Pre-Informe 3
Experiencia 3

Damian Ascui
201511020-4

Investigacion de Conceptos

1.-Un lenguaje de descripción de hardware (HDL, hardware description language) es un lenguaje de programación especializado que se utiliza para definir la estructura, diseño y operación de circuitos electrónicos, y más comúnmente, de circuitos electrónicos digitales, como el convertidor analógico-digital o cualquier antena satelital. Así, los lenguajes de descripción de hardware hacen posible una descripción formal de un circuito electrónico, y posibilitan su análisis automático y su simulación.

Los lenguajes de descripción de hardware se parecen mucho a otros lenguajes de programación de ordenadores tales como el C o Java: básicamente consisten en una descripción textual con expresiones, declaraciones y estructuras de control. Sin embargo, una importante diferencia entre los HDL y otros lenguajes de programación está en que el HDL incluye explícitamente la noción de tiempo.

2.-Tanto Wire,Reg y Logic son utilizados para almacenar una variable, en el caso de wire esta es usada para "Continuous Assignment" es decir para entradas que se iran revisando continuamente y que ademas usaran resultados intermedios para llegar al output final, por ejemplo, un conjunto de diferentes puertas, que se unan en una salida final, los pasos intermedios de estas corresponderan a los wires, ademas de esto son usados en "Structural Assignment" donde se general las puertas una por una, en lugar de usar la expresion boleana equivalente.

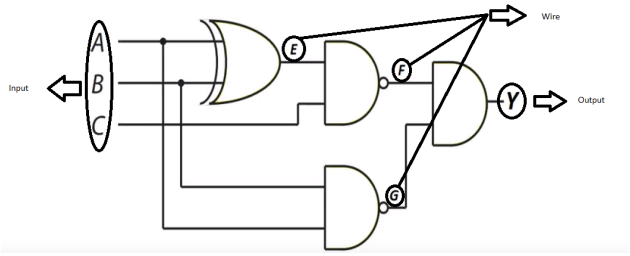

Figure 1: Ejemplo Wire

Reg se utiliza para los "Blocking Procedural Assignment" o "Not Blocking Procedural Assignment" estos tienen que ir siempre con un 'always@(condition)' o un 'initial' ya que se actualizaran en el momento que se cumpla la condicion o de inicio la secuencia.

Combinatorial Logic	Sequential Logic

reg a,b,c;
 	
always @(b or c)
 	
begin
 	
a = b | c;
 	
end
	reg a,b,c;
	
	always @(posedge c)
	
	begin
	
	a = a + b;
	
	end

Figure 2: Ejemplo Reg

Finalmente Logic es un tipo de data adicional que fue agregado para extender las funcionalidades de reg, por lo que este funciona en puertas al igual que wire, siendo esta versativilidad su principal diferencia entre los sistemas de almacenaje anteriormente mencionandos, ya que se puede utilizar tanto en Continuous Assignment como en Procedural Assignment.

3.-(assign) es usado en los "Continuous Assignment" y corresponde a una expresion boleana con inputs y output que sera envaluada constantemente, en estos es donde regularmente se utilizan wires, ademas cabe destacar que en versiones modernas de assign estos son transformados en always@* que seran revisados cuando alguno de los valores dentro de los inputs cambien con el objetivo de acelerar la simulacion.

(¡=) es usado en "Not Blocking Procedural Assignment", corresponde a un igual pero que se realizara en paralelo con el resto de procedimientos que tengan este signo, cada delay que se quiera utilizar debe ir despues del signo y no afectara a las siguientes lineas de codigo.

(=)es usado en "Blocking Procedural Assignment", al igual que el anterior es un igual pero este se ejecuta en el orden en que se escriba por los que los delays que se incoporen se acumularan orden por orden.

4.- 5.- el momento antes de la subida del clk el estado de first y second es 0 y 1 respectivamente, ya que el

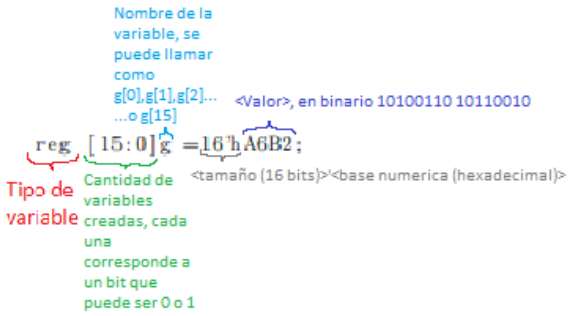

Figure 3: Explicacion de la expresion

cambio de valor no esta blockeado , al momento en que se activa el clk , first y second son anbos uno, ya que ¡= es non blocking en ese mismo instante se realiza el cambio de variable por lo que se cambia arr[1] por (arr[1]), resultando en 0111

Analisis de Codigo

En la tabla se muestra los valores de cada variable tanto en decimal como en binario

CLK	r	x	y	z	s	m	n
0	1	0	119	0	0	1	119
1	1	0	117	2	1	3	117
0	1	0	117	2	1	3	117
1	1	0	117	3	1	4	117
0	1	0	117	3	1	4	117
1	1	16	117	4	1	5	133
0	1	16	117	4	1	5	133
1	1	16	117	5	1	6	133
0	1	16	117	5	1	6	133
1	1	16	85	6	1	7	101

CLK	r	x	y	z	s	m	n
0	1	0	1110111	0	0	1	1110111
1	1	0	1110101	10	1	11	1110101
0	1	0	1110101	10	1	11	1110101
1	1	0	1110101	11	1	100	1110101
0	1	0	1110101	11	1	100	1110101
1	1	10000	1110101	100	1	101	10000101
0	1	10000	1110101	100	1	101	10000101
1	1	10000	1110101	101	1	110	10000101
0	1	10000	1110101	101	1	110	10000101
1	1	10000	1010101	110	1	111	1100101

Figure 4: Ruteo de valores con r=1

Diseño de Modulo

```
Ln#  1/3 Driver Lines
1  module up_counter(input clk, reset, output[4:0] counter
2      );
3  reg [4:0] counter_up;
4
5  // up counter
6  always @(posedge clk or posedge reset)
7  begin
8  if(reset)
9   counter_up <= 5'b0;
10 else
11  counter_up <= counter_up + 5'b1;
12 end
13 assign counter = counter_up;
14 endmodule
```

```
Ln#  1/3 Driver Lines
1  module upcounter_testbench();
2  reg clk, reset,bool;
3  wire [4:0] counter;
4  up_counter dut(clk, reset, counter);
5  initial begin
6  clk=0;
7  forever #5 clk=~clk;
8  end
9  initial begin
10 reset=1;
11 #20;
12 reset=0;
13 bool=1'b1;
14 #35 bool=1'b0;
15 #10 bool=1'b1;
16 #10 bool=1'b0;
17 #20 bool=1'b1;
18 #10 bool=1'b0;
19 #40 bool=1'b1;
20 #10 bool=1'b0;
21 #70 bool=1'b1;
22 #10 bool=1'b0;
23 #100 bool=1'b1;
24 end
25 endmodule
26
```

Figure 5: codigo

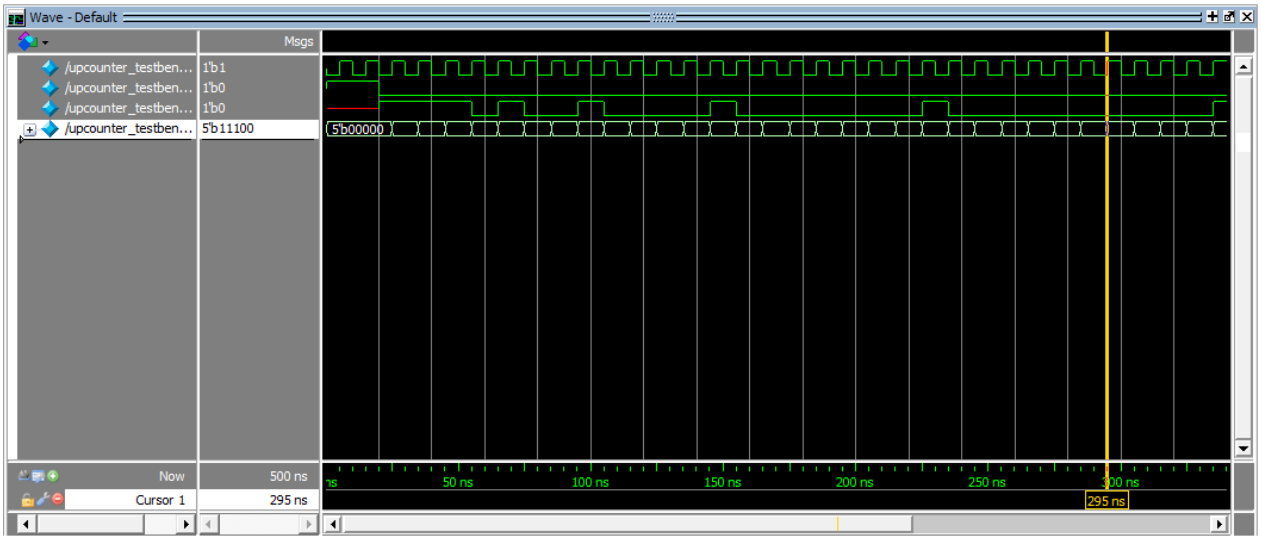

Figure 6: Wave