Nombre: José Alejandro Rodríguez Porras Carné:19131 Fecha: 29/10/2020

Laboratorio 9

Ejercicio 1

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

lab09dig > E1 > E1.v

```
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital 1
3   //Laboratorio 9
4   //Ejercicio 1
5   //Flip Flop tipo D de 1 bit con Enable
6   module FFD1(input clk, rst, enable, d, output reg q);
7       always @ (posedge clk or posedge rst) begin
8           if (rst)
9               q <= 0;
10          else if (enable)
11              q <= d;
12      end
13  endmodule
14  //Flip Flop tipo D de 2 bits con Enable
15  module FFD2(input clk, rst, enable, input [1:0]d, output [1:0]q);
16
17      FFD1 df1_1(clk, rst, enable, d[1], q[1]);
18      FFD1 df1_0(clk, rst, enable, d[0], q[0]);
19
20  endmodule
21  //Flip Flop tipo D de 4 bits con Enable
22  module FFD4(input clk, rst, enable, input [3:0]d, output [3:0]q);
23
24      FFD2 df2_1(clk, rst, enable, d[3:2], q[3:2]);
25      FFD2 df2_0(clk, rst, enable, d[1:0], q[1:0]);
26
27  endmodule
```

Módulos Flip Flop Tipo D.

Se implementaron 3 flip flops tipo D, uno de 1 bit, uno de 2 bits y uno de 4 bits. El flip flop de 2 bits se implementó usando el módulo del flip flop tipo D de 1 bit, para cada bit componente del flip flop de 2 bits. El flip flop de 4 bits se construyó usando 2 flip flops D de dos bits.

E1.v | E1_tb.v | E2.v | E2_tb.v | E3.v | E3_tb.v | E4.v | E4_tb.v

lab09dig > E1 > E1_tb.v

```
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital 1
3   //Laboratorio 9
4   //Ejercicio 1 Testbench
5   //Flip Flop tipo D de 1 bit con Enable, Flip Flop tipo D de 2 bit con Enable, Flip Flop tipo D de 4 bit con Enable
6   module testbench();
7   //Declaración de variables
8       reg clk, rst, enable;
9       reg [3:0]d;
10      wire [3:0]q;
11  //Asignación de módulos
12      FFD4 ffd4(clk, rst, enable, d, q);
13
14  //Prueba de funcionamiento del Flip Flop Tipo D con Enable de 1 bit
15  initial begin
16      $display("rst enable d | q");
17      $display("-------------|--");
18      $monitor("%b    %b    %b| %b", rst, enable, d, q);
19      clk = 0; rst = 0; enable = 0; d = 4'b0000;
20      //Prueba de reset
21      #1  rst = 1; enable = 0; d = 4'b0000;
22      #1  rst = 0; enable = 0; d = 4'b0000;
23      //Prueba de que cuando enable esta apagado q no se actualiza
24      #1  rst = 0; enable = 0; d = 4'b1001;
25      #5  rst = 0; enable = 0; d = 4'b0000;
26      //Prueba de que cuando enable esta encendido q se actualiza con los flancos positivos del clk
27      #5  rst = 0; enable = 1; d = 4'b0101;
28      #10 rst = 0; enable = 1; d = 4'b0000;
29      //Prueba de que cuando enable esta apagado q no se actualiza
30      #10 rst = 0; enable = 0; d = 4'b0111;
31      #10 rst = 0; enable = 0; d = 4'b0000;
32      //Prueba de reset
33      #10 rst = 0; enable = 1; d = 4'b1111;
34      #10 rst = 1; enable = 1; d = 4'b0000;
35      #1  rst = 0;
36      #80 $finish;
37  end
38  always
39          #5 clk = ~clk;
40      initial begin
41          $dumpfile("E1_tb.vcd");
42          $dumpvars(0,testbench);
43      end
44  endmodule
```

Testbench del Flip Flop Tipo D

En este testbench se probaron varias combinaciones usando un flip flop tipo D de 4 bits construido a base de los otros dos (1 bit y 2 bits) para comprobar el funcionamiento de todos indirectamente. Se probó el reset, y el funcionamiento del flip flop con el enable encendido y apagado.

```
PS D:\AlejandroDigital\electronica_digital1\lab09

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

iverilog -o E1_tb.out -D VCD_OUTPUT=E1_tb C:/Users
vvp E1_tb.out
rst enable d | q
-------------|--
VCD info: dumpfile E1_tb.vcd opened for output.
0    0   0000| xxxx
1    0   0000| 0000
0    0   0000| 0000
0    0   1001| 0000
0    0   0000| 0000
0    1   0101| 0000
0    1   0101| 0101
0    1   0000| 0101
0    1   0000| 0000
0    0   0111| 0000
0    0   0000| 0000
0    1   1111| 0000
0    1   1111| 1111
1    1   0000| 0000
0    1   0000| 0000
gtkwave E1_tb.vcd E1_tb.gtkw
```

Output del flip flop tipo D

En esta imagen se pueden observar los valores de inputs comparados a los valores de salida de q en forma de tabla.

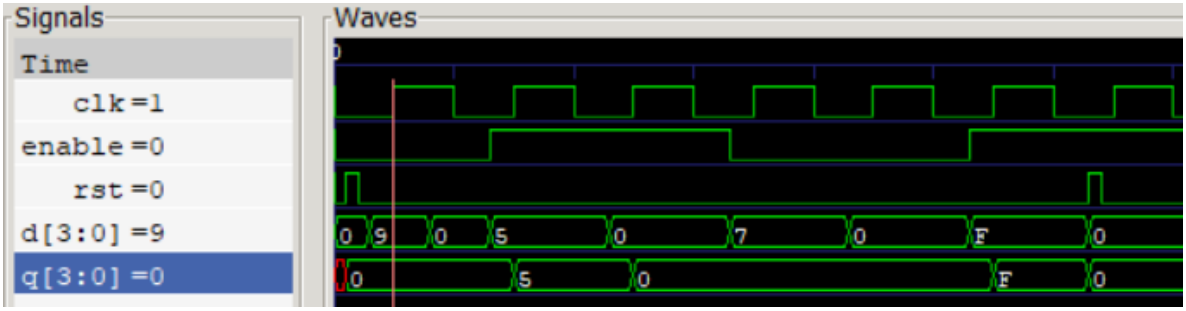


Diagrama de timing del flip flop tipo D

En este diagrama se puede observar que efectivamente solo cuando el enable esta encendido d pasa a q con los flancos positivos de reloj. También se comprobó que el reset funciona de manera asíncrona.

Ejercicio 2

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

ab09dig > E2 > E2.v

```
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital 1
3   //Laboratorio 9
4   //Ejercicio 2
5   //Flip Flop tipo T con Enable
6   module FFD1(input clk, rst, enable, d, output reg q);
7       always @ (posedge clk or posedge rst) begin
8           if (rst)
9               q <= 0;
10          else if (enable)
11              q <= d;
12          end
13  endmodule
14
15  module FFT(input clk, rst, enable, output q);
16      FFD1 ffd1(clk, rst, enable, ~q, q);
17  endmodule
```

Flip Flop Tipo T de 1 bit

Es un flip flop que cuando el enable está encendido, funciona como un “toggle” de la salida q, es decir que con cada flanco positivo de reloj la salida q se convierte en q negada de la actual, dando como resultado, que si el enable se queda encendido, entonces la salida q tiene un período el doble de largo que el del reloj entre q y q negada.

E1.v | E1_tb.v | E2.v | E2_tb.v | E3.v | E3_tb.v

lab09dig > E2 > E2_tb.v

```
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital 1
3   //Laboratorio 9
4   //Ejercicio 2 testbench
5   //Flip Flop tipo T con Enable
6   module testbench();
7   //Declaración de variables
8       reg clk, rst, enable;
9       wire q;
10  //Asignación de módulos
11      FFT fft(clk, rst, enable, q);
12
13  //Prueba de funcionamiento del Flip Flop Tipo D con Enable de 1 bit
14  initial begin
15      $display("rst enable | q");
16      $display("-----------|--");
17      $monitor("%b    %b   | %b", rst, enable, q);
18      clk = 0; rst = 0; enable = 0;
19      //Prueba del reset
20      #1  rst = 1; enable = 0;
21      #1  rst = 0; enable = 0;
22      //Prueba de que q se actualiza con el enable encendido en cada flanco de reloj
23      #10 rst = 0; enable = 1;
24      //Prueba de que q no se actualiza con el enable apagado
25      #30 rst = 0; enable = 0;
26      //Prueba del reset
27      #10 rst = 1; enable = 0;
28      //Prueba de que q se actualiza con el enable encendido en cada flanco de reloj
29      #1  rst = 0; enable = 1;
30      #20 rst = 0; enable = 0;
31      #80 $finish;
32  end
33  always
34          #5 clk = ~clk;
35      initial begin
36          $dumpfile("E2_tb.vcd");
37          $dumpvars(0,testbench);
38      end
39  endmodule
```

Testbench del flip flop tipo T

En este se prueba el funcionamiento del flip flop con el enable encendido y apagado, así como el reset asíncrono.

```
iverilog -o E2_tb.out -D VCD_OUTPUT=E2_tb C:/Users
vvp E2_tb.out
rst enable | q
-----------|--
VCD info: dumpfile E2_tb.vcd opened for output.
0    0   | x
1    0   | 0
0    0   | 0
0    1   | 0
0    1   | 1
0    1   | 0
0    1   | 1
0    0   | 1
1    0   | 0
0    1   | 0
0    1   | 1
0    1   | 0
0    0   | 0
gtkwave E2_tb.vcd E2_tb.gtkw
```

Output del programa de Flip Flop Tipo T

En esta imagen se pueden observar los valores de inputs comparados a los valores de salida de q en forma de tabla.

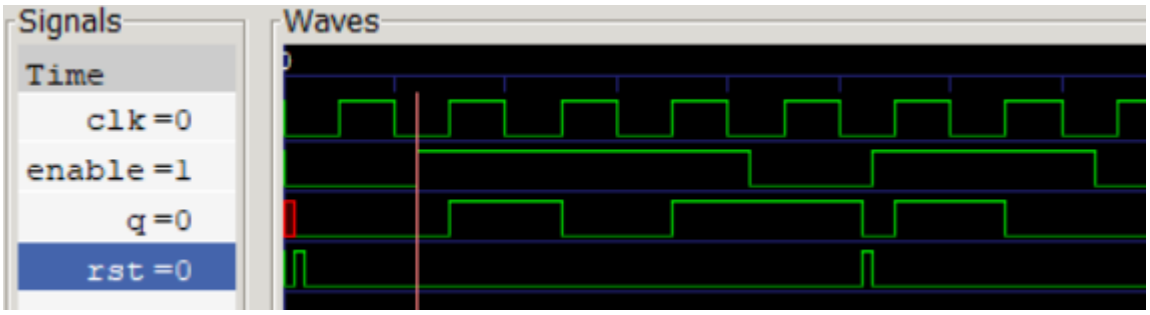
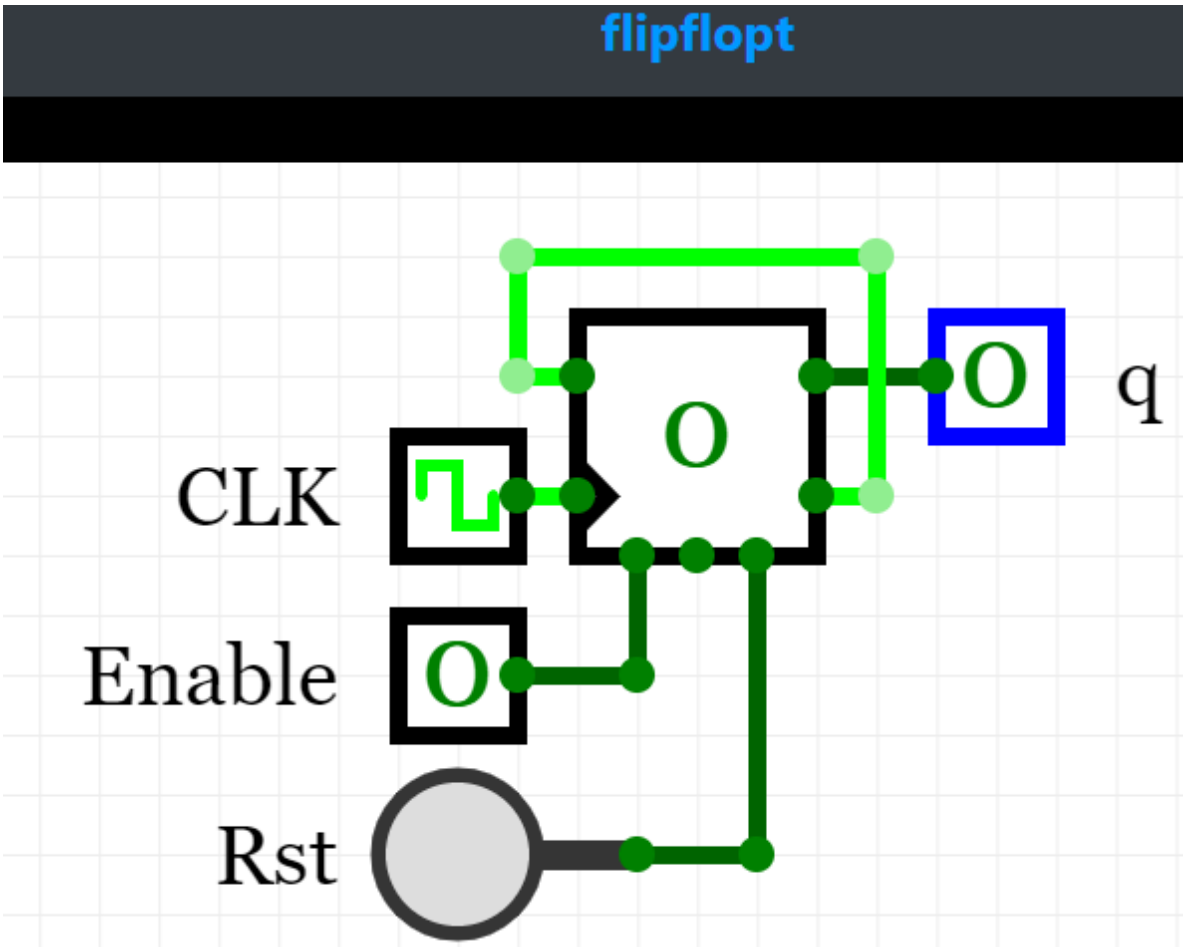


Diagrama de Timing del Flip Flop Tipo T

En esta imagen se puede observar que efectivamente cuando el enable está encendido, q cambia cada flanco de reloj positivo a q negado respecto al actual. También se observó que cuando se dejaba el enable encendido, el período de q a q negado y a q de nuevo era el doble que el del clock naturalmente, debido a que la salida q solo cambia con los flancos positivos. También se comprobó que el reset funciona de forma asíncrona.



Implementación en CircuitVerse del Flip Flop Tipo T

Ejercicio 3

E4_tb.v | E3.v | E4.v | E3_tb.v | E1.v | E1_tb.v | E2.v

ab09dig > E3 > E3.v

```
3   //Laboratorio 9
4   //Ejercicio 3
5   //Flip Flop JK
6   //Flip Flop Tipo D
7   module FFD1(input clk, rst, enable, d, output reg q);
8       always @ (posedge clk or posedge rst) begin
9           if (rst)
10              q <= 0;
11          else if (enable)
12              q <= d;
13          end
14  endmodule
15  //módulo del flip flop tipo JK con inputs J y K, con una nube combinacional previa para definir d
16  module FFJK(input clk, rst, enable, j, k, output q);
17      wire nk, nq, j_and_nq, nk_and_q, d;
18      not (nk, k);
19      not (nq, q);
20      and (j_and_nq, j, nq);
21      and (nk_and_q, nk, q);
22      or (d, j_and_nq, nk_and_q);
23      FFD1 ffd1(clk, rst, enable, d, q);
24  endmodule
```

Módulo del Flip Flop JK

El módulo se implementó tomando un flip flop tipo D como base, y se hizo un nuevo módulo que incluía las variables j y k, que determinaban el input d, dependiendo de las combinaciones en las que se presentaran. El módulo se implementó de forma structural para la nube combinacional que definía d.

E3.v | E3_tb.v | E1.v | E1_tb.v | E2.v

lab09dig > E3 > E3_tb.v

```
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital 1
3   //Laboratorio 9
4   //Ejercicio 3
5   //Flip Flop JK testbench
6   module testbench();
7   //Declaración de variables
8       reg clk, rst, enable, j, k;
9       wire q;
10  //Asignación de módulos
11      FFJK fft(clk, rst, enable, j, k, q);
12
13  //Prueba de funcionamiento del Flip Flop JK con Enable de 1 bit
14  initial begin
15      $display("rst enable j k | q");
16      $display("-----------|--");
17      $monitor("%b    %b   %b %b   | %b", rst, enable, j, k, q);
18      clk = 0; rst = 0; enable = 0; j = 0; k = 0;
19      //Prueba del reset
20      #1  rst = 1; enable = 0; j = 0; k = 0;
21      //Prueba del enable en 0
22      #1  rst = 0; enable = 0; j = 1; k = 0;
23      #10 rst = 0; enable = 0; j = 0; k = 0;
24      #10 rst = 0; enable = 0; j = 0; k = 1;
25      #10 rst = 0; enable = 0; j = 1; k = 1;
26      //Prueba del enable en 1
27      #10 rst = 0; enable = 1; j = 0; k = 0;
28      #10 rst = 0; enable = 1; j = 1; k = 0;
29      #10 rst = 0; enable = 1; j = 0; k = 0;
30      #10 rst = 0; enable = 1; j = 0; k = 1;
31      #10 rst = 0; enable = 1; j = 1; k = 1;
32      //Prueba del enable en 0
33      #10 rst = 0; enable = 0; j = 0; k = 1;
34      #10 rst = 1; enable = 0; j = 0; k = 0;
35      #1  rst = 0; enable = 0; j = 0; k = 0;
36      #150 $finish;
37  end
38  always
39          #5 clk = ~clk;
40      initial begin
41          $dumpfile("E3_tb.vcd");
42          $dumpvars(0,testbench);
43      end
44  endmodule
```

Testbench del Flip Flop JK

Se probó el reset, el enable en 0 y 1 con distintos valores de J y K.

```
iverilog -o E3_tb.out -D VCD_OU
vvp E3_tb.out
rst enable j k    | q
------------------|--
VCD info: dumpfile E3_tb.vcd op
0    0      0 0   | x
1    0      0 0   | 0
0    0      1 0   | 0
0    0      0 0   | 0
0    0      0 1   | 0
0    0      1 1   | 0
0    1      0 0   | 0
0    1      1 0   | 0
0    1      1 0   | 1
0    1      0 0   | 1
0    1      0 1   | 1
0    1      0 1   | 0
0    1      1 1   | 0
0    1      1 1   | 1
0    0      0 1   | 1
1    0      0 0   | 0
0    0      0 0   | 0
gtkwave E3_tb.vcd E3_tb.gtkw
```

Output del programa de Flip Flop JK

En esta imagen se pueden observar los valores de inputs comparados a los valores de salida de q en forma de tabla.

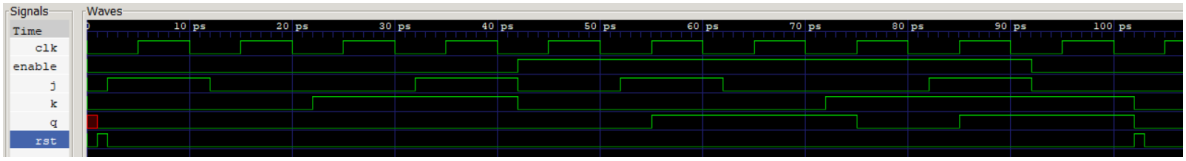
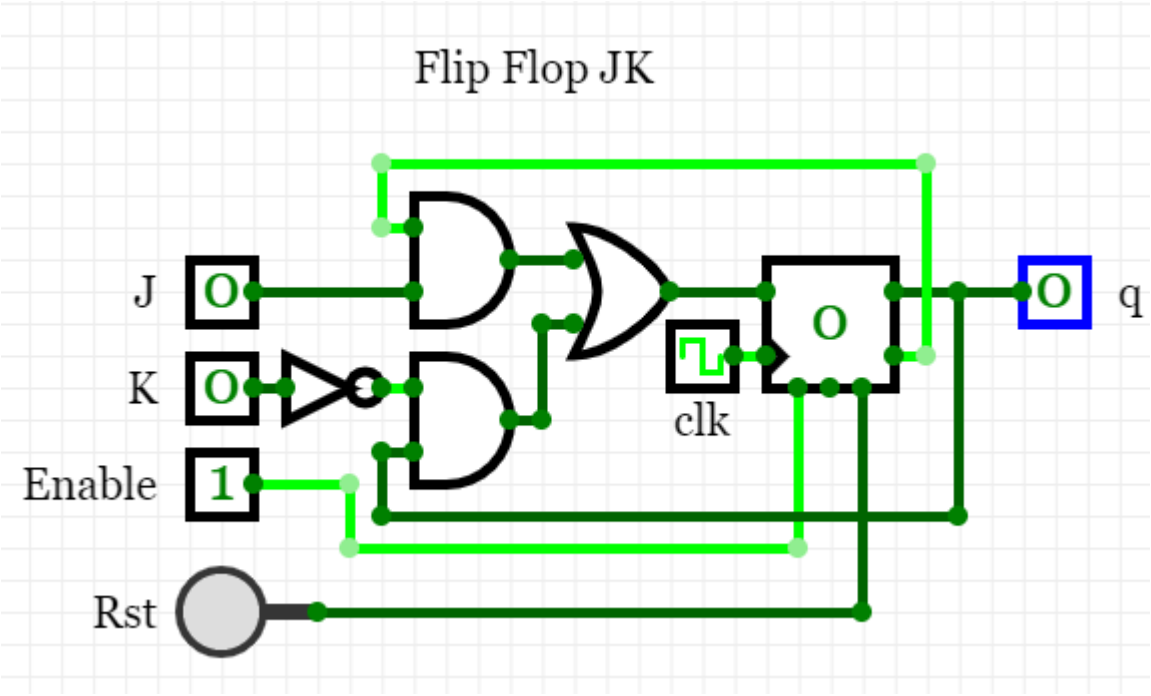


Diagrama de timing del Flip Flop JK

En este diagrama se puede observar que el enable desempeña su función exitosamente, ya que j y k hacen efecto únicamente cuando el enable este encendido. Se probaron todas las combinaciones de j y k, influyendo en el output de q, ya sea manteniendo su valor pasado, poniendo q en 0, poniendo q en 1 o poniendo a q en modo toggle. También se probó el reset asíncrono.



Flip Flop JK en Circuitverse

Ejercicio 4

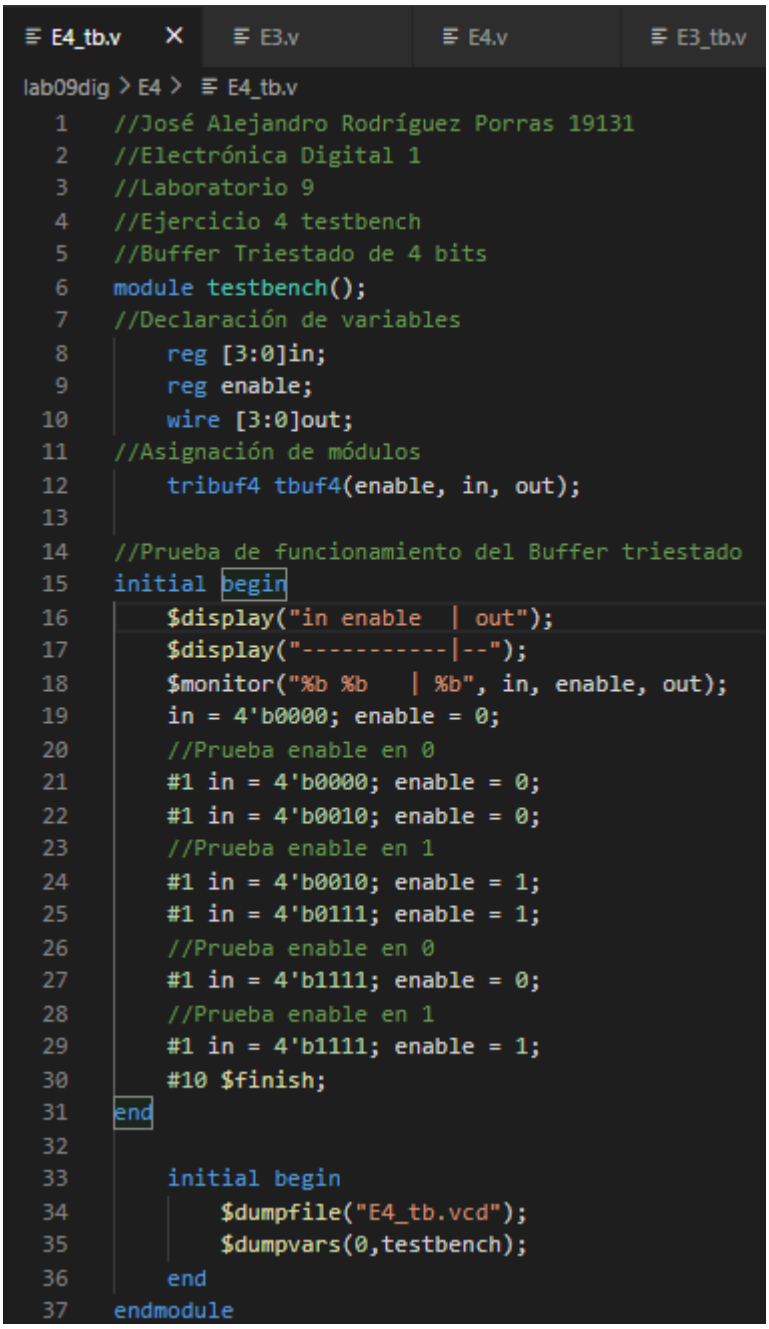
E3.v | E4.v | E3_tb.v | E1.v | E1_tb.v | E2.v

lab09dig > E4 > E4.v

```
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital 1
3   //Laboratorio 9
4   //Ejercicio 4
5   //Buffer Triestado de 4 bits
6   module tribuf4(input wire enable, input wire [3:0]in, output wire [3:0]out);
7       assign out = (enable) ? in:4'bz;
8   endmodule
```

Buffer Triestado de 4 bits

Cuando se activa el enable se dejan pasar los bits del input al ouput, mientras que si el enable está deshabilitado, se ponen los bits de salida en alta impedancia (z).



Testbench del buffer triestado de 4 bits

El testbench se diseño para observar el comportamiento de la salida al encender o apagar el enable, usando algunos valores aleatorios para el input.

```
vvp E4_tb.out
in enable  | out
-----------|--
VCD info: dumpfile E4_tb.vcd o
0000 0   | zzzz
0010 0   | zzzz
0010 1   | 0010
0111 1   | 0111
1111 0   | zzzz
1111 1   | 1111
gtkwave E4_tb.vcd E4_tb.gtkw
```

Output del buffer triestado de 4 bits

En esta imagen se pueden observar los valores de input comparados a los valores de output en forma de tabla. Se puede observar que cuando el enable es cero las salidas se ponen en zzzz, que significa alta impedancia.

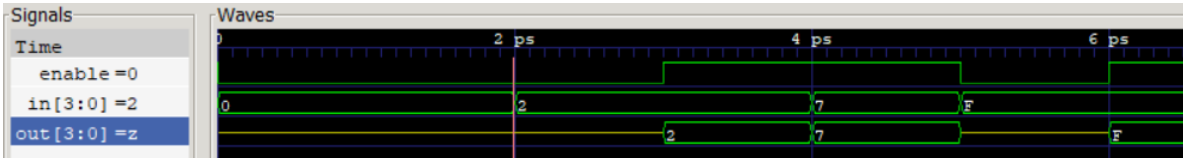
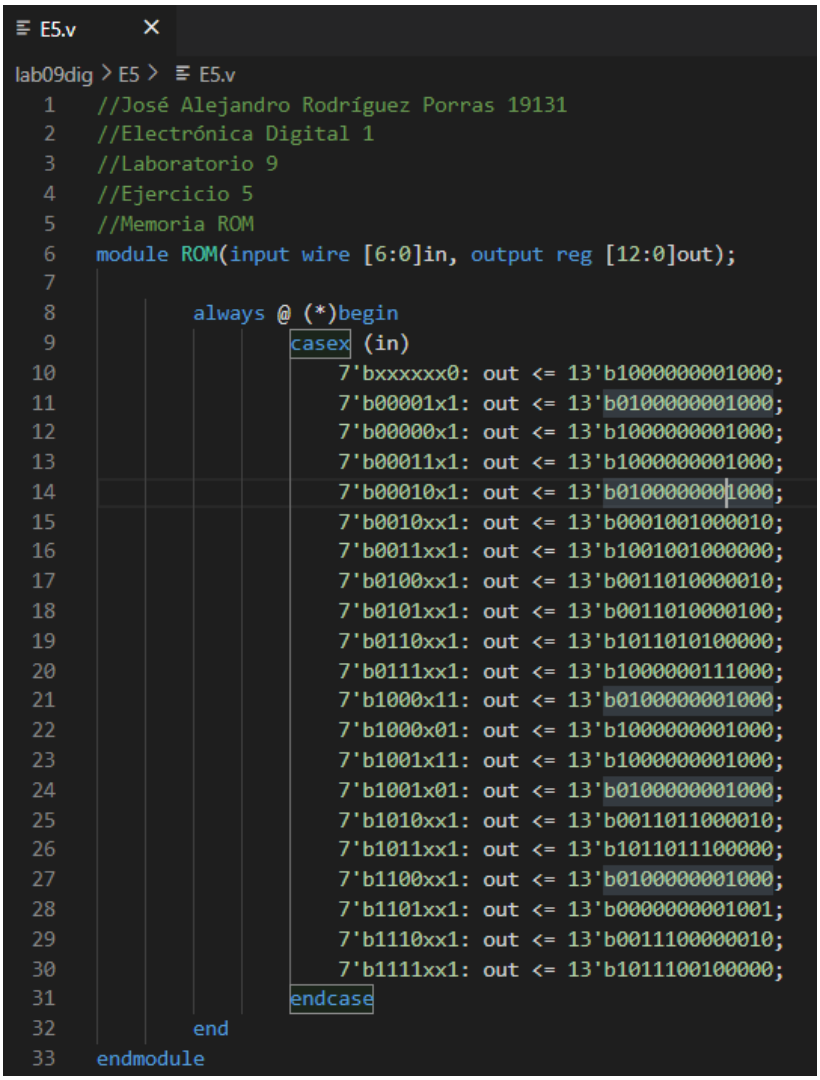


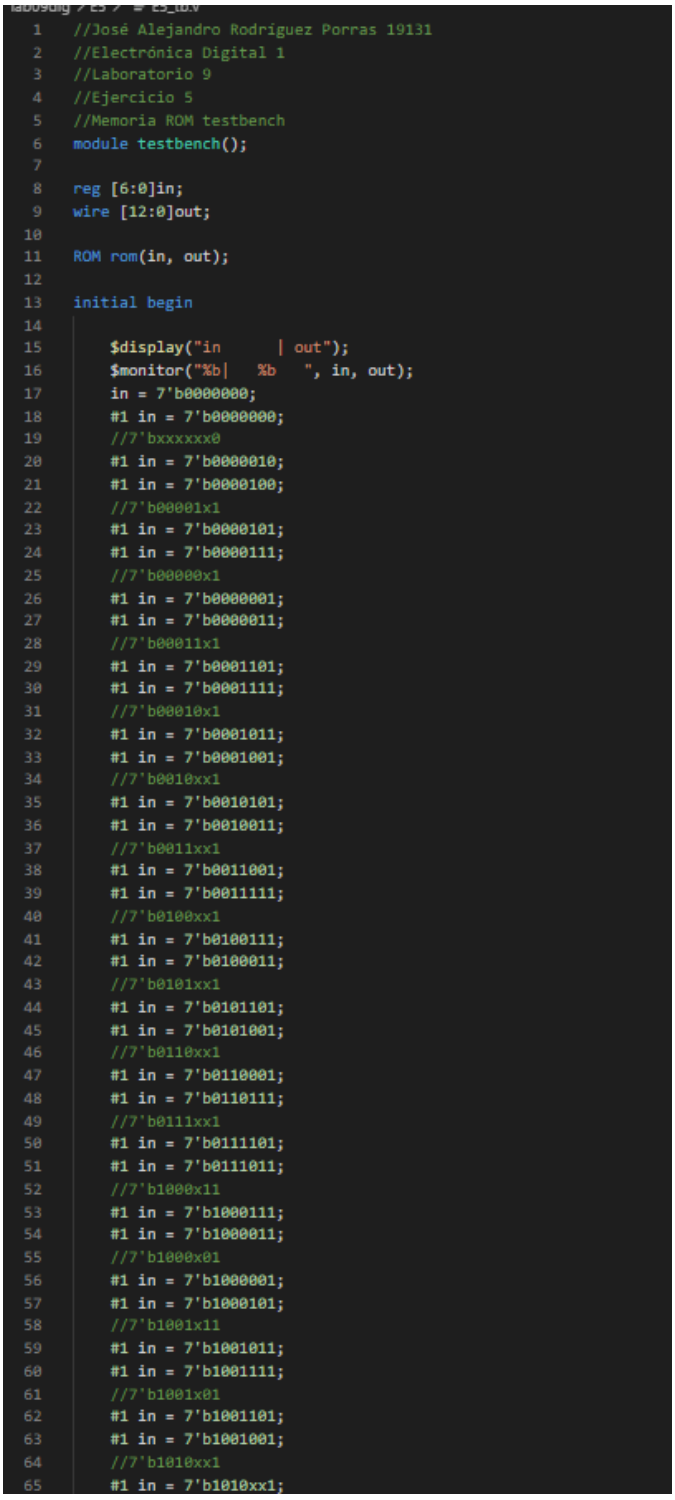
Diagrama de timing del buffer triestado de 4 bits

Se probaron diferentes valores de input, y como se puede ver en el diagrama de timing, estos solo pasaban a la salida cuando se activaba el enable. En cuanto el enable fuera 0, todas las salidas se ponen en alta impedancia (z), en el diagrama se ponen en amarillo en medio.

Ejercicio 5

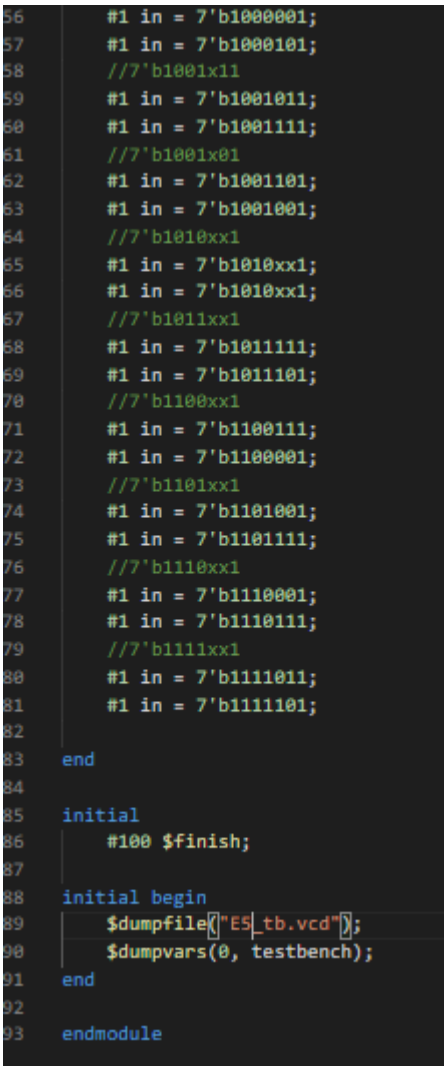


Código de la memoria ROM usando case, y don´t cares representados como x.



```
//José Alejandro Rodriguez Porras 19131
//Electrónica Digital 1
//Laboratorio 9
//Ejercicio 5
//Memoria ROM testbench
module testbench();

reg [6:0]in;
wire [12:0]out;

ROM rom(in, out);

initial begin

    $display("in      | out");
    $monitor("%b|   %b   ", in, out);
    in = 7'b0000000;
    #1 in = 7'b0000000;
    //7'bxxxxxx0
    #1 in = 7'b0000010;
    #1 in = 7'b0000100;
    //7'b00001x1
    #1 in = 7'b0000101;
    #1 in = 7'b0000111;
    //7'b00000x1
    #1 in = 7'b0000001;
    #1 in = 7'b0000011;
    //7'b00011x1
    #1 in = 7'b0001101;
    #1 in = 7'b0001111;
    //7'b00010x1
    #1 in = 7'b0001011;
    #1 in = 7'b0001001;
    //7'b0010xx1
    #1 in = 7'b0010101;
    #1 in = 7'b0010011;
    //7'b0011xx1
    #1 in = 7'b0011001;
    #1 in = 7'b0011111;
    //7'b0100xx1
    #1 in = 7'b0100111;
    #1 in = 7'b0100011;
    //7'b0101xx1
    #1 in = 7'b0101101;
    #1 in = 7'b0101001;
    //7'b0110xx1
    #1 in = 7'b0110001;
    #1 in = 7'b0110111;
    //7'b0111xx1
    #1 in = 7'b0111101;
    #1 in = 7'b0111011;
    //7'b1000x11
    #1 in = 7'b1000111;
    #1 in = 7'b1000011;
    //7'b1000x01
    #1 in = 7'b1000001;
    #1 in = 7'b1000101;
    //7'b1001x11
    #1 in = 7'b1001011;
    #1 in = 7'b1001111;
    //7'b1001x01
    #1 in = 7'b1001101;
    #1 in = 7'b1001001;
    //7'b1010xx1
    #1 in = 7'b1010xx1;
```

Testbench de la memoria ROM probando varias combinaciones parte 1



```
        #1 in = 7'b1000001;
        #1 in = 7'b1000101;
        //7'b1001x11
        #1 in = 7'b1001011;
        #1 in = 7'b1001111;
        //7'b1001x01
        #1 in = 7'b1001101;
        #1 in = 7'b1001001;
        //7'b1010xx1
        #1 in = 7'b1010xx1;
        #1 in = 7'b1010xx1;
        //7'b1011xx1
        #1 in = 7'b1011111;
        #1 in = 7'b1011101;
        //7'b1100xx1
        #1 in = 7'b1100111;
        #1 in = 7'b1100001;
        //7'b1101xx1
        #1 in = 7'b1101001;
        #1 in = 7'b1101111;
        //7'b1110xx1
        #1 in = 7'b1110001;
        #1 in = 7'b1110111;
        //7'b1111xx1
        #1 in = 7'b1111011;
        #1 in = 7'b1111101;

    end

    initial
        #100 $finish;

    initial begin
        $dumpfile("E5_tb.vcd");
        $dumpvars(0, testbench);
    end

    endmodule
```

Testbench de la memoria ROM probando varias combinaciones parte 2

```
in      | out
VCD info: dumpfile E5_tb.vcd opened
0000000|   1000000001000
0000010|   1000000001000
0000100|   1000000001000
0000101|   0100000001000
0000111|   0100000001000
0000001|   1000000001000
0000011|   1000000001000
0001101|   1000000001000
0001111|   1000000001000
0001011|   0100000001000
0001001|   0100000001000
0010101|   0001001000010
0010011|   0001001000010
0011001|   1001001000000
0011111|   1001001000000
0100111|   0011010000010
0100011|   0011010000010
0101101|   0011010000100
0101001|   0011010000100
0110001|   1011010100000
0110111|   1011010100000
0111101|   1000000111000
0111011|   1000000111000
1000111|   0100000001000
1000011|   0100000001000
1000001|   1000000001000
1000101|   1000000001000
1001011|   1000000001000
1001111|   1000000001000
1001101|   0100000001000
1001001|   0100000001000
1010001|   0011011000010
1010111|   0011011000010
1011111|   1011011100000
1011101|   1011011100000
1100111|   0100000001000
1100001|   0100000001000
1101001|   0000000001001
1101111|   0000000001001
1110001|   0011100000010
1110111|   0011100000010
1111011|   1011100100000
1111101|   1011100100000
gtkwave E5_tb.vcd E5_tb.gtkw
```

Output de la simulación de la memoria ROM probando varias combinaciones

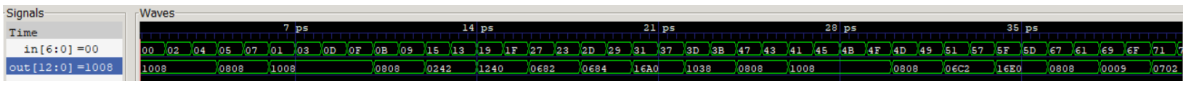


Diagrama de timing de la simulación probando varias entradas en la memoria ROM