Pre-Informe 3
HDL

Esteban Carrillo Naranjo
201773032-3

Pregunta 1

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

solución:

Es un elemento que permite unir la brecha entre la parte de diseño de un circuito, y la parte de verificación. Es decir, entre otras funciones, logra simular y analizar un diseño formalizado de un circuito, sin la necesidad de tenerlo físicamente construido. Tal como lo dice el nombre, es un lenguaje que describe el funcionamiento de un hardware.

Un lenguaje de programación, al contrario de un "HDL", es la herramienta que describe un algoritmo, a base de instrucciones. Por lo tanto, no está diseñado con el fin de describir componentes de hardware.

b) 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:

Aparentemente, ha existido por mucho tiempo problemas de confusión al respecto de sus diferencias. A partir de esto, se sabe que una variable `wire` se usa como un cable en sí mismo, donde sus asignaciones son continuas. Debido a esto, no se puede asignar un valor, sino que se lee y cambia dependiendo de a qué esté conectado.

Para el caso de `reg`, corresponde a un registro, pero este no está presente como tal en el hardware. Variables que tienen este tipo, retienen su valor hasta la siguiente asignación.

Finalmente, `logic` es un tipo que se ha añadido al lenguaje luego de los dos anteriores, con la diferencia de que elementos con `logic`, pueden cambiar su valor con asignaciones (tanto con un "=" o "blocking assignment", como "<=" o "non-blocking assignment") y pueden también derivados de otras variables, es decir, una mezcla de ambas.

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

solución:

El uso de "<=" se le llama también "non-blocking assignment" (como se menciona anteriormente), pues no restringe en qué momento se ejecutan las asignaciones siguientes al ser de lógica secuencial.

Esto es contrario a "=" o "blocking assignment", ya que este se ejecuta según el orden en que se presenta en el archivo. De esta forma, se usa en lógica de tipo combinacional (importa el orden).

En la sección "`always_ff`", por ejemplo, cuando se usa con el canto de subida de un CLK, permite (con "<=") una lógica secuencial sincrónica eficiente al tener que una de las variables (que puede ser el CLK mismo) afecte a otras partes del circuito al mismo timpo. En caso contrario, cuando necesitamos una lógica combinacional, se puede usar "`always_comb`" y con asignación de tipo "=".

d) Dada la siguiente expresión:

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

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

solución:

Analizamos por partes:

- `reg`: Corresponde al tipo de variable que estamos usando. En este caso, es un registro. Esto nos permite acceder a cada bit en particular, y efectuar cambios sobre estos.
- `[15:0]`: Se enumeran los bits desde el más significativo con un índice de 15, hasta el menos significativo con índice 0. Esto, además, determina que son 16 bits en total.
- `g`: Es el nombre de la variable con la que se accede en otras sentencias. Guarda los valores de bits que se le asignen.
- `h'`: Indica al compilador, que los siguientes valores son números hexadecimales. Este podría ser "`b'`" por binario, "`o'`" por octal, etc.
- `A6B2`: Es el número hexadecimal, que se guardará en 16 bits (lógicamente) con los valores: "`1010 0110 1011 0010`".

Luego, decimos que `g` es un registro con 16 bits, enumerados del 15 hasta el 0 (desde el bit más significativo al menos significativo) de valor igual a: "`1010011010110010`".

e) Si consideramos el siguiente modulo:

```
1   module Yaled (input logic clk);
2          reg [3:0] arr = 4'b0101;
3          reg first = 1'b0;
4          wire second;
5          assign second = (~(first));
6          always_ff@(posedge clk) begin
7                 first = 1;
8                 arr [first] <= (~(arr[second]));
9                 end
10  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:

Avanzaremos por las líneas de código:

(1) Se inicia el módulo, que recibe como input el `clk` (CLK).

(2, 3) Se declaran las variables `arr`: con 4 bits, enumerados desde 3 a 0 con los valores "0101" inicialmente, y `first`: con sólo 1 bit, que es un 0.

(4, 5) Se genera una variable de tipo `wire`, llamada `second`. Esta se puede considerar como un cable que se añade al que lleva `first`, pero esta extensión tiene un NOT.

(6) Se inicia un bloque que sólo se ejecutará cuando el `clk` esté en el canto de subida ("`posedge`" viene de "positive edge").

(7) La variable `first`, tiene valor 1.

(8) En la variable `arr`, el único bit que cambia es el de índice 1, ya que `first` siempre valdrá 1, desde que se le asigna el valor.

La primera vez que `clk` está en el canto de subida, decimos que `first` tendrá valor 1 y, en ese instante, es igual al valor que tiene `second` (pues este cambia a 0, luego de haber terminado ese bloque). Entonces, se le asigna al bit del índice 1 su propio valor negado, es decir, ahora el bit vale 1, pues antes era 0. Desde esa primera vez, el resto de las variables no cambian, ya que siempre se le asigna 1 a `first` y por consecuencia, `second` tendrá 0. Entonces la segunda vez que hay un canto de subida, se asigna al bit en el índice 1, el valor negado del bit en el índice 0 (que tiene valor 1) y se mantiene así (ver Figura 1).

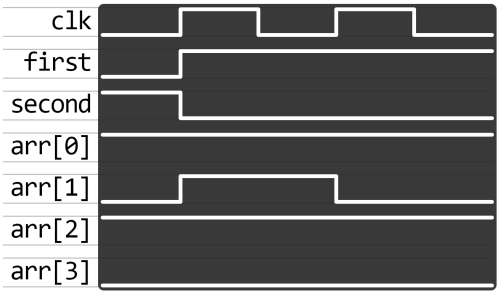

Figura 1. Los valores de la función `Yaled` en t=0.

(9, 10) Se finaliza el bloque y luego el módulo.

Entonces, el delay ocurre al ejecutarse los cambios una vez que el `clk` esté en el canto de subida, lo cual es el factor que permite atrasar las ejecuciones.

Respecto a los efectos de las asignaciones dentro del bloque, `arr[1]` obtiene valor 1, pues cuando se entra en el bloque `always_ff` tanto `first` como `second` tienen valor 1, entonces a `arr[1]` se le da el valor de ¬`arr[1]`. Luego de finalizado el bloque, `first` y `second` se mantienen con los valores que corresponde (1 y 0, respectivamente), por lo que siempre, a `arr[1]`, se le asigna el valor de ¬`arr[0]`.

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.

```
1        module Modulo (input logic [2:0]r,
2                       input logic clk,
3                       output logic [7:0]n);
4               reg [7:0]x = 8'b0;
5               reg [7:0]y = 8'b01110111;
6               reg [2:0]z = 3'b0;
7               wire [2:0]m;
8               assign n = (x+y)%256;
9               assign m = (z+1)%8;
10              reg s = 1'b0;
11              always ff@(posedge clk) begin
12                      if (~(s)) begin
13                              z <= r;
14                              s <= 1;
15                              end
16                      y[z] <= (~(y[m]));
17                      x[m] <= (~(y[z]));
18                      z <= (z+1)%8;
19                      end
20       endmodule
```

solución:

En la siguiente tabla, se muestran los valores obtenidos por cada canto de subida del `clk`. En la columna "***#posedge***", los valores representan cada iteración, donde en 0 están los valores iniciales. Sabiendo esto, usamos `r = 000` (valor simple) y obtenemos lo siguiente usando el simulador:

#posedge	***x***	***y***	***z***	***m***	***n***
0	0000 0000	0111 0111	000	001	0111 0111
1	0000 0000	0111 0110	001	010	0111 0110
2	0000 0000	0111 0100	010	011	0111 0100
3	0000 0000	0111 0100	011	100	0111 0100
4	0001 0000	0111 0100	100	101	1000 0100
5	0001 0000	0110 0100	101	110	0111 0100
6	0001 0000	0100 0100	110	111	0101 0100
7	0001 0000	0100 0100	111	000	0101 0100
8	0001 0001	1100 0100	000	001	1101 0101
9	0001 0011	1100 0101	001	010	1101 1000
10	0001 0111	1100 0101	010	011	1101 1100

Pregunta 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:

El módulo principal usado es:

```
module Counter(input logic clk, reset,
               output logic [4:0]count,
               output logic fib);

         reg [63:0]seq = 64'b11;
         reg [31:0]p1 = 32'b1;
         reg [31:0]p2 = 32'b0;
         reg [31:0]curr = 32'b1;

         always_ff@(posedge clk or posedge reset) begin
                     if (reset) count = 5'b00000;
                     else count = count + 1;

                     fib = seq[count];
         end

         always_ff@(negedge clk) begin
                     seq[curr] = 1;
                     p2 = p1;
                     p1 = curr;
                     curr = p1 + p2;
         end
endmodule
```

Donde el contador aumenta su valor dentro del bloque `always_ff`. Para saber si el número actual está en la secuencia de fibonacci, primero usamos `seq` que guarda si la secuencia tiene tal número, es decir, si `seq[n] = 1`, entonces `n` está en fibonacci. Por lo que constantemente sumamos desde los valores 0 y 1 en `p2` y `p1`, respectivamente, guardando el resultado como el valor 1. Luego, simplemente se accede a ese bit con el número actual, y si es 0, entonces no es parte de la secuencia.

Para el testbench, se usó:

```
‘timescale 1ns/1ns

module CounterRun;
        logic clk;
        reg reset;

        wire [4:0] counter;

        Counter dut(clk, reset, counter);

        initial begin
                clk = 0;
                reset = 1; #2;
                reset = 0;
        end

        always begin
                clk <= 0; #2;
                clk <= 1; #2;
        end
endmodule
```

La siguiente imágen, es lo que se obtiene analizando los elementos `clk`, `counter` y `fib`.

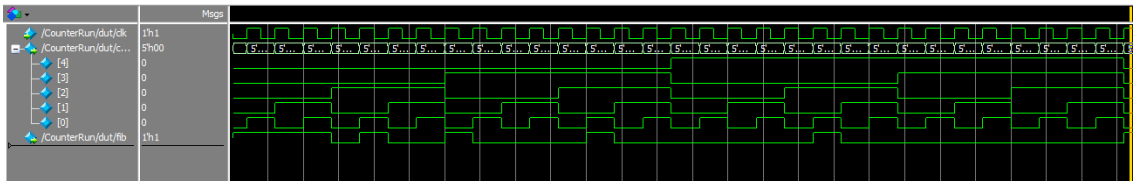

Figure 1: Simulación del modelo usado, en ModelSim