Pre-informe 3
Arquitectura de computadores

Nicolás Oporto
201773107-9

16 de agosto de 2019

1. Investigación de Conceptos

Solución:
a) 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. La diferencia entre el HDL con otros lenguajes de programación es que el HDL incluye explicitamente la noción de tiempo y que es exclusiva para operar componentes electronicos mientras que loslenguajes de programación como tal están enfocados en la creación de algoritmos que puedan ser interpretados por un computador.

b)
-wire: Representan conexiones estructurales entre componentes, no tienen capacidad de almacenamiento de información debido a que sirven para conectar elementos.
-reg: Representan variables con capacidad de almacenar información.
-logic: A diferencia de los elementos anteriores que también están presentes en Verilog, este elemento hizo su debut en Systemverilog, puede ser usado como un wire o como un reg.

c)
assign: Usado lógica combinacional, puede funcionar como un wire y cabe destacar que deben ser declaradas fuera de cualquier bloque (proceso).
¡=: Usado en lógica secuencial, en bloques always se ejecuta en paralelo a las otras líneas de codigo.
=: Usado en lógica combinacional, en bloques always la línea de codigo se ejecutará despues de que se haya ejecutado la línea anterior, es decir, se ejecuta una linea tras otra.

d) reg [15:0]g= h'A6B2;
reg: Tipo de variable (asignación continua)
[15 : 0]: indica el tamaño en bits (16 bits).
g: Nombre de variable
=: Asignación que indica un traspaso de valor inmediato.
16'hA6B2: Corresponde al número hexadecimal
1010 0110 1011 0010 (tamaño 16 bits)

e) Debido a que dentro del $always_ff$ se asigna a first el valor 1 (y second está asociado a first), y como second es un wire no va a cambiar hasta que termine el proceso entero del flip flop, por tanto va a mantener el valor hasta que termine el bloque always, lo que genera el delay.

2. Análisis de Código

Estado inicial:
x= 00000000
y= 01110111

z= 000
n= 01110111
m=001
y[m]=1
y[z]=1
Subida CLK 1:
x=00000000
y=01110110
z=001
n=01110110
m=010
y[m]=1
y[z]=1
Subida CLK 2:
x=00000000
y=01110100
z=010
n=01110100
m=011
y[m]=1
y[z]= 1
Subida CLK 3:
x=00000000
y=01110100
z=011
n=01110100
m=100
y[m]=0
y[z]=1
Subida CLK 4:
x=00010000
y=01110100
z=100
n=10000100
m=101
y[m]=1
y[z]=0
Subida CLK 5:
x=00010000
y=01100100
z=101
n=01110100
m=110
y[m]=1
y[z]=1
Subida CLK 6:
x=00010000
y=01000100
z=110
n=01010100
m=111
y[m]= 1
y[z]=1
Subida CLK 7:
x=00010000

y=01000100
z=111
n=01010100
m=000
y[m]=0
y[z]= 1
Subida CLK 8:
x=00010001
y=11000100
z=000
n=11010101
m=001
y[m]=0
y[z]=1
Subida CLK 9:
x=00010011
y=11000101
z=001
n=11011000
m=010
y[m]=0
y[z]=1
Subida CLK 10:
x=00010111
y=11000101
z=010
n=11011100
m=011
y[m]=1
y[z]=0

3. Diseño de Módulo

Solución:
Para comenzar el diseño del modulo usaremos mapas de Karnaugh para saber cuando un numero está dentro de la serie de Fibonacci: c[0]= E, c[1]= D, c[2]=C, c[3]=B, c[4]=A
fibo= $\sum$(0,1,2,3,5,8,13,21)

	$\overline{DE}$	$\bar{D}E$	DE	$D\bar{E}$
$\overline{ABC}$	1	1	1	1
$\overline{A\bar{B}C}$	0	1	0	0
$\bar{A}BC$	0	1	0	0
$\bar{A}B\bar{C}$	1	0	0	0
$A\overline{BC}$	0	0	0	0
$A\bar{B}C$	0	1	0	0
ABC	0	0	0	0
$AB\bar{C}$	0	0	0	0

Figura 1: Mapa de Kaunaugh

Por lo tanto fibo=$\overline{A}$ $\overline{B}$ $\overline{C}$+$\overline{A}$ $\overline{C}$ $\overline{D}$ $\overline{E}$+$\overline{A}$ C $\overline{D}$ E+$\overline{B}$ C $\overline{D}$ E
-Ahora adjuntar codigo del contador de 5 bits, codigo del testbench y pantallazo de la simulación:

C:/ModelSim/examples/fibonacci.sv - Default

```
Ln#
 1  module fiboc(clk, c, fibo);
 2          input logic clk;
 3          output logic[4:0] c;
 4          output logic fibo;
 5          reg[4:0]inicial=0;
 6          assign c=inicial;
 7          assign fibo= ((~c[4]) & (~c[4]) & (~c[4])) |
 8                       ((~c[4]) & (~c[4]) & (~c[4]) & (~c[4])) |
 9                       ((~c[4]) & (~c[4]) & (~c[4]) & (~c[4])) |
10                       ( (~c[4]) & (~c[4]) & (~c[4]) & (~c[4]));
11          always_ff@(posedge clk) begin
12                  inicial <= inicial+1'b1;
13          end
14  endmodule
15
```

Figura 2: Fibonacci

C:/ModelSim/examples/testbench.sv - Default *

```
Ln#
 1  module test;
 2          logic clk;
 3          fiboc test(clk);
 4          initial begin
 5                  forever begin
 6                  #10 clk=1;
 7                  #10 clk=1;
 8                  end
 9          end
10  endmodule
11
```

Figura 3: Testbench

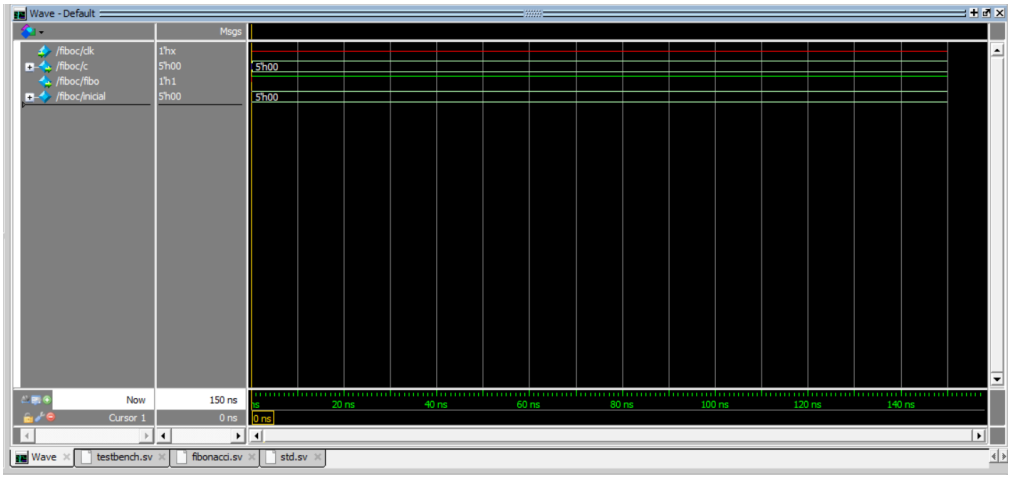

Figura 4: Simulación