Pre-Informe 3
INF-245

Javier Mendoza C
201773001-3

Pregunta 1

1. Un Lenguaje de Descripción de Hardware es un lenguaje de programación especializado que se utiliza para definir la estructura, diseño y operación de circuitos electrónicos; hacen posible una descripción formal del circuito y posibilitan su análisis automático y simulación.

   La mayor diferencia con un lenguaje de programación como tal es que un HDL utiliza una noción explícita de tiempo. En un lenguaje de programación las órdenes se ejecutan secuencialmente, mientras que en un HDL las órdenes se ejecutan de forma paralela.

2. - Variables de tipo **wire** se utilizan para conectar diferentes elementos en el código, básicamente representando los cables físicos asociados. No pueden mantener un valor, por lo que deben ser constantemente valorizados por un assign continuo o por un puerto de un módulo.
   - Variables de tipo **reg** se corresponden con elementos de almacenamiento de información, no necesariamente registros. Almacenan un valor hasta que se les asigne otro, al igual que una variable normal.
   - Variables de tipo **logic** son similares a los **reg**, con un nombre diferente para evitar confusión (El nombre **reg** da la impresión que representa un registro). Las diferencias entre **logic** y **reg** es que **logic** no permite múltiples drivers, sino que tomará siempre el último valor asignado, mientras que una variable **reg** o **wire** dará un valor **X** si tiene 2 asignaciones de valores diferentes.

3. - **assign** se utiliza para indicar las relaciones entre variables, por lo que se utiliza para definir lógica combinacional (fuera de las estructuras *always*@).
   - = se conoce como Asignación con Bloqueo. Se utiliza dentro de estructuras concurrentes para asignar valores de forma que cada asignación puede afectar al valor de la siguiente (considerando el orden). Por ejemplo, tener `x = a` y `y = x | b` resultará en `y = a | b`.
   - <= se conce como Asignación sin Bloqueo. Se utiliza dentro de estructuras concurrentes para asignar valores de forma que las asignaciones de tipo <= no afectan el valor de otras asignaciones <= (se dice que <= ejecuta con un cierto delay). En otras palabras, cada asignación se ejecuta como si no se hubieran ejecutado las demás. Esto es especialmente útil a la hora de hacer lógica secuencial, puesto que se puede describir el estado 'después del clock' en función del estado 'antes del clock'.

4.
```
reg [15:0]g = 16'hA6B2
```

   En esta expresión tenemos:

   - **reg:** Declaramos una variable de tipo *reg*.
   - **[15:0]:** La variable almacena 16 bits (del bit 0 al 15).
   - **g:** El nombre de la variable.
   - **16'h:** La variable almacena 16 bits, los cuales están en formato hexadecimal (h).
   - **A6B2:** El valor asignado a la variable (en hexadecimal debido al componente descrito en el punto anterior).

Luego, el valor definido en binario es: `g = 1010 0110 1011 0010`.

5. Al inicio de la ejecución, `reg = 0101, first = 0, second = 1`. Luego, cuando hay una subida del `clk` se le asigna a `first` el valor de 1 de forma secuencial, por lo que afectará en la ejecución de las líneas siguientes del bloque. Luego, al finalizar el bloque se ejecuta `arr[first] <= (~(arr[second]));`. Como second es un cable que depende del valor de first (debido al uso de `assign`), existe un delay en que se actualice su valor, mientras que el valor de first ya fue asignado (por el uso de `=`). Por lo tanto, la expresión se evalúa con `first = second = 1`:

```
arr[first] <= (~(arr[second]));
    arr[1] <= (~(arr[1]));
    arr[1] <= (~0);
    arr[1] <= 1;

    arr = 0111
```

Y así se explica el resultado obtenido.

Pregunta 2

A continuación se presenta una tabla con el ruteo correspondiente. Para cada cambio del clk hay más de una fila para considerar el delay en los *assign*:

x	y	z	m	n	r = 010	s	Estado
0000 0000	0111 0111	000	001	0111 0111	010	0	inicio
		010				1	clk up (1)
0000 0000	0111 0011	011					
			100	0111 0011			
0001 0000	0111 0011	100					clk up (2)
			101	1000 0011			
0001 0000	0110 0011	101					clk up (3)
			110	0111 0011			
0001 0000	0100 0011	110					clk up (4)
			111	0101 0011			
0001 0000	0100 0011	111					clk up (5)
			000	0101 0011			
0001 0001	0100 0011	000					clk up (6)
			001	0101 0100			
0001 0001	0100 0010	001					clk up (7)
			010	0101 0011			
0001 0001	0100 0010	010					clk up (8)
			011	0101 0011			
0001 1001	0100 0110	011					clk up (9)
			100	0101 1111			
0001 1001	0100 1110	100					clk up (10)
			101	0110 0111			

Luego, el valor final para n con $r = 010$ es $n = 01100111$.

Pregunta 3

El módulo pedido se muestra a continuación:

```
module up_counter(output logic [4:0] out,
                  output logic flag,
                  input logic clk,
                  output logic [4:0] first,
                  output logic [4:0] second);

initial begin
    out <= 1;
    first <= 5'b00001;
    second <= 5'b00001;
    flag <= 1;
end

always @(posedge clk) begin
    out = out + 1;
    if (first+second == out) begin
        second <= first;
        first <= out;
        flag <= 1;
    end
    else begin
        flag <= 0;
    end
end
endmodule
```

Pantallazo de la simulación: