Aplicación de Los Estilos de Descripción en VHDL

Daniel Camilo Rolón Quintero - *Cód.: 1160467*, Diego Andrés Parada Rozo - *Cód.: 1160449*

***Resumen*—En este documento se presentan tres estilos de descripción en VHDL: descripción algorítmica o funcional, descripción flujo de datos o Data Flow y descripción estructural. El procedimiento consiste en implementar los estilos mencionados a un ejemplo común, además de analizar la función lógica booleana que lo representa, encontrar la tabla de la verdad que describe su comportamiento, construir su símbolo y realizar su diagrama de tiempos. Todo ello implementando la herramienta Xilinx WebPack. El principal objetivo consiste en determinar cuáles son las ventajas y desventajas que se le atribuyen a cada estilo de descripción en VHDL.**

***Index Terms*—Función lógica booleana, when - else, componente, estilo de descripción funcional, data flow, estructural.**

I. Introducción

ELVHDL permite al diseñador describir un circuito a través de dos formas. Por un lado se puede describir indicando los diferentes componentes que lo forman y su interconexión, de esta manera se tiene especificado un circuito y se sabe cómo funciona. Las herramientas que utiliza esta forma son las capturas de esquemas y las de descripción *Netlist*[1,2].[1]

La segunda forma consiste en describir un circuito indicando lo que hace o cómo funciona, es decir, describiendo su comportamiento. Naturalmente esta forma de describir un circuito es preferible para un diseñador puesto que lo que realmente le interesa es el funcionamiento del circuito más que sus componentes. Por otro lado, al encontrarse lejos de lo que es realmente un circuito, se pueden plantear algunos problemas al momento de implementarlo a partir de la descripción de su comportamiento.[1]

El VHDL va a ser interesante puesto que va a permitir los dos tipos de descripciones[3][1]:

- *Estructura*: VHDL puede ser usado como lenguaje de Netlist común y corriente donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.
- *Comportamiento*: VHDL también se puede utilizar para la descripción funcional o comportamental de un circuito. Ésta es la diferencia con un lenguaje de *Netlist*. Sin necesidad de conocer la estructura interna de un circuito es posible describirlo explicando su funcionalidad. Esto es especialmente útil en simulación, ya que permite simular un sistema sin conocer su estructura interna. Así, este tipo de herramientas de síntesis permiten la creación automática de circuitos a partir de una descripción de su funcionamiento.

Muchas veces la descripción comportamental se divide a su vez en dos, dependiendo del nivel de abstracción y del modo en que se ejecuten las instrucciones. Estas dos formas comportamentales de describir circuitos son la de *flujo de datos* y la *algorítmica*.[4][1]

II. Objetivos

II-A. Objetivo General

Aplicar a un sistema electrónico digital los tres estilos básicos de descripción en lenguaje VHDL: funcional, flujo de datos, y estructural, para describir su comportamiento, identificar sus componentes fundamentales al interior de cada unidad básica de diseño y deducir las ventajas y desventajas de cada uno.

II-B. Objetivos Específicos

- Analizar la función lógica booleana que representa el sistema electrónico digital a implementar.
- Encontrar la tabla de la verdad que describe su comportamiento.
- Construir el símbolo esquemático que lo representa.
- Realizar el diagrama de tiempos que muestre el comportamiento de las señales de entrada y salida del sistema.

III. Marco Teórico

III-A. Estilos de Descripción en VHDL

VHDL presenta tres estilos de descripción de circuitos dependiendo del nivel de abstracción. El menos abstracto es una descripción puramente estructural. Los otros dos estilos representan una descripción comportamental o funcional, y la diferencia proviene de la utilización o no de la ejecución serie.[1]

La sintaxis del VHDL no es sensible a mayúsculas o minúsculas, por lo que se debe escribir como se prefiera. En primer lugar, sea el tipo de descripción que sea, hay que definir el símbolo o entidad del circuito. En efecto lo primero es precisar las entradas y salidas del circuito, es decir, la caja negra que lo define. Se la llama entidad porque en la sintaxis de VHDL se declara con la palabra `entity`. Esta definición de entidad, que suele ser la primera parte de toda descripción VHDL, se expone en la fig. 1 [1], [2].[5]

[1] El *Netlist*, o lista de conexiones, es la primera forma de describir un circuito mediante un lenguaje, y consiste en dar una lista de componentes, sus interconexiones y las entradas y salidas. No es un lenguaje de alto nivel, por lo que no describe cómo funciona el circuito, sino que simplemente se limita a describir los componentes que posee y las conexiones entre ellos.

[2] Pág. 30 de la referencia 1.

[3] Pág. 42 de la referencia 1.

[4] Pág. 42 y 43 de la referencia 1.

[5] Pág. 44 de la referencia 1, pág. 3 de la referencia 2.

```
Formato_de_la_entidad.vhd
File  Edit  View  Window
--Formato de la entidad

entity Nombre_de_la_entidad is
   generic(declaración_de_las_constantes);
   port(declaración_de_los_puertos);
end Nombre_de_la_entidad

--Formato de las constantes y los puertos:

-- generic(Nombre_de_las_constantes : tipo := valor);
-- port(Nombre_de_los_puertos : modo tipo);
```

Figura 1. Aproximación al formato de la entidad, las constantes y los puertos.

La entidad de un circuito es única. Sin embargo, un mismo símbolo, en este caso entidad, podía tener varias vistas, que en el caso de VHDL se llaman arquitecturas. Cada bloque de arquitectura, que es donde se describe el circuito, puede ser una representación diferente del mismo circuito. Por ejemplo, puede haber una descripción estructural y otra comportamental, las dos son descripciones diferentes, pero ambas corresponden al mismo circuito, símbolo o entidad. [1]

III-A1. Descripción Algorítmica o Funcional `if-then-else`*:* El formato de este estilo de descripción se muestra en la fig. 2 [2][6].

```
Formato_de_descripción_funcional.vhd
File  Edit  View  Window
architecture Nombre_de_la_arquitectura of Nombre_de_la_entidad is
begin
   process(lista_sensible)
   begin
      if condición_1 then
         secuencia_de_instrucciones;
      else if condición_n then
         secuencia_de_instrucciones;
      else instrucciones_por_defecto;
      end if;
   end process;
end Nombre_de_la_arquitectura;
```

Figura 2. Aproximación al formato de descripción funcional `if-then-else`.

Como primera aproximación, se considera que el bloque `process` es una especie de subrutina cuyas instrucciones se ejecutan secuencialmente cada vez que alguna de las señales de la *lista sensible* cambia. Ésta es una lista de señales que se suele poner junto a la clave `process`.[1]

Esta descripción comportamental es muy sencilla de entender, ya que se sigue un estructura parecida a los lenguajes de programación convencionales. Es por lo que se dice que se trata de una descripción comportamental algorítmica. Lo que se está indicando es simplemente que si se cumple con una(s) condición(es) específica(s) entonces se procede a cumplir con una secuencia de instrucciones. Esta forma tan sencilla de describir el circuito permite a ciertas herramientas sintetizar el diseño a partir de una descripción comportamental. La diferencia con un *Netlist* es directa: en una descripción comportamental no se están indicando ni los componentes ni sus interconexiones, sino simplemente lo que hace, es decir, su comportamiento o funcionamiento.[7][1]

III-A2. Descripción Flujo de Datos o Data Flow: La descripción anterior es puramente comportamental, de manera que con una secuencia sencilla de instrucciones se podría describir el circuito. Naturalmente, a veces resulta más interesante describir el circuito de forma que esté más cercano a una posible realización física del mismo. En este sentido, VHDL posee una forma de describir circuitos que además permite la paralelización de instrucciones[8], y que se encuentra más cercana a una descripción estructural del mismo, siendo todavía una descripción funcional. En la fig. 3 [9]se muestra el formato de una descripción concurrente, también llamada de flujo de datos o de *transferencia entre registros*, en dos modos: aplicando funciones lógicas booleanas y la clave `when - else`.[1]

```
Formato_de_descripción_flujo_de_datos.vhd
File  Edit  View  Window
architecture Nombre_de_la_arquitectura of Nombre_de_la_entidad is
begin
   Nombre del puerto,_señal,_etc. <= expresión_lógica_booleana;
end Nombre_de_la_arquitectura;
--                              (a)

architecture Nombre_de_la_arquitectura of Nombre_de_la_entidad is
begin
   Función_lógica_booleana when condición
                           else secuecuencia_de_instrucciones;
end Nombre_de_la_arquitectura;
--                              (b)
```

Figura 3. Aproximación al formato de descripción flujo de datos: (a) aplicando funciones lógicas booleanas y (b) aplicando la clave `when - else`.

En la descripción (a) de la fig.3 hay varias instrucciones concurrentes, es decir, se ejecutan cad vez que cambia alguna de las señales o puertos que intervienen en la asignación. Este primer caso es casi una descripción estructural, ya que de alguna manera se están describiendo las señales y los componentes que la definen; aunque no es estructural, ya que en realidad se trata de asignaciones a señales o a puertos y no a una lista de componentes y conexiones. El caso (b) es también una descripción de flujo de datos, solo que implementa condiciones para cumplir con la función lógica booleana predeterminada.[1]

III-A3. Descripción Estructural: El VHDL también permite ser usado como *Netlist* o lenguaje de descripción de estructura. Esta descripción estructural sirve también para realización de diseños jerárquicos. La descripción estructural también se incluye dentro de un bloque de arquitectura, si bien la sintaxis interna es completamente diferente (véase la fig. 4

[6]Pág. 5 de la referencia 2.

[7]Pág. 45 de la referencia 1.

[8]Un lenguaje que describa hardware debe permitir la ejecución paralela o que hayan instrucciones concurrentes.

[9]Pág. 46 de la referencia 1.

[10])[1], [3]

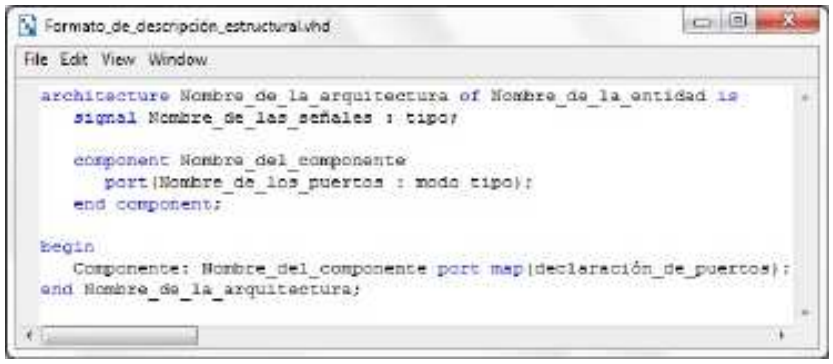

Figura 4. Aproximación al formato de descripción estructural.

Se observa fácilmente que esta descripción es un poco más larga y mucho menos clara que las anteriores. En el cuerpo de la arquitectura se hace lo que en una *Netlist* normal, es decir, se ponen los componentes y sus interconexiones. Para los componentes se utilizarán entidades que estarán definidas en alguna biblioteca, y para las conexiones se usarán señales que se declararán al principio de la arquitectura.[11][1]

Al igual que ocurre en cualquier *Netlist*, las señales o conexiones deben tener un nombre. En el esquema se le han puesto nombres a las líneas de conexión internas al circuito. Estas líneas hay que declararlas como `signal` en el cuerpo de la arquitectura y delante antes del `begin`. Una vez declaradas las señales que intervienen se procede a conectar entre sí las señales que representas componentes. Para ello la sintaxis es muy simple. Lo primero es identificar y poner cada componente, que es lo que comúnmente se conoce como *replicación*, es decir, asignarle a cada componente concreto un símbolo; en principio, el nombre puede ser cualquier identificador válido y la única condición es que no haya dos nombres iguales. Después se realizan las conexiones poniendo cada señal en su ligar correspondiente con las palabras `port map`. De esta forma se va creando el *Netlist* o la definición de la estructura.[12][1]

IV. Equipo Implementado

- Computadora.
- Herramienta de simulación Xilinx WebPack 10.1.
- Tarjeta PEGASUS de Digilent.
- Cable de conexión JTAG.
- Cable de alimentación para la PEGASUS.

V. Desarrollo de la Práctica

La práctica se desarrolla en torno a los objetivos planteados (teniendo como base el marco teórico expuesto) sobre el circuito lógico de la fig. 5.

[10]Pág. 2-6 de la referencia 3.
[11]Pág. 46 de la referencia 1.
[12]Pág. 47 de la referencia 1.

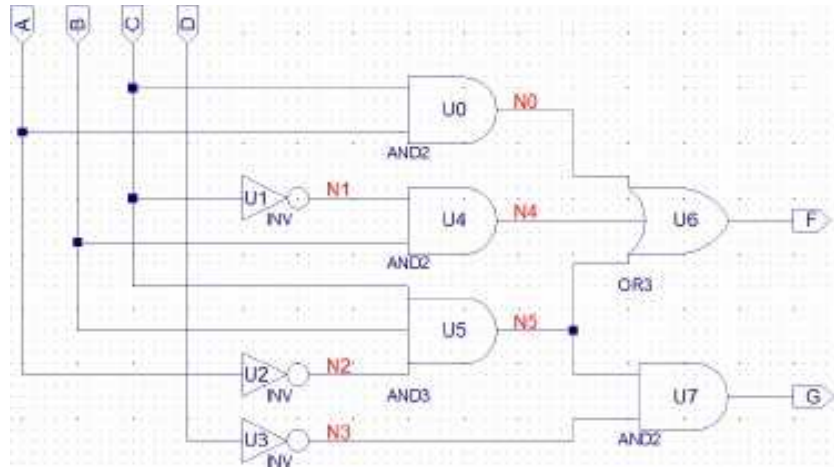

Figura 5. Circuito lógico digital a implementar.

V-A. *Expresión Booleana*

Teniendo en cuenta el número de entradas y salidas del circuito de la fig. 5, lo cual se muestra en el cuadro I, se llega a dos expresiones lógicas booleanas y son como se muestran a continuación.

$$F = A \cdot C + B \cdot \overline{C} + \overline{A} \cdot B \cdot C$$

$$G = \overline{A} \cdot B \cdot C \cdot \overline{D}$$

Como se verá más adelante, es necesario declarar las señales presentes internamente en el circuito. El cuadro I también incluye estas señales. La expresión booleana para cada una de ellas se muestra a continuación.

$$N0 = A \cdot C$$

$$N1 = \overline{C}$$

$$N2 = \overline{A}$$

$$N3 = \overline{D}$$

$$N4 = B \cdot N1$$

$$N5 = B \cdot C \cdot N2$$

Así, las salidas, F y G, se pueden expresar de la siguiente manera.

$$F = N0 + N4 + N5$$

$$G = N3 \cdot N5$$

Entradas	Señales	Salidas
	N0	
A	N1	
B	N2	E
C	N3	F
D	N4	
	N5	

Cuadro I
PUERTOS DE ENTRADA Y SALIDA Y SEÑALES INTERNAS DEL CIRCUITO DE LA FIG.5.

V-B. *Tabla de Verdad*

La tabla de verdad que modela el circuito de la fig. 5 se muestra explícitamente en el cuadro 6.

A	B	C	D	N0	N1	N2	N3	N4	N5	F	G
0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	1	0	1	1	0	0	0	0	0
0	0	1	0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	1	0	0	0	0	0
0	1	0	0	0	1	1	1	1	0	1	0
0	1	0	1	0	1	1	0	1	0	1	0
0	1	1	0	0	0	1	1	0	1	1	1
0	1	1	1	0	0	1	0	0	1	1	0
1	0	0	0	0	1	0	1	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	0	1	0
1	0	1	1	1	0	0	0	0	0	1	0
1	1	0	0	0	1	0	1	1	0	1	0
1	1	0	1	0	1	0	0	1	0	1	0
1	1	1	0	1	0	0	1	0	0	1	0
1	1	1	1	1	0	0	0	0	0	1	0

Figura 6. Tabla de verdad del circuito de la fig. 5.

V-C. *Diseño en VHDL*

A continuación se muestra los programas en VHDL creados para el circuito de la fig. 5. Se ha tenido en cuenta 4 estilos de descripción (ramificados de los expuestos en la sección 3). Estos son:

- Estilo funcional.
- Estilo flujo de datos a partir de ecuaciones booleanas.
- Estilo flujo de datos a partir de las instrucciones `when-else`.
- Estilo estructural.

V-C1. Estilo Funcional: La fig. 7 muestra el programa creado a partir de este estilo.

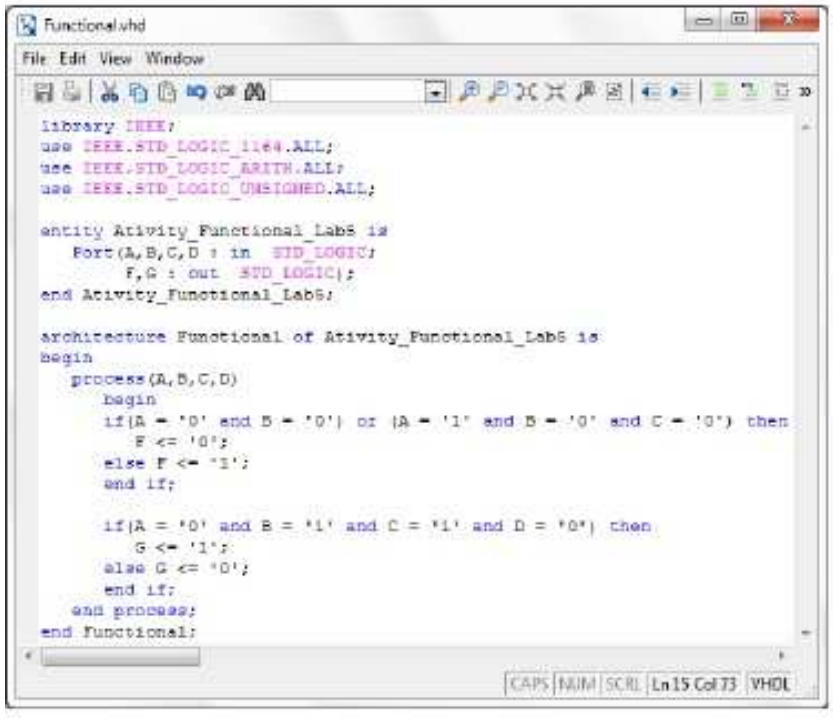

Figura 7. Estilo de descripción funcional para el circuito de la fig. 5.

V-C2. Estilo Flujo de Datos (Ecuaciones Booleanas): La fig. 8 muestra el programa creado a partir de este estilo.

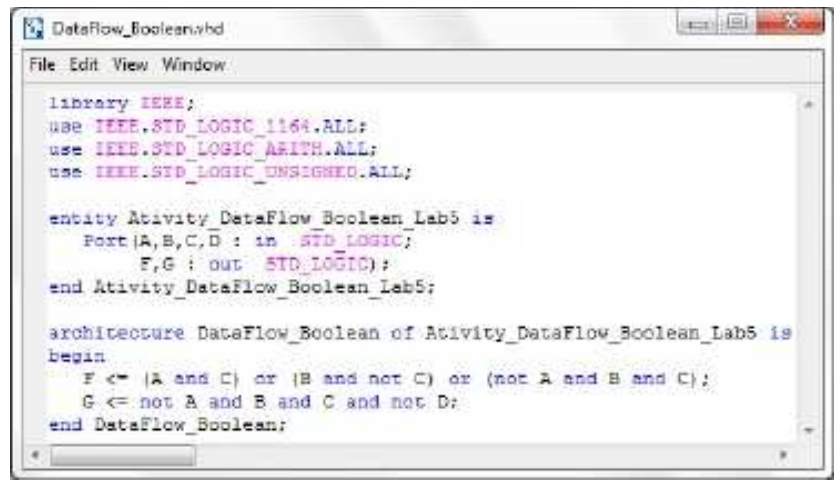

Figura 8. Estilo de descripción flujo de datos a partir de ecuaciones booleanas para el circuito de la fig. 5.

V-C3. Estilo Flujo de Datos (`when-else`): La fig. 9 muestra el programa creado a partir de este estilo.

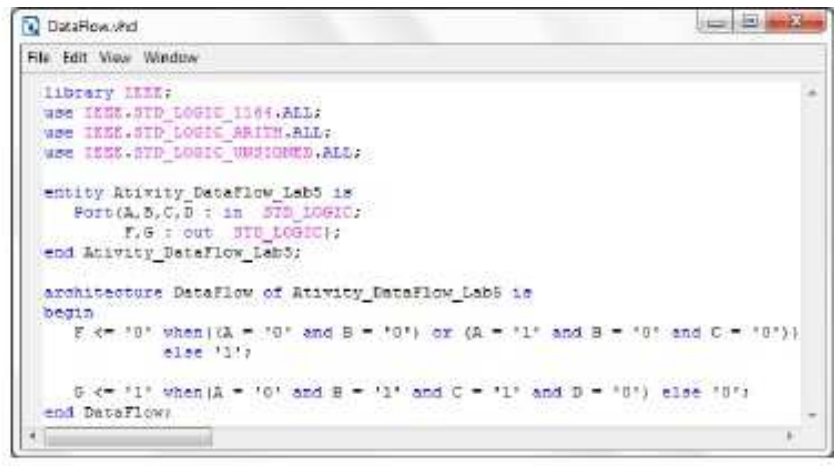

Figura 9. Estilo de descripción flujo de datos a partir de las instrucciones `when-else` para el circuito de la fig. 5.

V-C4. Estilo Estructural: La fig. 10 muestra el programa creado a partir de este estilo.

La implementación de este estilo de descripción requiere de la declaración o creación de subrutinas. Precisamente, estas subrutinas son:

- Compuerta not de una entrada (no) (véase la fig. 11).
- Compuerta and de dos entradas (and_2) (véase la fig. 12).
- Compuerta and de tres entradas (and_3) (véase la fig. 13).
- Compuerta or de tres entradas (or_3) (véase la fig. 14).

Structural.vhd

File Edit View Window

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Activity_Structural_Lab6 is
    Port(A,B,C,D : in  STD_LOGIC;
         F,G : out  STD_LOGIC);
end Activity_Structural_Lab6;

architecture Structural of Activity_Structural_Lab6 is
    signal N0,N1,N2,N3,N4,N5 : STD_LOGIC;

    component no
       port(X1 : in STD_LOGIC;
            Y1 : out STD_LOGIC);
    end component;

    component and_2
       port(X1,X2 : in STD_LOGIC;
            Y1 : out STD_LOGIC);
    end component;

    component and_3
       port(X1,X2,X3 : in STD_LOGIC;
            Y1 : out STD_LOGIC);
    end component;

    component or_3
       port(X1,X2,X3 : in STD_LOGIC;
            Y1 : out STD_LOGIC);
    end component;

begin
    U0: and_2 port map(A,C,N0);
    U1: no port map(C,N1);
    U2: no port map(A,N2);
    U3: no port map(D,N3);
    U4: and_2 port map(B,N1,N4);
    U5: and_3 port map(B,C,N2,N5);
    U6: or_3 port map(N0,N4,N5,F);
    U7: and_2 port map(N3,N5,G);
end Structural;
```

Figura 10. Estilo de descripción estructural para el circuito de la fig. 5.

no.vhd

File Edit View Window

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity no is
    Port(X1 : in  STD_LOGIC;
         Y1 : out  STD_LOGIC);
end no;

architecture arq_no of no is
begin
    Y1 <= not X1;
end arq_no;
```

Figura 11. Programa que declara la subrutina "no" para complementar el programa estructural de la fig. 10.

and_2.vhd

File Edit View Window

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity and_2 is
    Port(X1,X2 : in  STD_LOGIC;
         Y1 : out  STD_LOGIC);
end and_2;

architecture arq_and_2 of and_2 is
begin
    Y1 <= X1 and X2;
end arq_and_2;
```

Figura 12. Programa que declara la subrutina "and_2" para complementar el programa estructural de la fig. 10.

and_3.vhd

File Edit View Window

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity and_3 is
    Port(X1,X2,X3 : in  STD_LOGIC;
         Y1 : out  STD_LOGIC);
end and_3;

architecture arq_and_3 of and_3 is
begin
    Y1 <= X1 and X2 and X3;
end arq_and_3;
```

Figura 13. Programa que declara la subrutina "and_3" para complementar el programa estructural de la fig. 10.

or_3.vhd

File Edit View Window

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity or_3 is
   Port(X1,X2,X3 : in  STD_LOGIC;
        Y1 : out  STD_LOGIC);
end or_3;

architecture arq_or_3 of or_3 is
begin
   Y1 <= X1 or X2 or X3;
end arq_or_3;
```

Figura 14. Programa que declara la subrutina "or_3" para complementar el programa estructural de la fig. 10.

V-D. Símbolo

Para crear el símbolo del sistema se sigue que:

- Seleccionar el archivo del programa al cual se le va a crear el símbolo. Esto es, activar el programa .vhd a implementar.
- En la ventana "Process" abrir el menú "Desisng Utilities". Luego ejecutar con doble click la opción "Create Schematic Symbol".

Ello se sigue para todos los archivos creados (en los diferentes modos) de la misma manera. Se anota que estos símbolos quedan guardados por defecto en el directorio donde se encuentra el proyecto. Además, se recuerda que el símbolo es esquemático, entonces para implementarlo se requiere crear una fuente de tipos "Schematic".

El símbolo del estilo funcional, del estilo data flow (booleano), del estilo data flow (`when-else`) y del estilo estructural se muestran en la fig. 15, fig. 16, fig. 17 y fig. 18 respectivamente.

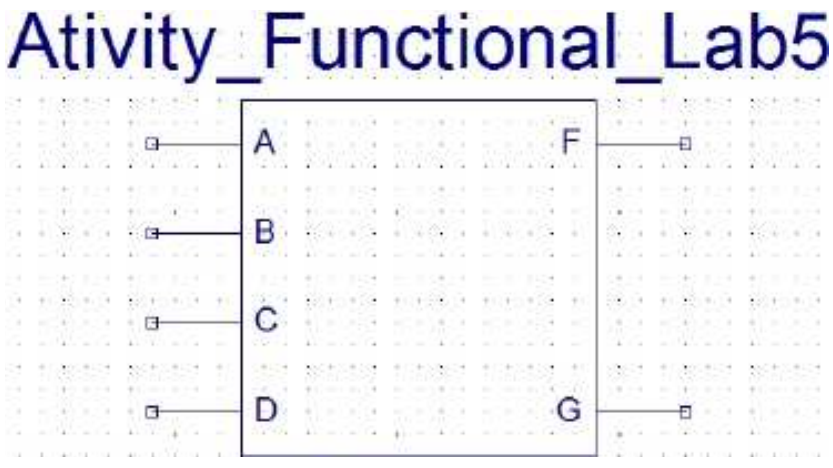

Figura 15. Símbolo del programa de estilo funcional de la fig. 7.

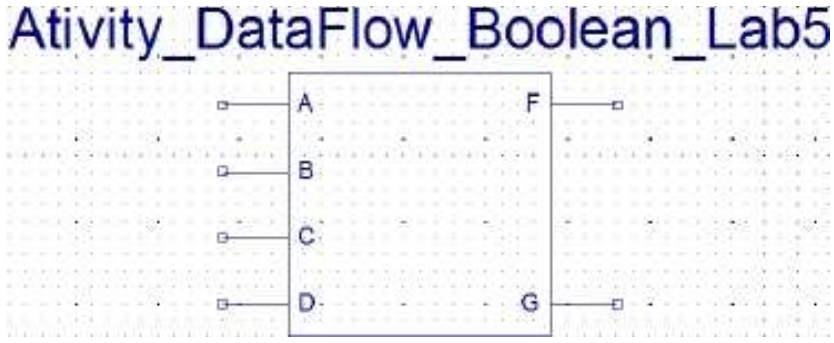

Figura 16. Símbolo del programa de estilo flujo de datos (booleano) de la fig. 8.

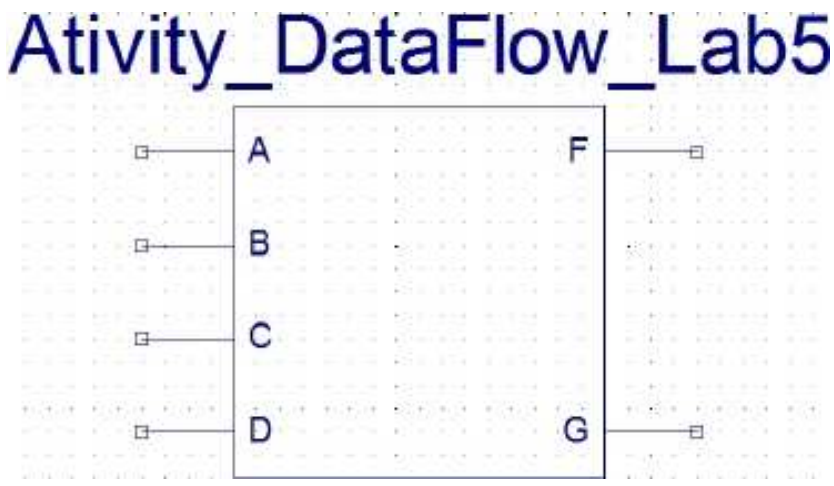

Figura 17. Símbolo del programa de estilo flujo de datos (`when-else`) de la fig. 9

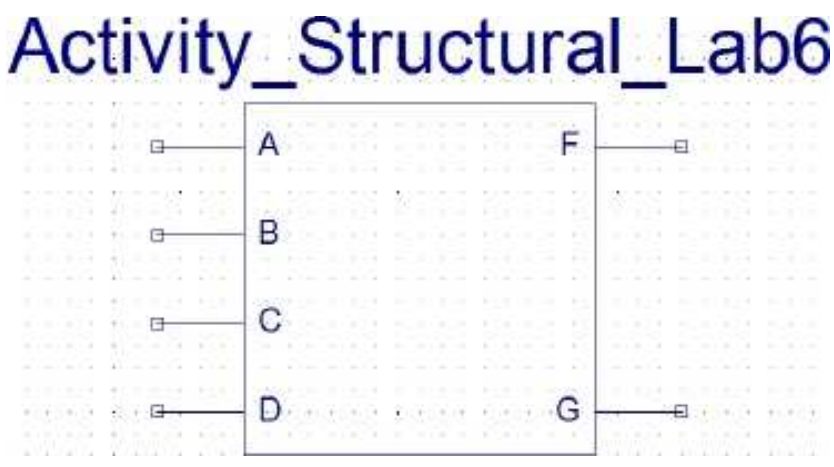

Figura 18. Símbolo del programa de estilo estructural de la fig. 10

V-E. Diagrama de Tiempos

Aplicando la opción de simulación "Behavioral Simulation" de Xilinx, se procede a declarar las señales de entrada (A, B, C y D) como se muestra en la fig. 19. Obsérvese que se hace coincidir con la asignación presentada en el cuadro 6 (tabla de verdad). Al correr la simulación, se obtienen las señales presentadas en la fig. 20. De nueva cuenta, obsérvese que las señales de salida (E y F) corresponden a las mostradas en el cuadro 6.

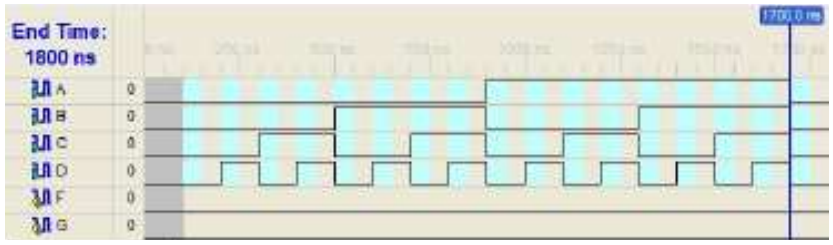

Figura 19. Asignación de las señales de entrada.

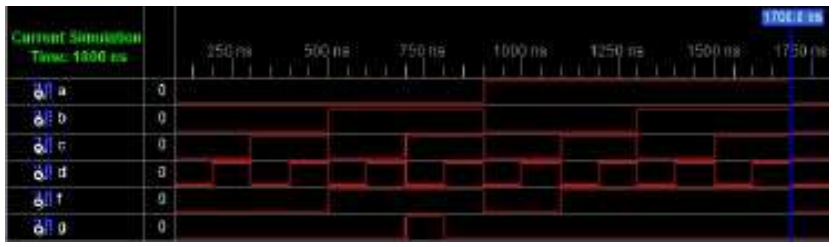

Figura 20. Diagrama de tiempos presentado por el archivo de simulación .tbw creado para cada estilo de descripción.

VI. Conclusiones

Cumpliendo con el principal objetivo de esta práctica, determinar cuáles son las ventajas y desventajas que poseen los estilos de descripción VHDL, se ha llegado a las siguientes conclusiones.

- El estilo de descripción menos complejo y que requiere de menos instrucciones, es el flujo de datos a partir de expresiones booleanas. No obstante, este estilo se hace obsoleto al momento de no tener a disposición la ecuación lógica booleana que describa el circuito. Por otro lado, este estilo no permite tener noción alguna acerca de cómo funciona el circuito. Ello hace destacar al estilo flujo de datos por `when-else`, ya que este permite dar una primera aproximación al comportamiento del circuito, estableciendo para ello condiciones que introduzcan a lo dicho.
- El estilo funcional permite en gran medida la noción de qué y cómo se comporta el circuito a modelar. Si bien, requiere de más líneas de código en comparación con el Data Flow, ello le permite intrínsecamente adoptar la esencia de las condiciones que tiene que cumplir el circuito para poder funcionar correctamente. Algo que se debe destacar, es que éste estilo no requiere de la expresión lógica booleana para poder describir el circuito a modelar.
- El estilo estructural es el más engorroso y el menos atractivo para un diseñador al momento de describir un circuito. Sin embargo, ello no siempre es así. La gran ventaja de poder crear varias subrutinas en un mismo programa le da a este estilo un valor agregado. Esto se refiere precisamente a aquellas ocasiones en las que el diseñador quiera modelar un sistema que esté compuesto por una serie de subsistemas, entonces, es allí donde el estilo estructural entra en acción produciendo las subrutinas que se necesite y, lo que es aún mejor, en cualquier otro estilo de descripción VHDL.
- Si bien, algunos de los estilos de descripción son más complejos que otros, para cada aplicación existe al menos un estilo que se acomode a su funcionalidad y finalidad. De ello se deduce que así como algunos de los estilos pueden funcionar para ciertas aplicaciones, hay aplicaciones que son exclusivamente aplicables con cierto estilo; No siempre se puede contar con todos los estilos para solucionar un problema. Además, se anota que puede haber casos en el que sea necesario involucrar varios estilos de descripción en una misma secuencia de asignaciones.

Referencias

[1] Fernando Pardo Carpio and José A. Boluda Grau, *"VHDL Lenguaje para síntesis y modelado de circuitos"*, Tercera ed. México D.F., México: Alfaomega Grupo Editor, S.A. de C.V., 2011.

[2] John Jairo Ramirez Mateus, *"Introducción al Lenguaje VHDL"*, Universidad Francisco de Paula Santander, San José de Cúcuta, Guía de laboratoio.

[3] John Jairo Ramírez Mateus, *"Estilo de Programación Estructural con VHDL"*, Universidad Francisco de Paula Santander, San José de Cúcuta, Guía de laboratorio.