Práctica 4. Modelado de comportamiento Y Multiplexor.

4.1. Introducción.

El modelado de comportamiento representa los circuitos digitales en un nivel funcional y algorítmico. Se le utiliza primordialmente para describir circuitos secuenciales, pero sirve también para describir circuitos combinacionales.

Las descripciones de comportamiento emplean la palabra clave **always** seguida de una lista de enunciados de asignación procedimentales. La salida deseada de los enunciados de asignación procedimentales debe ser del tipo de datos **reg.** A diferencia del tipo de datos **wire,** en el que la salida deseada de una asignación se puede actualizar continuamente, el tipo de datos **reg** conserva su valor hasta que se le asigna uno nuevo.

```
//Descripción de comportamiento del multiplexor 2 a 1
module mux2x1_bh(A,B,select,OUT);
   input A,B,select;
   output OUT;
   reg OUT;
   always @ (select or A or B)
         if (select == 1) OUT = A;
         else OUT = B;
endmodule
```

En el ejemplo anterior muestra la descripción de comportamiento de un multiplexor de 2 líneas a 1 (compare con el ejemplo HDL 4-6). Dado que la variable OUT es una salida deseada, debemos declararla como dato **reg** (además de la declaración **output**). Los enunciados de asignación procedimentales dentro del bloque **always** se ejecutan cada vez que hay un cambio en cualquiera de las variables indicadas después del símbolo @. (Observe que no se escribe un (;) al final del enunciado **always.**) En este caso, la lista incluye las variables de entrada *A* y *B*, y *select*. Advierta que se usa la palabra clave **or** entre las variables en lugar del operador de OR lógico "|". El enunciado condicional **if-else** permite tomar una decisión con base en el valor de la entrada *select*. El enunciado **if** se puede escribir sin el símbolo de igualdad:

```
if (select) OUT = A ;
```

Este enunciado implica que se examina *select* para ver si es 1 lógico.

```
//Descripción de comportamiento del multiplexor 4 a 1

module mux4x1_bh (i0,i1,i2,i3,select,y);
   input i0,i1,i2,i3;
   input [1:0] select;
   output y;
   reg y;
   always @ (i0 or i1 or i2 or i3 or select)
            case (select)
               2'b00: y = i0;
               2'b01: y = i1;
               2'b10: y = i2;
               2'b11: y = i3;
            endcase
endmodule
```

Otro ejemplo, se describe la función de un multiplexor de 4 líneas a 1. La entrada *select* se define como un vector de dos bits, y la salida "*y"* se declara como dato **reg.** El enunciado **always** tiene un bloque secuencial delimitado por las palabras clave **case** y **endcase.** El bloque se ejecuta cada vez que cambia el valor de cualquiera de las entradas indicadas después del símbolo @. El enunciado **case** es una condición de ramificación condicional multivías. La expresión **case** (select) se evalúa y se compara con los valores de la lista de enunciados que siguen. Se ejecuta el primer valor que coincide con la condición verdadera. Puesto que *select* es un número de dos bits, puede ser igual a 00, 01, 10 o 11. Los números binarios se especifican con una **b** precedida por un apóstrofo. Primero se escribe el tamaño del número y luego su valor. Así, 2_b01 especifica un número binario de dos dígitos cuyo valor es 01. También pueden especificarse números en decimal, octal o hexadecimal, con las letras ´**d,** ´**o** y ´**h,** respectivamente. Si no se especifica la base del número, se toma como decimal por omisión. Si no se especifica el tamaño del número, el sistema supondrá que es de 32 bits.

4.2. Previo Práctica 4.

1) Defina y escriba la sintaxis para hacer uso de la instrucción **case.**

2) Escriba la sintaxis para el uso de los condicionales **if-else**.

3)Investigue en el manual los pines asignados a los 4 display de 7 segmentos de la placa de desarrollo.

4.3. Trabajo de laboratorio

1) Diseñe un contador 0-9 con salida a un display de 7 segmentos.

2)Diseñe un contador de 0-99 con dos display y control de Start, Stop y Reset.

4.4. Conclusiones