Pre-Informe 3
INF-245

Felipe Quintanilla Godoy
201773047-1

Pregunta 1

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

Los lenguajes de descripción de hardware (HDL) son lenguajes formales utilizados para describir la estructura y comportamiento de un circuito electrónico. Estos permiten al diseñador de hardware modelar y simular el funcionamiento de un determinado componente antes de su construcción. La gran diferencia que tiene un HDL con un lenguaje de programación es que el primero tiene bien inserta la noción de temporalidad. Esto es debido a que el tiempo cumple un rol fundamental en los circuitos electrónicos reales, háblese de clock, delay, entre otros. Otra diferencia es que los HDL trabajan con el concepto de proceso, lo que permite ejecutar en paralelo.

b) Explique la funcionalidad de los tipos de variable wire, reg y logic, y cuáles son sus diferencias.

- **wire:** El wire es un tipo de variable "net", cumpliendo la función de ser una representación estructural de conexión entre componentes (como módulos o compuertas) por lo que no puede almacenar información. Su valor se deriva de lo que está siendo transmitido en un momento determinado por él. La única forma de asignar un valor es usando una asignación continua como "assign".
- **reg:** El reg es un tipo de variable que puede almacenar información. Esto quiere decir que el valor de estas variables no cambiará hasta que se le asigne otro valor.
- **logic:** El logic sirve para describir un tipo de dato genérico de 4 estados (1, 0, Z, X). La ventaja de las varaibles de tipo logic es que pueden soportar múltiples asignaciones sin generar errores de tiempo de compilación; no así las otras variables.

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

- **assign:** Se usa exclusivamente para modelar lógica combinacional y corresponde a una asignación de tipo continua. Esto quiere decir, por ejemplo, que si a a le asigno el valor de b, cada vez que b cambie de valor también lo hará a. El assign siempre se debe utilizar fuera de cualquier bloque de proceso (como un always o un initial).
- **<=:** Se usa para realizar asignaciones de forma paralela; osea que si tengo más de una de estas asignaciones dentro de un proceso, estas asignaciones ocurren en paralelo y sin un orden determinable. Esta asignación se usa dentro de un bloque de proceso.
- **=:** Este operador se usa para realizar asignaciones de forma secuencial. Si se usa más de un = dentro de un proceso, las asignaciones se harán una después de otra. Esta asignación se usa dentro de cualquier bloque de proceso.

d) Dada la siguiente expresión explique su estructura, qué significa cada una de sus partes y cuál es el equivalente binario del valor definido.

reg [15:0] g = h'A6B2;

- **reg:** Corresponde al tipo de variable, en este caso una variable tipo registro.
- **[15:0] g:** Se declara una variable de nombre "g" y de largo 16 bits.
- **=:** Asignación secuencial.
- **h'A6B2:** Corresponde a los bits que se le asignan a la variable "g". La h significa que el número está en hexadecimal y el A6B2 es el propio número hexadecimal.

El equivalente binario del número hexadecimal anterior es el siguiente:

$$A6B2_{16} = 1010011010110010_2$$

e) Si consideramos el siguiente módulo:

```
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 por qué el delay y las asignaciones dentro del bloque always ff influyen en este resultado.

En este ruteo las asignaciones cumplen un rol esencial. Antes del bloque always, los valores de la variables son:

$$\text{arr} = 0101 \quad \text{first} = 0 \quad \text{second} = 1$$

Cuando se entra al always, las operaciones $first = 1$ y $arr[first] <= ((arr[second]))$ se realizan "paralelamente", con el pero de que la operación $first = 1$ fuerza secuencialidad y se realizará antes. Hay que notar que el valor de *second* depende del valor de $first$, por lo que si cambia este último cambia el primero. Sin embargo, se sabe que todo cambio de valor conlleva un delay, por lo que el valor de *second* que entra en la operación $arr[first] <= ((arr[second]))$ es el antes asignado. Debido a esto se obtiene que $arr = 0111$ y el valor de *second* cambia a 0.

En el caso que dentro del always hubiese un = en vez de un <=, las operaciones se realizarían secuencialmente lo que generaría que el valor de *second* cambie antes de entrar a la segunda operación.

Pregunta 2

Realice un ruteo del valor del output "n" después 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
```

Considerando r = 000

- **Antes del proceso:**
  - x = 00000000
  - y = 01110111
  - z = 000
  - m = 001
  - **n = 01110111**
  - s = 0
- **Primera subida de CLK:**
  - $\sim(y[m]) = 0$
  - $\sim(y[z]) = 0$
  - x = 00000000
  - y = 01110110
  - z = 001
  - m = 010
  - **n = 01110110**
  - s = 1
- **Segunda subida de CLK:**
  - $\sim(y[m]) = 0$
  - $\sim(y[z]) = 0$
  - x = 00000000
  - y = 01110100
  - z = 010
  - m = 011
  - **n = 01110100**

- s = 1

- **Tercera subida de CLK:**
  - $\leftarrow (y[m]) = 1$
  - $\leftarrow (y[z]) = 0$
  - x = 00000000
  - y = 01110100
  - z = 011
  - m = 100
  - **n = 01110100**
  - s = 1

- **Cuarta subida de CLK:**
  - $\leftarrow (y[m]) = 0$
  - $\leftarrow (y[z]) = 1$
  - x = 00010000
  - y = 01110100
  - z = 100
  - m = 101
  - **n = 10000100**
  - s = 1

- **Quinta subida de CLK:**
  - $\leftarrow (y[m]) = 0$
  - $\leftarrow (y[z]) = 0$
  - x = 00010000
  - y = 01100100
  - z = 101
  - m = 110
  - **n = 01110100**
  - s = 1

- **Sexta subida de CLK:**
  - $\leftarrow (y[m]) = 0$
  - $\leftarrow (y[z]) = 0$
  - x = 00010000
  - y = 01000100
  - z = 110
  - m = 111
  - **n = 01010100**
  - s = 1

- **Séptima subida de CLK:**
  - $\leftarrow (y[m]) = 1$
  - $\leftarrow (y[z]) = 0$

- x = 00010000
- y = 01000100
- z = 111
- m = 000
- **n = 01010100**
- s = 1

- **Octava subida de CLK:**
  - $\leftarrow (y[m]) = 1$
  - $\leftarrow (y[z]) = 1$
  - x = 00010001
  - y = 11000100
  - z = 000
  - m = 001
  - **n = 11010101**
  - s = 1

- **Novena subida de CLK:**
  - $\leftarrow (y[m]) = 1$
  - $\leftarrow (y[z]) = 1$
  - x = 00010011
  - y = 11000101
  - z = 001
  - m = 010
  - **n = 11011000**
  - s = 1

- **Décima subida de CLK:**
  - $\leftarrow (y[m]) = 0$
  - $\leftarrow (y[z]) = 1$
  - x = 00010111
  - y = 11000101
  - z = 010
  - m = 011
  - **n = 11011100**
  - s = 1

Pregunta 3

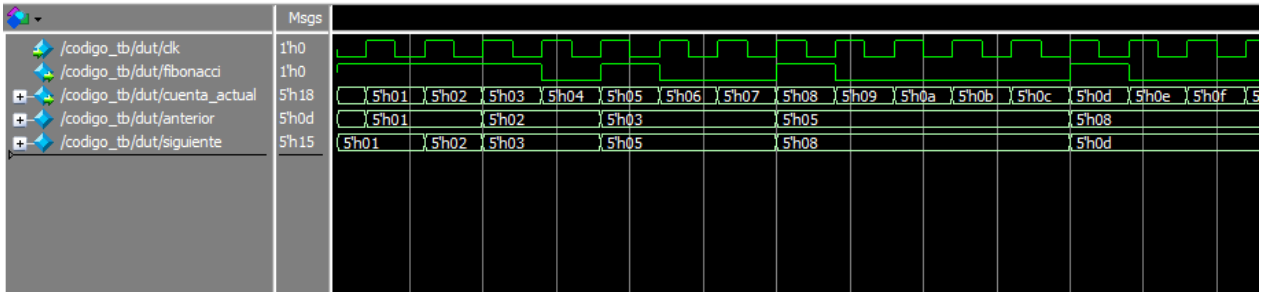
Escriba un módulo en System Verilog para la función siguiente: un contador de 5 bits que tenga de outputs la cuenta actual y un booleano que indique con 1 si es que el número actual se encuentra en al serie de Fibonacci, o con un 0 si no.

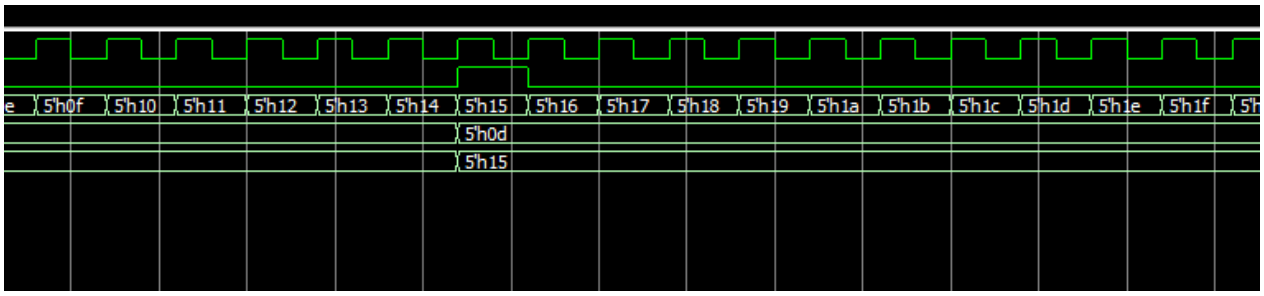
El módulo creado para resolver el problema es el siguiente:

```
module codigo(
input logic clk,
output logic fibonacci,
output logic [4:0] cuenta_actual
);

logic [4:0] anterior;
logic [4:0] siguiente;

initial begin
        fibonacci = 1;
        cuenta_actual = 5'b00000;
        anterior = 5'b00000;
        siguiente = 5'b00001;
end

always @(posedge clk) begin
        cuenta_actual = cuenta_actual + 1;
        if (cuenta_actual == siguiente + anterior) begin
                fibonacci <= 1;
                anterior <= siguiente;
                siguiente <= cuenta_actual;
        end
        else begin
                fibonacci<= 0;
        end
end
endmodule
```

La variable Fibonacci corresponda al booleano que indica si el número que está en cuenta actual pertenece a la serie o no.

Ahora, la simulación solicitada:


Msgs
/codigo_tb/dut/clk
/codigo_tb/dut/fibonacci
/codigo_tb/dut/cuenta_actual
/codigo_tb/dut/anterior
/codigo_tb/dut/siguiente
1'h0
1'h0
5'h18
5'h0d
5'h15
5'h01 5'h02 5'h03 5'h04 5'h05 5'h06 5'h07 5'h08 5'h09 5'h0a 5'h0b 5'h0c 5'h0d 5'h0e 5'h0f
5'h01 5'h02 5'h03 5'h05 5'h08 5'h0d


5'h0f 5'h10 5'h11 5'h12 5'h13 5'h14 5'h15 5'h16 5'h17 5'h18 5'h19 5'h1a 5'h1b 5'h1c 5'h1d 5'h1e 5'h1f
5'h0d
5'h15