INF245 - Arquitectura y Organización de computadores
Preinforme Laboratorio Experiencia 3

Diego Altamirano T
201773017-K

16 de Agosto, 2019

1 Conceptos

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

Los lenguajes de descripción de hardware se especializan en la descripción de una estructura, diseño y funcionamiento de un circuito electrónico (por lo general digitales). Estos lenguajes permiten acercar la alta complejidad de los circuitos modernos a un lenguaje estandar, el cual puede ser compilado y simulado.

Las grandes diferencias entre estos y los lenguajes de programación son, primero su fin, dado que, como fue recién dicho, los HDL se especializan en descripción de componentes electrónicos en un circuito, y segundo, los HDL incluyen en sus reglas la sintaxis de tiempo.

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

- **wire:** Las variables *wire* actúan como un cable físico, tal que pueden ser leidas y modificadas, pero no es como si se almacenara un valor, si no que estas "tendrán un valor asociado" solo bajo asignamiento continuo.
- **reg:** Representan elementos de almacenaje en SystemVerilog. Su diferencia con las variables *wire* que una ves se les asigne un valor, estos lo retendrán hasta que se le asigne otro.
- **logic:** Superficialmente, alacenan bits igual que los demás. Su gran diferencia es como lidian con múltiples asignaciones; utilizan la lógica de "la última asignación gana", mientras que otras variables si se les hace más de una asignación diferente quedan en un valor X, las variables *logic* se afirman de la última asignación que se les dió.

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

- <=: Son asignaciones que se evalúan en paralelo una ves son ejecutadas, pero no se les puede predecir en que orden lo harán. Se colocan dentro de secciones *always*.

- **assign =**: Son asignaciones continuas, o sea que si el valor asignado se modifica, también lo hará la variable asignada a este. Note que estas asignaciones solo son posibles fuera de un *always*.
- **=**: Es como la antagonista de las asignaciones <=, dado que fuerzan el orden secuencial entre ellas. Estas solo pueden darse dentro de modulos *always*.

d) Dada la siguiente expresión:

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

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

Vamos por partes (dijo el descuartizador):

- **reg**: Define que será una variable registro.
- **[15:0] g**: Con un tamaño de 16 bits, se le asignará a la variable *g*.
- **=**: Será una asignación que forzará secuencialidad.
- **16'h**: Define que el número acontinuación tendrá 16 bits y estará escrito en sistema *hexadecimal*.
- **A6B2**: Es el número respectivo, de valor decimal 42674.

Desde acá, al transformar a binario, nos queda que:

$$A6B2_{16} = 1010011010110010_2$$

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 porque el delay y las asignaciones dentro del bloque always ff influyen en este resultado.

Las asignaciones son las que juegan el gran rol en el módulo. Con un ruteo normal uno caería en la conclusión que arr[first] debería quedar con el valor 0, el principal fundamento de por qué esto no funciona es el funcionamiento del *always_ff*.

Como este tipo de always busca imitar el funcionamiento de un flip-flop, también "imita" la acción de entregar toda su información al mismo tiempo, por lo que cuando se ejecuta la asignación *first = 1*, aunque se haga con un "=", la siguiente operación no tendrá delay suficiente para hacerce con el cambio, por lo que llevará a que el registro *arr* quede con el valor binario 0111.

2 Análisis de Código

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;
wire [2:0]m;
assign n = (x+y)%256;
assign m = (x+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
```

El ruteo queda como (dado un $r = 3'b0$):

clk	n
0	0x77
1	0x76
0	0x76
1	0x74
0	0x74
1	0x74
0	0x74
1	0x7e
0	0x7e
1	0x6e
0	0x6e
1	0x4e
0	0x4e
1	0x0e
0	0x0e
1	0x16
0	0x16
1	0x16
0	0x16
1	0x16
0	0x16

3 Diseño de modulo

El código utilizado es:

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

    logic [4:0] prim;
    logic [4:0] seg;

    initial begin
       out <= 0;
       prim <= 5'b00001;
       seg <= 5'b00000;
       flag <= 1;
    end
    always @(posedge clk) begin
        out = out + 1;
        if (prim+seg == out) begin
          seg <= prim;
          prim <= out;
          flag <= 1;
        end
        else begin
            flag <= 0;
        end
    end
    endmodule
```

Y sus simulaciones fueron: