Introducción General Lenguaje Descriptivo de Hardware Verilog

Vargas Meza Aldo Alexandro, 213495653

Con la resiente utilización del software de ModelSim y QuestaSim, llega como nuevo reto la implementación de instancias dentro de un módulo de programación, esto con objetivo de comunicar y "conectar" módulos separados para que puedan ser usado por un tercer módulo, o entre ellos.

I. Introducción

El programa a llevar a cabo, es una calculadora lógica, que nos pueda dar 4 operaciones lógicas básicas (AND, OR, NAND y XOR), controlada por un selector y con una única salida de datos. A grandes rasgos, el programa se compone de dos entradas de datos de 4 bits que se llamarán A y B, un selector de la función lógica de 2 bits, y tendrá una salida en 4 bits. Para programar el correcto funcionamiento del programa, se utilizarán módulos separados para llevar a cabo el manejo por etapas de los datos.

II. Composición del código

El primer módulo del código, es el decodificador, su trabajo será obtener una entrada de 2 bits, que se convertirá en una salida de 4. Pero para esto, una segunda entrada de 1 bit, debe de tener un estado lógico HIGH para poder cumplir con la condición para llevar a cabo la conversión de la entrada.

```
1   //Decodificador
2   module decodificador (
3     input wire dec_enable,
4     input wire[1:0]dec_selector,
5     output reg[3:0]dec_out
6     );
7   //Proceso
8   always @(*)
9     begin
10      //Condicional enable
11      if (dec_enable==1)
12        //Condicional selector
13        case (dec_selector)
14        2'b00:
15          dec_out= 4'b0001;   //AND
16        2'b01:
17          dec_out=4'b0010; //OR
18        2'b10:
19          dec_out=4'b0100; //NAND
20        2'b11:
21          dec_out=4'b1000;   //XOR
22        endcase
23      else
24       dec_out=0;
25    end
26  endmodule
```

Imagen 01"Modulo decodificador"

El segundo módulo, contenido en un archivo diferente, es en realidad un conjunto de 4 módulos, que desempeñarán las funciones lógicas necesarias. Los módulos son simples; dos entradas de 4 bits, y una salida de 4 bits.

```
12  //Compuerta logica or
13  module or_gate(
14    input wire [3:0]or_a,
15    input wire [3:0]or_b,
16    output reg [3:0]or_out
17  );
18
19  always @ (*)
20    or_out= or_a|or_b; //SALIDA OR
21  endmodule
```

Imagen 02"Modulo compuerta OR"

El siguiente modulo, desarrolla todo el manejo de datos, desde las 4 salidas proporcionadas por los módulos de las compuertas, hasta la salida del decodificador, que utiliza para seleccionar que entrada de las 4 es la que será digitada en la salida del mismo. El modulo Multiplexor, además contiene las instancias de los demás módulos para llevar a cabo las conexiones virtuales.

```
1   //Multiplexor
2   module multiplexor(
3   output reg[3:0]mux_out,              //SALIDA
4   input wire [3:0]mux_selector,        //Selector de datos
5   //Lectura valores logicos
6   input wire[3:0]mux_and,  //Entrada AND
7   input [3:0]mux_or,   //Entrada OR
8   input [3:0]mux_nand, //Entrada NAND
9   input [3:0]mux_xor   //Entrada XOR
10  );
11
12  and_gate and_data(
13    .and_out(mux_and)
14  );
15  or_gate or_data(
16    .or_out(mux_or)
17  );
18  nand_gate nand_data(
19    .nand_out(mux_nand)
20  );
21  xor_gate xor_data(
22    .xor_out(mux_xor)
23  );
24  decodificador dec_data(
25    .dec_out(mux_selector)
26  );
```

Imagen 03"Multiplexor"

Una vez “conectados” los módulos, es momento de hacer una simulación de los comportamientos esperados que podría tener el circuito. Para ello es necesario hacer un módulo extra conocido como “test_bench” que se encarga de estimular las entradas de los módulos, y conectarse con una entrada propia a la salida de los mismos, esto con la finalidad de monitorear los valores que el estímulo mandado debería de generar.

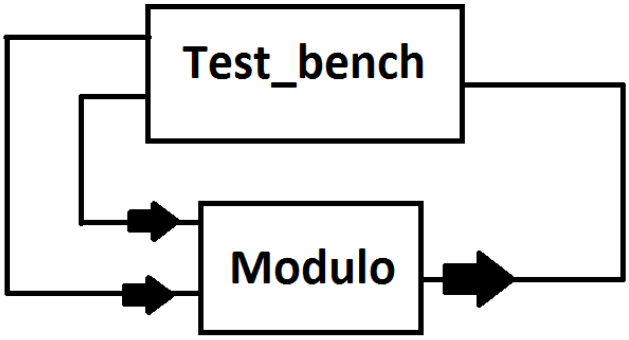


Imagen 05”Funcionamiento Test Bench”

El funcionamiento del test bench se caracteriza porque las salidas del mismo, van a ir a las entradas del programa, esto para generar las salidas que se espera operen en el circuito, las cuales van a ir conectadas a la entrada del test bench, que las visualizará en la simulación en el mismo software.

```
1   module test_bench(
2     output reg[3:0]a,
3     output reg[3:0]b,
4     output reg dec_enable,
5     output reg[1:0]dec_selector,
6     input wire[3:0]or_out,
7     input wire[3:0]nand_out,
8     input wire[3:0]xor_out,
9     input wire[3:0]and_out,
10    input [3:0]mux_out
11    );
12
13    decodificador tb1(
14      .dec_enable(dec_enable),
15      .dec_selector(dec_selector)
16    );
17    and_gate tb2(
18      .and_a(a),
19      .and_b(b),
20      .and_out(and_out)
21    );
22     multiplexor tb3(
23      .mux_out(mux_out)
24    );
25    or_gate tb4(
26      .or_a(a),
27      .or_b(b),
```

Imagen 06”Modulo test_bench”

III. Simulación

Para simular, primero es necesario llevar a cabo la compilación del código y de cada uno de los módulos, así como una compilación completa de cada uno.
El software utiliza un compilador propio, que hace un chequeo de sintaxis y una vez compilados todos los módulos involucrados es posible llevar la simulación.

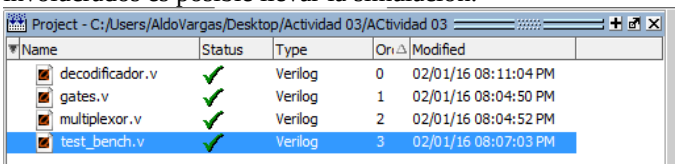


Imagen 06 “Archivos compilados”

La simulación, se lleva a cabo en la misma ventana con un menú contextual, o se puede también configurar un archivo para iniciar la simulación con configuración predeterminada.

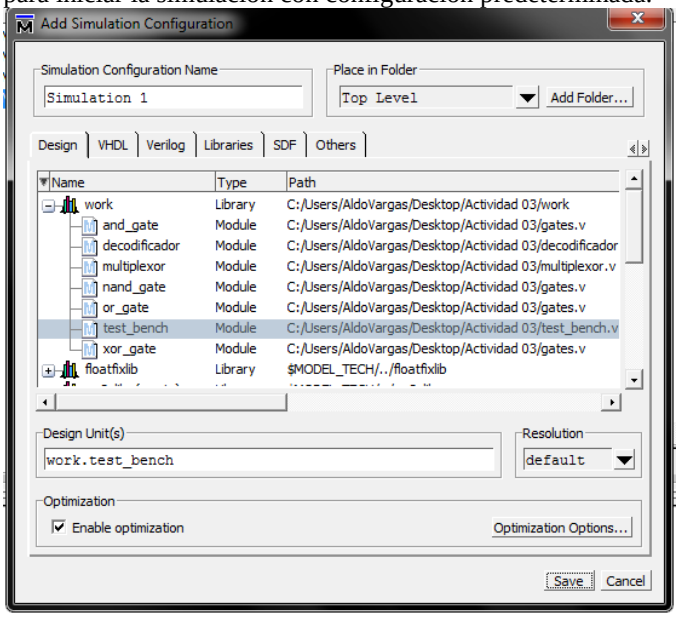
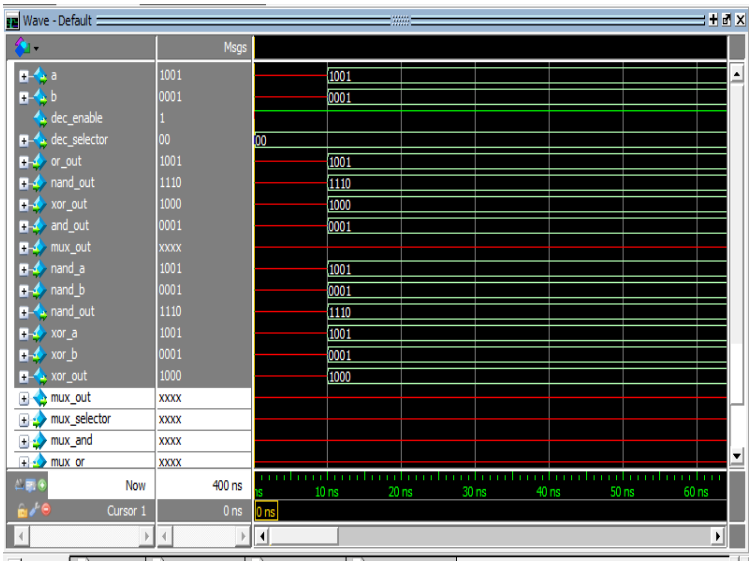


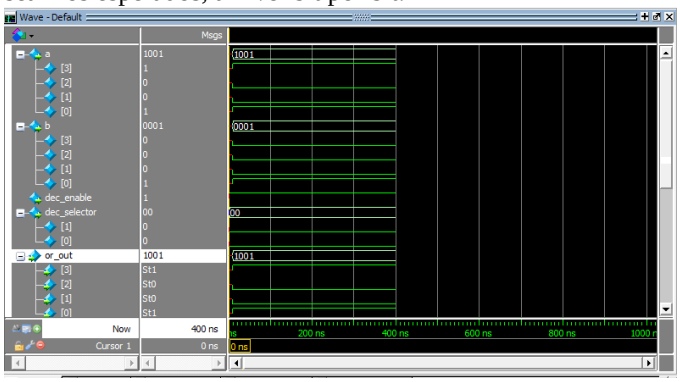
Imagen 07”Configuracion de simulación”

La salida es monitoreada por las entradas del test bench y se hace un barrido de datos de los diferentes resultados.



IV. Comprobación de datos

Una vez simulado, es posible checar que los datos que salen sean los esperados, a nivel bit por bit.



V. Conclusión

El circuito involucró dentro de sí, comportamientos con diferentes variables, dependientes los unos de los otros. Algo que me sirvió mucho en lo personal, fue la organización de todas las señales de entrada y salida, con nombres de etiqueta y con separadores únicos.

Una de las principales ventajas de este modo de trabajo es la utilización de las instancias, que fue el tema principal de esta práctica, que ordena todo por módulos, y pueden ser utilizados por los demás elementos del proyecto.

VI. Bibliografía

Asic World. (s.f.). Obtenido de http://www.asic-world.com/verilog/operators1.html

ECCE. COLORADO. (s.f.). Obtenido de Colorado Education: http://ecee.colorado.edu/~ecen5837/cadence/problem_images/synth_code2.JPG

IUMA oficial. (s.f.). Obtenido de http://www.iuma.ulpgc.es/~nunez/clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf

Tamu.EDu. (s.f.). Obtenido de http://students.cs.tamu.edu/tanzir/csce350/reference/verilog.html

Verilog Oficial Site. (s.f.). Obtenido de http://www.verilog.com/

Wikibooks. (s.f.). Obtenido de https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Verilog/