltos simples
- Acceso a FIFOs o directos a PEs
- Si el PE final se encuentra en el bus actual

Todo define el tipo de acceso del TX a otro TX, o a un PE por medio de uno de los UBCs conectados.

El árbol de decisión para generar el RBU correspondiente se muestra en la Figura 1.

La generación del bloque Request Buffer toma en cuenta las parejas de procesos comunicantes, y las direcciones de los canales: se evalúa cada proceso en cada canal para examinar si la ruta asignada incluye el transductor a generar. Si este es el caso, se generan las direcciones utilizadas por el canal y se genera los comandos IF y las acciones que revisan el FIFO para recibir o enviar el paquete.

Finalmente, para generar el módulo superior, se usan lazos anidados alrededor de los objetos del EDS para:

- Instar cada clase generada: buses, PEs, transductores
- Enlazar buses, puertos, conectores, a procesos y transductores

4. RESULTADOS EXPERIMENTALES

Los algoritmos mostrados fueron implementados en C++ en una herramienta para generar TLMs usando como entrada una aplicación en C y especificaciones

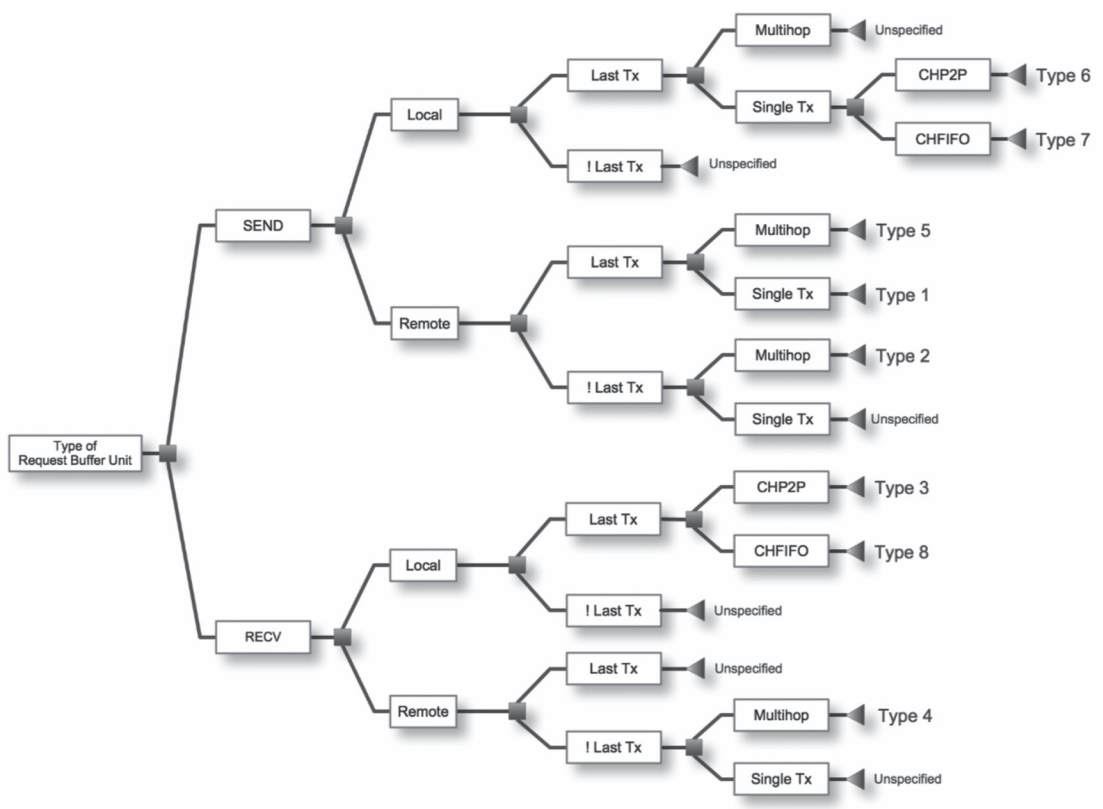

Figura 1. Árbol de decisión del tipo de RBU.

de plataformas. La entrada al generador fue el archivo EDS y el código en C. Se seleccionaron dos aplicaciones industriales grandes: un decodificador MP3 de audio y un decodificador H264 de video. Todas las pruebas se efectuaron en una computadora con Pentium 4, 3Ghz de frecuencia, 1GB de memoria, ejecutando un sistema operativo Linux con un kernel 2.6. El código de referencia en C del MP3 consistía en 9463 líneas de código y la entrada a decodificar era de 138Kb de tamaño. El código del H264 era de 3419 líneas de código y se decodificó un video de 27kb de tamaño 352 por 288 pixeles. Se probaron varias plataformas con estas aplicaciones, se muestran las plataformas usadas para decodificar MP3 en la Figura 2. Las plataformas para el decodificador H264 se muestran en la Figura 3. Los bloques rectangulares consisten en elementos de procesamiento o IPs, y los procesos son representados como cuadrados redondeados al interior de los rectángulos. Las líneas horizontales representan los buses de comunicación.

Los resultados experimentales se muestran en en la tabla 1.

La herramienta genera una serie de archivos (.h, .cpp, y makefiles) en SystemC, y los compila, produciendo un TLM totalmente ejecutable. Al ejecutarse, se decodifica el archivo de audio o video y graba la salida. La tabla muestra los tiempos de generación, y simulación. Como referencia se

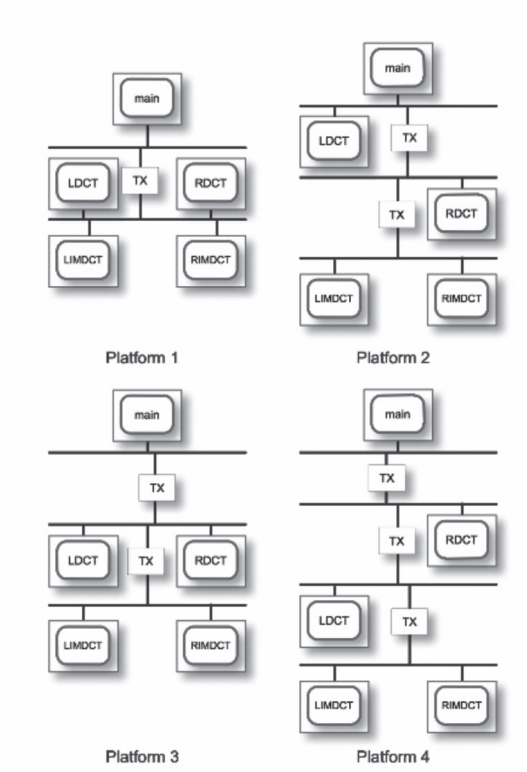

Figura 2. Plataformas usadas para decodificación de MP3.

muestra la cantidad de líneas de código generado y el tiempo aproximado para codificarlas manualmente.

Se observa que la herramienta genera miles de líneas de código en una fracción de segundo.El tiempo manual de codificación tomaría decenas de horas que se pueden ahorrar utilizando el generador automático. Se muestra la comparación entre tiempos de generación y simulación de ambas plataformas en las Figuras 4 y 5.

En cuanto a la calidad de los modelos, se puede ver que el tiempo de simulación está en el mismo orden de magnitud, por ende, la calidad de los modelos es alta (se ha definido un modelo a nivel de sistemas con mucho más detalle de la implementación y se ejecuta rápido).

5. CONCLUSIONES

1. Se definieron modelos correctos de un sistema computador para los elementos básicos de un modelo a nivel de transferencia.
2. Se presentó una metodología para la generación automática de modelos a nivel de transferencia, en base a una especificación formal de un sistema incrustado.
3. Con una especificación formal y detallada del sistema, es posible crear algoritmos de generación automática con niveles de complejidad de baja a alta, e implementarlos en una herramienta con interfase gráfica.

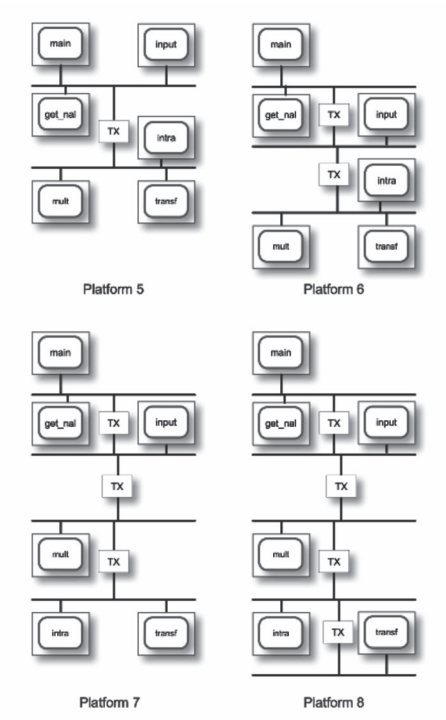

Figura 3. Plataformas usadas para decodificación de H264.

4. Los modelos ejecutables de sistemas incrustados se pueden generar en fracciones de segundo y simulan confiablemente un sistema real, permitiendo usar la herramienta para exploración de diseño, de una forma rápida y confiable.
5. Se permite utilizar código legado escrito en lenguaje C en un sistema nuevo y examinar su comportamiento sin necesidad de reescribir código o aprender un lenguaje de diseño de sistemas nuevo como SpecC o SystemC.

6. REFERENCIAS BIBLIOGRÁFICAS

Baghdadi, A., Zergainoh, N., Cesario, W., & Jerraya, A (2002). Combining a performance estimation methodology with a hardware/ software codesign flow supporting multiprocessor systems. *IEEE Transactions on Software Engineering,* 28:822–831.

Bruschi, F., Di Nitto, E., & Sciuto, D. (2003) *SystemC code generation from uml model*. *Proceedings Int*. *Forum on Specication and Design Languages*. FDL'04, Frankfurt.

Cai, L., & Gajski, D. (2003). Transaction level modeling: an overview. CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP *International Conference on Hardware/ software Codesign and System Synthesis,* pp. 19–24.

Donlin, A. (2004). *Transaction level modeling: Flows and use models*. CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP

Cuadro 1. Tiempo de generación y calidad en diferentes diseños MPSoC.

Aplicación	Config. Plataforma	SystemC líneas código	Tiempo manual est.	Tiempo generación	Tiempo simulación
MP3	Ref C	-	-	-	1.29s
MP3	Plat 1	2095	104 hrs	0.633s	3.268s
MP3	Plat 2	2894	144 hrs	0.661s	5.519s
MP3	Plat 3	3148	157 hrs	0.645s	5.764s
MP4	Plat 4	3653	183 hrs	0.741s	7.424s
H.264	Ref C	-	-	-	2.027s
H.264	Plat 5	1722	86 hrs	0.245s	7.542s
H.264	Plat 6	2796	140 hrs	0.244s	9.935s
H.264	Plat 7	3853	192 hrs	0.267s	13.326s
H.264	Plat 8	4910	245 hrs	0.260s	15.415s

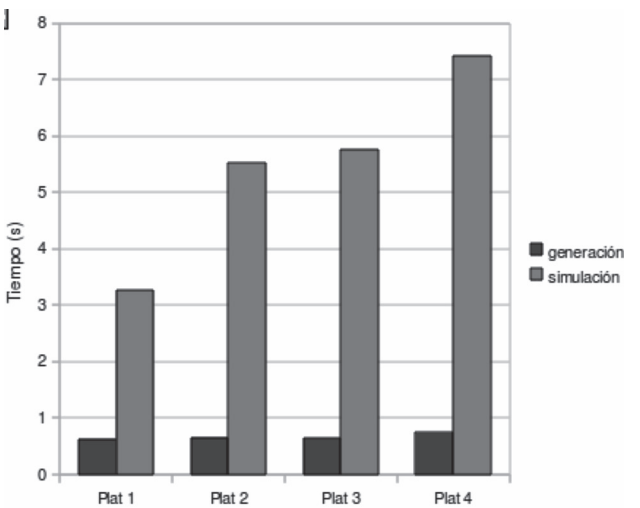

Figura 4. Tiempos de generación y simulación de plataformas con MP3.

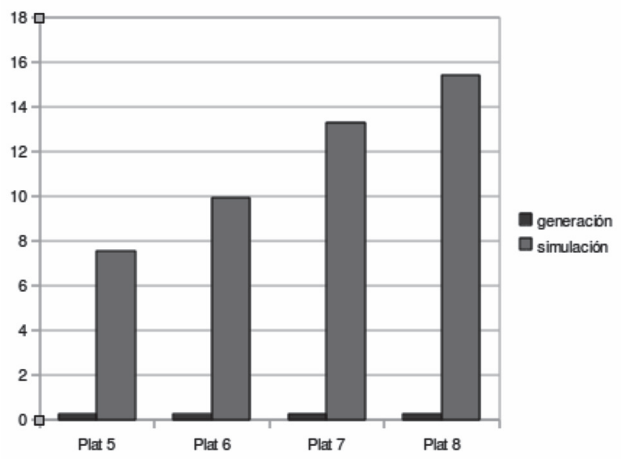

Figura 5. Tiempos de generación y simulación de plataformas con H264.

International Conference on Hardware/ software Codesign and System Synthesis, pp. 75–80

Findenig, R. (2013). *Transcation-Level Modeling and Refinement Using State Charts*. Springer-Verlag, Berlin Heidelberg.

Gaither, D. (2013). *Toward Denotational Semantics of Domain-Specific Modeling Languages for Automated Code Generation*. Springer-Verlag, Berlin Heidelberg.

Grotker, H (2002). *System Design with SystemC*. Kluwer Academic Publishers , Norwell, MA.

Ghenassia, F (2005) *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer.

Kangas, T (2006) Uml-based multiprocessor soc design framework. *ACM Transactions on Embedded Computing Systems,* 5(2):281–320.

Ogawa, O., Bayon de Noyer, S., Chauvet, P., Shinohara, K., Watanabe, Y., Niizuma, H., Sasaki, T., & Takai, Y (2003). A practical approach for bus architecture optimization at transaction level. DATE '03: *Proceedings of the conference on Design, Automation and Test in Europe,* p 20176.

Sarmento, A., Cesario, W., & Jerraya, A. (2004) Automatic building of executable models from abstract soc architectures made of heterogeneous subsystems. *Proceedings of the 15th IEEE International Workshop on Rapid System Prototyping*.

Wu, M. (2010) *Automatic Generation of Software TLM in Multiple Abstraction Layers for Efficient HW/SW Co-simulation*. Proceedings of the Design Automation and Test in Europe 2010.

SOBRE EL AUTOR

Lucky Lochi Yu Lo
Universidad de Costa Rica, Escuela de Ingeniería Eléctrica. Ph. D. en Ingeniería Eléctrica.
Profesor del Departamento de Automática y Digitales.
Correo electrónico: lochiyu@eie.ucr.ac.cr