Pre-Informe 3

Rodrigo Díaz Del Valle
201773012-9

Pregunta 1

a) Un lenguaje de descripción de hardware (HDL) busca simular y sintetizar lógica. Al simular se crean inputs en el modulo y se revisa que el output se obtenga sin errores para que el modulo sea valido, y durante la síntesis, la descripción del modulo se transforma en puertas lógicas.

A diferencia de un lenguaje de programación el cual utiliza programas para resolver problemas, el HDL esta enfocado en simular un hardware y asi evitar gastar tiempo en testear en la vida real si es viable el hardware que se quiere crear.

b) *Wire* es utilizado generalmente para representar una conexion por cable, por lo que sirve para conectar distintos modulos.

Reg viene de *register* y su objetivo es mantener registrado o guardado un valor o variable, es necesario que la asignacion sea proceduralmente asignado (por ejemplo en un always@).

Finalmente el *logic* funciona de manera similar al reg al tener la funcion de guardar variables, sin embargo a diferencia del *reg*, el *logic* es mas flexible al asignar y manipular las variables, tambien puede manejarse a traves de asignacion continua o por *blocking/non blocking assignment*

c) *assign* sirve para asignarle un valor (fijo o dependiente de variables) a un dato tipo *wire*. El operador = es un *blocking assignment*, lo que significa que debe ejecutarse primero esta operacion antes de seguir "leyendo" el codigo. En cambio el <= es un *nonblocking assignment*, esto significa que todas las lineas de codigo se realizan a la vez, y por lo tanto el cambio de valor por un < + no afecta a otros <= si se realizan en un bloque de codigo seguido.

Los *blocking assignment* (=) pueden usarse en initial, bloques always y assign, en cambio los *nonblocking assignment* (<=) pueden usarse en initial y bloques always solamente.

d) reg significa que se esta declarando un registro, [15 : 0] indica que se usaran 16 bits (desde el indice 15 al 0), g es la variable a utilizar, = indica que se esta asignado el valor del lado derecho, el 16' que deberia estar antes de la "h", indica la cantidad de bits que se usaran, h indica que el valor es del tipo hexadecimal y A6B2 es el valor hexadecimal entregado.

El equivalente binario del valor A6B2 es 1010 0110 1011 0010

e) Lo que sucede en este modulo, es que first comienza con un valor igual a 0 en binario, luego second toma el valor 1 (NOT first). Al entrar al always_ff primero se realiza el cambio de first al valor 1, sin embargo *second* no cambia debido a que no existen cambios fuera del bloque hasta llegar al "end", luego se procede a cambiar el valor del arr[first] a NOT arr[second] y con los nuevos valores, arr[first] = arr[1] = 01**0**1, y NOT arr[second] = NOT arr[1] = NOT 0 = 1, por lo tanto el nuevo valor del indice 1 pasa a ser 1.

Pregunta 2

A continuacion se presenta la tabla con el ruteo de la pregunta:

nro. subida	x	y	z	m	n	s	clk	r
	00000000	01110111	000	001	01110111	0	0	000
1							1	
			000			1		
	00000000	01110110	001					
				010	01110110			
2							$0 \longrightarrow 1$	
	00000000	01110100	010					
				011	01110100			
3							$0 \longrightarrow 1$	
	00000000	01110100	011					
				100	01110100			
4							$0 \longrightarrow 1$	
	00010000	01110100	100					
				101	10000100			
5							$0 \longrightarrow 1$	
	00010000	01100100	101					
				110	01110100			
6							$0 \longrightarrow 1$	
	00010000	01000100	110					
				111	01010100			
7							$0 \longrightarrow 1$	
	00010000	01000100	111					
				000	01010100			
8							$0 \longrightarrow 1$	
	00010001	11000100	000					
				001	11010101			
9							$0 \longrightarrow 1$	
	00010011	11000101	001					
				010	11011000			
10							$0 \longrightarrow 1$	
	00010111	11000101	010					
				011	11011100			

si tomamos r como 000, el resultado después de 10 subidas del CLK es 11011100 tal y como se ve en el final de la tabla

Pregunta 3

Para este caso se utiliza un clk de periodo 20 ns en un rango de 640 ns con un contador que comienza en 0.

El modulo creado y los resultados obtenido en un ciclo (hasta que el contador llega a su valor máximo y se reinicia), se ve en las siguientes imágenes:

C:/Modeltech_pe_edu_10.4a/examples/tarea2.sv (/cont_fib) - Default

```
1  module cont_fib(input logic clk,
2                  output logic [4:0]c,
3                  output logic isfib);
4    initial begin         //definimos valores iniciales para testear
5      isfib = 1'b0;
6      c = 5'b0;
7    end
8    logic [4:0]fib1 = 5'b0; //se definen los primeros 2 num. de fibonacci
9    logic [4:0]fib2 = 5'b1;
10   always @(posedge clk) begin
11     if ((c + 1)%32 == 5'b0) begin // verificamos que el contador no llegue por sobre
12       c = 5'b0;                   // su capacidad de 5 bits y si es asi, vuelve a 0
13       fib1 = 5'b0;                // si c vuelve a 0, fibonacci vuelve a su estado
14       fib2 = 5'b1;                // inicial y se repite el ciclo
15       end
16     else c = c+1;
17     if ((c == 5'b0) || (c == fib1 + fib2)) begin // se verifica fibonacci
18       isfib = 1'b1;
19       if (c > 5'b0) begin
20         fib1 <= fib2;
21         fib2 <= fib1 + fib2;
22         end
23       end
24     else isfib = 1'b0;
25   end
26 endmodule
```

Figure 1: Modulo creado con ModelSim en SystemVerilog

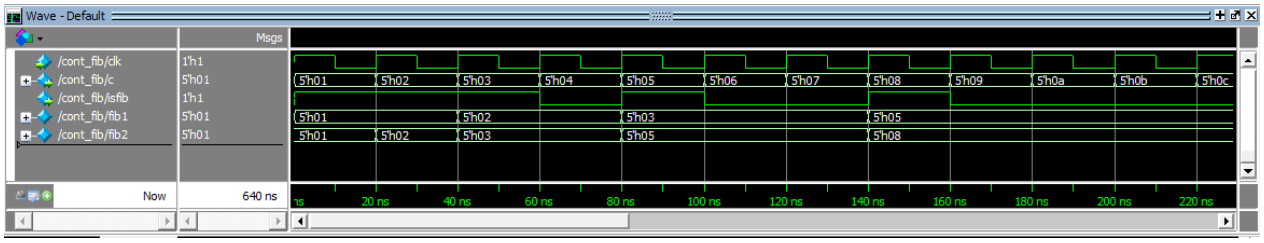

Figure 2: Primera parte de la ejecución del modulo, valores desde los 0 a 220 ns

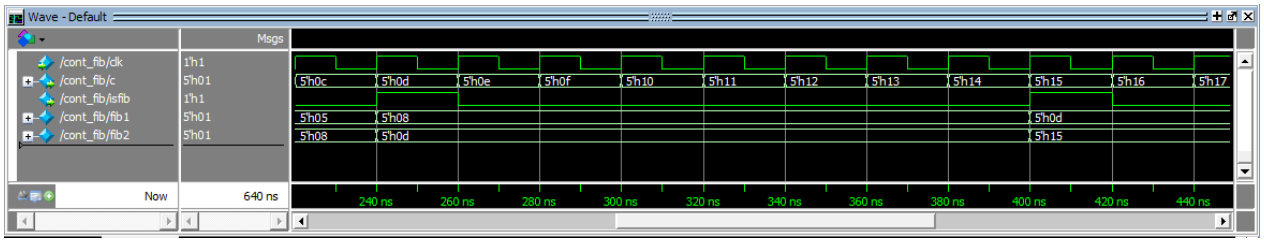

Figure 3: Segunda parte de la ejecución del modulo, valores desde los 220 a 440 ns

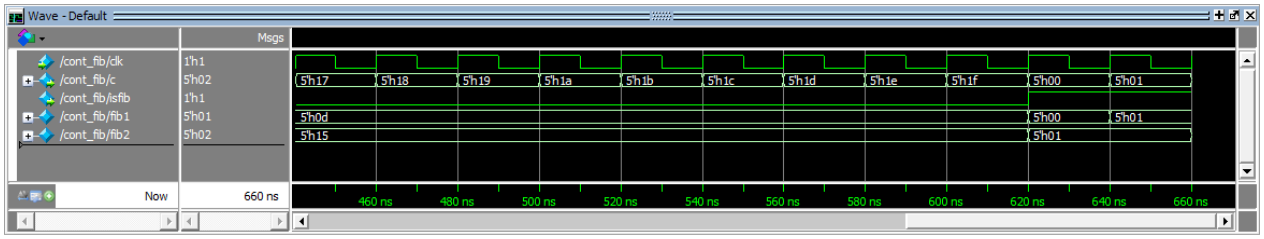

Figure 4: Tercera parte de la ejecución del modulo, valores desde los 440 a 660 ns (el ciclo se reinicia en 620 ns)