Citación: Vargas, J.R. (2010). Herramienta de software para el desarrollo de descripciones de hardware utilizando VHDL a partir de modelos gráficos. En: *Ingeniería, Vol. 15, No. 1.* 35-46.

Reporte de Caso

Herramienta de software para el desarrollo de descripciones de hardware utilizando VHDL a partir de modelos gráficos

A software tool for the development of graphic-model-based hardware descriptions using VHDL

José Roberto Vargas Rivero
Universidad Distrital
Francisco José de Caldas
Facultad de Ingeniería
jrvargasr@correo.udistrital.edu.co

Resumen

Con el objetivo de agilizar y facilitar el proceso de diseño de hardware se han realizado investigaciones relacionadas con la generación automática de descripciones VHDL a partir de modelos gráficos. En este documento se describe una herramienta de software que permite obtener la descripción VHDL a partir de un diagrama ASM o de un diagrama de componentes. Esta herramienta tiene la característica de que pueden reutilizarse descripciones ya creadas como nuevos componentes y la información contenida en diagramas ASM como parte de nuevos diagramas ASM. La herramienta es utilizada para generar la descripción VHDL de 15 circuitos comunes a partir de sus diagramas de componentes y ASMs, obteniéndose en un 53% de los casos descripciones equivalentes y en el resto de casos aproximadas.

Palabras clave: VHDL, ASM, componentes, herramienta, XML, jerarquía.

Abstract

With the purpose of making the process of hardware design quicker and easier, a number of investigations have been conducted concerning the automatic generation of VHDL descriptions based on graphic models. In this document, a software tool that generates VHDL descriptions based on either ASM diagrams or components diagrams will be presented. The characteristics of this tool allow a VHDL description already created to be reused like a new component. Additionally, the information contained in an ASM diagram can be reused as part of new ASM diagrams. The tool is used to generate the VHDL description of 15 well-known circuits from both their component diagrams and their ASM diagrams, obtaining an equivalent description in 53% of the cases and approximate descriptions in the remaining cases.

Key words: VHDL, ASM, components, tool, XML, hierarchy.

1. Introducción

A medida que el tamaño de los transistores disminuye más funcionalidad puede ser incluida en un dispositivo como una FPGA (*Field Programmable Gate Array*) y el sistema digital a desarrollar puede crecer en tamaño y complejidad; se utilizan dos técnicas para tratar con esta complejidad [1], [2]:

- Considerar niveles de jerarquía.
- Aumentar el nivel de abstracción del lenguaje en el cual el sistema puede ser desarrollado, hacia lenguajes gráficos.

Teniendo como objetivo implementar estas características se han desarrollado herramientas libres [2],[3],[4],[5],[6] que permiten la construcción de modelos gráficos de diagramas FSM (*Finite State Machine*) [3] y recientemente de diagramas de estado UML (*Unified Modeling Language*) [2], a partir de los cuales puede obtenerse de forma automática la descripción VHDL (*Very-high-speed integrated circuits Hardware Description Language*) correspondiente. Sin embargo los diagramas ASM (*Algorithmic State Machine*) son más descriptivos que los FSM y se adaptan mejor a diseños más complejos debido a su estructura similar a un diagrama de flujo [4], además de ser una representación alternativa mucho más común que los diagramas UML para los diseñadores de hardware [1],[7]. Por otra parte los diagramas mencionados permiten hacer descripciones funcionales [1] y no se cuenta con elementos para desarrollar descripciones estructurales. Por último no se cuenta con una herramienta libre que permita desarrollar diseños gráficos jerárquicos utilizando diagramas ASM y componentes en conjunto.

Con base en lo anterior se plantea el desarrollo de una herramienta de software que permita a partir de modelos gráficos simples de diagramas ASM y componentes extraer código VHDL sintetizable con la característica que estos modelos puedan organizarse de forma jerárquica.

2. Referentes

Ogubi y David [4] muestran las ventajas en tiempo de diseño y tamaño del circuito al emplear un lenguaje de nivel intermedio, entre aquellos de alto nivel como Java o C y los de descripción de hardware como VHDL, inspirado en la estructura de los ASM, que resulte más familiar a los programadores de software que los HDLs (*Hardware Description Language*) y a la vez permita un manejo de las temporizaciones más preciso que otras herramientas existentes como Handel-C y HardwareC, y que adicionalmente pueda ser sintetizado en alto nivel. Para este lenguaje intermedio se desarrolla un compilador. Como se evidenciará en este artículo la representación de ASM en texto facilita que desarrolladores de software implementen circuitos sin un conocimiento profundo de diseño hardware, reduciendo el tiempo de diseño.

En [8] se desarrolla una investigación sobre la optimización de descripciones VHDL a nivel funcional. Para lograrlo se interpretan las estructuras implícitas en una descripción VHDL. Dentro de esta investigación se desarrolló una herramienta de software. Uno de los elementos que conforman este software reconoce las palabras clave presentes en una descripción VHDL. A partir de estas palabras se crea una estructura jerárquica de obje-

tos en la cual se almacena cada tipo de elemento existente. De la declaración de entidad se extraen los puertos de entrada y salida mientras que de la arquitectura se extraen los elementos definidos por las palabras clave: *case*, *component*, *if*, *else*, *elseif*, *null*, *portmap*, *process* y *when*. A partir de las palabras clave y los elementos contiguos se generan objetos Java.

Con relación a herramientas con interfaz gráfica, en [9] se presenta una herramienta capaz de obtener el código VHDL funcional de una representación FSM gráfica. La metodología que los autores manejan es partir de una herramienta de edición gráfica ya existente llamada Graphviz que les permite a los usuarios de forma interactiva explorar grandes espacios de estados y diagramar gráficas de estado transición (STG) del diseño requerido. La salida de este programa es un archivo en lenguaje (*dot*) el cual es una representación textual de la máquina FSM. La herramienta convierte el archivo *dot* a formato estándar *kiss2* usado para representar FSMs. Este archivo *kiss2* se utiliza por otro módulo de la herramienta para directamente generar código VHDL.

En [5] se presenta una aproximación al proceso inverso que es la extracción de un modelo gráfico, es decir un STG a partir del código HDL, antecediéndolo un proceso de optimización de los FSM obtenidos, teniendo como única limitante sobre el código HDL que este sea sintetizable.

Con relación a los modelos gráficos a utilizar, en [6] se propone un estándar gráfico para la representación de VHDL estructural, dejando abierto a los vendedores de herramientas de software la representación gráfica funcional, de forma similar a como se realizó con lenguajes de descripción formales como SDL y MSC.

Más recientemente, en [2] se describe un proceso y una herramienta de software para pasar de los diagramas de estado de UML a código VHDL automáticamente. Los elementos que los autores contemplan en tal proceso son los siguientes:

1. Editor de diagrama de estado: Provee las herramientas necesarias para permitir que el usuario ingrese especificaciones de diagramas de estado de forma cómoda, los autores comentan que para esta etapa hay dos alternativas, la primera utilizar una herramienta ya creada y emplear un formato de intercambio o construir una nueva herramienta.
2. Metamodelo de diagrama de estado y metamodelo de VHDL: En esta etapa se indica que el metamodelo de diagrama de estado para UML está definido de manera estándar, pero en el caso de VHDL los autores lo definen.
3. Transformación de modelo de diagrama de estado a modelo VHDL: En esta etapa se aplican un conjunto de reglas para la transformación de cada uno de los elementos del metamodelo de diagrama de estado al de VHDL.
4. Obtención del código VHDL: La etapa final del proceso consiste en pasar del modelo VHDL obtenido a texto utilizando nuevamente un conjunto de reglas.

3. Desarrollo

El proceso de obtención de la descripción VHDL es abordado utilizando como interfaz gráfica la herramienta Draw de OpenOffice.org [10] que tiene como característica gene-

rar un archivo estándar compuesto por un conjunto de documentos XML, los cuales almacenan la información sobre la estructura y contenido de los modelos gráficos. Examinando el código XML de estos archivos, se puede extraer la información necesaria para la reconstrucción de la descripción VHDL. Esta información debe organizarse y los elementos de sintaxis necesarios deben agregarse.

Por otra parte, la posibilidad de hacer uso de niveles de jerarquía en modelos gráficos de ASM y componentes, se logra mediante la definición de elementos que permiten la inclusión de un diagrama ya elaborado, dentro de otro.

Por último, mediante un software desarrollado en el lenguaje de programación Java se aplican los métodos propuestos. Esta herramienta permite corroborar la metodología al utilizarla para generar descripciones VHDL a partir de varios modelos gráficos de diagramas ASMs y de componentes, en los cuales se utilizan niveles de jerarquía.

3.1 Elementos gráficos utilizados para la representación de diagramas ASM y de componentes

A continuación se presentan algunos equivalentes entre el modelo gráfico convencional de los elementos que conforman un diagrama ASM y el modelo obtenido utilizando elementos gráficos provistos por la herramienta Draw de OpenOffice.org, que fue utilizada en esta propuesta.

En relación con los diagramas ASM, se dispone de bloques de estado, condición, y condicionales (ver Figura 1). Adicionalmente ya que a partir de los modelos gráficos desarrollados con esta herramienta se debe obtener código VHDL sintetizable, es necesario que se definan elementos que no se encuentran en un diagrama ASM convencional. Por ejemplo, se requiere que el usuario tenga la posibilidad de definir cuales serán las entradas, salidas y señales (*signals*) que compondrán la declaración de entidad de la descripción VHDL, para este propósito se definió el bloque I/O.

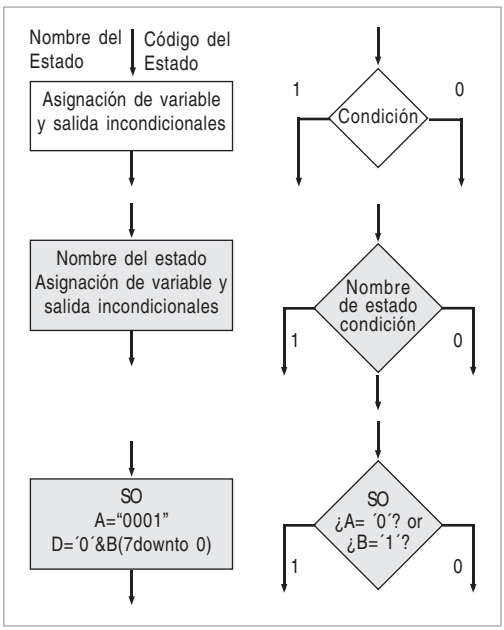

Figura 1. Diagrama de un bloque de estado y condición, (arriba) modelo, (medio) estructura, (abajo) ejemplo

A su vez para identificar los puntos en los cuales comienza y termina un diagrama ASM se definieron los bloques de RESET y de FIN. Una representación de la forma en que se conectarían estos bloques se presenta en la Figura 2, el bloque de FIN es opcional debido a que el último estado puede saltar a un estado anterior.

Adicionalmente, dado que se requiere que el usuario pueda reutilizar estructuras para poder implementar niveles de jerar-

quía se utiliza la palabra reservada *HNAME*, la cual en caso de incluirse al inicio del bloque I/O indica que el diagrama ASM será reutilizado como parte de un nuevo diagrama, en este caso los indicadores A y B se usan para marcar el inicio y fin de la estructura, como se ilustra en la Figura 3.

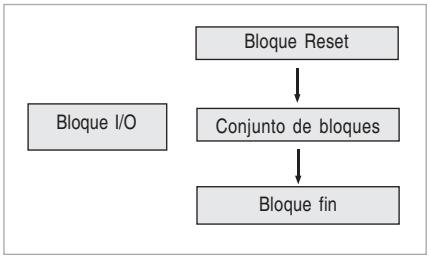

Figura 2. Diagrama de conexión bloques adicionales

La información que se extrae de un gráfico ASM para reutilización se guarda en un archivo de texto simple (.txt) en una serie de caracteres con sus elementos separados por viñetas y organizados por secciones que luego pueden ser leídas con facilidad, este tipo de elementos se llamarán elementos heredados. Un ejemplo se presenta en la Figura 4.

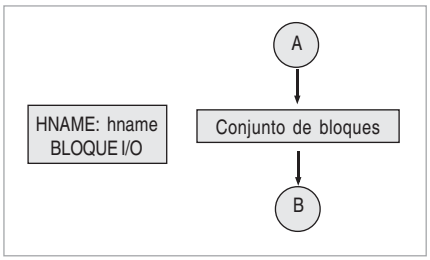

Figura 3. Diagrama de conexión para reutilización

Con relación a los diagramas de componentes se utilizan compuertas AND, OR y NOT dado que con estas se puede generar cualquier otra compuerta. Sin embargo dado que no es práctico elaborar circuitos complejos únicamente a partir de estas compuertas, se emplean los componentes VHDL los cuales son creados a partir de una descripción VHDL. Así por ejemplo a partir del archivo 'xor_of.vhd' que contiene la descripción funcional o estructural de una compuerta 'xor' se genera automáticamente un elemento gráfico que puede ser conectado a otros elementos permitiendo la creación de circuitos cada vez más complejos como se ilustra en la Figura 5. En ella puede observarse la estructura de un componente VHDL, este elemento esta constituido por un número (*m*) de entradas y un número (*p*) de salidas que dependerán de la declaración de entidad de la cual se genere. Y cuyo nombre sigue la estructura: 'nombre del archivo sin extensión' +'.'+un número único de componente 1,2,…*n*.

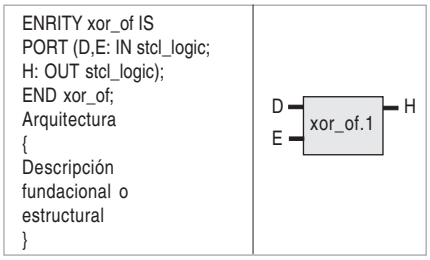

Figura 4. Ejemplo de generación de un componente VHDL

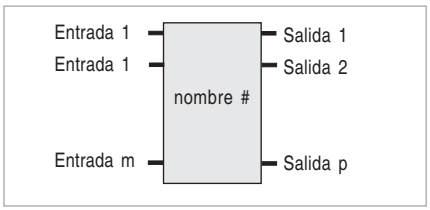

Figura 5. Modelo general de un componente VHDL

Los componentes que se pueden importar no están limitados a aquellos cuya descripción VHDL ha sido generada con la herramienta, lo cual permite al diseñador hacer uso de su librería particular de componentes para incorporar al modelo gráfico.

En la Figura 6 se muestra el elemento de entrada, salida o señal (I/O/S), el cual se utiliza para representar las entradas, salidas y señales en un diagrama estructural.

3.2 Estructura y características del código XML generado

El archivo context.xml es uno de los archivos que conforman un documento guardado en Draw de OpenOffice.org. Este contiene la información sobre todas las figuras que hayan sido creadas. Cada figura se transforma en un elemento XML específico que tiene un grupo de atributos como un identificador "draw:id", la posición del elemento "svg:x" y "svg:y", entre otros. El texto interno se transforma en elementos subordinados tipo "text:p". En la Figura 7 se presenta un elemento tipo "rect" en el cual se transforman los componentes VHDL, nótese la presencia de elementos hijos tipo draw:gluepoint que corresponden a las entradas o salidas del componente. Los elementos I/O/S a su vez se transforman en elementos tipo "circle" o "ellipse".

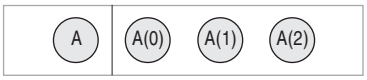

Figura 6. Elemento I/O/S, (izquierda) elemento de 1 bit (derecha) vector

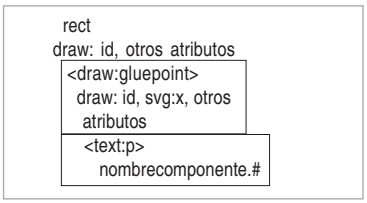

Figura 7. Elemento XML tipo rect

A nivel de los diagramas ASM también encontramos elementos tipo "rect" o "custom-shape", el elemento subordinado tipo "text:p" contiene la información sobre el nombre del estado y las asignaciones que en este se realicen, como se observa en la Figura 8.

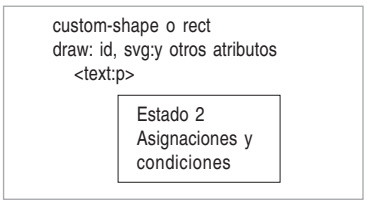

Figura 8. Elemento XML tipo custom-shape o rect

En ambos tipos de diagramas la interconexión de los diferentes elementos se hace con el uso de conectores los cuales se transforman en elementos tipo "connector". Estos elementos tienen los atributos "draw:start-shape" y "draw:end-shape" que corresponden al atributo "draw:id" del elemento del cual parten y al cual llegan respectivamente, esto permite identificar las conexiones en un diagrama de componente y la secuencia de estados en un diagrama ASM (ver Figura 9).

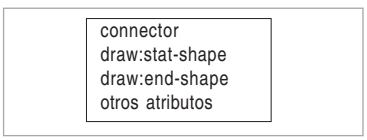

Figura 9. Elemento XML tipo connector

3.3 Extracción del contenido del código XML requerido para construir la descripción VHDL

La información contenida en cada uno de los elementos XML debe extraerse para poder construir la descripción VHDL correspondiente; con este objetivo se utilizó XPath [11] el cual es un lenguaje de consulta que utiliza expresiones con sintaxis simple, compuesta por una ruta y uno o más predicados opcionales, y que retorna todos los nodos que concuerdan con la expresión [8].

La estructura genérica de una consulta XPath es:

(Tipo de variable retornada) nombre = xpath.evaluate (ruta+[expresión], documento, XPathConstants.(Tipo de variable)) (1)

Así por ejemplo para extraer el número de elementos tipo "circle" se utiliza la expresión:

Número de elementos tipo circulo=

XPath(documentcontent/body/drawing/page/circle, doc,XPathConstants.NUMBER) (2)

3.4 Construcción de objetos VHDL para diagramas de componentes y ASMs

A partir de las variables obtenidas con base en las consultas XPath, es necesario reconstruir la descripción VHDL tanto de un diagrama de componentes como ASM. Para esto se requiere definir un método en el cual se organice la información de una manera mas adecuada. Con este objetivo se definieron los objetos VHDL, que son piezas de la descripción VHDL en las cuales se agrupan los datos extraídos del XML.

A continuación se describe de forma general la secuencia de pasos a seguir para cada tipo de diagrama.

Para *diagramas de componentes*:

1) Identificar las entradas (Objeto entradas).
2) Identificar las salidas (Objeto salidas).
3) Identificar elementos comunes a entradas y salidas que corresponden a señales (Objeto señales).
4) Relacionar las entradas, las salidas y las señales con los componentes utilizando los atributos "draw:id".
5) Clasificar los componentes según su texto interno.
6) Organizar las entradas y salidas en listas de asociación (Objeto instancias de componentes).
7) Añadir las declaraciones de los componentes a partir de la carpeta que agrupa los archivos VHDL del usuario (Objeto declaraciones de componentes).

Para *diagramas ASM*:

1) Identificar si se debe generar un arreglo o una descripción VHDL.

Si es el primer caso generar el arreglo a partir de la secuencia de bloques y el bloque I/O. Si es el segundo caso:

2) Construir la entidad a partir del bloque I/O (Objeto entradas, Objeto salidas y Objeto señales).
3) Organizar los bloques a partir de su posición vertical.
4) Leer la secuencia a partir del bloque Reset, utilizando los atributos "draw:id" para identificar los estados siguientes (Objeto asignaciones de reset).
5) Añadir la información de los elementos heredados.

6) Identificar los nombres de los estados y las acciones que se realizan en cada estado.

7) Identificar los bloques de decisión y los estados siguientes (Objeto estados).

3.5 Construcción de la descripción VHDL a partir de los objetos VHDL

Utilizando las reglas de sintaxis del lenguaje VHDL y los objetos obtenidos, es posible obtener una descripción VHDL sintetizable correspondiente al diagrama inicial. La estructura que se propone para cada tipo de diagrama es la siguiente:

Para *diagramas de componentes*:

Entidad{
Objeto entradas, Objeto salidas
}

Objeto componente
Arquitectura{
Objeto señales
Objeto declaraciones de componentes
Objeto instancias de componentes
}

Para *diagramas ASM*:

Entidad{
Objeto entradas, Objeto salidas
}

Objeto componente
Arquitectura{
Objeto señales
Objeto asignaciones de reset
Objeto estados
}

4. Resultados

4.1 Ejemplos

A continuación se presentan tres ejemplos contruidos con la herramienta: el primero es un diagrama ASM para la serie de Fibonacci (Figura 11); el segundo es un diagrama de componentes correspondiente a un contador ascendente descendente de 4 bits (Figura 10) con su modelo gráfico; el tercero es un diagrama ASM de un circuito arbitrador con prioridad alternada utilizando niveles de jerarquía (Figura 12) en el cual se reutiliza dos veces la estructura del arbitrador simple; además se presenta la arquitectura de la descripción VHDL obtenida en los dos primeros casos.

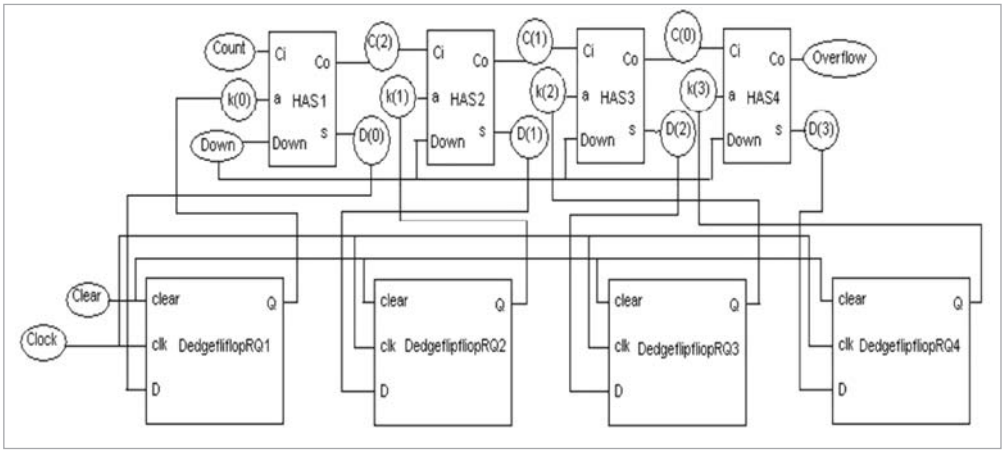

Figura 10. Diagrama ASM del contador ascendente descendente de 4 bits

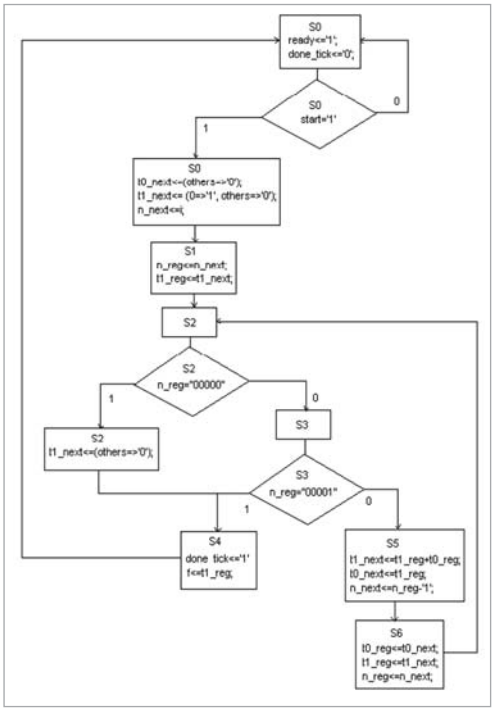

Figura 11. Diagrama ASM para el cálculo de la serie de Fibonacci

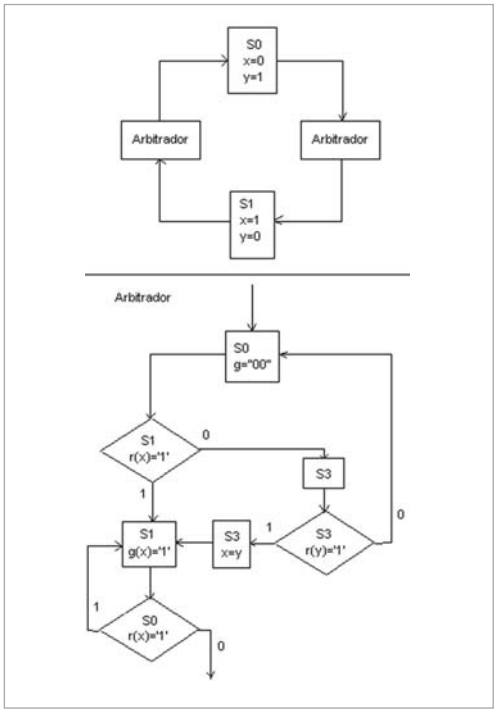

Figura 12. Diagrama ASM para un arbitrador con prioridad alternada utilizando niveles de jerarquía

Nótese que en el caso del diagrama de componentes, el HAS (Half adder substractor) y el DedgeflipflopRQ (flip-flop tipo D con disparo por flanco positivo) son incorporados automáticamente como componentes al igual que su entidad.

4.1.1. Serie de Fibonacci

```
when S0 =>
ready<='1';
done_tick<='0';
if (start='1') then
t0_next<=(others=>'0');
t1_next<= (0=>'1', others=>'0');
n_next<=i;
sx<=S1;
else
sx<=S0;
end if;
when S1 =>
n_reg<=n_next;
t1_reg<=t1_next;
sx<=S2;
when S2 =>
null;
 if (n_reg="00000") then
t1_next<=(others=>'0');
sx<=S4;
else
sx<=S3;
end if;
when S3 =>
null;
 if (n_reg="00001") then
sx<=S4;
else
sx<=S5;
end if;
when S4 =>
done_tick<='1';
sx<=S0;
f<=t1_reg;
when S5 =>
t1_next<=t1_reg+t0_reg;
t0_next<=t1_reg;
n_next<=n_reg-'1';
sx<=S6;
when S6 =>
t0_reg<=t0_next;
t1_reg<=t1_next;
n_reg<=n_next;
sx<=S2;
end case;
 end if;
 end process;
```

4.1.2. Contador ascendente descendente de 4 bits

```
component HAS is
port ( Ci : in std_logic;
a : in std_logic;
Down : in std_logic;
Co : out std_logic;
s : out std_logic);
end component;
component DedgeflipflopRQ is
port ( clear : in std_logic;
clk : in std_logic;
D : in std_logic;
Q : out std_logic);
end component;

begin
HAS4: HAS port map (C(2),K(3),Down,Overflow,D(3));
DedgeflipflopRQ4: DedgeflipflopRQ port map (Clear,Clock,D(3),K(3));
HAS3: HAS port map
  (C(1),K(2),Down,C(2),D(2));


DedgeflipflopRQ3: DedgeflipflopRQ port map (Clear,Clock,D(2),K(2));
HAS2: HAS port map
(C(0),K(1),Down,C(1),D(1));
DedgeflipflopRQ2: DedgeflipflopRQ port map (Clear,Clock,D(1),K(1));
HAS1: HAS port map (Count,K(0),Down,C(0),D(0));
DedgeflipflopRQ1: DedgeflipflopRQ port map (Clear,Clock,D(0),K(0));
Q(3)<=K(3);
Q(2)<=K(2);
Q(1)<=K(1);
Q(0)<=K(0);
```

4.2 Simulaciones

La herramienta de software se creó utilizando el lenguaje de programación Java el cual es libre y multiplataforma. Para propósitos de validación de la herramienta, se generaron las descripciones VHDL de 15 circuitos; 5 a partir de su diagrama de componentes, 5 a partir de su diagrama ASM y 5 en los que se aplican características adicionales como el uso de vectores, niveles de jerarquía y reutilización de código existente [1], [7], [9], [12].

En la Tabla I se resumen los resultados obtenidos comparando las simulaciones de la descripción VHDL original y la obtenida con la herramienta utilizando el simulador Modelsim XEIII 3.4b. La comparación se basa en la equivalencia funcional (EF) según la tabla de verdad o descripción del circuito.

Con respecto a los resultados obtenidos la equivalencia funcional parcial que se observa en algunas simulaciones es consecuencia de que el número total de estados es mayor en las descripciones obtenidas con la herramienta desarrollada. Esto se debe a que los bloques ASM complejos deben descomponerse en bloques que solo tengan 1 bloque de decisión. Esta limitante del algoritmo se basa en el número creciente de posibilidades que se generan sobre la descripción VHDL a obtener, cuando aumenta el número de bloques de decisión en un mismo estado. En estos casos la descripción generada por la herramienta y la original corresponden a diferentes circuitos. Se establece el calificativo de parcial para indicar que aunque el circuito obtenido es distinto la misma funcionalidad se puede replicar.

Otro punto a mencionar es que en la Tabla I se presentan tiempos de generación mayores cuando la descripción VHDL se obtiene a partir de un diagrama de componentes que cuando se obtiene a partir de un diagrama ASM, lo cual se puede explicar a raíz de que el análisis de la estructura de este tipo diagramas es computacionalmente mas exigente que el de un diagrama ASM, el cual es primordialmente lineal. Por último puede observarse el impacto en el tiempo de generación en los últimos cinco circuitos en los cuales se implementaron niveles de jerarquía.

Tabla I. Resultados obtenidos

Circuito	EF
Codificador BCD 7 segmentos	Si
Sumador Restador	Si
Decodificador 3 a 8	Si
Flip-Flop tipo D con disparo por flanco positivo	Si
Ejemplo de un FSM de Moore	Si
Tiempo promedio de generación (segs.)	13.32
Cálculo polinomio	Parcial
Promedio	Parcial
Raíz cuadrada	Parcial
Medidor de periodo	Parcial
Serie de Fibonacci	Parcial
Tiempo promedio de generación (segs.)	2.74
Unidad aritmética lógica	Si
Contador ascendente-descendente 4 bits	Si
Raíz cuadrada con niveles de jerarquía	Parcial
Medidor de baja frecuencia	Si
Arbitrador	Parcial
Tiempo promedio de generación (segs.)	***44.8***

4.3 Trabajo futuro

El algoritmo para interpretación de diagramas ASM debe mejorarse de tal forma que se puedan interpretar diagramas con más de un bloque de decisión por estado. Al mismo tiempo el algoritmo que realiza la lectura de las declaraciones de entidad para crear los componentes VHDL puede mejorarse de tal forma que existan menos limitantes con respecto al contenido y organización de las declaraciones que se pueden interpretar. También puede mejorarse el manejo de vectores ya que si se genera una entrada o salida por cada elemento de un vector, el tamaño del componente VHDL a nivel gráfico aumenta drásticamente; por otra parte, si solo se genera una entrada o salida para todo el vector se pierde la posibilidad de conectar individualmente cada uno de sus elementos.

Finalmente se planea lograr una mayor integración con el trabajo desarrollado en [8] de tal forma que se pueda realizar optimización sobre la descripción VHDL generada a partir del modelo gráfico.

5. Conclusiones

En este estudio se encontró que el uso de XML como lenguaje intermedio para la transformación entre un diagrama de componentes o ASM y una descripción VHDL genera ventajas en cuanto a la portabilidad al ser un archivo de texto y la facilidad de extracción de la información con herramientas como XPath.

De la misma forma, se evidencia que las descripciones VHDL de diagramas de componentes o ASMs pueden ser representadas en modelos gráficos con el uso de un número pequeño de elementos de construcción.

Por último, se resalta que el uso de niveles de jerarquía en modelos gráficos de diagramas ASMs y de componentes puede implementarse a partir de la definición de elementos que simbolicen la información contenida en una descripción VHDL o algoritmo previamente elaborado.

6. Referencias bibliográficas

[1] Pong P Chu. (2006). RTL Hardware Design Using VHDL. *Jhon Wiley & Sons, Inc. Hoboken, New Jersey*. 21, 25,317,357,474,475.

[2] Wood, Steve K, Akehurst David H, Uzenkov Oleg, Howells W, McDonald-Maier, Klaus D. (2008). A Model-Driven Development Approach to Mapping UML State Diagrams to Synthesizable VHDL. *IEEE Transactions on Computers. Vol. 57*. 1357-1371.

[3] Abdel-Hamid, Mohamed Zaki, Sofiéne Tahar. (2004). A tool converting finite state machine to VHDL. *Canadian Conference on Electrical and Computer Engineering. Vol. 4, 2-5*. 1907-1910.

[4] Etienne Ogoubi, Jean Pierre David. (2004). Automatic synthesis from high level ASM to VHDL: a case study. *The 2nd Annual IEEE Northeast Workshop on Circuits and Systems*. 81-84.

[5] Chien-Nan Liu, Jing-Yang Jou. (2000). An Automatic controller extractor for HDL Descriptions at the RTL. *IEEE Design & Test of Computers. Vol. 17*. 72-77.

[6] Thomas Hadlich. (1997). Proposing graphic extensions to VHDL. *VHDL International User's Forum (VIUF '97)*. 109-115.

[7] Enoch O Wang. (2005). Digital Logic and Microprocessor Design with VHDL Brooks/Cole. 80, 84, 110, 116, 118, 185, 210, 271, 367, 368.

[8] Oscar Javier Méndez Zuluaga. (2007). Modificación de ASMs sobre descripciones funcionales de VHDL. *Trabajo de grado Universidad Distrital Francisco José de Caldas*.

[9] Volnei A Pedroni. (2004). Circuit Design with VHDL. *MIT Press. Cambridge Massachusetts*. 3.

[10] Open Office. (2010). OpenOffice.org 3.3.0. OpenOffice.org project. 2011. Consultado: http://www.openoffice.org. (7 de septiembre, 2010).

[11] Oracle. (2010). Package javax.xml.xpath . Oracle. 2011. Consultado: http://download.oracle.com/javase/1,5.0/docs/api/javax/xml/xpath/package-summary.html. (7 de septiembre, 2010).

[12] Pong P Chu. (2008). FPGA prototyping by VHDL examples. *Jhon Wiley & Sons, Inc. Hoboken, New Jersey*. 109,110,153.

Jose Roberto Vargas Rivero

Ingeniero Electrónico de la Universidad Distrital Francisco José de Caldas, de Bogotá, Colombia. Miembro del grupo de investigaciòn LAMIC, Universidad Distrital Francisco Jose de Caldas, Bogota, Colombia.