Pre-Informe Experiencia 1

Bastián Solar Vargas
201773003-k

1 Investigación de conceptos.

1. ¿Qué es un lenguaje de descripción de hardware? ¿Qué lo diferencia de un lenguaje de programación?

   Solución:

   Un Lenguaje de Descripción de Hardware, o por sus siglas en ingles: HDL, es un lenguaje computacional usado para describir el comportamiento y estructura de circuitos electrónicos, principalmente circuitos lógico-digitales. Luego un HDL no corresponde a una serie de ordenes a realizar como lo seria un lenguaje de programación, si no que describe el esquema o las conexiones de un chip. Así entonces una diferencia con respecto a un lenguaje de Programación es el manejo de la noción temporal. Un lenguaje de programación (por ejemplo Assembly) entrega comandos a realizar en un orden secuencial, en cambio un lenguaje de descripción de hardware describen una conexión arbitraria de circuitos digitales, permitiendo así un paralelismo no alcanzable en lenguajes de programación.

2. Explique la funcionalidad de los tipos de variable wire, reg y logic, y cuales son sus diferencias. (HINT: Recordar que los registros dependen del CLK)

   Solución:

   En Verilog se observan dos tipos de datos: redes y variables. La principal diferencia entre ellos radica en que una variable representa una pieza de almacenamiento, de modo es su función es de almacenar valores en situaciones específicas, y una red se representa la conexión entre piezas, de modo que permite asignaciones continuas. Ante esto un "reg" es tipo variable, almacenando valores almacenando valores en casi cualquier entorno del modulo, principalmente dentro de un bloque always y/o dado el comportamiento de un CLK (situaciones específicas); "wire" en cambio es tipo red, de modo que se utilizan para unión de outputs e imputs de modulos u otras asignaciones mas frecuentes. En SystemVerilog se implementó el tipo "Logic", el que resultó ser más flexible, pudiéndose usar en casi todas las situaciones, excepto cuando una conexión incluía múltiples controladores (multi-drivers), para lo cual se **debe** utilizar una variable tipo net, como cuando se lleva el output de un modulo al input de otro.

3. ¿Cuál es la diferencia entre los operadores de asignación assign, <= y =, y en que secciones del módulo se utilizan?

   Solución:

   - **assign** se utiliza junto al operador "=" para fijar un valor o una union de piezas, donde no se utiliza sensibilidad o un evento "gatillador" para asignaciones. Siempre se utiliza fuera de bloques de proceso (como always o initial).
   - = representa una asignación bloqueante (*blocking assignment*), lo que significa que la asignación se realizará sólo después que la linea anterior se haya ejecutado, y se continuará sólo después de realizada la asignación. Se utiliza (por si solo o sin el operador assign) dentro de procesos always o initial de lógica combinacional (sin latch).
   - <= representa una asignación no-bloqueante (*non-blocking assignment*), lo que significa que la asignación se realizará en paralelo al tiempo que tenga asociado, de modo que la evaluación de la expresión se hace al inicio del bloque temporal pero su asignación se realiza al final de éste. Se utiliza dentro de procesos always o initial de lógica secuencial (con latch o memoria).

4. Dada la siguiente expresión:

```
reg[15:0]g = 16'hA6B2;
```

Explique su estructura, que significa cada una de sus partes, y cual es el equivalente binario del valor definido.

Solución:

Descibiendo su estructura por separado encontramos:

- **tipo - "reg"**, representa el tipo de la variable a definir, en este caso de tipo registro.
- **tamaño - "[15:0]"**, representa el tamaño o cantidad de bits (e indices en que se encuentra cada uno) que tendrá la variable, en este caso 16 bits del 15 al 0.
- **nombre - "g"**, es el nombre dado a la variable.
- **operador de asignacion - "="**, operador para realizar la asignación de la variable.
- **expresion a asignar - "16'hA6B2"**, es el valor a asignar, el cual se separa en 3 partes:
  (a) **16'** tamaño del numero a representar, en este caso 16 bits.
  (b) **h** tipo del numero a representar, en este caso Hexadecimal.
  (c) **A6B2** numero a representar en el tipo específicado, en este caso el 42674_{10} o 1010011010110010_2 forma en la que ocupa 16 bits.

5. Si consideramos el siguiente modulo

```
module Yaled (input logic clk);
    reg [3:0]arr = 4'b0101;
    reg first = 1'b0;
    wire second;
    assign second = (~(first));
    always_ff@(posedge clk) begin
        first = 1;
        arr [first] <= (~(arr[second]));
    end
endmodule
```

Después de una subida del "clk", el registro "arr" resulta tener el valor binario 0111. Explique porque el delay y las asignaciones dentro del bloque always_ff influyen en este resultado.

Solución:

El delay y tipo de asignaciones influyen en el resultado dado el tiempo de ejecución. En este caso, en la primera linea del bloque *always* la asignacion de *first* es de tipo bloqueante, de modo que se debe terminar la asignación para poder continuar causando delay, por lo cual en la linea dos se accede al bit según el nuevo valor de *first*, que es 1, asignándole el negado del valor del indice *second* (vale decir, indice 1 que almacena el valor 0, el cual al negarlo termina siendo 1).

Si por el contrario, la primera linea del bloque *always* usase la asignación no bloqueante <=, dicho delay no existiría y ambas lineas se ejecutarían en paralelo realizando las asignaciones al final del bloque temporal, cambiando el valor del bit 0 de *arr* a 1, lo que lo mantendría igual.

2 Realice un ruteo del valor del output "n" despues de 10 subidas del CLK Defina usted el valor de entrada de 'r', y explicítelo junto al ruteo.

```
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
```

Solución:

Nro	r	CLK	z	m	$\sim y[m]$	$\sim y[z]$	x_{nuevo}	y_{nuevo}	z_{nuevo}	m_{nuevo}	n	s
0	010	0					00000000	01110111	000	001	01110111	0
1		1							010			1
			010	001	0	0	00000000	01110011	011		01110011	
		0								100		
2		1	011	100	0	1	00010000	01110011	100		10000011	
		0								101		
3		1	100	101	0	0	00010000	01100011	101		01110011	
		0								110		
4		1	101	110	0	0	00010000	01000011	110		01010011	
		0								111		
5		1	110	111	1	0	00010000	01000011	111		01010011	
		0								000		
6		1	111	000	0	1	00010001	01000011	000		01010100	
		0								001		
7		1	000	001	0	0	00010001	01000010	001		01010011	
		0								010		
8		1	001	010	1	0	00010001	01000010	010		01010011	
		0								011		
9		1	010	011	1	1	00011001	01000110	011		01011111	
		0								100		
10		1	011	100	1	1	00011001	01001110	100		01100111	
		0								101		

Por lo tanto, para un $r = 010$ el valor de n luego de 10 subidas y bajadas del CLK será de 01100111.

3 Escriba un módulo en System Verilog para la función siguiente:

Un contador de 5 bits que tenga como outputs la cuenta actual, y un booleano que indique con 1 si es que el numero actual se encuentra en la serie de Fibonacci, o con un 0 si no.

Para el desarrollo de esta pregunta es obligatorio utilizar ModelSim al momento de escribir su código. Debe tomar una captura de pantalla de la simulación una vez el contador alcance su máximo valor, y presentarla junto al modulo escrito.

Solución:

Si bien SystemVerilog incluye el operador While para poder mantener un ciclo que aumente el contador, manejar el contador con un reloj o un *CLK* es más sincrónico y simple de manejar, por lo cual se define el módulo con un input que será dicho reloj.

```
module countFibonacci(      input logic clk,
                    output logic [4:0] num,
                    output logic fib);
    reg [4:0] value = 5'b0;
    reg [31:0] fibonaccis = 32'b00000000001000000010000100101111;
    assign num = value;
    assign fib = fibonaccis[value];
    always_ff @(posedge clk) begin
            value <= value + 1;
    end
endmodule
```

Luego al simular el módulo y representar los valores como ondas o *waves* resulta:

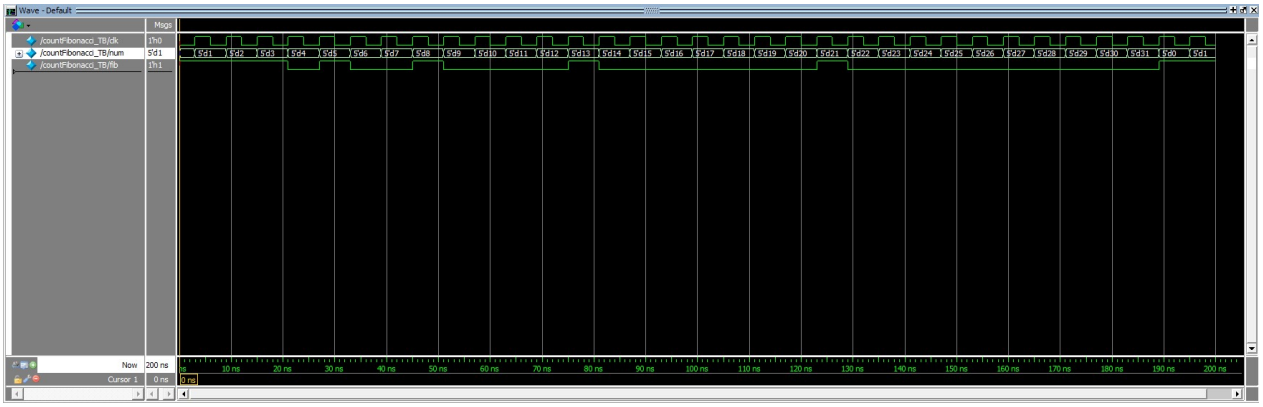

Figure 1: Ondas de las variables del modulo durante simulación.