Pre-Informe 3

Diego Norambuena V.
201704002-5

Pregunta 1

a) Un HDL (Hardware Description Language) es un lenguaje especializado utilizado para para definir la estructura, diseño y operación de circuitos electrónicos,comunmente digitales. Su objetivo es describir un circuito mediante un conjunto de instrucciones de alto nivel de abstracción para que el programa de síntesis genere (ensamble) un circuito que pueda ser implementado físicamente, posibilitando su análisis automático y su simulación.
Las principales diferencias con los lenguajes de programación convencionales son: que el HDL incluye explícitamente la noción de tiempo (podemos definir la unidad de tiempo a utilizar); la mayoría de las expresiones se "ejecutan" concurrentemente, es decir, simultáneamente; cada expresión o "instrucción" corresponde a la operación de un bloque de circuito.

b) **Wire:** usado para conectar entidades del hardware, como puertas lógicas, o como nodos intermedios. Es usado en asignaciones continuas, por lo que no puede almacenar un valor ya que este viene derivado desde lo que esté manejando su controlador (driver). Normalmente se asocia a la función de un alambre convencional.
Reg: Usado generalmente para modelar registros de hardware pues es capaz de almacenar datos, aunque también puede representar lógica combinacional, por ejemplo dentro de un bloque **always_comb**
Logic: es un tipo de dato de 4 estados introducido en SystemVerilog, y puede ser manejado tanto en bloques de procedimiento (*always o initial*) como en sentencias de asignación continua(*assign*), por lo que su uso es más general que **reg** o **wire**.

c) La asignación continua (*assign*)se utiliza exclusivamente para modelar lógica combinacional, donde no se necesita de una lista de sensibilidad para realizar la asignación. La variable a la que se realiza la asignación continua sólo puede ser declarada de tipo net, es decir wire. Y la asignación sólo puede ser declarada fuera de cualquier proceso y nunca dentro de bloques always o initial.
La asignación procedural (= y <=) es donde a las variables se le asigna un valor dentro de un proceso always o initial, el tipo de variable a la que se le asigna el valor puede ser de cualquier tipo. La diferencia entre estos 2 últimos operadores, es que el primero (=) es bloqueante, lo que implica que debe de calcularse la expresión dada antes de continuar con las instrucciones siguientes, mientras que el segundo (<=) es no bloqueante, lo que significa que la expresión se calculará en el instante actual pero no se *asignará* hasta que termine dicho instante.

d) Expresión corregida: reg [15:0]g = 16'hA6B2;
Esta expresión corresponde a la creación de una variable de tipo **reg**, en este caso, es un vector cuyo MSB es g[15] y su LSB es g[0] (de acuerdo a los corchetes), de nombre "g". 16 es el número de bits expresado en decimal de la cantidad que viene a continuación en base hexadecimal (h) cuya magnitud es A6B2. La comilla separa la base de la magnitud por sintaxis del lenguaje. El símbolo de punto y coma termina la expresión.
El equivalente binario es 1010 0110 1011 0010

e) Primero tenemos una asignación con bloqueo, donde first toma el valor de 1. Luego, en la asignación no bloqueante, tomamos el bit arr[first], es decir, arr[1] y le asignamos ¬(arr[second]), que corresponde al mismo bit pero negado, puesto que *second = 1*. El delay de esta asignación es #0, por lo que el cálculo se hace de inmediato y no espera a que transcurra una unidad de tiempo. Después de la primera subida, el arreglo ya ha sido modificado.

Pregunta 2

Situación inicial

$$x = 00000000$$
$$y = 01110111$$
$$z = 000$$
$$n_0 = 01110111$$
$$m_0 = 001$$
$$r = 000$$
$$s = 0$$

Subida CLK 1

$$\neg(y[m]) = 0$$
$$\neg(y[z]) = 0$$
$$z = 0$$
$$s = 1$$
$$y = 01110110$$
$$x = 00000000$$
$$z = 000$$
$$n = 01110110$$
$$m = 001$$

Subida CLK 2

$$\neg(y[m]) = 0$$
$$\neg(y[z]) = 0$$
$$y = 01110100$$
$$x = 00000000$$
$$z = 010$$
$$n = 01110100$$
$$m = 011$$

Subida CLK 3

$$\neg(y[m]) = 1$$
$$\neg(y[z]) = 0$$
$$y = 01110100$$
$$x = 00000000$$
$$z = 011$$
$$n = 01110100$$
$$m = 100$$

Subida CLK 4

$\neg(y[m]) = 0$
$\neg(y[z]) = 1$
$y = 01110100$
$x = 00010000$
$z = 100$
$n = 10000100$
$m = 101$

Subida CLK 5

$\neg(y[m]) = 0$
$\neg(y[z]) = 0$
$y = 01100100$
$x = 00010000$
$z = 101$
$n = 01110100$
$m = 110$

Subida CLK 6

$\neg(y[m]) = 0$
$\neg(y[z]) = 0$
$y = 01000100$
$x = 00010000$
$z = 110$
$n = 01010100$
$m = 111$

Subida CLK 7

$\neg(y[m]) = 0$
$\neg(y[z]) = 0$
$y = 01000100$
$x = 00010000$
$z = 111$
$n = 01010100$
$m = 000$

Subida CLK 8

$\neg(y[m]) = 1$
$\neg(y[z]) = 1$
$y = 11000100$
$x = 00010001$
$z = 000$
$n = 11010101$
$m = 001$

Subida CLK 9

$$\neg(y[m]) = 1$$
$$\neg(y[z]) = 1$$
$$y = 11000101$$
$$x = 00010111$$
$$z = 001$$
$$n = 11100000$$
$$m = 010$$

Subida CLK 10

$$\neg(y[m]) = 0$$
$$\neg(y[z]) = 1$$
$$y = 11000101$$
$$x = 00010111$$
$$z = 010$$
$$n = 11011100$$
$$m = 011$$

Pregunta 3

Código del módulo

```
module counter(input logic clk, output [4:0] data, output logic fibo);
//-- La salida es un registro de 5 bits, inicializado a 0
reg [4:0] data = 0;
//-- Sensible al flanco de subida
always_ff@(posedge clk) begin
//-- Incrementar el registro
    data $<=$ data + 1;
end
endmodule
```