Valerie Valdez
Universidad del Valle de Guatemala

Carné: 19659
Sección: 20

Electrónica digital
Laboratorio 8

Ejercicio 01: Contador

```
// Implementar un contador de 12 bits con un enable y capaz de hacer un load

module contador(input wire clk, rst, en, blo,
  input wire [11:0]load,
  output reg [11:0]val);

// Non-blocking assigment
    always @(posedge clk or posedge rst or posedge blo) begin

      if(rst == 1) // Colocar todos los valores en 0
        val <= 12'b000000000000;

      else if(en == 1) //Enabled encendido cuenta en cada flanco de reloj
        val <= val + 1;

      else if(blo == 1) // Blo encendido entonces Load comienza a contar
        val <= load;

        end
endmodule
```

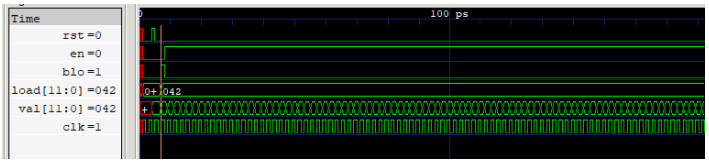
Testbench

```
module testbench();
  reg clk, rst, en, blo;
  reg [11:0]load;
  wire [11:0]val;

contador A1(clk, rst, en, blo, load, val);

initial begin
#1
$display("\n");
$display(" Contador de 12 bits ");
$display("\n");
$display("|-------------------------|-------------|");
$display("|clk  rst  en  blo  load  |     val     |");
$display("|-------------------------|-------------|");
$monitor("|%b  %b  %b  %b  %b | %b |", clk, rst, en, blo, load, val);


clk = 0; rst = 0; en = 0; blo = 0; load = 12'b000000000000;
#1 rst = 0;
#1 rst = 0;
#1 rst = 1;
#1 rst = 0;
#2 blo = 1; load = 12'b000001000010;
#1 blo = 0; en = 1;
#1 en = 1;

end

always
#1 clk = ~clk;

initial
#200 $finish;

initial begin
  $dumpfile("Contador_tb.vcd");
  $dumpvars(0, testbench);
end
endmodule
```

Tabla generada por el código

```
Contador de 12 bits


|-------------------------|-------------|
|clk  rst  en  blo  load  |     val     |
|-------------------------|-------------|
|1  0  0  0  000000000000 | xxxxxxxxxxxx |
|0  0  0  0  000000000000 | xxxxxxxxxxxx |
|1  0  0  0  000000000000 | xxxxxxxxxxxx |
|0  1  0  0  000000000000 | 000000000000 |
|1  0  0  0  000000000000 | 000000000000 |
|0  0  0  0  000000000000 | 000000000000 |
|1  0  0  1  000001000010 | 000001000010 |
|0  0  1  0  000001000010 | 000001000010 |
|1  0  1  0  000001000010 | 000001000011 |
|0  0  1  0  000001000010 | 000001000011 |
|1  0  1  0  000001000010 | 000001000100 |
|0  0  1  0  000001000010 | 000001000100 |
|1  0  1  0  000001000010 | 000001000101 |
|0  0  1  0  000001000010 | 000001000101 |
|1  0  1  0  000001000010 | 000001000110 |
|0  0  1  0  000001000010 | 000001000110 |
|1  0  1  0  000001000010 | 000001000111 |
|0  0  1  0  000001000010 | 000001000111 |
|1  0  1  0  000001000010 | 000001001000 |
|0  0  1  0  000001000010 | 000001001000 |
|1  0  1  0  000001000010 | 000001001001 |
|0  0  1  0  000001000010 | 000001001001 |
|1  0  1  0  000001000010 | 000001001010 |
|0  0  1  0  000001000010 | 000001001010 |
|1  0  1  0  000001000010 | 000001001011 |
|0  0  1  0  000001000010 | 000001001011 |
|1  0  1  0  000001000010 | 000001001100 |
|0  0  1  0  000001000010 | 000001001100 |
|1  0  1  0  000001000010 | 000001001101 |
|0  0  1  0  000001000010 | 000001001101 |
|1  0  1  0  000001000010 | 000001001110 |
|0  0  1  0  000001000010 | 000001001110 |
|1  0  1  0  000001000010 | 000001001111 |
```

Diagrama de Timing



Para realizar el módulo del contador se utilizó un bus de 12 bits para almacenar los valores y uno para establecer hasta cuánto llegaría el load, el cual era controlado por un bit (blo). Por otro lado, se utilizó el *non-blocking assigment* para que todas las condicionales se comprueben en paralelo. Para finalizar, en el

módulo del *testbench* se realizaron distintas pruebas con el fin de probar todas las funciones del contador (Contar, cargar, contar después de cargar, no contar).

Ejercicio 02: Memoria ROM

```
// Implementación de la memoria ROM de 4Kx8
module Mrom(input wire [11:0]address,
  output wire [7:0]D); // Tamaño del dato de 8 bits

  reg[7:0] Mrom[0:4095]; // Ancho y localidades de memoria

// Inicializar la memoria
  initial begin
    $readmemb("Mrom.list", Mrom); // Leer los valores de la lista en binario
  end

  assign D = Mrom[address]; // Asignarle el valor en la dirección dada
endmodule
```

Testbench

```
// Módulo para la memoria ROM
module testbench();

reg [11:0]address; // Dirección de almacenamiento
wire [7:0]D;// Variable de 7 bits para el módulo

//Llamar al módulo
Mrom A1(address, D);

initial begin
#1
$display("\n");
$display(" Memoria ROM ");
$display("\n");
$display("|----------------|------------|");
$display("|    address     |    Dato    |");
$display("|----------------|------------|");
$monitor("|  %b  |  %b  |", address, D);

// Se declaran las 10 diferentes direcciones
address = 12'b000000000000;
  #1 address = 12'b000000000001;
  #1 address = 12'b000000000010;
  #1 address = 12'b000000000011;
  #1 address = 12'b000000000100;
  #1 address = 12'b000000000101;
  #1 address = 12'b000000000110;
  #1 address = 12'b000000000111;
  #1 address = 12'b000000001000;
  #1 address = 12'b000000001001;
  #1 address = 12'b000000001010;
  end
initial
#200 $finish;

//GTKwave
initial begin
  $dumpfile("Memoria_rom_tb.vcd");
  $dumpvars(0, testbench);
end
endmodule
```

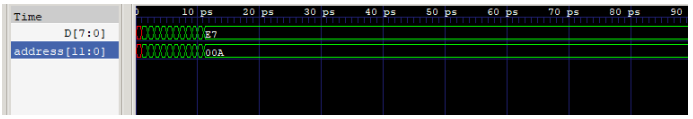
Lista de valores

```
0000_0000
1111_0000
0011_1010
1110_1100
0110_0000
0111_1000
0110_0110
1100_0110
0010_1110
1110_0100
1110_0111
```

Tabla generada por el código

address	Dato
000000000000	00000000
000000000001	11110000
000000000010	00111010
000000000011	11101100
000000000100	01100000
000000000101	01111000
000000000110	01100110
000000000111	11000110
000000001000	00101110
000000001001	11100100
000000001010	11100111

Diagrama de Timing



Un *array* de datos son matrices utilizadas para agrupar elementos a objetos multidimensionales para que puedan ser manipulados con facilidad. Por cada *array* es necesario establecer la dimensión por lo que el valor mínimo y máximo se declaran entre corchetes y los índices de cada matriz se pueden escribir en cualquier dirección.

Las instrucciones $*readmemb* y $*readmemh* se utilizan para leer los datos almacenados en binario y hexagesimal respectivamente.

Para realizar el código se utilizó un bus de 12 bits para la entrada (*address*) la cual indica la dirección de almacenamiento y una salida de 8 bits la cual representa el dato guardado en la lista creada y almacenada en una localidad específica. La idea es tener 10 datos

almacenados en una lista externa al código, y con $*readmem*b leer cada dato los cuales fueron asignados en una localidad específica en el *testbench.*

Ejercicio No. 3: ALU

```
// Módulo del la ALU
module ALU(input wire [3:0]D, Q,
           input wire [2:0]S,
           output reg [3:0]Y);

    always @ (D, Q, S) begin //Registrar los cambios realizados en D o Q
      case (S)
        3'b000: Y <= D & Q ; // Función: D AND Q
        3'b001: Y <= D | Q ; // Función: D OR Q
        3'b010: Y <= D + Q ; // Función: D ADD Q

        3'b011: Y <= Y ; // Colocar el valor pervio

        3'b100: Y <= D & (~Q) ; // Función D AND ~Q
        3'b101: Y <= D | (~Q) ; // Función D OR ~Q
        3'b110: Y <= D - Q ; // Función D LESS Q
        3'b111: Y <= (D < Q) ? 1:0 ; // Función STL. (Comparación entre D

        default: Y <= 4'b0000 ;// Todo en 0
    endcase
  end
endmodule
```

Testbench

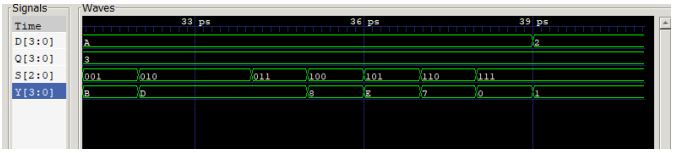
```
//Módulo
module testbench();

  reg [3:0]D, Q; // Input de la ALU
  reg [2:0]S; // Variable de selección
  wire [3:0]Y;// Salida de 4 bits

// Llamar al módulo
  ALU A3(D, Q, S, Y);

// Inicializar el módulo
  initial begin
  #30
  $display("\n");
  $display("  A L U  ");
  $display("\n");
  $display("|----------------|------|");
  $display("|  D    Q     S  |  Y   |");
  $display("|----------------|------|");
  $monitor("|%b  %b  %b | %b |", D, Q, S, Y);

// Funciones a realizar
  D = 4'b1010; Q = 4'b0011; S = 3'b000; //FUNCION  D AND Q
  #1 S = 3'b001; // Función D OR Q
  #1 S = 3'b010; // Función D ADD Q
  #1 S = 3'b010; // Función D ADD Q

  #1 S = 3'b011; //Colocar el valor previo

  #1 S = 3'b100; // Función D AND ~Q
  #1 S = 3'b101; // Función D OR ~Q
  #1 S = 3'b110; // Función D LESS Q
  #1 S = 3'b111; // Función STL
  #1 D = 4'b0010; Q = 4'b0011; // Colocar todo en 0

  #2 $finish;
  end
```

```
//GTKwave
  initial begin
        $dumpfile("alu_tb.vcd");
        $dumpvars(0, testbench);
      end

endmodule
```

Tabla generada por el código

```
A L U

|----------------|------|
|  D    Q     S  |  Y   |
|----------------|------|
|1010  0011  000 | 0010 |
|1010  0011  001 | 1011 |
|1010  0011  010 | 1101 |
|1010  0011  011 | 1101 |
|1010  0011  100 | 1000 |
|1010  0011  101 | 1110 |
|1010  0011  110 | 0111 |
|1010  0011  111 | 0000 |
|0010  0011  111 | 0001 |
```

Diagrama de Timing



La ALU combina varias operaciones lógicas y matemáticas en una unidad. Por lo que, para implementar la ALU se creó un modulo que incluye las operaciones lógicas y comparaciones las cuales operan dentro de un always, el cual permite que si existen cambios en las variables dependientes de las funciones se registren simultáneamente.

Link:
https://github.com/valeelorraine/Laboratorios_Digital