Fernando Arribas

Carné: 19030

# Lab 09 Digital 1

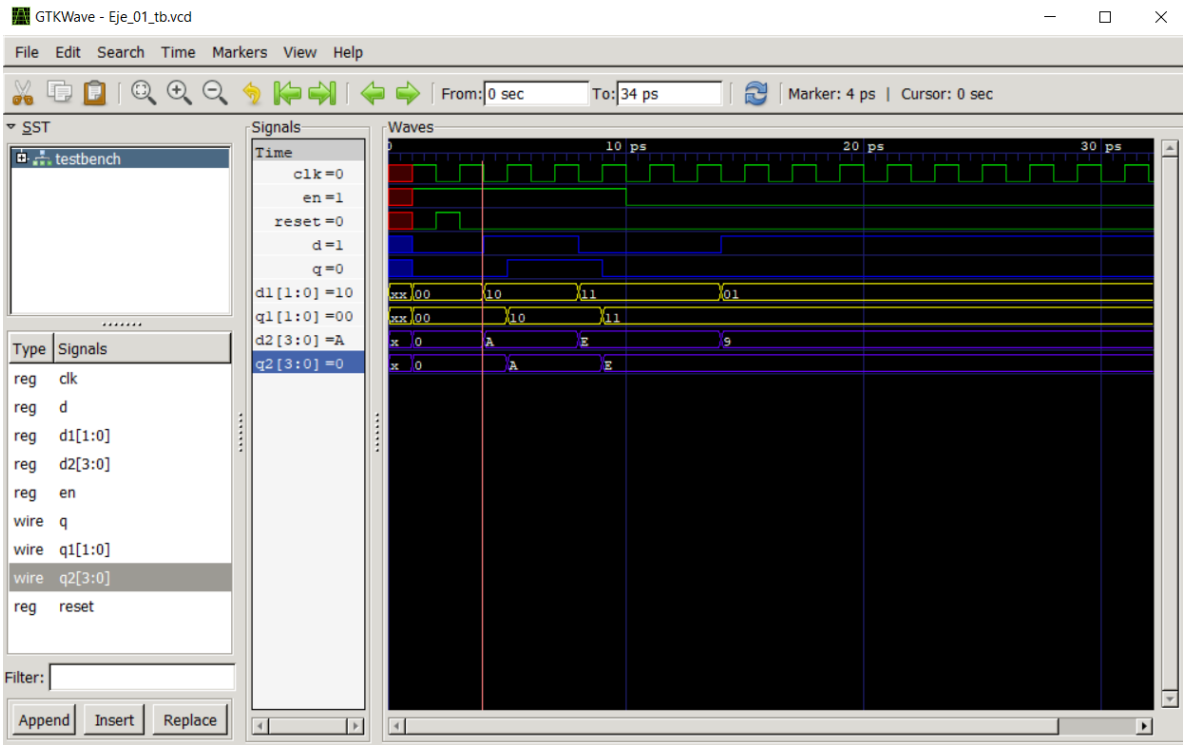
## Ejercicio 1

```
module ffd(input clk, input reset, input en, input d, output reg q);

  always @ (posedge clk, posedge reset)
  if (reset) q <= 1'b0;
  else if (en == 1) q <= d;

endmodule

module ffd2(input clk, reset, en, input [1:0] d, output [1:0] q);

  ffd a0(clk, reset, en, d[0], q[0]);
  ffd a1(clk, reset, en, d[1], q[1]);

endmodule

module ffd4(input clk, reset, en, input [3:0] d, output [3:0] q);

  ffd a0(clk, reset, en, d[0], q[0]);
  ffd a1(clk, reset, en, d[1], q[1]);
  ffd a2(clk, reset, en, d[2], q[2]);
  ffd a3(clk, reset, en, d[3], q[3]);

endmodule
```

En el primer ejercicio se realizo un modulo principal, el cual es un flip flop tipo d, con el cual se prosiguieron a realizar otros dos flip flops tipo de d. Los cuales eran de 2 y 4 bits, lo cual se logró llamando el módulo principal las veces necesarias para lograr nuestra cantidad de bits. En el caso del de 2 bits se llamo un total de 2 veces el FFD, mientras que para el de 4 bits se llamo un total de 4 veces, aunque también se pudo llamar el FFD de 2 bits un total de 2 veces para acortar un poco el código.

```
module testbench();

  reg clk, reset, en, d;
  reg [1:0] d1;
  reg [3:0] d2;
  wire q;
  wire [1:0] q1;
  wire [3:0] q2;

  ffd a1(clk, reset, en, d, q);
  ffd2 a2(clk, reset, en, d1, q1);
  ffd4 a3(clk, reset, en, d2, q2);

  //FF de 1 bit
  initial begin

  #1
  clk = 0; en = 1; reset = 0; d = 0; d1 = 2'b00; d2 = 4'b0000;
  $display("Ejercicio 01");
  $display("| D | Q | 1 bit | D  | Q  | 2 bits | D    | Q    | 4 bits ");
  $monitor("| %b | %b |       | %b | %b |        | %b | %b |", d, q, d1, q1, d2, q2);

  #1 reset = 1;
  #1 reset = 0;
  #1 d = 1; d1 = 2'b10; d2 = 4'b1010;
  #4 d = 0; d1 = 2'b11; d2 = 4'b1110;
  #2 en = 0;
  #4 d = 1; d1 = 2'b01; d2 = 4'b1001;

  #20 $finish;
  end
```

```
32
33      always
34      begin
35        #1 clk = ~clk;
36        end
37
38      initial begin
39        $dumpfile("Eje_01_tb.vcd");
40        $dumpvars(0, testbench);
41      end
42
43
44    endmodule
```

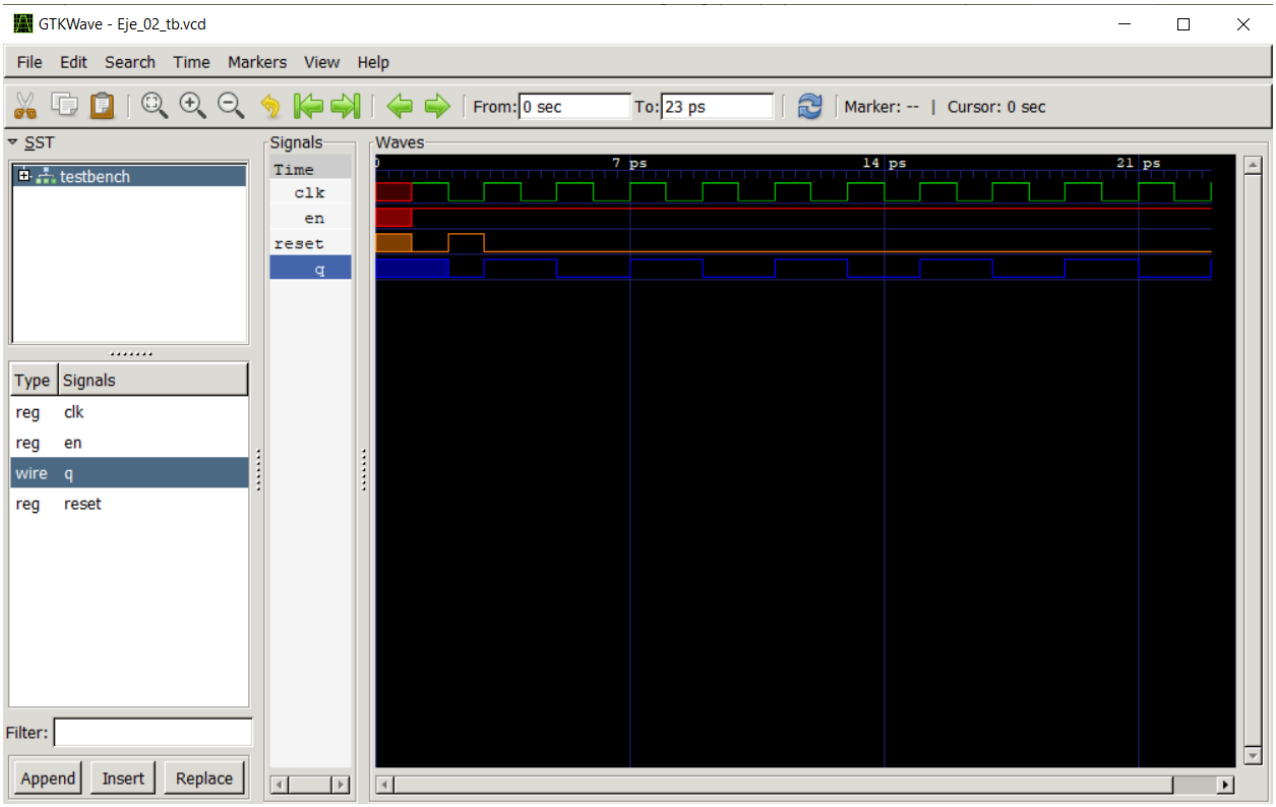


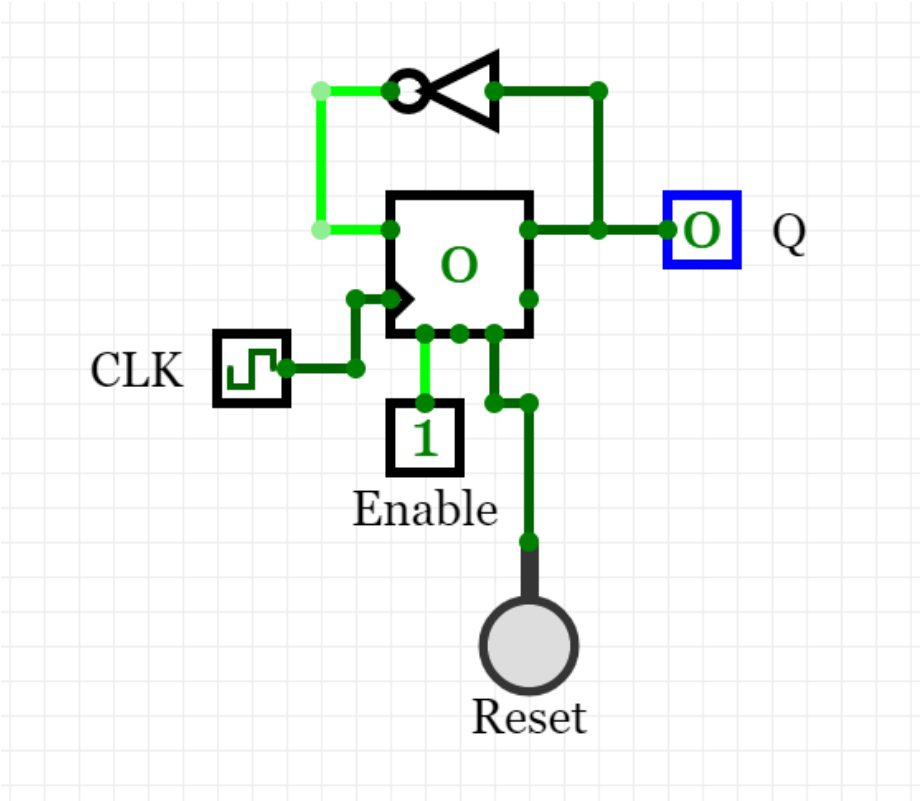
## Ejercicio 2

```
module ffd(input clk, input reset, input en, input d, output reg q);

  always @ (posedge clk, posedge reset)
  if (reset) q <= 1'b0;
  else if (en) q <= d;

endmodule

module fft(input clk, en, reset, output q);

  wire w;
  assign w = ~q;

  ffd a(clk, reset, en, w, q);

endmodule
```

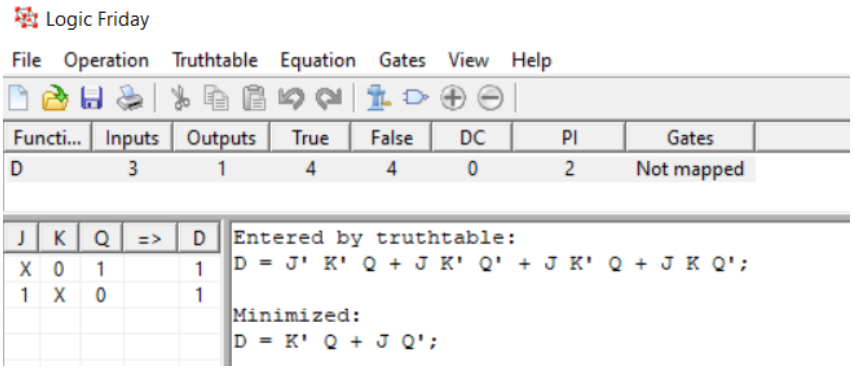
Primero se construyo un flip flop tipo d, luego se prosiguió con la construcción del toggle FF. Para lograrlo se utilizó un cable, el cual conecta la salida Q del FFD a una compuerta NOT, la cual entrara en la entrada D del FFD. Logrando así al llamar el módulo del FFD un ToggleFF.

```
module testbench();

  reg clk, reset, en;
  wire q;

  fft a(clk, en, reset, q);

  initial begin

  #1
  clk = 0; en = 1; reset = 0;
  $display("Ejercicio 02: Flip FLop T");
  $display("| Q |");
  $monitor("| %b |", q);
  #1 reset = 1;
  #1 reset = 0;

  #20 $finish;
  end

  always
   #1 clk = ~clk;

   initial begin
     $dumpfile("Eje_02_tb.vcd");
     $dumpvars(0, testbench);
   end

endmodule
```


GTKWave - Eje_02_tb.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 23 ps
Marker: -- | Cursor: 0 sec
SST
testbench
Type Signals
reg clk
reg en
wire q
reg reset
Filter:
Append Insert Replace
Signals
Time
clk
en
reset
q
Waves
7 ps
14 ps
21 ps


D
Q
CLK
1
Enable
Reset

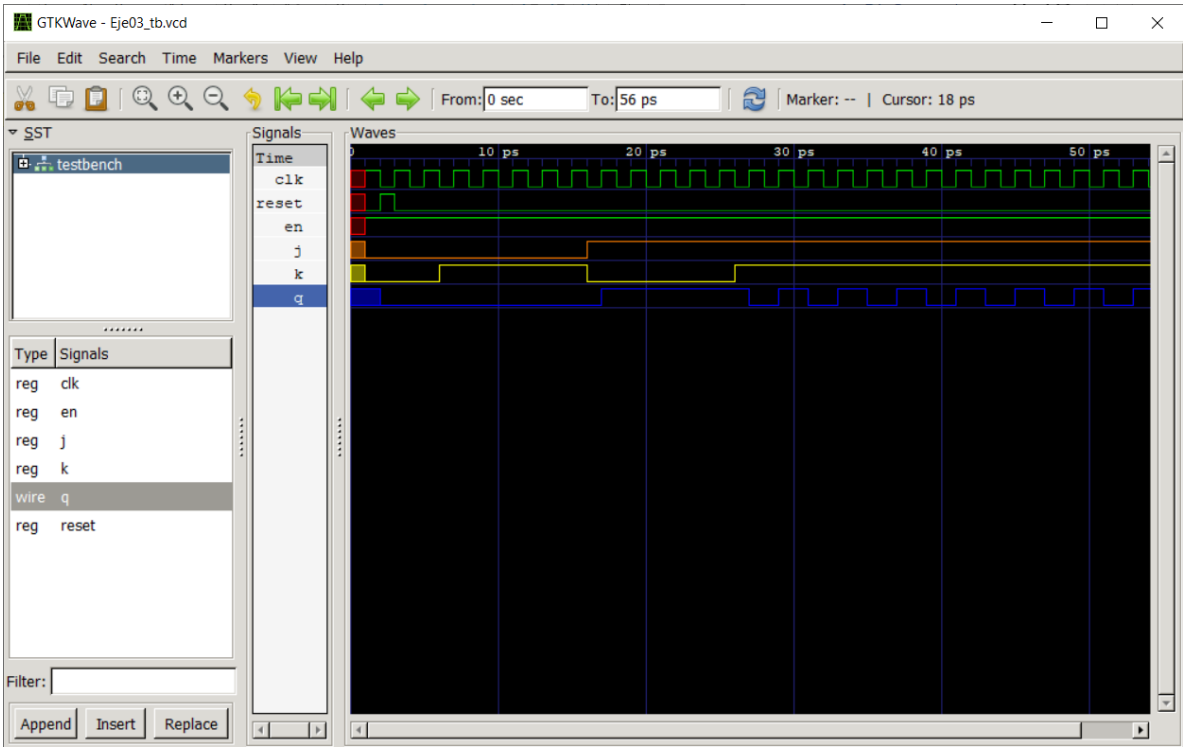
## Ejercicio 3

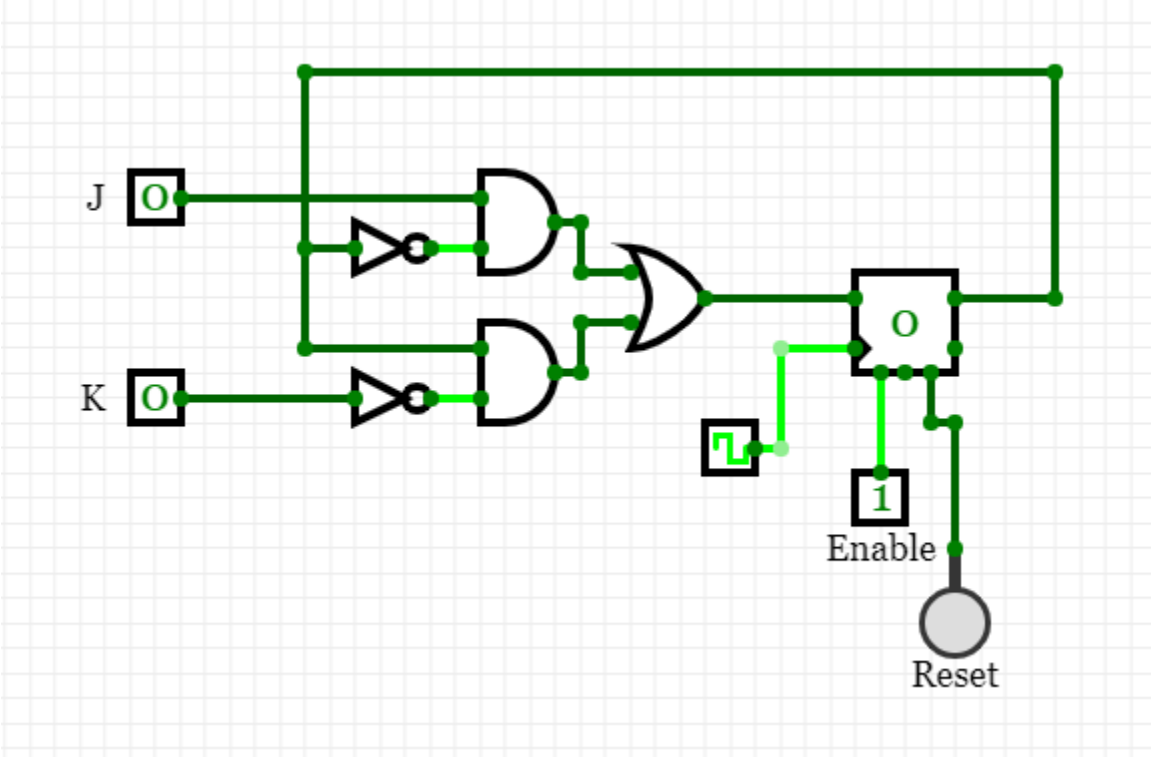



```
module ffd(input clk, input reset, input en, input d, output reg q);

  always @ (posedge clk, posedge reset)
  if (reset) q <= 1'b0;
  else if (en == 1) q <= d;

endmodule

module ffjk(input clk, reset, en, j, k, output q);

  wire d;

  assign d = (~k & q) | (j & ~q);

  ffd a1(clk, reset, en, d, q);

endmodule
```

Primero se construyó una tabla de verdad, la cual describiría el comportamiento de un FFJK por medio de una nube combinacional. Luego se prosiguió a construir el módulo con el FFD, el cual fue llamado a otro modulo, en donde se primero se aplico la nube combinacional que entraría en la entrada D del FFD y se llamo al módulo el FFD.

```
module testbench();

  reg clk, reset, en, j, k;
  wire q;

  ffjk a1(clk, reset, en, j, k, q);

  initial begin
    #1

    clk = 0; reset = 0; en = 1; j = 0; k = 0;
    $display("Ejercicio 3: Flip Flop JK");
    $display("J | K | Q");
    $monitor("%b | %b | %b", j, k, q);
    #1 reset = 1;
    #1 reset = 0;
    #3 k = 1;
    #10 j = 1; k = 0;
    #10 k = 1;

    #30 $finish;
    end

  always
  begin
    #1 clk = ~clk;
    end

  initial begin
    $dumpfile("Eje03_tb.vcd");
    $dumpvars(0, testbench);
  end

```


GTKWave - Eje03_tb.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 56 ps
Marker: -- | Cursor: 18 ps
SST
testbench
Type Signals
reg clk
reg en
reg j
reg k
wire q
reg reset
Filter:
Append Insert Replace
Signals
Time
clk
reset
en
j
k
q
Waves
10 ps
20 ps
30 ps
40 ps
50 ps

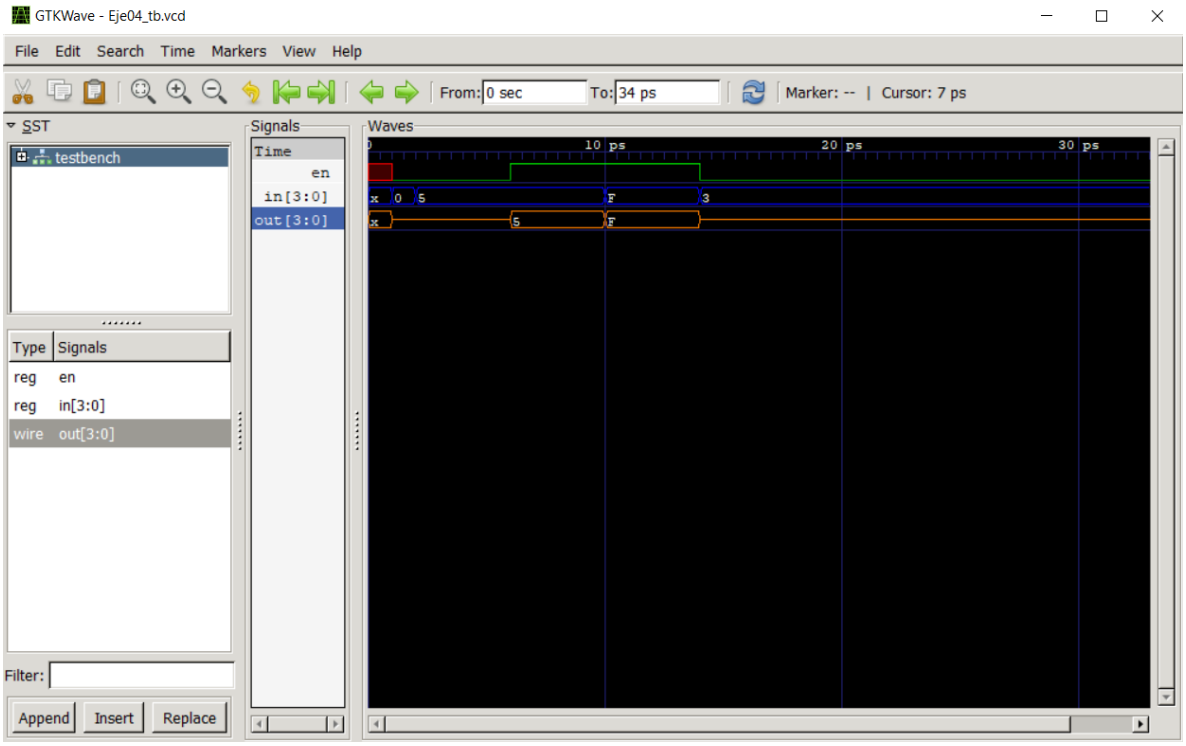

J
K
O
1
Enable
Reset

**Ejercicio 4**

```
module buftri(input wire en, input wire [3:0]in, output wire [3:0]out);

  assign out = (en == 1) ? in :
               (en == 0) ? 4'bz : 4'bx;

endmodule
```

Primero se llamaron las variables, de entrada, tenemos el enable y una de 4 bits llamada in, mientras que de salida tenemos una variable de 4 bits llamada out. Luego empezamos a construir el buffer triestado, asignamos a out un condicional en donde si en es igual a 1 deja pasar a in, de lo contrario estará en alta impedancia.

```
module testbench();

  reg en;
  reg [3:0] in;
  wire [3:0] out;

  buftri a1(en, in, out);

  initial begin

    #1

    en = 0; in = 4'b0000;
    $display("Ejercicio 4: Buffer tri estado de de 4 bits");
    $display("EN | In | OUT");
    $monitor("%b | %b | %b", en, in, out);
    #1 in = 4'b0101;
    #4 en = 1;
    #4 in = 4'b1111;
    #4 en = 0; in = 4'b0011;

    #20 $finish;
    end

    initial begin
      $dumpfile("Eje04_tb.vcd");
      $dumpvars(0, testbench);
    end

endmodule
```


GTKWave - Eje04_tb.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 34 ps
Marker: -- | Cursor: 7 ps
SST
testbench
Type Signals
reg en
reg in[3:0]
wire out[3:0]
Filter:
Append Insert Replace
Signals
Time
en
in[3:0]
out[3:0]
Waves
10 ps
20 ps
30 ps

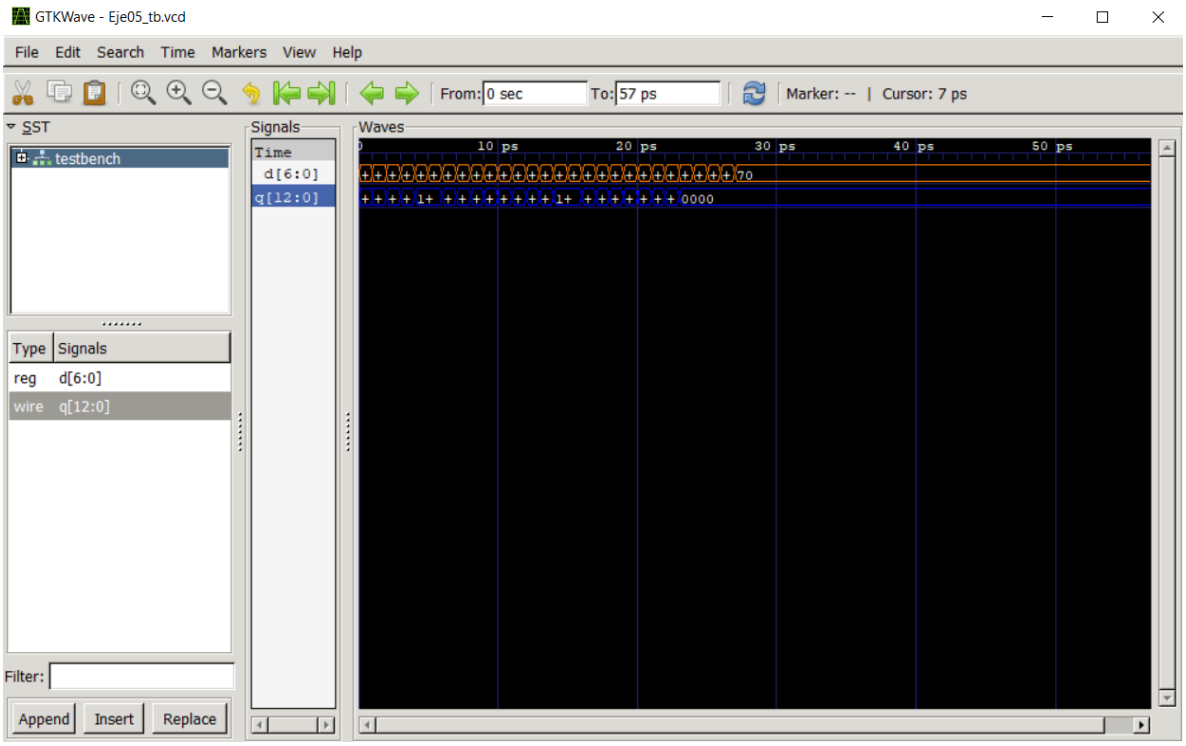
## Ejercicio 5

```
module ROM(input [6:0] d, output reg [12:0] q);

always @ (d) begin

    case(d)
      7'b??????0: q = 13'b1000000001000;
      7'b00001?1: q = 13'b0100000001000;
      7'b00000?1: q = 13'b1000000001000;
      7'b00011?1: q = 13'b1000000001000;
      7'b00010?1: q = 13'b0100000001000;
      7'b0010??1: q = 13'b0001001000010;
      7'b0011??1: q = 13'b1001001100000;
      7'b0100??1: q = 13'b0011010000010;
      7'b0101??1: q = 13'b0011010100000;
      7'b0110??1: q = 13'b1011010100000;
      7'b0111??1: q = 13'b1000000111000;
      7'b1000?11: q = 13'b0100000001000;
      7'b1000?01: q = 13'b1000000001000;
      7'b1001?11: q = 13'b1000000001000;
      7'b1001?01: q = 13'b0100000001000;
      7'b1010??1: q = 13'b0011011000010;
      7'b1011??1: q = 13'b1011011100000;
      7'b1100??1: q = 13'b0100000001000;
      7'b1101??1: q = 13'b0000000001001;
      7'b1110??1: q = 13'b0011100000010;
      7'b1111??1: q = 13'b1011100100000;
      default: q = 13'b0;

    endcase
  end


endmodule
```

La memoria ROM se construye de manera sencilla con un case, en el cual hay una entrada de 7 bits llamada d y una salida de 13 bits llamada q. En el case se colocaron las 21 combinaciones posibles que existen en la tabla, siendo los don´t cares representados en verilog por un signo ‘?’.

```
module testbench();

  reg [6:0] d;
  wire [12:0] q;

  ROM a1(d, q);

  initial begin

    #1
    d = 7'b0000000;
    $display("Ejercicio 5: Memoria Rom con Cases");
    $display("D       |      Q");
    $monitor("%b | %b", d, q);
    #1 d = 7'b??????0;
    #1 d = 7'b00001?1;
    #1 d = 7'b00000?1;
    #1 d = 7'b00011?1;
    #1 d = 7'b00010?1;
    #1 d = 7'b0010??1;
    #1 d = 7'b0011??1;
    #1 d = 7'b0100??1;
    #1 d = 7'b0101??1;
    #1 d = 7'b0110??1;
    #1 d = 7'b0111??1;
    #1 d = 7'b1000?11;
    #1 d = 7'b1000?01;
    #1 d = 7'b1001?11;
    #1 d = 7'b1001?01;
    #1 d = 7'b1010??1;
    #1 d = 7'b1011??1;
    #1 d = 7'b1100??1;
    #1 d = 7'b1101??1;
```

```
34          #1 d = 7'b1110??1;
35          #1 d = 7'b1111??1;
36          #1 d = 7'b0101010;
37          #1 d = 7'b1110001;
38          #1 d = 7'b0000000;
39          #1 d = 7'b1100110;
40          #1 d = 7'b1110000;
41
42          #30 $finish;
43          end
44
45          initial begin
46            $dumpfile("Eje05_tb.vcd");
47            $dumpvars(0, testbench);
48          end
49
50
51      endmodule
```




Link Repositorio: https://github.com/fernando19030/LaboratoriosElectronica_Digital_1-19030.git