Nombre: José Alejandro Rodríguez Porras Carné: 19131 Fecha: 24/10/2020

Laboratorio 8

Ejercicio 1

Contador de 12 bits con Enabled, Load y Reset

El contador en cuestión tiene una capacidad de 12 bits, es decir puede contar hasta 4095, y cuenta con las funciones de Enabled, Load y Reset. El código se implementó de manera que el load y reset funcionaran de forma asíncrona, es decir que no necesitaran un flanco de reloj positivo para que el cambio se detectara y tuviera efecto en la salida del contador. Si se activaba el reset, la salida del contador vuelve a todos sus bits en 0, mientras que si se activa el load, la salida del contador toma el valor del input Loadvalue en ese momento, y se queda en ese valor mientras no se desactive el load. Una vez se desactiva el load, el contador vuelve a contar desde el valor impuesto por el previo load. Por último, el contador solo cuenta cuando el Enabled está encendido. La cuenta es de 1 en 1, y en el código se implementó de forma que se sumara uno al valor previo del counter, de esta forma actualizándolo. Cuando el contador llega a 4095 la siguiente cuenta son todos los bits en 0.

E1.v | E3.v | E1_tb.v | E2.v | E2_tb.v

E1 > E1.v

```
1  //José Alejandro Rodríguez Porras 19131
2  //Electrónica Digital 1
3  //Lab 8 Ejercicio 1
4  //Contador de 12 bits, con Load, Enabled y Reset
5  module up_counter(input clk, rst, Load, Enabled, input [11:0]Loadvalue, output [11:0]counter);
6  reg [11:0] counter_up;
7  // Load y reset asíncronos
8  always @(posedge clk or posedge Load or posedge rst)
9  // El reset pone todos los bits del contador en 0 de forma asíncrona
10 begin
11 if (rst)
12     counter_up <= 12'b000000000000;
13 // El Load asigna un valor específico(Loadvalue) al contador mientras Load sea igual a 1
14 else if(Load)
15     counter_up <= Loadvalue;
16 //Mientras Enable sea igual a 1 el contador va sumando uno al valor previo, "va contando"
17 else begin
18 //Si el contador llega a su valor máximo, es decir 4095 el siguiente valor que se le asigna es 0
19 if (Enabled == 1 & counter_up == 12'b111111111111)
20     counter_up = 12'd0;
21 //El contador cuenta
22 else if (Enabled == 1)
23     counter_up <= counter_up + 12'd1;
24 end
25 end
26 //Se asigna el valor del contador a la salida
27 assign counter = counter_up;
28 endmodule
```

Código en Verilog (Módulo de contador

E1_tb.v | E1.v | E3.v | E2.v | E2_tb.v

E1 > E1_tb.v

```
1   //José Alejandro Rodríguez Porras
2   //Electrónica Digital 1
3   //Lab 8 Ejercicio 1 testbench
4   module testbench();
5   reg clk, rst, Load, Enabled;
6   reg [11:0]Loadvalue;
7   wire [11:0]counter;
8
9   up_counter count(clk, rst, Load, Enabled, Loadvalue, counter);
10  initial begin
11  forever #5 clk=~clk;
12  end
13  initial begin
14  $display("Load Enabled Loadvalue | counter ");
15  $display("----------------------|---------");
16  $monitor("%d      %d      %d     || %d", Load, Enabled, Loadvalue, counter);
17  //Valores iniciales de los inputs, el clock y el rst
18  clk=0; rst = 0; Load = 0;  Enabled = 0; Loadvalue = 12'b000000000000;
19  //Prueba de que el contador no cuenta si el Enabled es 0
20  #1  rst = 1;
21  #1  rst = 0;
22  //Prueba de que el contador comienza a contar cuanto se el Enabled es 1
23  #40  Enabled = 1;
24  //Prueba de una carga de un valor al contador de forma asíncrona cuando el Load es 1
25  #100 Load = 1; Loadvalue = 12'b000000011010;
26  //La salida del contador es el valor cargado por el Load mientras el Load se quede en 1
27  // Cuando el Load es 0 el contador comienza a contar desde el valor asignado cuando se activó el Load
28  #100 Load = 0;
29  //Prueba de que el contador deja de contar cuando el Enabled es 0.
30  #100 Enabled = 0;
31  // Se carga el valor de 4094 con el Load al contador
32  #100 Enabled = 1; Loadvalue = 12'b111111111110;
33  #60 Load = 1;
34  //Prueba de que el contador vuelve todos sus bits a 0 luego de llegar a su valor máximo (4095)
35  #10  Load = 0;
36  //Prueba del reset asíncrono
37  #100 rst = 1;
38  #1   rst = 0;
39  //final de la simulación
40  #40 $finish;
41  end
42  //creación del archivo vcd  (diagrama de timing)
43  initial begin
44          $dumpfile("E1_tb.vcd");
45          $dumpvars(0,testbench);
46      end
47  endmodule
```

Código del testbench de contador

```
---> WARNING: no PCF file found (.pcf)

iverilog -o E1_tb.out -D VCD_OUTPUT=E1_tb C:/Use
vvp E1_tb.out
Load Enabled Loadvalue | counter
-----------------------|---------
VCD info: dumpfile E1_tb.vcd opened for output.
0      0         0     |    x
0      0         0     |    0
0      1         0     |    0
0      1         0     |    1
0      1         0     |    2
0      1         0     |    3
0      1         0     |    4
0      1         0     |    5
0      1         0     |    6
0      1         0     |    7
0      1         0     |    8
0      1         0     |    9
0      1         0     |   10
1      1        26     |   26
0      1        26     |   26
0      1        26     |   27
0      1        26     |   28
0      1        26     |   29
0      1        26     |   30
0      1        26     |   31
0      1        26     |   32
0      1        26     |   33
0      1        26     |   34
0      1        26     |   35
0      1        26     |   36
0      0        26     |   36
0      1      4094     |   36
0      1      4094     |   37
0      1      4094     |   38
0      1      4094     |   39
0      1      4094     |   40
0      1      4094     |   41
0      1      4094     |   42
1      1      4094     | 4094
0      1      4094     | 4094
0      1      4094     | 4095
0      1      4094     |    0
0      1      4094     |    1
0      1      4094     |    2
0      1      4094     |    3
0      1      4094     |    4
0      1      4094     |    5
0      1      4094     |    6
0      1      4094     |    7
0      1      4094     |    8
0      1      4094     |    0
0      1      4094     |    1
0      1      4094     |    2
0      1      4094     |    3
0      1      4094     |    4
gtkwave E1_tb.vcd E1_tb.gtkw
```

Output del programa en la ventana de comandos

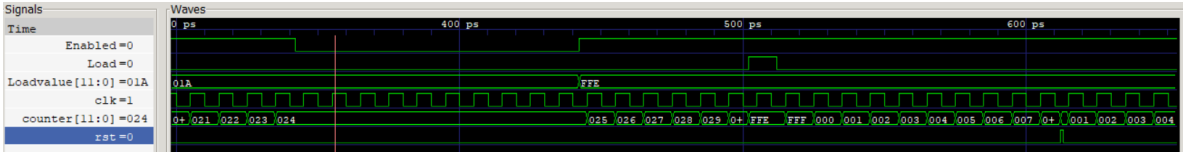


Diagrama de timing parte 1 del contador

.

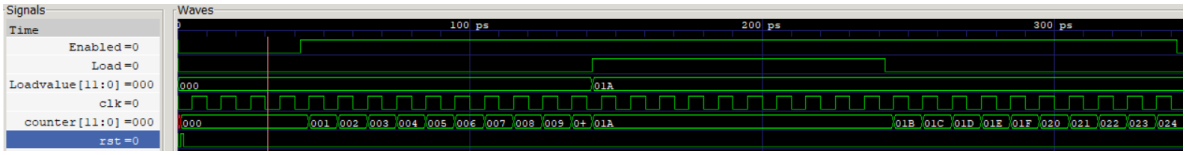
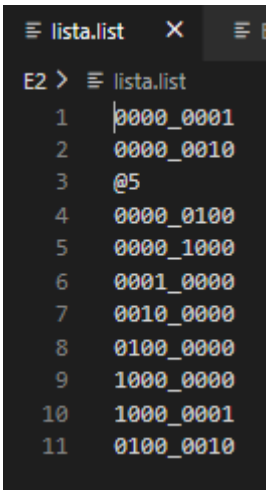


Diagrama de timing parte 2 del contador

Ejercicio 2

Memoria ROM de 4k x 8

Para implementar un array de datos en verilog primero se debe tener un archivo de datos desde el cual “leer” dicha información y agregarla a un arreglo que consta de una dirección (en este caso por tener 12 bits hay 4096 direcciones o localidades de memoria), y los cuales pueden guardar hasta 8 bits por fila. La instrucción $readmemb sirve para leer los datos guardados en el archivo lista, y cada fila de esta lista es guardada o asignada en una respectiva variable en el array. La diferencia entre $readmemb y $readmemh es la última letra, ya que cuando es b, esta representa que el dato está en binario, mientras que la letra h representa que el dato está en decimal.



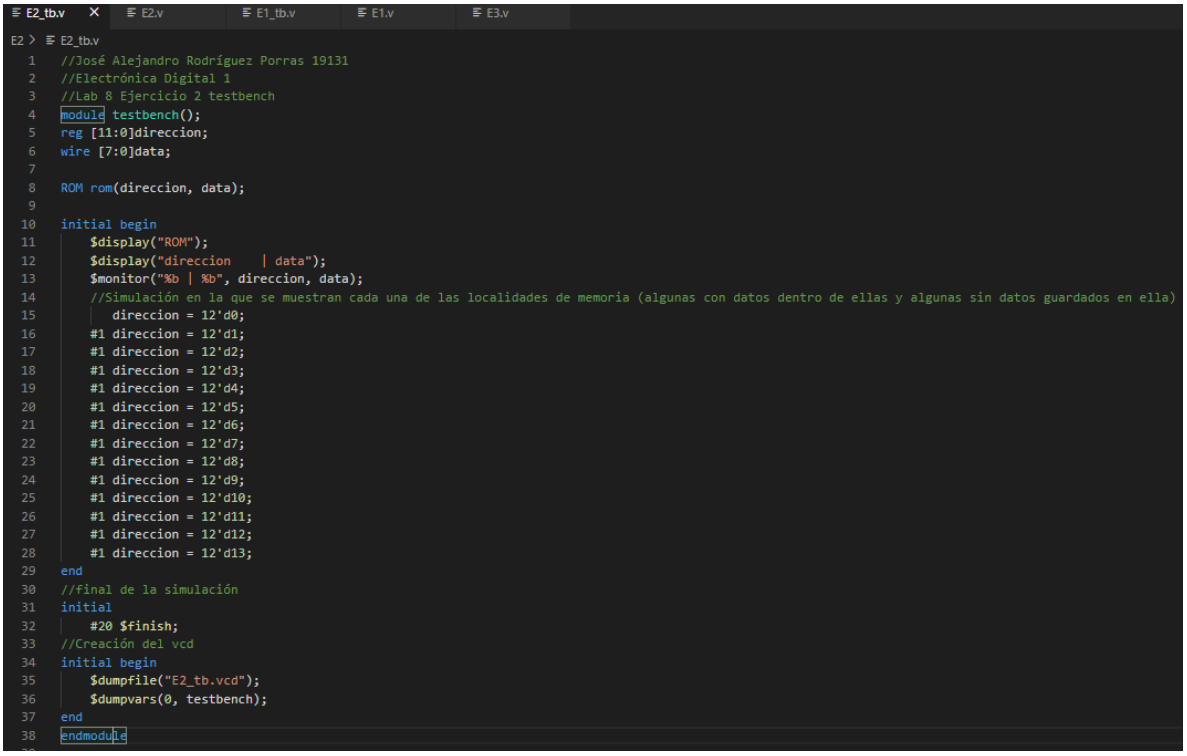
Archivo .list con datos

E2.v | E1_tb.v | E1.v | E3.v | E2_tb.v

E2 > E2.v

```v
//José Alejandro Rodríguez Porras 19131
//Electrónica Digital 1
//Lab 8 Ejercicio 2
//Memoria ROM de 4k x 8
//input es la dirección de memoria en la que se guarda el valor
//output es el dato que se almacena en la dirección
module ROM(input wire [11:0]direccion, output wire [7:0]data);
reg [7:0] memoria[0:4095];
initial begin
// readmemb lee el archivo lista y almacena sus datos en una variable
    $readmemb("lista.list", memoria);
end
//asignación del dato en la dirección de la memoria
assign data = memoria[direccion];
endmodule
```

Código en verilog de la memoria ROM



E2_tb.v | E2.v | E1_tb.v | E1.v | E3.v

E2 > E2_tb.v

```v
//José Alejandro Rodríguez Porras 19131
//Electrónica Digital 1
//Lab 8 Ejercicio 2 testbench
module testbench();
reg [11:0]direccion;
wire [7:0]data;

ROM rom(direccion, data);

initial begin
    $display("ROM");
    $display("direccion    | data");
    $monitor("%b | %b", direccion, data);
    //Simulación en la que se muestran cada una de las localidades de memoria (algunas con datos dentro de ellas y algunas sin datos guardados en ella)
       direccion = 12'd0;
    #1 direccion = 12'd1;
    #1 direccion = 12'd2;
    #1 direccion = 12'd3;
    #1 direccion = 12'd4;
    #1 direccion = 12'd5;
    #1 direccion = 12'd6;
    #1 direccion = 12'd7;
    #1 direccion = 12'd8;
    #1 direccion = 12'd9;
    #1 direccion = 12'd10;
    #1 direccion = 12'd11;
    #1 direccion = 12'd12;
    #1 direccion = 12'd13;
end
//final de la simulación
initial
    #20 $finish;
//Creación del vcd
initial begin
    $dumpfile("E2_tb.vcd");
    $dumpvars(0, testbench);
end
endmodule
```

Código del testbench de la memoria ROM

```
PS D:\AlejandroDigital\electronica_digital1\lab08dig\E2> apio sim

---> WARNING: no PCF file found (.pcf)

iverilog -o E2_tb.out -D VCD_OUTPUT=E2_tb C:/Users/AlejandroRodriguez
vvp E2_tb.out
ROM
direccion    | data
VCD info: dumpfile E2_tb.vcd opened for output.
000000000000 | 00000001
000000000001 | 00000010
000000000010 | xxxxxxxx
000000000011 | xxxxxxxx
000000000100 | xxxxxxxx
000000000101 | 00000100
000000000110 | 00001000
000000000111 | 00010000
000000001000 | 00100000
000000001001 | 01000000
000000001010 | 10000000
000000001011 | 10000001
000000001100 | 01000010
000000001101 | xxxxxxxx
gtkwave E2_tb.vcd E2_tb.gtkw
```

Output de la memoria ROM en la ventana de comandos

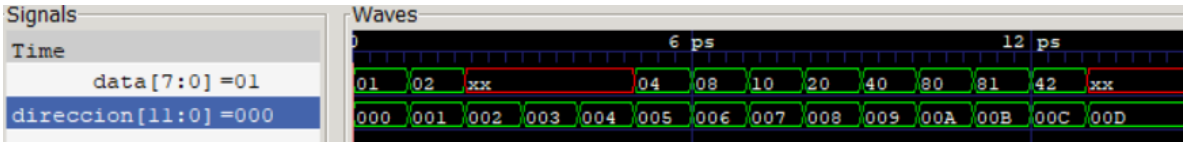


Diagrama de timing de la memoria ROM

Ejercicio 3

Unidad Lógica/Aritmética (ALU) de 4 bits

La ALU es una unidad que puede hacer varias operaciones con inputs en la misma, que devuelve el resultado de la operación elegida. Los inputs en este caso son de A y B, mientras que Y es la salida. Esta ALU no cuenta con carry in ni carry out. La ALU se implementó con un bloque always en el que cual usando un case dependiendo del código del operador, una operación específica (usando la forma behavioural en la programación) se realizaría con los inputs A y B dando un resultado de dicha operación. El operador “opcode” servía para elegir la operación que se deseaba realizar con A y B, ya fueran lógicas o aritméticas.

E3.v E2_tb.v E2.v E1_tb.v E1.v

E3 > E3.v

```v
//José Alejandro Rodríguez Porras 19131
//Electrónica Digital 1
//Lab 8 Ejercicio 3
//ALU
module ALU(input wire [3:0]A, input wire [3:0]B, input wire [2:0]opcode, output reg [3:0]Y);
    always @(opcode or A or B) begin
        //case para elegir cada uno de los modos de operación
        case(opcode)
        3'b000: Y <= A & B; //A AND B
        3'b001: Y <= A | B; // A OR B
        3'b010: Y <= A + B; // A + B
        3'b011: Y <= 4'b0000; // Sin uso
        3'b100: Y <= A & ~B; //A AND B'
        3'b101: Y <= A | ~B; //A OR B'
        3'b110: Y <= A - B; //A - B
        3'b111: Y <= (A < B) ? 1:0; // Y es 1 solo cuando A es menor que B
        //valor por defecto de salida
        default: Y <= 4'b0000;
        endcase
    end
endmodule
```

Código del módulo ALU en verilog

E3_tb.v

E3 > E3_tb.v

```
//José ALejandro Rodríguez Porras 19131
//Electrónica Digital 1
//Lab 8 Ejercicio 3 testbench
module testbench();
    reg [3:0]A;
    reg [3:0]B;
    reg [2:0]opcode;
    wire [3:0]Y;

ALU alu(A, B, opcode, Y);
//Comienzo de la simulación
initial begin
   #1 $display("A       B    opcode |    Y");
      $display("--------------------|-----------");
    $monitor("%b    %b    %b  |   %b", A, B, opcode, Y);
// Prueba A AND B
    A = 4'b0000; B = 4'b0000; opcode = 3'b000;
   #1 A = 4'b0001; B = 4'b0000; opcode = 3'b000;
   #1 A = 4'b0000; B = 4'b0001; opcode = 3'b000;
   #1 A = 4'b0001; B = 4'b0001; opcode = 3'b000;
   #1 A = 4'b0110; B = 4'b0110; opcode = 3'b000;
   #1 A = 4'b0011; B = 4'b0010; opcode = 3'b000;
//Prueba A OR B
   #1 A = 4'b0000; B = 4'b0000; opcode = 3'b001;
   #1 A = 4'b0000; B = 4'b0001; opcode = 3'b001;
   #1 A = 4'b0001; B = 4'b0000; opcode = 3'b001;
   #1 A = 4'b1010; B = 4'b0010; opcode = 3'b001;
//Prueba A + B
   #1 A = 4'b0000; B = 4'b0000; opcode = 3'b010;//0+0 = 0
   #1 A = 4'b1000; B = 4'b0011; opcode = 3'b010;//8+3 = 11
   #1 A = 4'b0000; B = 4'b0001; opcode = 3'b010;//0+1 = 1
   #1 A = 4'b0001; B = 4'b0010; opcode = 3'b010;//1+2 = 3
   #1 A = 4'b0001; B = 4'b0101; opcode = 3'b010;//1+5 = 6
//Prueba sin uso (salida es 0 siempre)
   #1 A = 4'b0000; B = 4'b0000; opcode = 3'b011;
   #1 A = 4'b0001; B = 4'b0110; opcode = 3'b011;
//Prueba A AND B'
   #1 A = 4'b0000; B = 4'b0000; opcode = 3'b100;
   #1 A = 4'b0001; B = 4'b0000; opcode = 3'b100;
   #1 A = 4'b0011; B = 4'b0001; opcode = 3'b100;
   #1 A = 4'b0100; B = 4'b0010; opcode = 3'b100;
//Prueba A OR B'
   #1 A = 4'b0000; B = 4'b0000; opcode = 3'b101;
   #1 A = 4'b0000; B = 4'b0001; opcode = 3'b101;
   #1 A = 4'b0010; B = 4'b0000; opcode = 3'b101;
//Prueba A - B
   #1 A = 4'b0000; B = 4'b0000; opcode = 3'b110;// 0 - 0 = 0
   #1 A = 4'b0001; B = 4'b0000; opcode = 3'b110;// 1-0 = 1
   #1 A = 4'b0010; B = 4'b0010; opcode = 3'b110;//2-2 = 0
   #1 A = 4'b0100; B = 4'b0001; opcode = 3'b110;//4-1 = 3
//Prueba Y = 1 solo cuando A<B
   #1 A = 4'b0011; B = 4'b0011; opcode = 3'b111;
   #1 A = 4'b0110; B = 4'b0111; opcode = 3'b111;
   #1 A = 4'b0100; B = 4'b0010; opcode = 3'b111;
   #1 A = 4'b0101; B = 4'b0010; opcode = 3'b111;
end
//final de la simulación
initial
    #40 $finish;
//Creación del vcd
initial begin
    $dumpfile("E3_tb.vcd");
    $dumpvars(0, testbench);
end
endmodule
```

Código del testbench del ALU

```
---> WARNING: no PCF file found

iverilog -o E3_tb.out -D VCD_OUT
vvp E3_tb.out
VCD info: dumpfile E3_tb.vcd ope
A        B    opcode |    Y
--------------------|---------
0000    0000    000  |   0000
0001    0000    000  |   0000
0000    0001    000  |   0000
0001    0001    000  |   0001
0110    0110    000  |   0110
0011    0010    000  |   0010
0000    0000    001  |   0000
0000    0001    001  |   0001
0001    0000    001  |   0001
1010    0010    001  |   1010
0000    0000    010  |   0000
1000    0011    010  |   1011
0000    0001    010  |   0001
0001    0010    010  |   0011
0001    0101    010  |   0110
0000    0000    011  |   0000
0001    0110    011  |   0000
0000    0000    100  |   0000
0001    0000    100  |   0001
0011    0001    100  |   0010
0100    0010    100  |   0100
0000    0000    101  |   1111
0000    0001    101  |   1110
0010    0000    101  |   1111
0000    0000    110  |   0000
0001    0000    110  |   0001
0010    0010    110  |   0000
0100    0001    110  |   0011
0011    0011    111  |   0000
0110    0111    111  |   0001
0100    0010    111  |   0000
0101    0010    111  |   0000
gtkwave E3_tb.vcd E3_tb.gtkw
```

Output de la simulación de la ALU

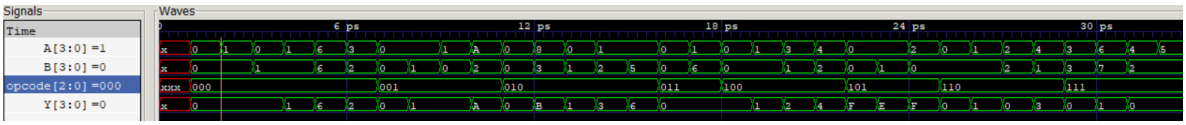


Diagrama de timing de la ALU