Valerie Lorraine Sofia Valdez Trujillo
Universidad del Valle de Guatemala

Carné: 19659
Sección: 20

Electrónica digital
Laboratorio 09

Ejercicio 01

Módulos de los Flip Flop de 1, 2 y 4 bits

```
// Flip Flop D 1 bit

module FFD(input wire clk, EN, reset, input wire C, output reg D);
  always @ (posedge clk, posedge reset) begin

// Non-blockinf assigment
    if(reset) // Colocar todos los valores en 0
      D <= 1'b0;

    else if(EN == 1) // Enabled encendido cuenta cuenta cada flanco del reloj
      D <= C;
  end
endmodule


// Flip Flop 2 bits

module FFD2(input wire clk, EN, reset, input wire [1:0]C, output [1:0]D);

  FFD A1(clk, EN, reset, C[0], D[0]);
  FFD A2(clk, EN, reset, C[1], D[1]);

endmodule


// Flip Flop 4 bits

module FFD4(input wire clk, EN, reset, input wire [3:0]C, output [3:0]D);

  FFD2 B1(clk, EN, reset, C[3:2], D[3:2]);
  FFD2 B2(clk, EN, reset, C[1:0], D[1:0]);

endmodule
```

Testbench

```
module testbench();

  //inputs
  reg clk, EN, reset;
  reg C;
  reg [1:0]C1;
  reg [3:0]C2;

  //outputs
  wire D;
  wire [1:0]D1;
  wire [3:0]D2;

// Clock
always
  begin
    clk <= 1; #1 clk <= ~clk; #1;
end

// Llamar a los módulos
  FFD A1(clk, EN, reset, C, D); // Flitidiflop 1 bit

  FFD2 B1(clk, EN, reset, C1[1:0], D1[1:0]); // Flitidiflop 2 bits

  FFD4 R1(clk, EN, reset, C2[3:0], D2[3:0]); // Flitidiflop 4 bits

  initial begin
    #1
    $display("\n");
    $display("Flip Flop de 1 bit");
    $display("\n");
    $display("-------------------|-----|");
    $display(" clk  EN  reset  C  |  D  |");
    $display("-------------------|-----|");
    $monitor("%b    %b    %b    %b   |  %b  |", clk, EN, reset, C, D);

    #1 clk = 0; reset = 0; EN = 1; C = 0;
    #5 reset = 1; EN = 1; C = 0;
    #5 reset = 0; EN = 1; C = 0;
    #5 C = 1;
    #5 C = 0;
    #5 EN = 0;
    #5 C = 1;

  end
```

```
// Prueba del FFD de dos bits
  initial begin
    #34
    $display("\n");
    $display("Flip Flop de 1 bit");
    $display("\n");
    $display("-------------------|-----|");
    $display(" clk  EN  reset  C  |  D  |");
    $display("-------------------|-----|");
    $monitor("%b    %b    %b    %b   |  %b  |", clk, EN, reset, C1, D1);

    #1 clk = 0; reset = 0; EN = 0; C = 2'b00;
    #5 reset = 1; EN = 1; C1 = 2'b00;
    #5 reset = 0; EN = 1; C1 = 2'b00;
    #5 C1 = 2'b11;
    #5 C1 = 2'b00;
    #5 EN = 0;
    #5 C1 = 2'b11;

  end

// Prueba del FFD de 4 bits
initial begin
  #66
  $display("\n");
  $display("Flip Flop de 1 bit");
  $display("\n");
  $display("---------------------|-------|");
  $display(" clk  EN  reset   C   |   D   |");
  $display("---------------------|-------|");
  $monitor("%b    %b    %b    %b   |  %b   | ", clk, EN, reset, C2, D2);

  #1 clk = 0; reset = 1; EN = 1; C2 = 4'b0000;
  #5 reset = 0; EN = 1; C2 = 4'b0000;
  #5 reset = 0; EN = 1; C2 = 4'b0000;
  #5 C2 = 4'b1111;
  #5 C2 = 4'b0000;
  #5 EN = 0;
  #5 C2 = 4'b1111;

end
```

```
  initial
    #100 $finish;

// GTKwave para el diagrama de timing
  initial begin
    $dumpfile("Ejercicio_01_tb.vcd");
    $dumpvars(0, testbench);
  end
endmodule
```

Para crear 3 Flip Flops tipo D de 1, 2 y 4 bits respectivamente se utilizaron los flancos del reloj, un reset y un enable. Por lo que, para crear un FF de 2 bits se utilizaron dos FF de 1 bit, y para el FF de 4 bits, se utilizaron 2 FF de 2 bits. Para finalizar, se realizó un testbench para verificar que los módulos funcionaran como debían.

Flip Flop de 1 bit

clk	EN	reset	C	D
x	x	x	x	x
0	1	0	0	x
1	1	0	0	0
0	1	0	0	0
1	1	0	0	0
0	1	0	0	0
1	1	1	0	0
0	1	1	0	0
1	1	1	0	0
0	1	1	0	0
1	1	1	0	0
0	1	0	0	0
1	1	0	0	0
0	1	0	0	0
1	1	0	0	0
0	1	0	0	0
1	1	0	1	1
0	1	0	1	1
1	1	0	1	1
0	1	0	1	1
1	1	0	1	1
0	1	0	0	1
1	1	0	0	0
0	1	0	0	0
1	1	0	0	0
0	1	0	0	0
1	0	0	0	0
0	0	0	0	0
1	0	0	0	0
0	0	0	0	0
1	0	0	0	0
0	0	0	1	0
1	0	0	1	0

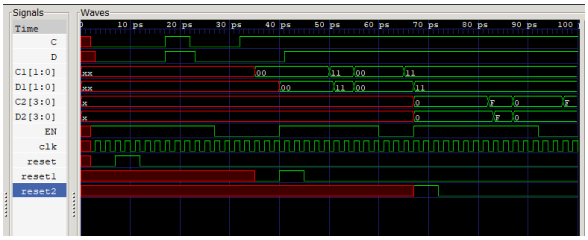
Flip Flop de 2 bit

clk	EN	reset	C	D
0	0	x	xx	xx
1	0	0	00	xx
0	0	0	00	xx
1	0	0	00	xx
0	0	0	00	xx
1	0	0	00	xx
0	1	1	00	00
1	1	1	00	00
0	1	1	00	00
1	1	1	00	00
0	1	1	00	00
1	1	0	00	00
0	1	0	00	00
1	1	0	00	00
0	1	0	00	00
1	1	0	00	00
0	1	0	11	00
1	1	0	11	11
0	1	0	11	11
1	1	0	11	11
0	1	0	11	11
1	1	0	00	00
0	1	0	00	00
1	1	0	00	00
0	1	0	00	00
1	1	0	00	00
0	0	0	00	00
1	0	0	00	00
0	0	0	00	00
1	0	0	00	00
0	0	0	00	00
1	0	0	11	00
0	0	0	11	00
1	0	0	11	00

Flip Flop de 4 bit

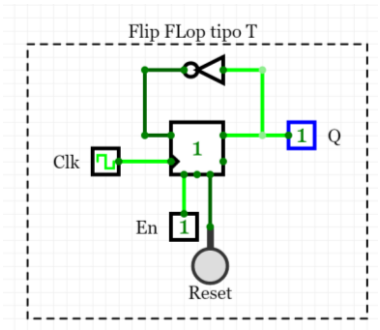
clk	EN	reset	C	D
0	0	x	xxxx	xxxx
1	1	1	0000	0000
0	1	1	0000	0000
1	1	1	0000	0000
0	1	1	0000	0000
1	1	1	0000	0000
0	1	0	0000	0000
1	1	0	0000	0000
0	1	0	0000	0000
1	1	0	0000	0000
0	1	0	0000	0000
1	1	0	0000	0000
0	1	0	0000	0000
1	1	0	0000	0000
0	1	0	0000	0000
1	1	0	0000	0000
0	1	0	1111	0000
1	1	0	1111	1111
0	1	0	1111	1111
1	1	0	1111	1111
0	1	0	1111	1111
1	1	0	0000	0000
0	1	0	0000	0000
1	1	0	0000	0000
0	1	0	0000	0000
1	1	0	0000	0000
0	0	0	0000	0000
1	0	0	0000	0000
0	0	0	0000	0000
1	0	0	0000	0000
0	0	0	0000	0000
1	0	0	1111	0000
0	0	0	1111	0000
1	0	0	1111	0000
0	0	0	1111	0000
1	0	0	1111	0000
0	0	0	1111	0000
1	0	0	1111	0000

GTKWAVE



Ejercicio 02

Circuitverse



Módulo de los Flip Flops tipo D y T

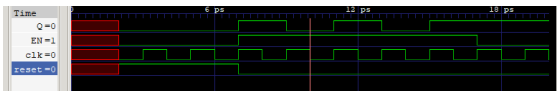
```
// Flip Flop D 1 bit
module FFD(input wire clk, EN, reset, input wire C, output reg D);
  always @ (posedge clk, posedge reset) begin

// Non-blockinf assigment
    if(reset) // Colocar todos los valores en 0
      D <= 1'b0;

    else if(EN == 1) // Enabled encendido cuenta cuenta cada flanco del reloj
      D <= C;
  end
endmodule

// Módulo flip flop tipo T
module FFT(input wire clk, EN, reset, output D);
    wire w;
    assign w = ~D;

    FFD A1(clk, EN, reset, w, D);
endmodule
```

Testbench

```
module testbench();

  reg clk, EN, reset;
  wire Q;

// Clock
  always
    #1 clk = ~clk;

// Llamar al módulo del FF tipo T
  FFT B1(clk, EN, reset, Q);

// Inicializar la Prueba
  initial begin
  #1
  $display("\n");
  $display(" Flip Flop tipo T de 1 bit ");
  $display("\n");
  $display("--------------|-----|");
  $display(" clk  EN  reset |  Q  |");
  $display("--------------|-----|");
  $monitor("%b    %b    %b    |  %b  |", clk, EN, reset, Q);

  #1 clk = 0; reset = 1; EN = 0;
  #5 reset = 0; EN = 1;
  #5 reset = 0; EN = 1;
  #5 EN = 0;

  end

  initial
    #20 $finish;

// GTKwave para el diagrama de timing
  initial begin
    $dumpfile("Ejercicio_02_tb.vcd");
    $dumpvars(0, testbench);
  end
endmodule
```

```
Flip Flop tipo T de 1 bit

---------------|-----|
clk  EN  reset  |  Q  |
---------------|-----|
x    x    x     |  x  |
0    0    1     |  0  |
1    0    1     |  0  |
0    0    1     |  0  |
1    0    1     |  0  |
0    0    1     |  0  |
1    1    0     |  1  |
0    1    0     |  1  |
1    1    0     |  0  |
0    1    0     |  0  |
1    1    0     |  1  |
0    1    0     |  1  |
1    1    0     |  0  |
0    1    0     |  0  |
1    1    0     |  1  |
0    1    0     |  1  |
1    0    0     |  1  |
0    0    0     |  1  |
1    0    0     |  1  |
0    0    0     |  1  |
gtkwave Ejercicio_02_tb.vc
```

GTKWAVE



Para crear un Flip Flop tipo T a partir de uno D se utilizó el código del FF de 1 bit, del ejercicio anterior y fue instanciado en el módulo del FF tipo T de 1 bit. Su funcionamiento consta de utilizar los flancos del reloj, para que, cuando varíen y el enable esté encendido, la entrada del FF sea Q negado.

Ejercicio 3

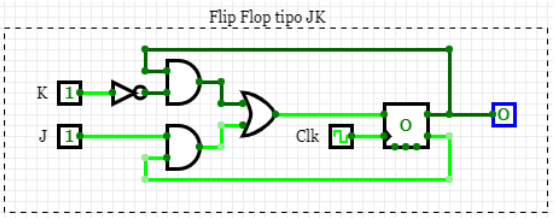
Tabla para el crear el Flip Flop tipo JK

Term	J	K	Q	=>	D
0	0	0	0		0
1	0	0	1		1
2	0	1	0		0
3	0	1	1		0
4	1	0	0		X
5	1	0	1		1
6	1	1	0		1
7	1	1	1		0

Ecuación minimizada

```
Entered by truthtable:
D = J' K' Q + J K' Q + J K Q';

Minimized:
D = K' Q + J Q';
```

Circuitverse



Módulo

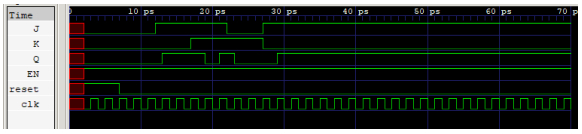
```
// Flip Flop D 1 bit

module FFD(input wire clk, EN, reset, input wire C, output reg D);
  always @ (posedge clk, posedge reset) begin

// Non-blockinf assigment
    if(reset) // Colocar todos los valores en 0
      D <= 1'b0;

    else if(EN == 1) // Enabled encendido cuenta cuenta cada flanco del reloj
      D <= C;
  end
endmodule


// Flip FLop JK

module FFJK(input wire clk, EN, reset, input wire J, K, output Q);
  wire D, QN, KN, Y1, Y2;

  not(QN, Q);
  not(KN, K);
  and(Y1, KN, Q);
  and(Y2, J, QN);
  or(D, Y1, Y2);

  FFD A1(clk, EN, reset, D, Q);

endmodule
```

Testbench

```
module testbench();

  // inputs
  reg clk, EN, reset;
  reg J, K;

  // Outputs
  wire Q;

  // Clock
  always
    #1 clk = ~clk;

  // Instanciar el módulo
  FFJK A1(clk, EN, reset, J, K, Q);

  initial begin
    #1
    $display("\n");
    $display("Flip Flop JK");
    $display("\n");
    $display("-----------------------|-----|");
    $display(" clk  EN  reset  J   K   | Q  |");
    $display("-----------------------|-----|");
    $monitor("%b    %b    %b     %b   %b   |  %b  |", clk, EN, reset, J, K, Q);

    #1 clk = 0; reset = 1; EN = 1; J = 0; K = 0;
    #5 reset = 0; EN = 1;
    #5 J = 1;
    #5 K = 1;
    #5 J = 0;
    #5 J = 1; K = 0;

  end

    initial
      #70 $finish;

  // GTKwave para el diagrama de timing
    initial begin
      $dumpfile("Ejercicio_003_tb.vcd");
      $dumpvars(0, testbench);
    end
endmodule
```

```
Flip Flop JK


------------------------|-----|
clk  EN  reset  J   K   |  Q  |
------------------------|-----|
x    x    x     x   x   |  x  |
0    1    1     0   0   |  0  |
1    1    1     0   0   |  0  |
0    1    1     0   0   |  0  |
1    1    1     0   0   |  0  |
0    1    1     0   0   |  0  |
1    1    0     0   0   |  0  |
0    1    0     0   0   |  0  |
1    1    0     0   0   |  0  |
0    1    0     0   0   |  0  |
1    1    0     0   0   |  0  |
0    1    0     1   0   |  0  |
1    1    0     1   0   |  1  |
0    1    0     1   0   |  1  |
1    1    0     1   0   |  1  |
0    1    0     1   0   |  1  |
1    1    0     1   1   |  1  |
0    1    0     1   1   |  1  |
1    1    0     1   1   |  0  |
0    1    0     1   1   |  0  |
1    1    0     1   1   |  1  |
0    1    0     0   1   |  1  |
1    1    0     0   1   |  0  |
0    1    0     0   1   |  0  |
1    1    0     0   1   |  0  |
0    1    0     0   1   |  0  |
1    1    0     1   0   |  0  |
0    1    0     1   0   |  0  |
1    1    0     1   0   |  1  |
0    1    0     1   0   |  1  |
1    1    0     1   0   |  1  |
0    1    0     1   0   |  1  |
1    1    0     1   0   |  1  |
0    1    0     1   0   |  1  |
1    1    0     1   0   |  1  |
0    1    0     1   0   |  1  |
1    1    0     1   0   |  1  |
0    1    0     1   0   |  1  |
```

GTKWAVE



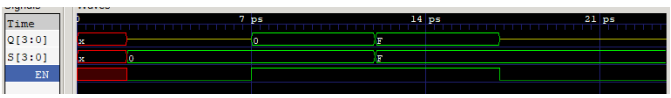
Para crear un Flip Flop tipo JK se utilizó lógica combinacional con el módulo del Flip Flop tipo D de un bit. Se requirieron 2 compuertas not, and y una or. Para finalizar, se realizó un testbench para verificar el funcionamiento del mismo.

Testbench

```systemverilog
module testbench();
  reg EN;
  reg [3:0]S;
  wire [3:0]Q;

// Instanciar el módulo
  BT A1(EN, S, Q);

// Inicialización de la prueba del Buffer
  initial begin
    #1
    $display("\n");
    $display("Buffer triestado de 4 bits");
    $display("\n");
    $display("------------|--------|");
    $display(" EN   S      |  Q     |");
    $display("------------|--------|");
    $monitor("%b   %b    |  %b  |", EN, S, Q);

    #1 EN = 0; S = 4'b0000;
    #5 EN = 1;
    #5 S = 4'b1111;
    #5 EN = 0;

    end

    initial
      #25 $finish;

  // GTKwave para el diagrama de timing
  initial begin
    $dumpfile("Ejercicio_4_tb.vcd");
    $dumpvars(0, testbench);
  end
endmodule
```

```
Buffer triestado de 4 bits


------------|--------|
EN   S      |  Q     |
------------|--------|
x   xxxx    |  xxxx  |
0   0000    |  zzzz  |
1   0000    |  0000  |
1   1111    |  1111  |
0   1111    |  zzzz  |
gtkwave Ejercicio_4_tb.vcd Ejercicio_4_tb.gtkw
```

GTKWAVE



Ejercicio 4

Módulo del buffer triestado

```systemverilog
module BT(input wire EN,
  input wire [3:0]S, // Entradas
  output wire [3:0]Q); // Salidas

    assign Q = (EN) ? S : 4'bz; // Cuando EN = 0 la salida está en alta impedancia
endmodule
```

Para crear un buffer triestado se requirió de un enable, un input de 4 bits, un output de 4 bits y se tomó en cuenta que su funcionamiento se basa en que la salida es igual a la entrada a menos que enable esté apagado, ya que, en ese momento, se evidencia alta impedancia como es visto de color Amarillo en el diagrama de timing.

Ejercicio 5

Módulo

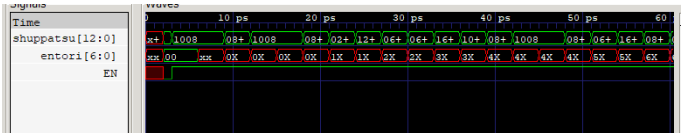
```
// Implementación de la tabla como memoria ROM utilizando case

module MROM(input wire EN,
            input wire [6:0]entori,
            output reg [12:0]shuppatsu);

              always @ (EN or entori)begin
                  shuppatsu = 0;
                if (EN)
                    casex (entori)

                        7'bxxxxxx0: shuppatsu <= 13'b1000000001000;
                        7'b00001x1: shuppatsu <= 13'b0100000001000;
                        7'b00000x1: shuppatsu <= 13'b1000000001000;
                        7'b00011x1: shuppatsu <= 13'b1000000001000;
                        7'b00010x1: shuppatsu <= 13'b0100000001000;
                        7'b0010xx1: shuppatsu <= 13'b0001001000010;
                        7'b0011xx1: shuppatsu <= 13'b1001001100000;
                        7'b0100xx1: shuppatsu <= 13'b0011010000010;
                        7'b0101xx1: shuppatsu <= 13'b0011010000100;
                        7'b0110xx1: shuppatsu <= 13'b1011010100000;
                        7'b0111xx1: shuppatsu <= 13'b1000000111000;
                        7'b1000x11: shuppatsu <= 13'b0100000001000;
                        7'b1000x01: shuppatsu <= 13'b1000000001000;
                        7'b1001x11: shuppatsu <= 13'b1000000001000;
                        7'b1001x01: shuppatsu <= 13'b0100000001000;
                        7'b1010xx1: shuppatsu <= 13'b0011011000010;
                        7'b1011xx1: shuppatsu <= 13'b1011011100000;
                        7'b1100xx1: shuppatsu <= 13'b0100000001000;
                        7'b1101xx1: shuppatsu <= 13'b0000000001001;
                        7'b1110xx1: shuppatsu <= 13'b0011100000010;
                        7'b1111xx1: shuppatsu <= 13'b1011100100000;

                    endcase
                end
endmodule
```

Testbench

```
module testbench();

  reg EN;
  reg [6:0]entori;
  wire [12:0]shuppatsu;

  // Instanciar el módulo
  MROM A1(EN, entori, shuppatsu);

  initial begin
    #1
    $display("\n");
    $display("Memoria ROM");
    $display("\n");
    $display("---------------|-----------------|");
    $display(" EN    entori   |   shuppatsu    |");
    $display("---------------|-----------------|");
    $monitor("%b   %b    | %b  |", EN, entori, shuppatsu);

    #1 EN = 0; entori = 7'b0000000;
    #1 EN = 1;
    #3 entori = 7'bxxxxxx0;
    #3 entori = 7'b00001x1;
    #3 entori = 7'b00000x1;
    #3 entori = 7'b00011x1;
    #3 entori = 7'b00010x1;
    #3 entori = 7'b0010xx1;
    #3 entori = 7'b0011xx1;
    #3 entori = 7'b0100xx1;
    #3 entori = 7'b0101xx1;
    #3 entori = 7'b0110xx1;
    #3 entori = 7'b0111xx1;
    #3 entori = 7'b1000x11;
    #3 entori = 7'b1000x01;
    #3 entori = 7'b1001x11;
    #3 entori = 7'b1001x01;
    #3 entori = 7'b1010xx1;
    #3 entori = 7'b1011xx1;
    #3 entori = 7'b1100xx1;
    #3 entori = 7'b1101xx1;
    #3 entori = 7'b1110xx1;
    #3 entori = 7'b1111xx1;

    end

    initial
      #69 $finish;

  // GTKwave para el diagrama de timing
  initial begin
  $dumpfile("Ejercicio_5_tb.vcd");
  $dumpvars(0, testbench);
  end
  endmodule
```

```
---------------|-----------------|
EN    entori   |   shuppatsu     |
---------------|-----------------|
x   xxxxxxx    |  xxxxxxxxxxxxx  |
0   0000000    |  0000000000000  |
1   0000000    |  1000000001000  |
1   xxxxxx0    |  1000000001000  |
1   00001x1    |  0100000001000  |
1   00000x1    |  1000000001000  |
1   00011x1    |  1000000001000  |
1   00010x1    |  0100000001000  |
1   0010xx1    |  0001001000010  |
1   0011xx1    |  1001001100000  |
1   0100xx1    |  0011010000010  |
1   0101xx1    |  0011010000100  |
1   0110xx1    |  1011010100000  |
1   0111xx1    |  1000000111000  |
1   1000x11    |  0100000001000  |
1   1000x01    |  1000000001000  |
1   1001x11    |  1000000001000  |
1   1001x01    |  0100000001000  |
1   1010xx1    |  0011011000010  |
1   1011xx1    |  1011011100000  |
1   1100xx1    |  0100000001000  |
1   1101xx1    |  0000000001001  |
1   1110xx1    |  0011100000010  |
1   1111xx1    |  1011100100000  |
gtkwave Ejercicio_5_tb.vcd Ejercici
```

GTKWAVE



Para implementar la tabla proporcionada como memoria ROM se utilizó la función de case que permite establecer todas las condiciones sin la necesidad de utilizar el commando de if para cada una. Asimismo, se realizó un módulo de testbench para corroborar que el funcionamiento del mismo era el correcto. Para esto, se probaron 21 posibles combinaciones y en efecto, el funcionamiento era el debido.

Link al repositorio:

https://github.com/valeelorraine/Laboratorios_Digital

Link a circuitverse:

https://circuitverse.org/users/29247/projects/laboratorio-09