Laboratorio 9 - Electrónica Digital 1

- Ejercicio 1 - Flip Flop tipo D de 1 bit

Este ejercicio consistió en la construcción de un Flip Flop tipo D de 1 bit, el cual debía contar con un enable, que al estar encendido, permitiera pasar la señal de entrada. A partir de este Flip Flop tipo D de 1 bit, se partió para la construcción de un Flip Flop tipo D de 2 y 4 bits.

Ejercicio1.v | Ejercicio2.v | Ejercicio3.v | Ejercicio4.v | Ejercicio5.v

Ejercicio1 > Ejercicio1.v

```
// Luis Pedro Molina Velásquez
// Carné 18822 - Sección 12
// Laboratorio 9
// Ejercicio 1 - Flip Flops tipo D

// Flip Flop tipo D de 1 bit
module FFD1B(input wire D, input wire clk, input wire reset, input wire enable, output reg Q);

    always @ (posedge clk or posedge reset) begin
        if (reset)
            Q <= 1'b0;
        else if (enable)
            Q <= D;
        end

endmodule

// Flip Flop tipo D de 2 bits
module FFD2B(input wire [1:0]D, input wire clk, input wire reset, input wire enable, output [1:0]Q);
FFD1B FFD1B_1 (D[0], clk, reset, enable, Q[0]);
FFD1B FFD1B_2 (D[1], clk, reset, enable, Q[1]);
endmodule

// Flip Flop tipo D de 4 bits
module FFD4B(input wire [3:0]D, input wire clk, input wire reset, input wire enable, output [3:0]Q);
FFD2B FFD2B_1 (D[1:0], clk, reset, enable, Q[1:0]);
FFD2B FFD2B_2 (D[3:2], clk, reset, enable, Q[3:2]);
endmodule
```

Módulo Flip Flop tipo D

Ejercicio1_tb.v | Ejercicio2_tb.v | Ejercicio3_tb.v | Ejercicio4_tb.v | Ejercicio5_tb.v

Ejercicio1 > Ejercicio1_tb.v

```
module testbench ();

// Flip Flop tipo D de 4 bits
reg [3:0]D;
reg clk, reset, enable;
wire [3:0]Q;

FFD4B ffd4b (D, clk, reset, enable, Q);

initial begin
   clk = 1 ;
   forever #1 clk = ~clk ;
end

    initial begin

        #1
        $display (" ");
        $display(" Ejercicio 1 - Flip Flops tipo D");
        $display(" ");
        $display(" Reset  Enable     D  |  Q ");
        $display("--------------------|-----");
        $monitor("  %b        %b       %b  |  %b", reset, enable, D, Q);
           reset = 0 ; enable = 0 ; D = 4'b0000;
        #1 reset = 1 ; enable = 0 ; D = 4'b0000;
        #1 reset = 0 ; enable = 0 ; D = 4'b0001;
        #1 reset = 0 ; enable = 1 ; D = 4'b1000;
        #1 reset = 0 ; enable = 0 ; D = 4'b0110;
        #1 reset = 0 ; enable = 0 ; D = 4'b0001;
        #1 reset = 0 ; enable = 0 ; D = 4'b0101;
        #1 reset = 0 ; enable = 1 ; D = 4'b1001;
        #1 reset = 0 ; enable = 1 ; D = 4'b0110;
        #1 reset = 0 ; enable = 1 ; D = 4'b0001;
        #1 reset = 0 ; enable = 1 ; D = 4'b0011;
        #1 reset = 1 ; enable = 0 ; D = 4'b0100;
        #1 reset = 0 ; enable = 1 ; D = 4'b1000;
    end

    initial
    #25  $finish;

initial
    begin
        $dumpfile("Ejercicio1_tb.vcd");
        $dumpvars(0, testbench);
    end
```

Módulo testbench Flip Flop tipo D

Ejercicio 1 - Flip Flops tipo D

Reset	Enable	D	Q
0	0	0000	xxxx
1	0	0000	0000
0	0	0001	0000
0	1	1000	0001
0	0	0110	0001
0	0	0001	0001
0	0	0101	0001
0	1	1001	0101
0	1	0110	0101
0	1	0001	0110
0	1	0011	0110
1	0	0100	0000
0	1	1000	0000
0	1	1000	1000

Tabla 1

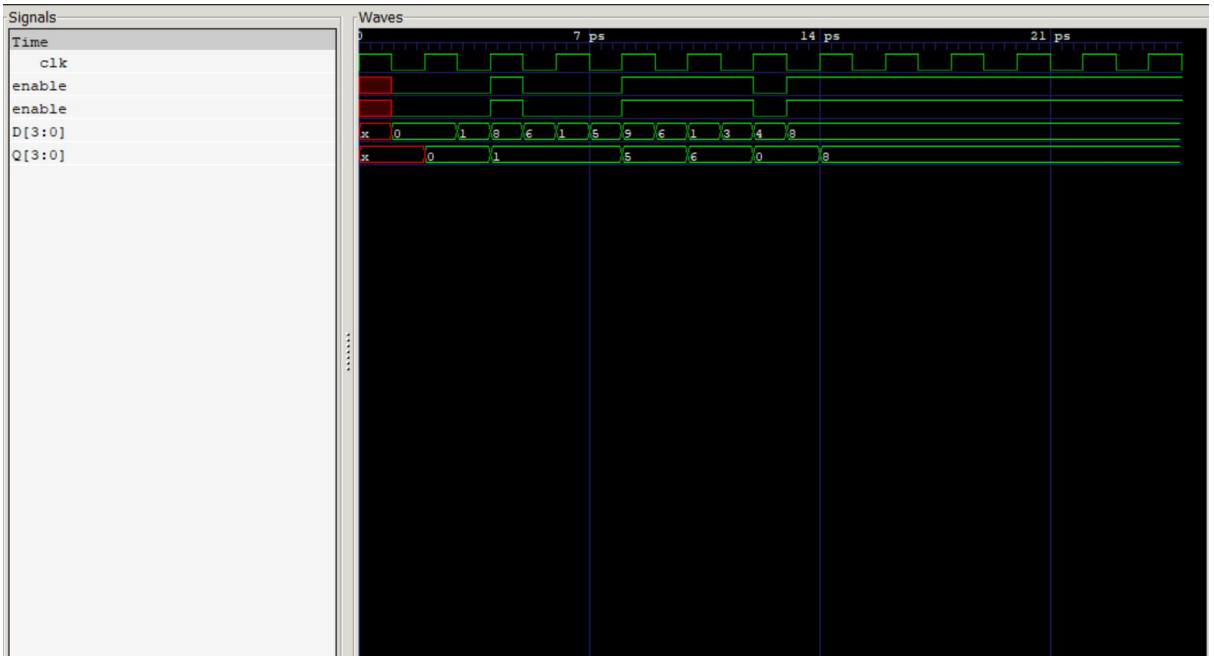


Diagrama de timing - Flip Flop tipo D

- Ejercicio 2 - Flip Flop tipo T de 1 bit

Este ejercicio consistió en la construcción de un Flip Flop tipo T de 1 bit, basado en la estructura de un Flip Flop tipo D de 1 bit. El funcionamiento de dicho Flip Flop consiste en ir variando con los flancos de reloj siempre que el enable esté activado, con una duración de 2 flancos, es decir, 1 periodo. En este caso la señal de entrada es ~Q.

Ejercicio2 > Ejercicio2.v

```
// Luis Pedro Molina Velásquez
// Carné 18822 - Sección 12
// Laboratorio 9
// Ejercicio 2 - Flip Flop tipo T

// Flip Flop tipo D de 1 bit
module FFD1B (input wire D, input wire clk, input wire reset, input wire enable, output reg Q);

    always @ (posedge clk or posedge reset) begin
        if (reset)
            Q <= 1'b0;
        else if (enable)
            Q <= D;
    end

endmodule

// Flip Flop tipo T de 1 bit
module FFTT1B (input wire clk, input wire reset, input wire enable, output Q);
FFD1B ffd1b (~Q, clk, reset, enable, Q);

endmodule
```

Módulo Flip Flop tipo T

Ejercicio2 > Ejercicio2_tb.v

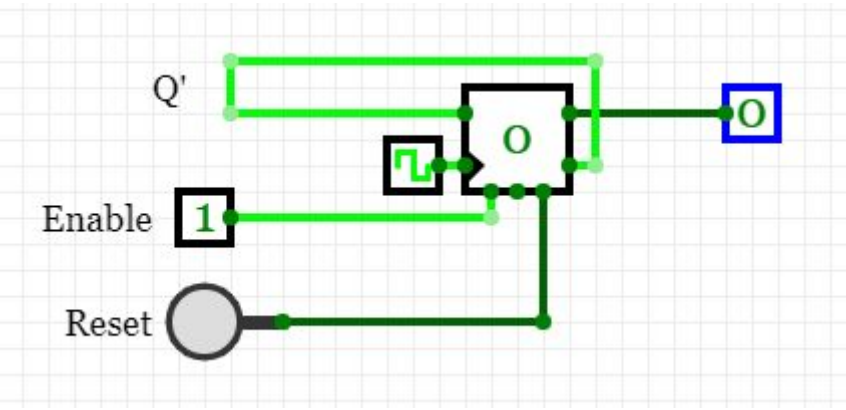
```
module testbench ();

reg clk, reset, enable;
wire Q;

FFTT1B fftt1b (clk, reset, enable, Q);

initial begin
    clk = 1 ;
    forever #1 clk = ~clk ;
end

    initial begin

        #1
        $display (" ");
        $display(" Ejercicio 2 - Flip Flop tipo T");
        $display(" ");
        $display(" Reset  Enable  |  Q ");
        $display("--------------|-----");
        $monitor("  %b        %b      |  %b", reset, enable, Q);
           reset = 0 ; enable = 0 ;
        #1 reset = 1 ; enable = 0 ;
        #1 reset = 0 ; enable = 1 ;
        #1 reset = 0 ; enable = 0 ;
        #1 reset = 0 ; enable = 0 ;
        #1 reset = 0 ; enable = 1 ;
        #1 reset = 0 ; enable = 1 ;
        #1 reset = 0 ; enable = 1 ;
        #1 reset = 0 ; enable = 1 ;
        #1 reset = 1 ; enable = 0 ;
        #1 reset = 0 ; enable = 0 ;
        #1 reset = 0 ; enable = 1 ;
        #1 reset = 0 ; enable = 1 ;
        #1 reset = 0 ; enable = 1 ;


    end

    initial
    #30  $finish;

initial
    begin
        $dumpfile("Ejercicio2_tb.vcd");
        $dumpvars(0, testbench);
```

Módulo testbench Flip Flop tipo T

Ejercicio 2 - Flip Flop tipo T

Reset	Enable	Q
0	0	x
1	0	0
0	1	0
0	0	0
0	1	1
0	1	0
1	0	0
0	0	0
0	1	1
0	1	0
0	1	1
0	1	0
0	1	1
0	1	0
0	1	1
0	1	0
0	1	1
0	1	0

Tabla 2



Flip Flop tipo T

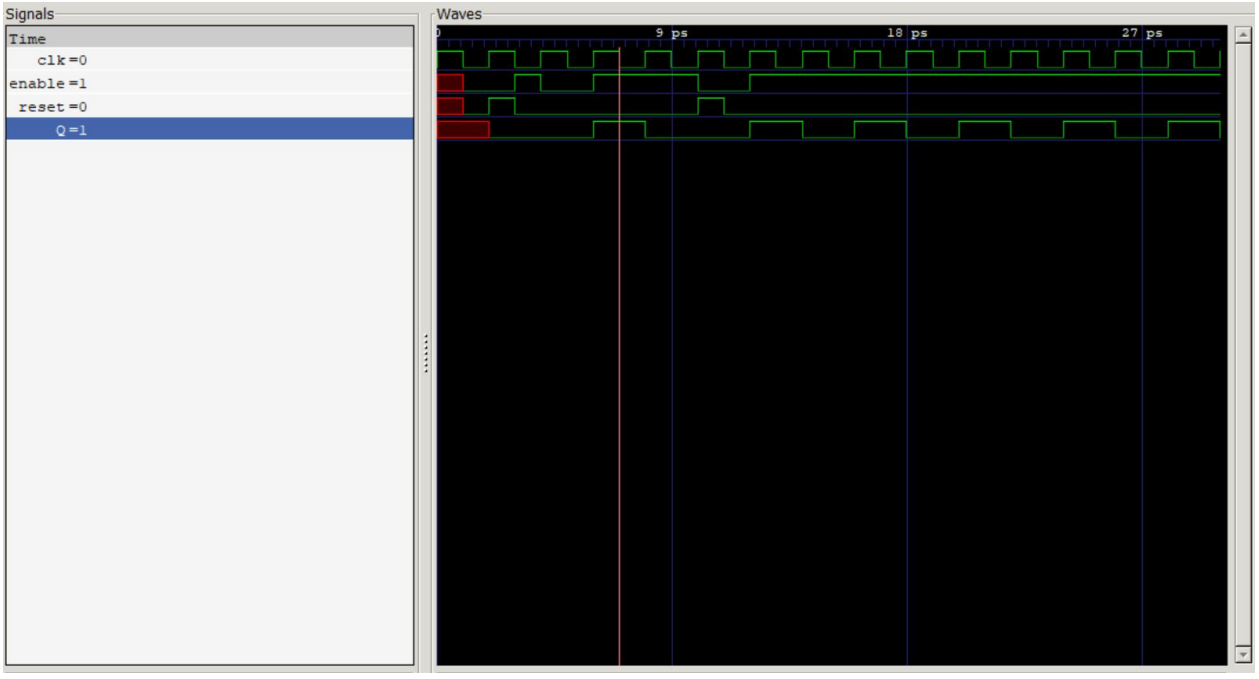
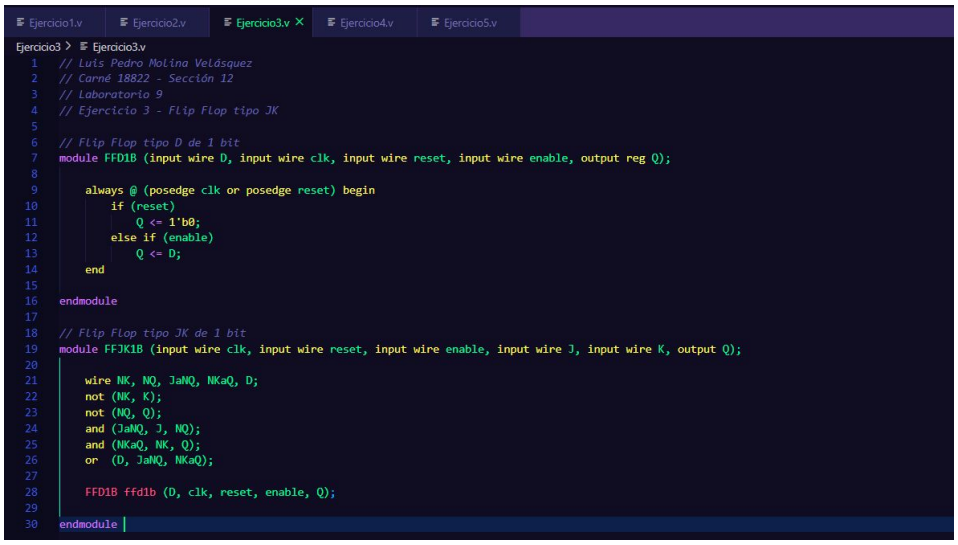


Diagrama de timing - Flip Flop tipo T

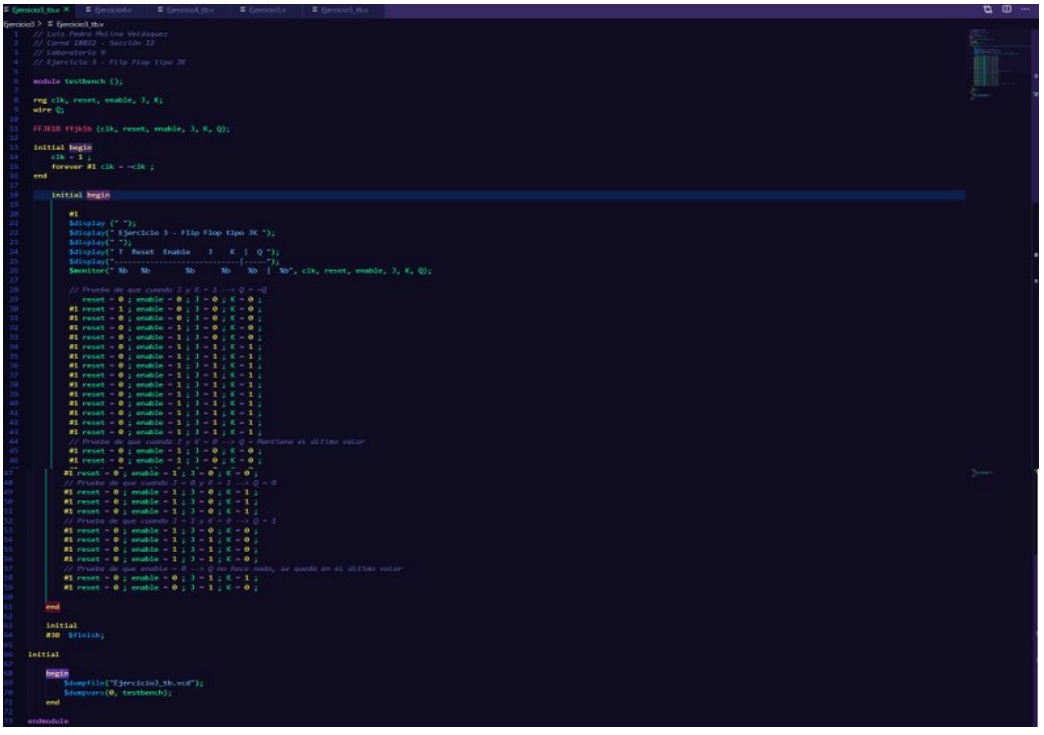
- Ejercicio 3 - Flip Flop tipo JK

Este ejercicio consistió en la construcción de un Flip Flop tipo JK de 1 bit. Al igual que en el anterior, se partió de un Flip Flop tipo D de 1 bit. El funcionamiento de este Flip Flop es el siguiente:

- ❏ J y K = 0 → Q = Mantiene su valor anterior
- ❏ J = 1 y K = 0 → Q = 1
- ❏ J = 0 y K = 1 → Q = 0
- ❏ J y K = 1 → Q = El valor de Q va alternando en cada flanco de reloj. Q → ~Q



Módulo Flip Flop JK

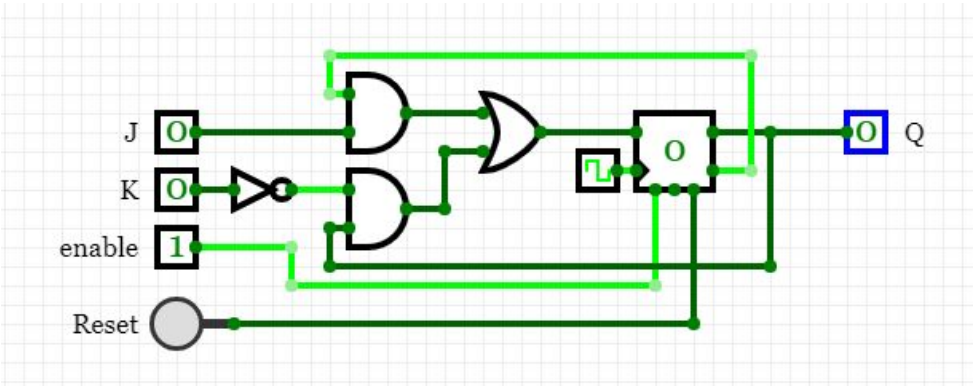


Módulo testbench Flip Flop JK

Ejercicio 3 - Flip Flop tipo JK

T	Reset	Enable	J	K	Q
0	0	0	0	0	x
1	1	0	0	0	0
0	0	0	0	0	0
1	0	1	0	0	0
0	0	1	0	0	0
1	0	1	1	1	0
0	0	1	1	1	0
1	0	1	1	1	1
0	0	1	1	1	1
1	0	1	1	1	0
0	0	1	1	1	0
1	0	1	1	1	1
0	0	1	1	1	1
1	0	1	1	1	0
0	0	1	1	1	0
1	0	1	0	0	1
0	0	1	0	0	1
1	0	1	0	0	1
0	0	1	0	1	1
1	0	1	0	1	0
0	0	1	0	1	0
1	0	1	0	0	0
0	0	1	1	0	0
1	0	1	1	0	1
0	0	1	1	0	1
1	0	0	1	1	1
0	0	0	1	0	1
1	0	0	1	0	1
0	0	0	1	0	1
1	0	0	1	0	1

Tabla 3



Flip Flop tipo JK

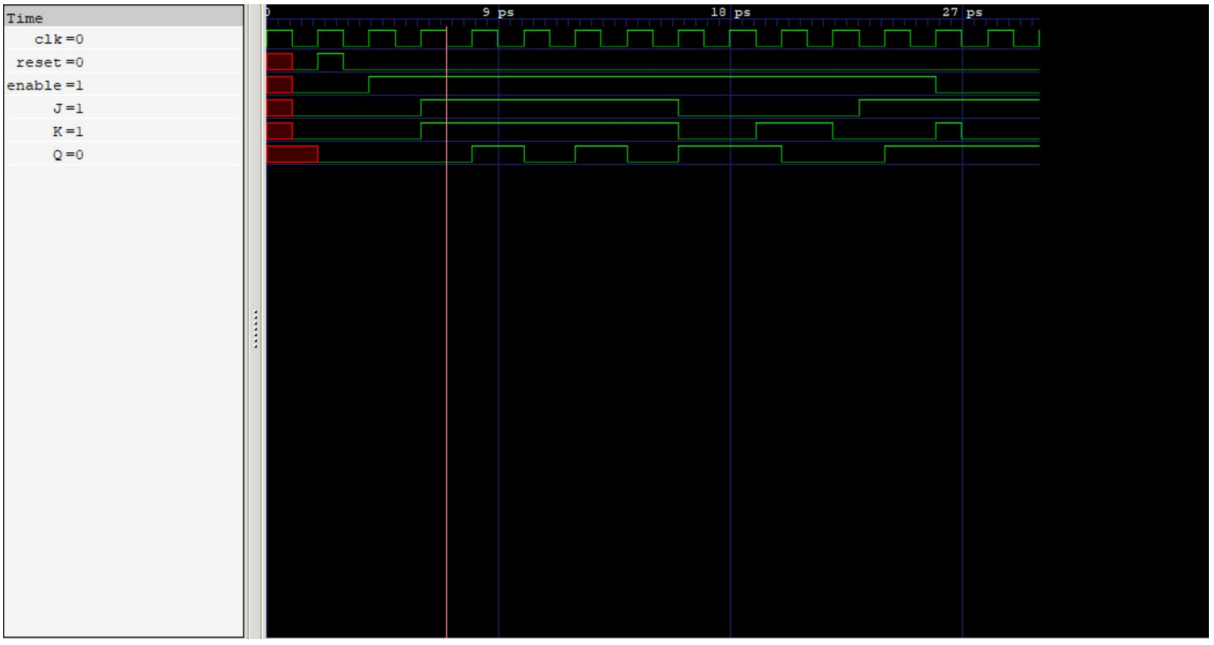
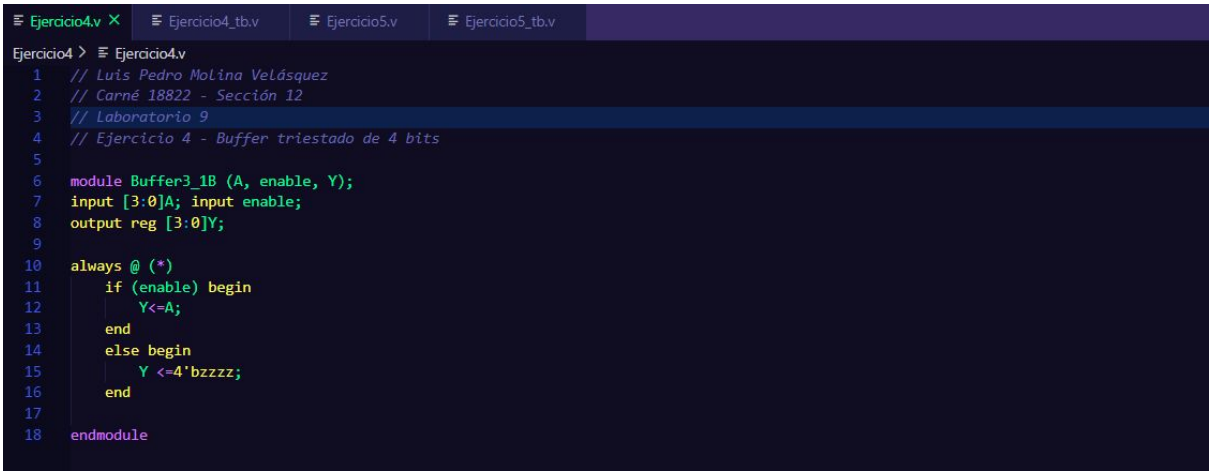


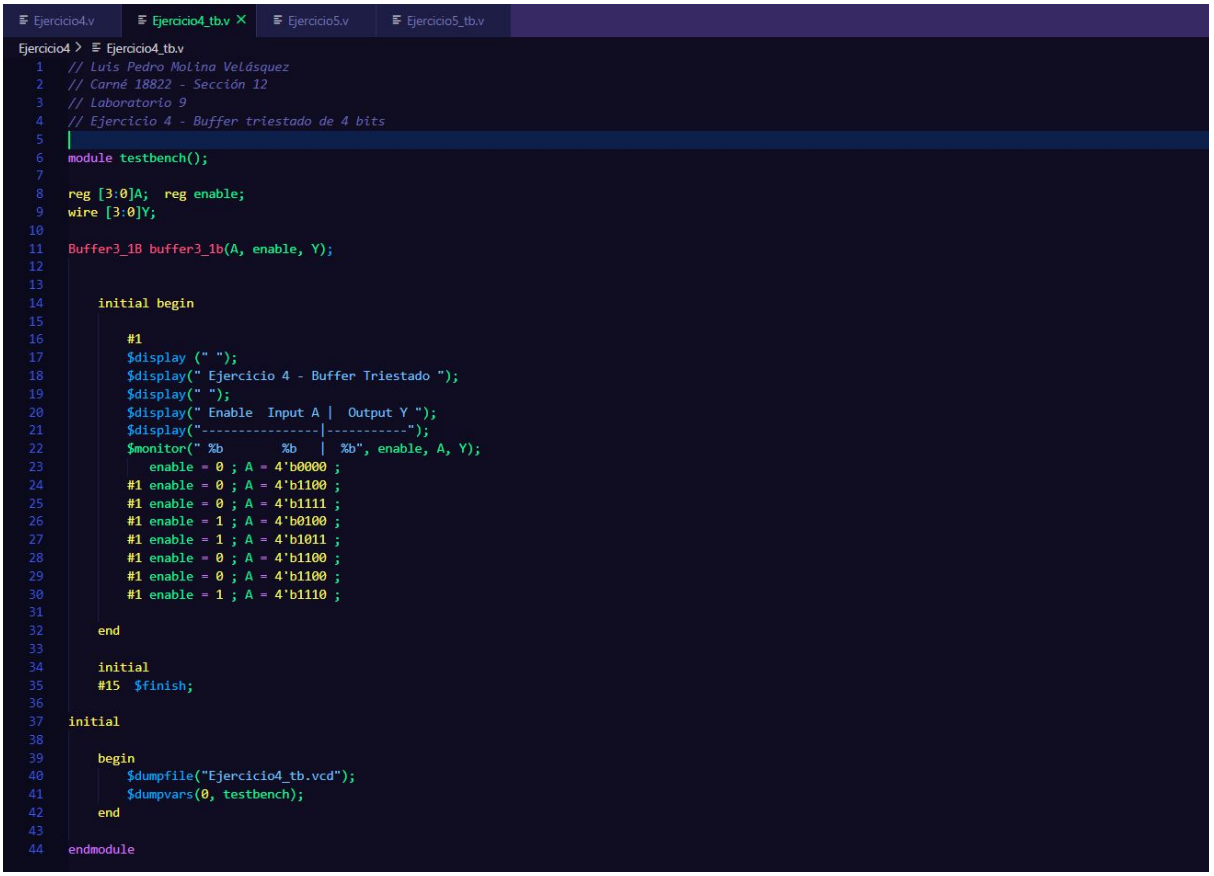
Diagrama de timing- Flip Flop JK

- Ejercicio 4 - Buffer triestado

Este ejercicio consistió en la construcción de un Buffer Triestado de 4 bits. El funcionamiento es básicamente que cuando el enable esté activado, la señal de entrada pasa a ser la señal de salida, pero cuando el enable está apagado, la señal de salida esta en alta impedancia.



Módulo Buffer triestado



Módulo testbench Buffer triestado

Ejercicio 4 - Buffer Triestado

Enable	Input A	Output Y
0	0000	zzzz
0	1100	zzzz
0	1111	zzzz
1	0100	0100
1	1011	1011
0	1100	zzzz
1	1110	1110

Tabla 4

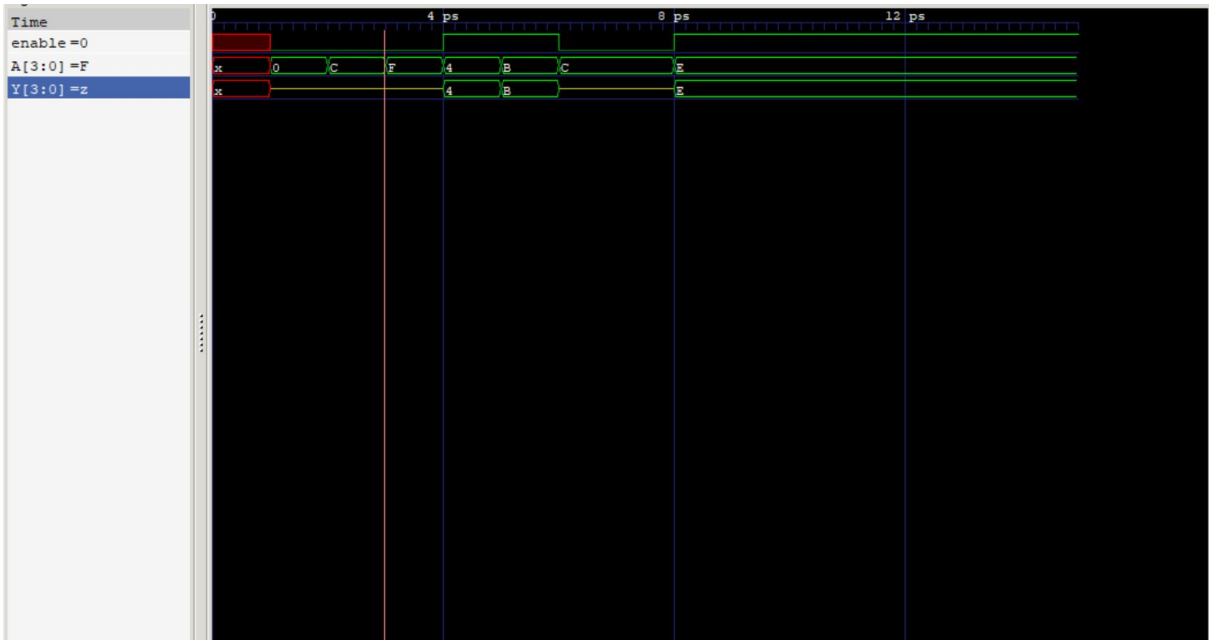
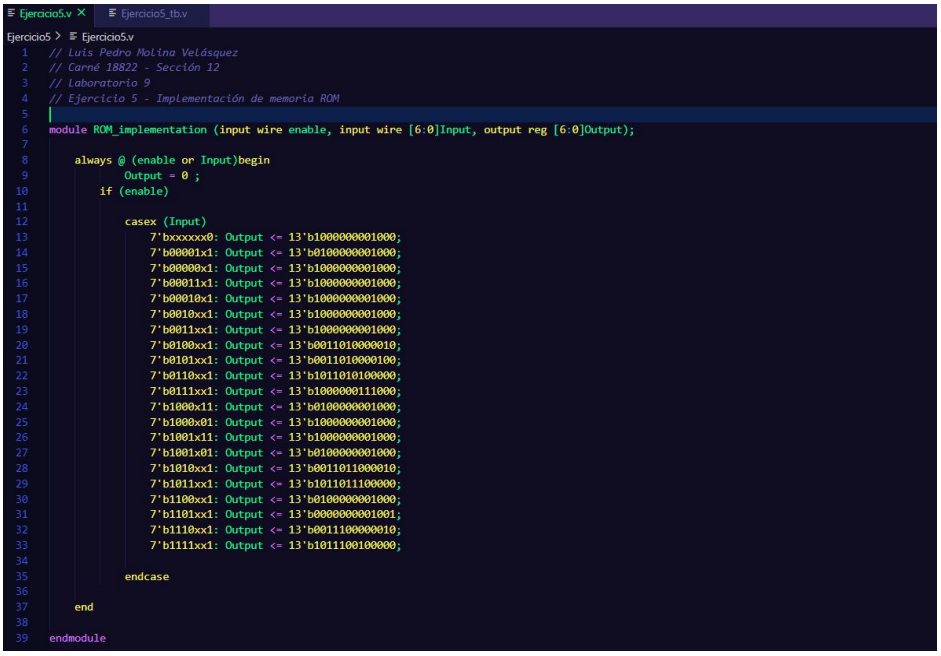


Diagrama de timing - Buffer triestado

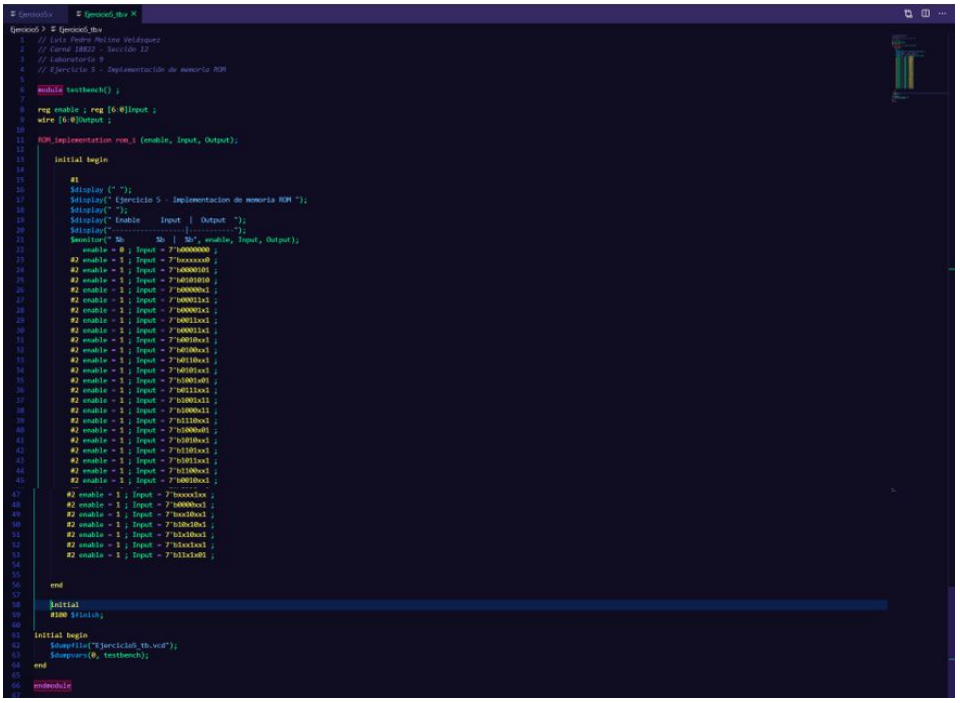
- Ejercicio 5 - Implementación de memoria ROM

Este ejercicio consistió en la construcción de una memoria ROM de 7 bits de entrada y 13 bits de salida para luego implementarla con ciertos parámetros. En este caso, básicamente era darle una instrucción a la memoria, es decir, indicarle la localidad requerida para que la memoria nos desplegara lo que sea que estuviera en dicha localidad. Se utilizó un casex, con el propósito de tener la posibilidad de asignarle don't cares (x) a los módulos de la memoria y que fuera capaz de interpretarlos.



```v
// Luis Pedro Molina Velásquez
// Carné 18822 - Sección 12
// Laboratorio 9
// Ejercicio 5 - Implementación de memoria ROM

module ROM_implementation (input wire enable, input wire [6:0]Input, output reg [6:0]Output);

    always @ (enable or Input)begin
            Output = 0 ;
        if (enable)

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

            endcase

    end

endmodule
```

Módulo de memoria ROM



Módulo testbench memoria ROM

Ejercicio 5 - Implementacion de memoria ROM

Enable	Input	Output
0	0000000	0000000
1	xxxxxx0	0001000
1	0000101	0001000
1	0101010	0001000
1	00000x1	0001000
1	00011x1	0001000
1	00001x1	0001000
1	0011xx1	0001000
1	00011x1	0001000
1	0010xx1	0001000
1	0100xx1	0000010
1	0110xx1	0100000
1	0101xx1	0000100
1	1001x01	0001000
1	0111xx1	0111000
1	1001x11	0001000
1	1000x11	0001000
1	1110xx1	0000010
1	1000x01	0001000
1	1010xx1	1000010
1	1101xx1	0001001
1	1011xx1	1100000
1	1100xx1	0001000
1	0010xx1	0001000
1	0110xx1	0100000
1	xxxx1xx	0001000
1	0000xx1	0001000
1	xx10xx1	0001000
1	10x10x1	0001000
1	1x10xx1	1000010
1	1xx1xx1	0001000
1	11x1x01	0001001

Tabla 5

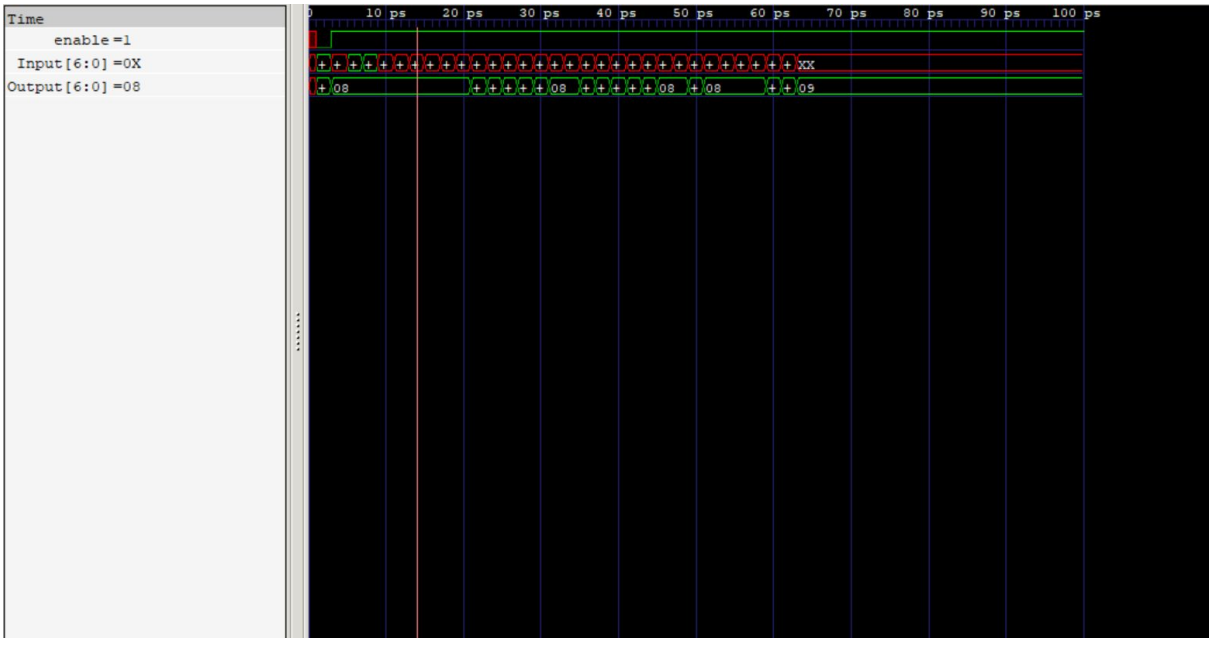


Diagrama de timing - Memoria ROM