Hw #09 { Descripción estructural y IPCore}
Lenguajes de descripción de hardware

Hernández Castellanos César Uriel

1 Sección uno

La descripción estructural permite en los lenguajes de descripción de hardware crear módulos (verilog) o componentes (VHDL) cuyo código puede reutilizarse tantas veces como se desee y que además permite dar estructura a los diseños y repartir un proyecto grande entre varios diseñadores. Cuando se tiene un módulo o componente, éste puede instanciarse una, dos o más veces en un diseño, tantas veces como se necesite. Suponer que se quiere crear un módulo o un componente de un multiplexor de dos entradas y una salida como el de la siguiente figura:

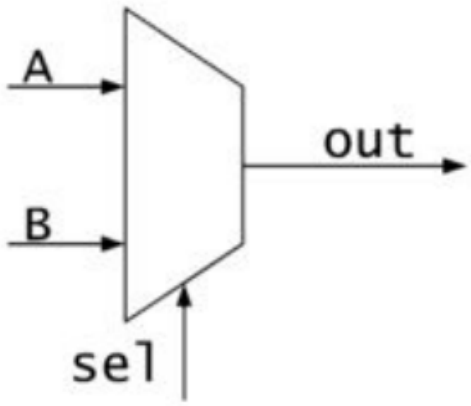


Figure 1: *Multiplexor*

Y que se quiere utilizar en una descripción estructural. Consideremos primero como hacerlo en VHDL y posteriormente se realizará en verilog.

1.1 Descripción estructural en VHDL.

Considerar el siguiente código escrito en VHDL:

1.2 Entidad principal

```
library ieee;
use ieee.std_logic_1164.all;
entity prueba2 is
 port(a,b,sel: in std_logic;
 s: out std_logic);
end prueba2;
architecture estructural of prueba2 is
 component mimux
 port(in1,in2,control: in std_logic;
 sal: out std_logic);
 end component;
begin
 mux0: mimux port map(a,b,sel,s);
end estructural;
```

1.3 Componente mux

```
library ieee;
use ieee.std_logic_1164.all;
entity mimux is
 port(in1,in2,control: in std_logic;
 sal: out std_logic);
end mimux;
architecture rtl of mimux is
begin
 sal <= in1 when control='0' else in2;
end rtl;
```

Este código muestra como describir en forma estructural el uso de un componente de nombre mimux que describe un multiplexor de dos entradas y una salida de un bit. Notar que en VHDL cuando se va a instanciar un componente, es necesario declarar el prototipo de éste en la zona declarativa de la arquitectura. El prototipo indica los puertos y el orden de éstos en el componente

(primero las tres entradas: in1, in2 y control; luego la salida: sal para este ejemplo).

```
component mimux
 port(in1,in2,control: in std_logic;
 sal: out std_logic);
 end component;
```

Para instanciarlo en la arquitectura, se usa la siguiente sentencia:

```
 mux0: mimux port map(a,b,c,s);
```

En donde utilizamos un nombre o etiqueta de la instancia que es útil además para diferenciar entre múltiples instancias del mismo componente, el nombre del componente y las señales que vamos a conectar al componente en el orden en que son definidas en el prototipo. Se pueden hacer tantas instancias de un componente como se requiera en el diseño. Cabe hacer notar que tanto la entidad principal como el componente pueden ir en el mismo archivo o en archivos distintos que pertenezcan al mismo proyecto. Notar que aunque la entidad principal y el componente estén en el mismo archivo, es necesario hacer dos veces la declaración de bibliotecas, ya que su alcance en la compilación llega hasta el final de su correspondiente arquitectura.

1.4 Descripción estructural en verilog

Considerar el siguiente código escrito en verilog:

1.5 Módulo principal

```
module prueba(a,b,c,f);
input a,b,c;
output f;
 mimux u0(a,b,c,f);
endmodule
```

1.6 Módulo multiplexor

```
module mimux(in1,in2,control,out);
input in1,in2,control;
output out;
 assign out=(control==0)?in1:in2;
endmodule
```

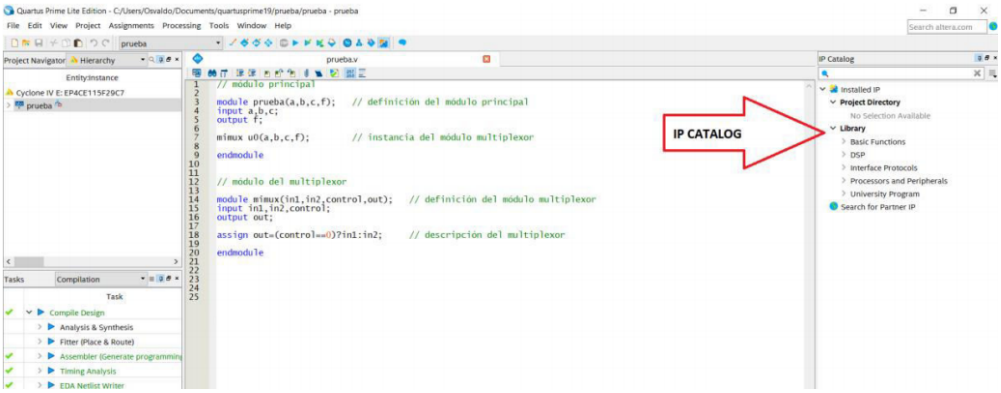
Al igual que en VHDL, tanto el módulo principal como el del multiplexor pueden estar en el mismo archivo o en archivos distintos pertenecientes al mismo proyecto. En verilog es más sencilla la descripción estructural ya que no se declara prototipo del módulo a instanciar. Sólo es necesario usar una sentencia como la siguiente para instanciar al módulo mimux:

```
mimux u0(a,b,c,f);
```

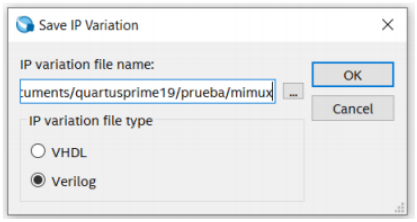
Aquí se usa primero el nombre del módulo a instanciar, seguido del nombre o etiqueta de la instancia para posteriormente poner las señales que se conectarán al módulo multiplexor en el orden en que aparecen definidas en el módulo: primero las tres entradas y después la salida.

1.7 Uso de IPCore en Quartus

Esta herramienta es parte de Quartus y permite generar algunas funciones digitales incluidas en su catálogo, entre las que se encuentran funciones aritméticas de enteros, de punto flotante, memorias, de entrada/salida etc. Esto agiliza el proceso de diseño al evitar que el diseñador codifique dichas funciones. IPCore genera archivos que se instancian en descripción estructural. Para realizar el proyecto del multiplexor usando IPCore, debemos ubicar el IPCatalog que se encuentra en el extremo derecho de la pantalla de Quartus como se muestra en la siguiente figura:



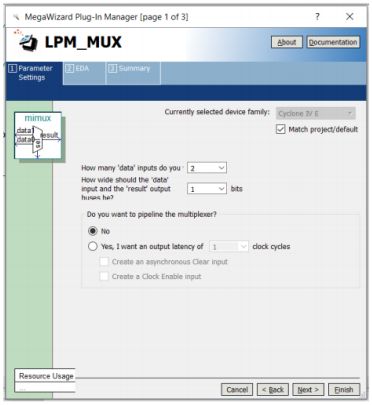
Cuando el IP Catalog no aparece, podemos mostrarlo entrando al menú TOOLS y seleccionando la opción IP Catalog. Se pueden expandir las categorías para ver el tipo de funciones que proporcionan. En este caso se expande la categoría "Basic Functions" y posteriormente se expande la categoría "Miscelaneous", ahí aparece la opción LPMMUX. Seleccionarla dando doble click para mostrar la ventana inicial siguiente:



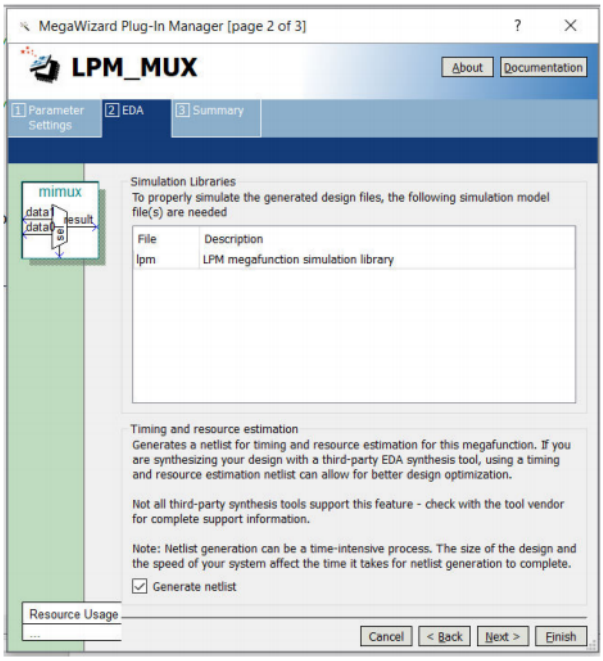
En esta ventana debemos dar nombre al archivo que contendrá el código del multiplexor, en este caso mimux (por default ya aparece la ruta en que se encuentra el proyecto y sólo hay que agregar el nombre al final de la ruta). También debe seleccionarse el lenguaje en que se está trabajando el proyecto. Dar click en OK. Aparecerá la primera ventana de un asistente que pedirá la configuración del multiplexor que se desea generar. En ella debe indicarse:

1. El número de entradas del multiplexor (en este caso 2).
2. En ancho de bits de las entradas (en este caso 1 bit).
3. Si se desea segmentar (pipeline) el multiplexor (No en este caso).

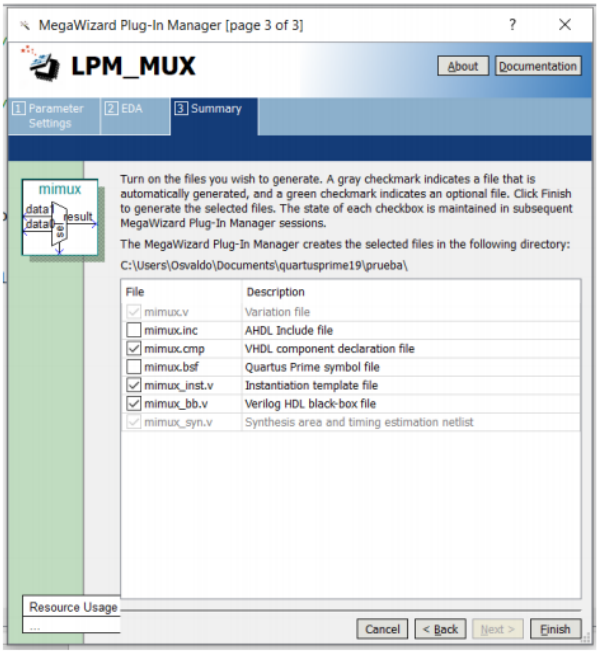
La ventana tiene la siguiente apariencia:



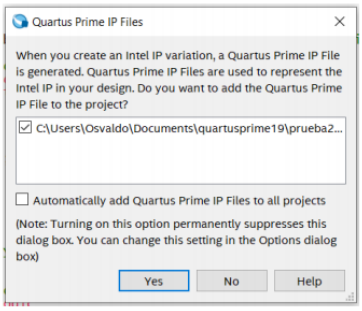
Al terminar de configurar los parámetros debe darse click en NEXT. A continuación aparece la siguiente ventana:



Seleccionar la opción “Generate netlist” la cual permite generar archivos necesarios para la correcta simulación del multiplexor. Debe darse click en NEXT para que aparezca la siguiente ventana:



Esta ventana muestra un resumen de los archivos que se van a generar. Es importante seleccionar el archivo mimux-inst, en este archivo se muestra la forma de instanciar el multiplexor generado en IP Core. Cuando el proyecto se ha escrito en VHDL, se puede seleccionar el archivo mimuxcmp que muestra la forma de declarar el prototipo del componente. Debe darse a continuación click en Finish. Finalmente, aparecerá una ventana que nos preguntará si se adiciona al proyecto un archivo relacionado con la función creada por IPCore. Este archivo es importante si posteriormente se desean hacer modificaciones a la función creada. Dar click en “yes” en la siguiente ventana:



Finalmente, instanciar la función creada en el código del proyecto que lo requiere.

2 Sección dos

2.1 Inciso uno

Hacer el ejercicio de describir en forma estructural el multiplexor en el lenguaje de su elección. Instanciar el módulo o componente que se encuentre en el mismo archivo. Simular el multiplexor.

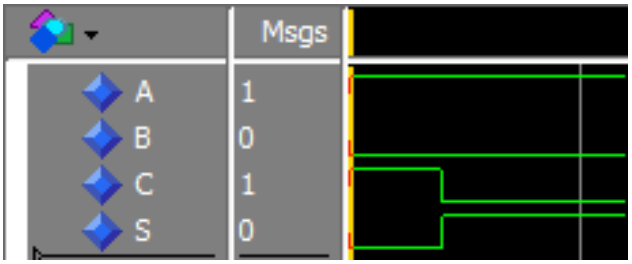
2.2 Componente multiplexor

```
ARCHITECTURE MuxA OF Multiplexer_2_To_1T IS
BEGIN
        C <= B when S else A;
END MuxA;
```

2.3 Componente principal

```
ARCHITECTURE Beh OF prueba IS
COMPONENT Multiplexer_2_To_1T IS
        PORT (
                A : IN  STD_LOGIC;
                B : IN  STD_LOGIC;
                S : IN  STD_LOGIC;
                C : OUT STD_LOGIC
        );
END COMPONENT;
BEGIN
mux : Multiplexer_2_To_1T PORT MAP (
                A=>A,
                B=>B,
                S=>S,
                C=>C
);
END Beh;
```

2.4 Simulación



2.5 Inciso dos

Describir el multiplexor en forma estructural usando una función creada por el IP Core siguiendo los pasos descritos en este documento. Simular el multiplexor.

2.6 Componente principal

```
ARCHITECTURE Beh OF prueba IS
COMPONENT mimux IS
        PORT (
                data0 : IN  STD_LOGIC;
                data1 : IN  STD_LOGIC;
                sel   : IN  STD_LOGIC;
                result: OUT STD_LOGIC
        );
END COMPONENT;
BEGIN
mimux_inst : mimux PORT MAP (
                data0=> A,
                dataa1=> B,
                sel=> S,
                result => C
);
END Beh;
```

2.7 Simulación