HLStool: Una herramienta de Síntesis de Alto Nivel para el aprendizaje del estudiante en asignaturas de ATC

Alberto A. Del Barrio, Guillermo Botella

Departamento de Arquitectura de Computadores y Automática, Facultad de Informática de la Universidad Complutense de Madrid
{abarriog, gbotella}@ucm.es

Resumen. La utilización de herramientas de Síntesis de Alto Nivel (SAN) es una práctica habitual en las empresas dedicadas al diseño de circuitos. El principal beneficio de estas herramientas se basa en la reducción del "tiempo de lanzamiento al mercado", ya que permiten evaluar múltiples soluciones en un tiempo reducido. En esta contribución, se propone el uso de una herramienta de síntesis altamente visual y amigable que contenga los algoritmos clásicos para enseñar al alumno las técnicas básicas de SAN, y adicionalmente familiarizarle con el método de trabajo de las compañías de diseño de circuitos. Con esta contribución se propone ofrecer una formación más completa a los futuros graduados que cursen asignaturas del área de la Arquitectura y Tecnología de Computadores

Palabras Clave: Síntesis de Alto Nivel (SAN), planificación, asignación, diseño de circuitos integrados, Grado en Informática, Plan de Estudios, EEES.

Abstract. The use of High-Level Synthesis (HLS) tools is a common practice in circuit design companies. The main benefit of these tools consists of diminishing time to market, as they provide multiple solutions to explore in a reduced amount of time. In this paper we propose the use of a highly visual synthesis tool, which implements the classic algorithms to teach students the basic HLS concepts and to get them closer to the companies' methodology. In this way, Computer Architecture and Technology related subjects will offer a more complete programme to the future graduates.

Keywords: High-Level Synthesis, scheduling, binding, Integrated Circuit Design, Computer Science Degree, Syllabus, EEES.

1 Introducción

Actualmente, la automatización del proceso de diseño (*aka*. Diseño Automático) se ha convertido en una tarea esencial, debido a la cada vez mayor complejidad de los diseños y al menor tiempo destinado a ellos por parte de las compañías. Sin embargo,

las titulaciones actuales en las ramas relacionadas con la Informática no ponen suficiente énfasis en esta área.

La Tabla I muestra un ejemplo de los contenidos y competencias que pueden encontrarse en 3 asignaturas del área de la Arquitectura y Tecnología de Computadores (ATC) en las titulaciones de Grado en Informática, Grado en Ingeniería de Computadores y Máster en Ingeniería Informática ofertados por la Universidad Complutense de Madrid (UCM) en el curso académico 2013/2014 [1]. La asignatura Tecnología y Organización de Computadores (TOC) se encuentra ubicada en el 2º curso de ambos grados, Diseño Automático de Sistemas (DAS) puede encontrarse como asignatura optativa en ambos grados también, mientras que la asignatura Sistemas Empotrados Distribuidos (SED) se ubica en el 1er curso del Máster de Ingeniería Informática. Tal y como puede observarse, en todas las asignaturas se enfoca la enseñanza de sistemas de acuerdo a un esquema algorítmico, es decir, el objetivo es aprender a diseñar *IP core*s [2] o subsecuentemente los *kernels* que los componen con mayor o menor granularidad. Dichos *kernels* normalmente consisten en funciones que requieren de una alta capacidad de cálculo, como los que ofrece la suite de *PolyBench* [3].

Sin embargo, el diseño que realizan los alumnos en las asignaturas previamente mencionadas es plenamente manual, mediante un lenguaje de descripción hardware de alto nivel como VHDL [4,5] o adicionalmente con el editor de esquemáticos de la herramienta Xilinx ISE [6]. Mediante esta filosofía de diseño manual, -usando VHDL- los estudiantes deciden el diseño que aparecerá en la implementación final. Desgraciadamente este diseño está sujeto a errores frecuentes. Sin embargo, si se usa el editor de esquemáticos, se subsanan muchos de los errores que aparecen en la metodología anterior, pero se sacrifica el control sobre el diseño final. Asimismo, como se ha mencionado previamente, hoy en día las compañías utilizan herramientas de Síntesis de Alto Nivel (SAN) [7] que permiten producir circuitos correctos por construcción y en un tiempo muy reducido, lo que les proporciona enormes beneficios [8].

En este artículo se pretende dar una alternativa para resolver los problemas mencionados por medio de la utilización de una herramienta de SAN muy visual y amigable, de tal forma que los alumnos puedan evaluar múltiples soluciones en un tiempo reducido, sin perder al mismo tiempo la perspectiva real de la implementación del circuito. De esta manera, los estudiantes aprenderán una metodología más cercana a la forma de trabajar en las empresas que diseñan circuitos. Además, la visualización de los resultados redundará en una mejor comprensión de los algoritmos de planificación y asignación utilizados.

El resto del artículo está organizado de la siguiente manera: la Sección 2 dará una visión global sobre el uso de las herramientas de SAN, mientras que la Sección 3 describe los detalles de la nuestra herramienta propuesta -HLStool-. La Sección 4 presentará un ejemplo ilustrativo para comprender su potencial aplicación y finalmente la Sección 5 expondrá nuestras conclusiones.

Tabla 1. Contenidos y Competencias adquiridas en 3 asignaturas ofertadas por la UCM en el curso 2013/2014 en los grados de Ingeniería de Computadores e Ingeniería Informática.

Asignatura	Contenidos	Competencias
Tecnología y Organización de Computadores	Circuitos Aritméticos, Sistemas Algorítmicos, Lenguajes de Descipción de HW	Conocimiento de la Estructura e Interconexión de los Sistemas Informáticos
Diseño Automático de Sistemas	Diseño Automático de Sistemas, Síntesis sobre FPGAs	Conocimiento de la Estructura y Arquitectura de un Sistema y Componentes que los conforman
Sistemas Empotrados Distribuidos	Componentes de Sistemas Empotrados	Capacidad para Diseñar Sistemas Empotrados y Ubicuos

2 Herramientas de Síntesis de Alto Nivel

Desde finales de los años 80 han aparecido múltiples compañías ofreciendo herramientas de SAN. Actualmente Synopsys con la herramienta *Synphony* [9], Cadence con *C-to-Silicon* [12], Calypto con *Catapult C* [11] y Xilinx con *Vivado* [10] son las compañías que dominan el mercado. Además de las soluciones comerciales, desde el sector académico se han desarrollado herramientas muy completas como *SPARK* [13], *xPilot* [14], *ROCCC* [15] o *LegUp* [16]. De hecho, *xPilot*, desarrollada en la Universidad de California en Los Ángeles (UCLA), es el núcleo de lo que hoy es *Vivado*. En conclusión, tal y como corroboran los datos de la figura 1, la SAN es un mercado que está en creciente auge y expansión. Por tanto, es importante proporcionar alumnado una formación básica en el contexto del Diseño Automático.

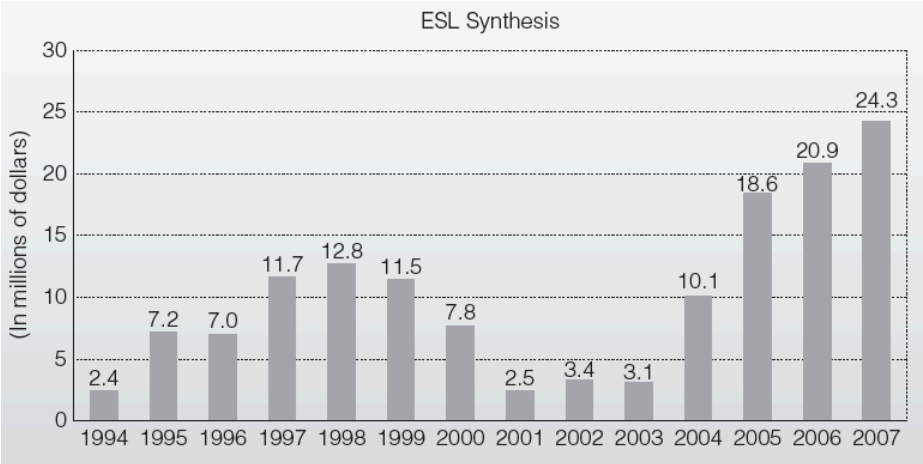

Figura 1. Venta de herramientas de Síntesis de Alto Nivel [17]

Quedando fuera de este artículo detalles más profundos de síntesis automática de alto nivel, podemos no obstante, abordar una breve descripción de la topología general de una herramienta SAN, que parte de un lenguaje de alto nivel, que desde comienzo de este siglo suele ser C [9-11] o variaciones como SystemC [12]. A continuación, transforma dicha descripción de alto nivel en un formato intermedio por medio de un *front-end* como el que ofrece LLVM [18]. En el caso de LLVM la representación sigue el formato AST (*Abstract Syntax Tree*). Posteriormente dicha información se procesa con algoritmos basados en grafos y se genera la descripción RTL que implementará el circuito.

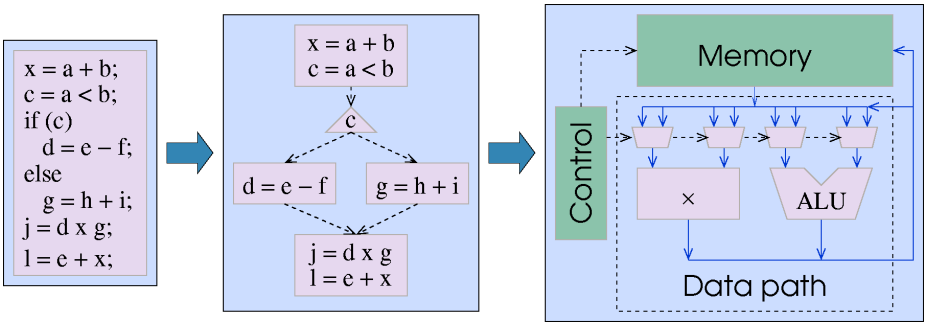

Figura 2. Flujo de Diseño en la Síntesis de Alto Nivel [11] desde C hasta RTL.

El flujo completo de diseño puede observarse en la Figura 2. Una vez obtenida la información de la descripción de entrada, las tareas realizadas por las herramientas de síntesis antes de generar la descripción RTL básicamente son 4:

1) Particionamiento (*partitioning*), o división del problema en subproblemas tratables.
2) Asignación de un número y tipo de recursos (*allocation*) suficientes para implementar el circuito.
3) Planificación (*scheduling*) de las operaciones en ciclos o control-steps (*csteps*).
4) Asignación (*binding*) de cada operación a instancias específicas de cada recurso.

Nuestra herramienta (HLStool) parte directamente de una representación en forma de grafo, y se centra fundamentalmente en las tareas 3 y 4. El objetivo es aportar una serie de interfaces gráficas que permitan al alumno comprender rápidamente que pequeñas modificaciones de las restricciones impuestas por el diseñador, pueden producir circuitos muy diferentes. Además, la visualización de los resultados permitirá comprender el funcionamiento de los distintos algoritmos de planificación y asignación utilizados.

3 HLStool

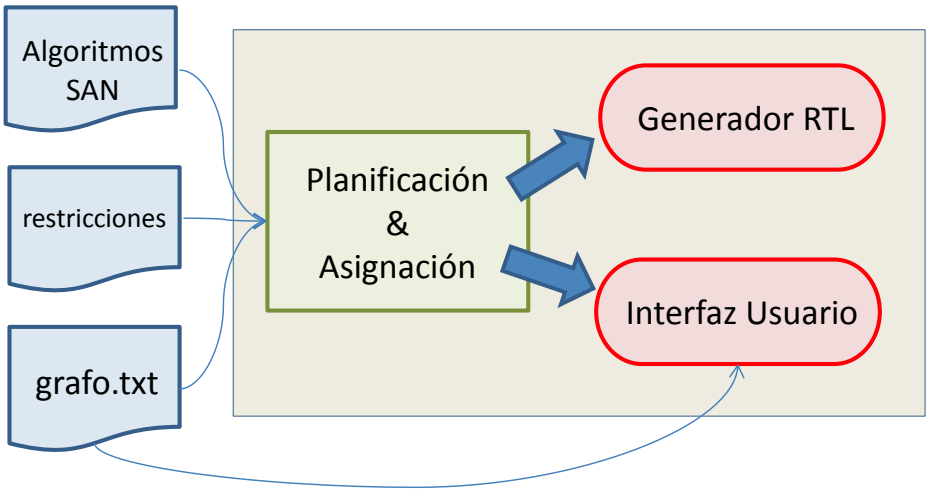

Figura 3. Estructura de la herramienta de síntesis propuesta (HLStool)

La estructura de HLStool puede observarse en la Figura 3. Inicialmente, se recibe un grafo a través de un fichero de texto. Dicho grafo puede mostrarse mediante un panel (*Dataflow Graph Panel*, o *DFG Panel*), aportando información acerca de las operaciones y sus relaciones. Además de ello, la herramienta recibe una configuración inicial que determinará qué algoritmos de planificación y asignación deben ejecutarse, y una serie de restricciones que especificarán detalles como la latencia del circuito o el número, tipo y latencia de los recursos utilizados (sumadores, multiplicadores, etc.). En otras palabras, la anteriormente mencionada tarea de *allocation* se realiza por medio de una restricción de usuario.

Nuestra herramienta implementa diversos algoritmos clásicos de planificación [7]:

1) As Soon As Possible (*ASAP*). Este algoritmo planifica las operaciones tan pronto como sea posible, es decir, en cuanto los operandos de entrada estén listos.
2) As Late As Possible (*ALAP*). Dada una latencia de circuito fijada por el usuario, el algoritmo planifica las operaciones tan tarde como sea posible.
3) Planificación basada en listas con prioridad. En cada *cstep* las operaciones disponibles se añaden a una lista de operaciones candidatas ordenadas por prioridad. De entre la lista de candidatas se seleccionarán las que se adecuen a las restricciones de usuario. Por ejemplo, si hay 4 sumas candidatas y tan solo 2 sumadores disponibles, se planificarán las 2 sumas de mayor prioridad.
4) Force Directed Algorithm (*FDS*) [19]. Dada una latencia de circuito, el algoritmo basado en fuerzas distribuye las operaciones de la forma más homogénea posible, para reducir el número de recursos necesarios para implementar el circuito.

Respecto a los algoritmos de asignación de operaciones, HLStool implementa las siguientes opciones [7]:

1) Algoritmos voraces para la asignación de operadores [7]. Dado un número de operadores de un tipo, y una planificación, cada operación se asigna al primer operador disponible.
2) Algoritmo *Left-Edge* o del Lado Izquierdo [7] para la asignación de registros. Dada una planificación, dicho algoritmo calcula el número mínimo de registros necesario para almacenar todas las variables, además de asignarlos.
3) Política *Least Recently Used* (LRU) para la asignación de registros. Para cada variable, esta política asigna el registro menos recientemente usado. En el contexto de la SAN clásica, tendría un efecto parecido al algoritmo *Left-Edge*, pero con la aparición de operadores de latencia variable [21,22] y la consecuente introducción de algoritmos de planificación dinámica para rutas de datos [20,21], puede tener un efecto apreciable sobre el rendimiento.

El resultado de ambas tareas se muestra por medio de un panel (*Operations Panel*), representando las operaciones con óvalos y los registros con rectángulos. Los colores determinarán a qué recurso en concreto está asignada cada operación. Un ejemplo de estas interfaces gráficas se describe en la siguiente sección. Además de la información visual, nuestra herramienta ofrece la funcionalidad de crear una descripción RTL generando sendas ruta de datos y controlador especificados en VHDL.

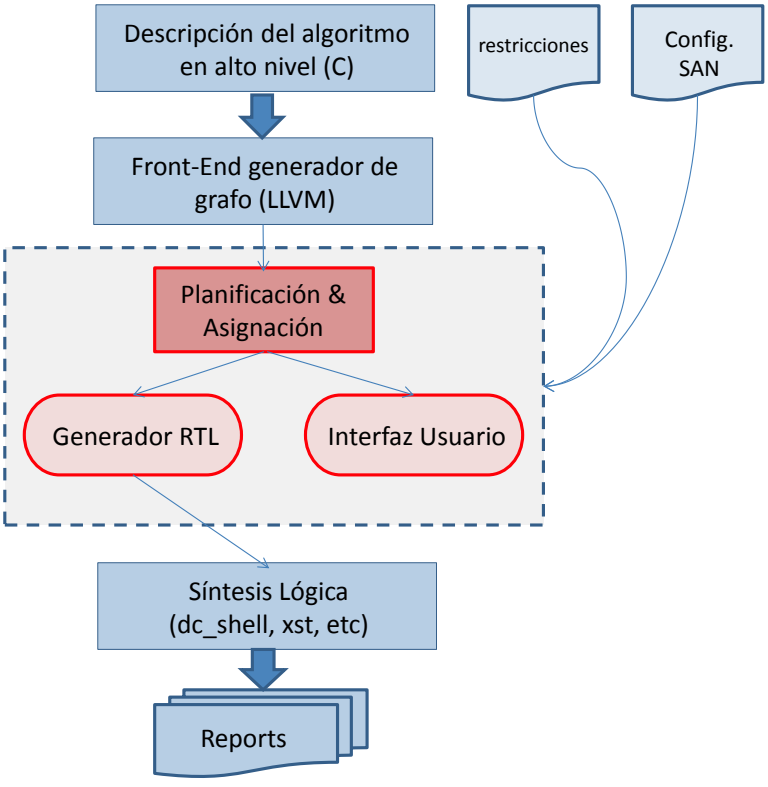

Figura 4. Flujo completo de diseño

Finalmente, la figura 4 muestra cómo se integraría HLStool en el flujo completo de diseño. Tan solo sería preciso proporcionar el grafo de entrada, que en nuestro caso es un fichero .txt. Para ello, bastaría con utilizar un frontend como *Clang*, proporcionado por LLVM [18]. *Clang* es capaz de construir un árbol en formato AST partiendo de un lenguaje de alto nivel como C. Por tanto, el AST nos proporcionaría el grafo. Por último, la descripción RTL generada por HLStool podría utilizarse para obtener datos de tiempo, área, consumo, etc. por medio de una herramienta de síntesis lógica.

4 Ejemplo Ilustrativo

En esta sección describiremos gráficamente los resultados correspondientes a utilizar la herramienta sobre el filtro Least Mean-Square (*LMS*) [23] aplicando el algoritmo de planificación basado en listas y el algoritmo Left-Edge para la asignación de registros. El código del LMS en C se encuentra en la figura 5 a). El panel del Dataflow Graph se encuentra en la Figura 5 b). Tal y como puede observarse, en este caso se trata de una instancia del código de la figura 5 a) en la que *length*=4.

a)

```
X_expected = 0;
for(i = 0; i < length; i++)
    X_expected += k[i]*x[i];
E = (X - X_expected)*Beta;
for(i = 0; i < length; i++)
    k[i] += E*k[i];
```

b)

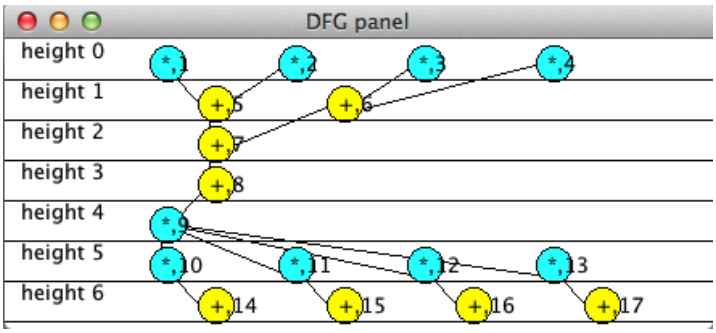

Figura 5. a) Código C del LMS filter, b) Dataflow Graph Panel para el filtro LMS [23]

Como puede observarse, cada operación aparece identificada por un *id* numérico, y el tipo de la misma viene determinado por un color: azul para las multiplicaciones y amarillo para las sumas. En la Figura 6 se muestra el resultado de la síntesis

utilizando como restricciones de entrada 4 multiplicadores de 4 ciclos de latencia, y 4 sumadores de 1 ciclo de latencia.

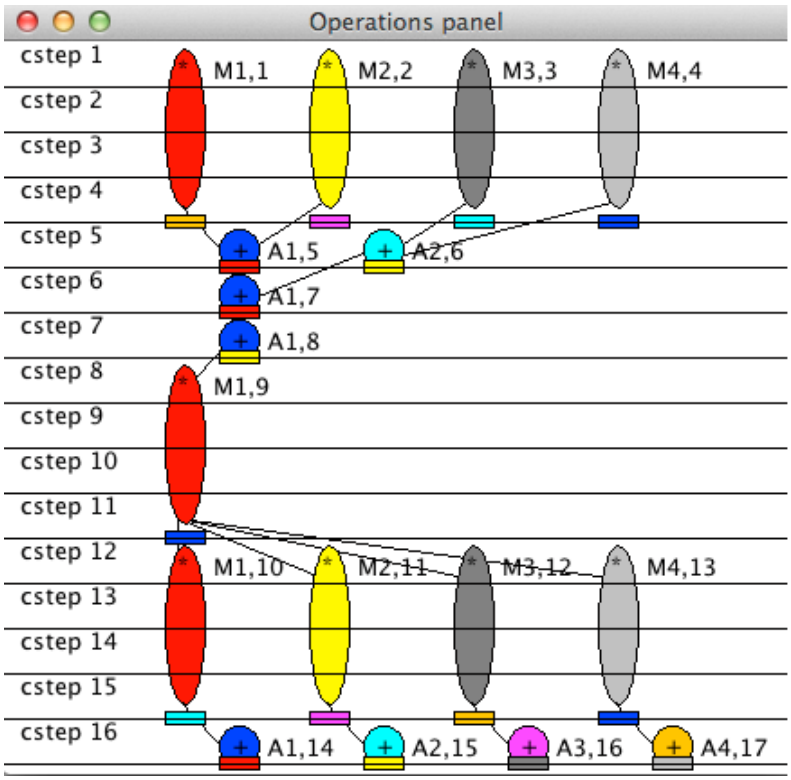

Figura 6. Panel de operaciones para el LMS [23], utilizando 4 multiplicadores de 4 ciclos de latencia y 4 sumadores de 1 ciclo de latencia.

El resultado producido por la herramienta sería una planificación de 16 ciclos de latencia y con la asignación mostrada por los colores de los recursos. Además, en el caso de los operadores dicha información viene complementada por una etiqueta identificativa del operador en el cual está asignada la operación. Por ejemplo, en el multiplicador M1, de color rojo, se han asignado las operaciones 1, 9 y 10. Si ahora se cambiasen las restricciones de entrada, el estudiante podría ver fácilmente cómo varía el circuito sin necesidad de generar la especificación RTL.

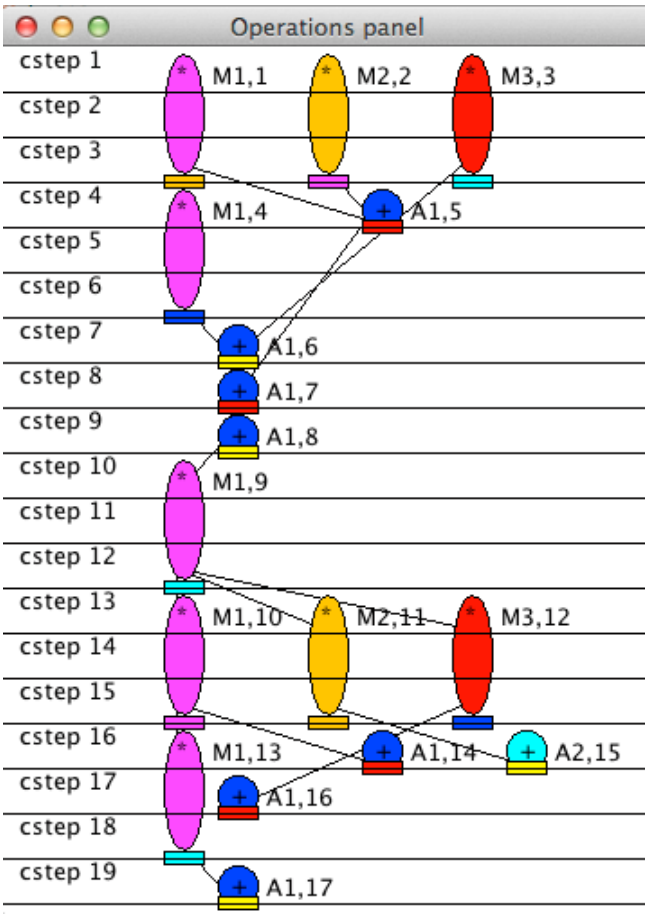

Figura 7. Panel de operaciones para el LMS [23], utilizando 3 multiplicadores de 3 ciclos de latencia y 2 sumadores de 1 ciclo de latencia.

La Figura 7 muestra el panel de operaciones tras cambiar las restricciones de entrada y utilizar 3 multiplicadores de 3 ciclos de latencia y 2 sumadores de 1 ciclo de latencia. En este caso, la latencia del circuito aumenta a 19 ciclos. Si quisiéramos resultados de área o potencia, generaremos la descripción RTL y utilizaremos una herramienta de síntesis lógica, como pueda ser *Design Compiler* de Synopsys [9] o *XST* de Xilinx [6].

5 Conclusiones

En este artículo se han descrito los potenciales beneficios del uso de una herramienta de Síntesis de Alto Nivel denominada HLStool, en asignaturas del área de Arquitectura y Tecnología de Computadores. En concreto, con la herramienta

propuesta, es muy sencillo evaluar el impacto que tendrían los cambios en las restricciones de entrada impuestas por el diseñador, por medio de una serie de interfaces gráficas que representan las operaciones, operadores y registros que aparecen en el circuito implementado. Todo esto sin la necesidad de realizar la síntesis lógica del circuito. Adicionalmente, el módulo de generación RTL permite obtener resultados de síntesis con cualquier herramienta comercial, lo que complementa la información de alto nivel proporcionada por la interfaz gráfica.

El uso de esta herramienta está pensada con un triple objetivo docente, por un lado formará de manera más completa a los estudiantes, por otro lado ayudará a que se familiaricen, en un tiempo reducido, a cómo trabajan las empresas relacionadas con el diseño de circuitos. Y adicionalmente, los alumnos aprenderán el funcionamiento de los algoritmos básicos usados en Síntesis de Alto Nivel.

Referencias

1. Facultad de Informática de la Universidad Complutense de Madrid: información docente, http://informatica.ucm.es/informatica/informacion-docente
2. Xilinx Inc.: IP cores, http://www.xilinx.com/products/intellectual-property/
3. PolyBench, http://www.cse.ohio-state.edu/~pouchet/software/polybench/
4. VHDL, http://www.vhdl.org/
5. N. Zainalabedin, VHDL:Modular Design and Synthesis of Cores and Systems, McGraw-Hill, 3ª ed., 2007.
6. Xilinx Inc.: ISE Design Suite, http://www.xilinx.com/products/design-tools/ise-design-suite/
7. Giovanni De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1ª ed., 1994.
8. D. McMillen, R. Camposano, D. Hill y T.W. Williams, An industrial view of Electronic Design Automation, IEEE Trans. On Computer-Aided Design, 2000, pp. 1428-1448.
9. Synphony, http://www.synopsys.com/Systems/BlockDesign/HLS/Pages/default.aspx
10. Vivado, http://www.xilinx.com/products/design-tools/vivado/
11. Catapult C, http://calypto.com/en/products/catapult/overview/
12. C-to-Silicon, http://www.cadence.com/products/sd/silicon_compiler/pages/default.aspx
13. SPARK, http://mesl.ucsd.edu/spark/methodology.shtml
14. xPilot, http://cadlab.cs.ucla.edu/soc/
15. ROCCC, http://www.jacquardcomputing.com/roccc/
16. LegUp, http://legup.eecg.utoronto.ca/
17. G. Martin y G. Smith, High-Level Synthesis: Past, Present, and Future, IEEE Design & Test of Computers, 2009, pp. 18-25.
18. LLVM, http://llvm.org/
19. P.G. Paulin y J.P. Knight, Force-Directed Scheduling in Automatic Data Path Synthesis, Design Automation Conference, 1987, pp. 195-202.
20. A.A. Del Barrio, S.O. Memik, M.C. Molina., J.M. Mendías y R. Hermida, "A Distributed Controller for Managing Speculative Functional Units in High Level Synthesis", IEEE Trans. On Computed-Aided Design, 2011, pp. 350-363.
21. A.A. Del Barrio, R. Hermida, S.O. Memik, J.M. Mendías y M.C. Molina, "Multispeculative Addition Applied to Datapath Synthesis", IEEE Trans. On Computer-Aided Design, 2012, pp. 1817-1830.
22. A.A. Del Barrio, R. Hermida y S.O. Memik, "Exploring the Energy Efficiency of Multispeculative Adders", IEEE International Conference on Computer Design, 2013, pp. 309-315.

23. E. Musoll y J. Cortadella, “Scheduling and resource binding for low power”, International Symposium on System Synthesis, 1995, pp. 104-109.