Pre-Informe 3
Lenguaje de Descripción de Hardware

Verónica Obilinovic Mercado
201513085-k

Pregunta 3.1: Investigación de Conceptos

a) ¿Qué es un lenguaje de descripción de hardware? ¿Qué lo diferencia de un lenguaje de programación?

El lenguaje de descripción de hardware o HDL es un lenguaje orientado a la descripción de estructuras y comportamiento de hardware, cuya función es describir las características de un circuito electrónico. Permite definir la estructura a través de la simulación para verificar el correcto funcionamiento del circuito. Además, realiza síntesis lógica, que sería equivalente a la compilación del software.

Podemos compararlo con un lenguaje de programación como C o Java ya que cada operación se describe explícitamente a través de expresiones, donde cada una de ellas corresponde a la operación de un bloque de circuito. Pero la diferencia con estos lenguajes esta en que el HDL incluye la noción del tiempo (CLK). Además, en HDL tenemos los procesos que se ejecutan en paralelo, a diferencia de otros lenguajes como por ejemplo C.

b) Explique la funcionalidad de los tipos de variables *wire, reg, logic* y cuales son sus diferencias.

- *wire:* En español: alambre La función de una conexión net es unir los componentes del circuito. *wire* es un tipo de net, la cual no posee la capacidad de almacenar información. Son alimentadas mediante la salida de un módulo o por la asignación *assign*. Si las señalas *wire* están desconectadas, automáticamente tomarán el valor desconocido o x.
- *reg:* También llamado registro, es un tipo de dato asociado a las salidas de flip-flops. A diferencia de *wire*, los registros sí poseen capacidad de información.
- *logic:* En SystemVerilog se introduce el tipo de variable logic como reemplazo de de reg y wire, para que automáticamente designe cual sera la variable a utilizar, dependiendo del comportamiento que posea.

Como mencionamos anteriormente, la principal diferencia entre *wire* y *reg* es que el primero no almacena información y el otro sí. Además, las variables tipo *reg* se asignan en un proceso *initial* o *always*, jamás se asignan en *wire*.

c) ¿Cuál es la diferencia entre los operadores de asignación *assign* <= y =, y en qué secciones del módulo se utilizan?

El operador <= es una asignación sin bloqueo o *nonblocking assignment* y = es una asignación bloqueada o *blocking assignment*
La asignación de bloqueo es ejecutada en serie, ya que bloquea la ejecución de la siguiente instrucción hasta que se complemente. Por esto, los resultados de la siguiente declaración pueden depender de que la primera haya sido completada.
En cambio, la asignación sin bloqueo se ejecuta en paralelo ya que describe las tareas que ocurran todas al mismo tiempo. Los resultados no dependerán de la primera declaración. assign es una asignación continua a realizar fuera de un *always*. El valor de LHS es actualizado cuando 1 RHS cambia.

Estos operadores son utilizados en el módulo dentro del proceso *always* o *initial*

d) Dada la siguiente expresión: $reg[15:0]g = 16'hA6B2$; Explique su estructura, qué significa cada una de sus partes y cuál es el equivalente binario del valor definido.

Es un tipo de dato de registro [15 : 0] donde 15 nos indica que es de 16 bits (el valor de quince no considera al cero, por eso al tomarlo en cuenta se le suma uno), y 0 es el bit menos significativo. Por lo tanto, se define g como la salida de un registro de 16 bits. 16'h nos dice que el registro se encuentra en el sistema hexadecimal, por lo tanto, el número correspondiente a g será $A6B2 = 1010011010110010 = 42674$

e) Si consideramos el módulo. Despúes de una subida del "clk", el registro "arr" resulta tener el valor binario 0111. Explique el delay y las asignaciones dentro del bloque alwaysff influyen en este resultado.

Reg[3:0]	Reg first	wire
0101		
	0	
		second
		1
	1	
0111		

Figure 1: Breve ruteo de lo que sucede en el módulo.

En la figura anterior podemos ver resumidamente lo que sucede en el modulo entregado. Ahora analizaremos las lineas mas importantes:

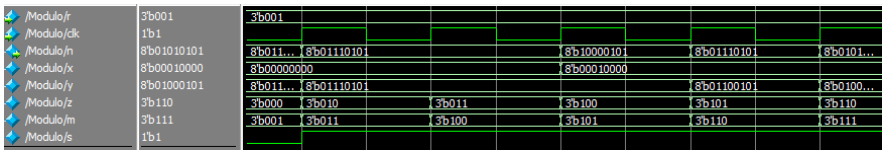
- $reg[3:0]arr$: Un registro de 4 bits definido como 0101
- $regfirst$: Un registro de 1 bits definido como 0
- $assignsecond = (first)$: Al dato wire definido como second, se le asigna el negado de first, que está definido como 0, por lo tanto, el negado es 1.
- $alwaysff@(posedgeclk)begin$: Mientras vaya cambiando el valor de clk, sucederá la siguiente instrucción. Cabe señalar que posedge signfica positivo, es decir, cuando clk tiene una transición de 0 a 1.
- $arr(first) <= (\ (arr[second]))$: Se toma el bit $first$ de arr, es decir, se toma el bit número 1 del arr 0101, osea, el 0, y se cambia por el negativo de $second$. En resumen, en el número binario 0101 se intercambio el 0 por un 1, quedando como 0111.

Pregunta 3.2: Análisis de Código

```
module Modulo(input logic [2:0]r,
        input logic clk,
        output logic [7:0]n);
reg [7:0]x = 8'b0;
reg [7:0]y = 8'b01110111;
reg [2:0]z = 3'b0;
wire [2:0]m;
assign n = (x+y)%256;
assign m = (z+1)%8;
reg s = 1'b0;
always_ff@(posedge clk) begin
        if (~(s)) begin
                z = r;
                s <= 1;
        end
        y[z] <= (~(y[m]));
        x[m] <= (~(y[z]));
        z = (z+1)%8;
        end
endmodule
```

R	CLK	N	X	Y	Z	M	S
3'b001	1'b0	8'b0	8'b0	8'b01110111	3'b0	3'b0	1'b0
3'b001	1'b1	8b'01110111	8'b0	8b'01110101	3'b000	3'b001	1'b1
3'b001	1'b0	8'b01110101	8'b0	8'b01110101	3'b010	3'b011	1'b1
3'b001	1'b1	8'b01110101	8'b0	8'b01110101	3'b010	3'b011	1'b1
3'b001	1'b0	8'b01110101	8'b0	8'b01110101	3'b011	3'b100	1'b1
3'b001	1'b1	8'b01110101	8'b0	8'b01110101	3'b011	3'b100	1'b1
3'b001	1'b0	8'b10000101	8'b00010000	8'b01110101	3'b100	3'b101	1'b1
3'b001	1'b1	8'b10000101	8'b00010000	8'b01110101	3'b100	3'b101	1'b1
3'b001	1'b0	8'b01110101	8'b00010000	8'b01100101	3'b101	3'b110	1'b1
3'b001	1'b1	8'b01110101	8'b00010000	8'b01100101	3'b101	3'b110	1'b1

Figure 2: Análisis realizado con un valor r de 3'b001



Pregunta 3.3: Diseño de Módulo

```
module Fibonacci (input logic [4:0]cont,
                  input logic clk,
                  output logic bool[1:0]);
reg s = 0;
reg [4:0]numprev = 5'b0;
reg [4:0]numactual = 5'b00001;
reg [4:0]sum = 5'b0;

always_ff@(posedge clk) begin
        if (~(s)) begin
                  sum = (numprev + numactual);
                  if ((sum == cont)) begin
                          bool = 1;
                  numprev = numactual;
                  numactual = (numactual+1);
        cont = cont+1;
        end
endmodule
```

Figure 3: Serie Fibonacci en VHDL