Carlos Gil 19443

Ejercicio # 1:

```
module contador (input wire clock, reset, load, en, input wire [11:0]load_1, output reg [11:0]count);

always @ (posedge clock or posedge reset or load or load_1) begin

if (reset) begin

count <= 12'b0000000000000;

end

else if (load) begin

count <= load_1;

end

else if (en) begin

count <= count + 12'b00000000001;

end

end

end
```

El codigo mostrado anteriormente funciona de la siguiente manera: Lo que está dentro del always va a suceder siempre que haya un flanco positivo del clock, un flanco positivo del reset, un cambio en el load o un cambio en la variable load_1. Dentro de las condiciones, describo que cuando el reset = 1, el valor del contador sea 0, cuando el load =1, el valor precargado en load_1 pase a ser el valor de contador y el contador únicamente incrementa cuando en = 1. En el caso que alguna de estas condiciones no se cumpla el valor de contador mantendrá su valor anterior, en este caso no se utilizo un "else" ya que el valor del contador es de tipo reg y si no se le hacen cambios, guarda su valor anterior.

simulación del contador:

C:\Windows\System32\cmd.exe

C:\Users\Carlos Gil\Desktop\Universida				
VCD info: dumpfile contador.vcd opened				
contador				
clk en load load_1	l count			
1 x x xxxxxxxxxxxx	0000000000000			
0 x x xxxxxxxxxxxx	000000000000			
1 x x xxxxxxxxxxxxx	000000000000			
0 x x xxxxxxxxxxxx	000000000000			
1 0 0 0000000000000	000000000000			
0 0 0 000000000000	000000000000			
1 0 0 0000000000000	000000000000			
0 0 0 000000000000000000000000000000000	000000000000			
1 0 0 0000000000000	000000000000			
0 0 0 000000000000	000000000000			
	000000000000			
	000000100000			
1 1 1 0000000000000	000000000000			
0 1 1 010010000000	010010000000			
1 1 1 010010000000	010010000000			
0 1 1 010010000000	010010000000			
1 1 0 010010000000	010010000010			
0 1 0 010010000000	010010000010			
1 1 0 010010000000	010010000011			
0 1 0 010010000000	010010000011			
1 1 0 010010000000	010010000100			
0 1 0 010010000000	010010000100			
1 1 0 010010000000	010010000101			
0 1 0 010010000000	010010000101			
1 1 0 010010000000	010010000110			
0 1 0 010010000000	010010000110			
1 1 0 010010000000	010010000111			
0 1 0 010010000000	010010000111			
1 1 0 010010000000	010010001000			
0 1 0 010010000000	010010001000			
1 1 0 010010000000	010010001001			
0 0 0 010010000000	010010001001			
1 0 0 010010000000	010010001001			
0 0 0 010010000000	010010001001			
1 0 0 010010000000	010010001001			
0 0 0 010010000000	010010001001			
1 0 0 010010000000	010010001001			
0 0 0 010010000000	010010001001			
1 0 0 010010000000	010010001001			
0 0 0 010010000000	010010001001			
1 0 0 010010000000	010010001001			
0 0 0 010010000000	010010001001			
1 0 0 010010000000	010010001001			
0 0 0 010010000000	010010001001			
1 0 0 010010000000	010010001001			

Ejercicio # 2:

Array: Un arreglo en verilog básicamente consiste en una variable que tiene dos dimensiones, como se ve el ejemplo siguiente.

Ejemplo:

```
reg [7:0] array [0:4095]; // Este arreglo es de 4k x 8
```

\$readmemb: La función de este comando es asignarle leer los datos de un archivo escrito en binario y guardarlo en el arreglo creado en verilog.

\$readmemh: La función de este comando es asignarle leer los datos de un archivo escrito en hexadecimal y guardarlo en el arreglo creado en verilog.

MEMOKIA KOM	
ADRESS	PALABRA
000000000000	00000001
0000000000001	00000010
000001010101	01010101
000001010110	01010110
000001010111	01010111
000001011000	01011000
000010001000	10001000
000010001001	10001001
000010001010	10001010
000010001011	10001011

Ejercicio #3:

```
module ALU (input wire [3:0]A, B, input wire [2:0]S, output reg [3:0]resultado);
always @ (*) begin
case(S)

0:
resultado <= A & B;

1:
resultado <= A | B;

2:
resultado <= A + B;

3:
resultado <= 4'b0;

4:
resultado <= A & ~B;
```

```
5:
resultado <= A | ~B;
6:
resultado <= A - B;
7:
resultado <= (A < B) ? 4'b1111: 4'b0;
default:
resultado <= 4'b0;
endcase
end
endmodule
```

La forma en la que implemente la ALU fue con una serie de casos que describen las diferentes funciones que quiero que haga. Inicialmente estipule que lo que está dentro del always suceda siempre, luego acorde con el S ejecuto una operación especifica entre A y B y lo muestro en resultado. Así mismo, establece que la función por defecto es que el resultado sea 0.

ALU			
Α	В	S	resultado
0101	0000	000	0000
0101	0000	001	0101
0101	0001	010	0110
0101	0000	011	0000
0101	0000	100	0101
0101	0000	101	1111
0101	0001	110	0100
0101	0000	111	0000

Link de github: https://github.com/gil19443/Digital_1.git