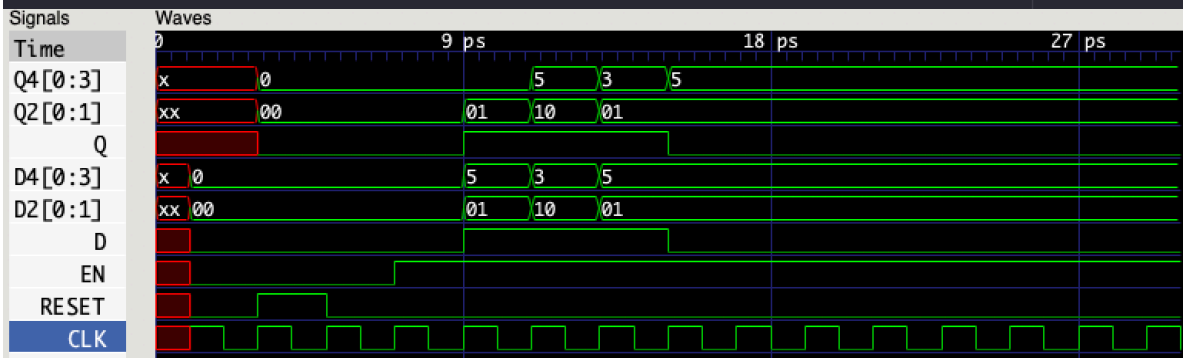
Universidad del Valle de Guatemala
Electronica Digital 1
Kurt Kellner
Josue Salazar
19420
Laboratorio 9

Ejercicio 1

uarto Semestre /Electronica /Lab 3/Laboratorios-/LAB09/Ejercicio1 lab9/Ejercicio1.v

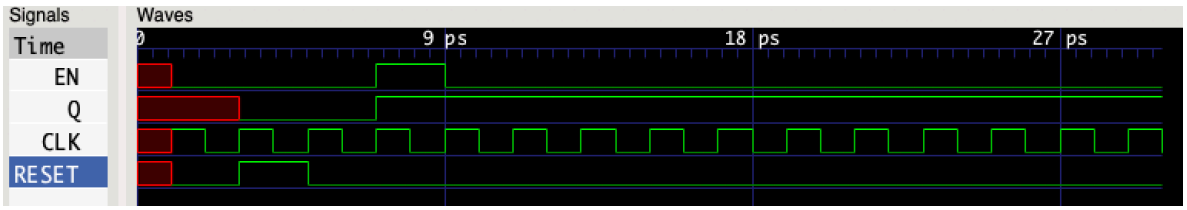
```
module flipD1(input wire CLK, RESET, EN, D, output reg Q);
  always @ (posedge CLK or posedge RESET)begin
    if(RESET)
      Q <= 1'b0;
    else if (EN)
      Q <= D;
  end
endmodule


//Flip FLop de 2 bits
module flipD2(input wire CLK, RESET, EN, input wire [0:1]D2, output wire [0:1]Q2);

flipD1 F2_1(CLK, RESET, EN, D2[0], Q2[0]);
flipD1 F2_2(CLK, RESET, EN, D2[1], Q2[1]);
endmodule

//Flip FLop de 4 bits
module flipD4(input wire CLK, RESET, EN, input wire [0:3]D4, output wire [0:3]Q4);
flipD2 F4_1(CLK, RESET, EN, D4[0:1], Q4[0:1]);
flipD2 F4_2(CLK, RESET, EN, D4[2:3], Q4[2:3]);
endmodule
```

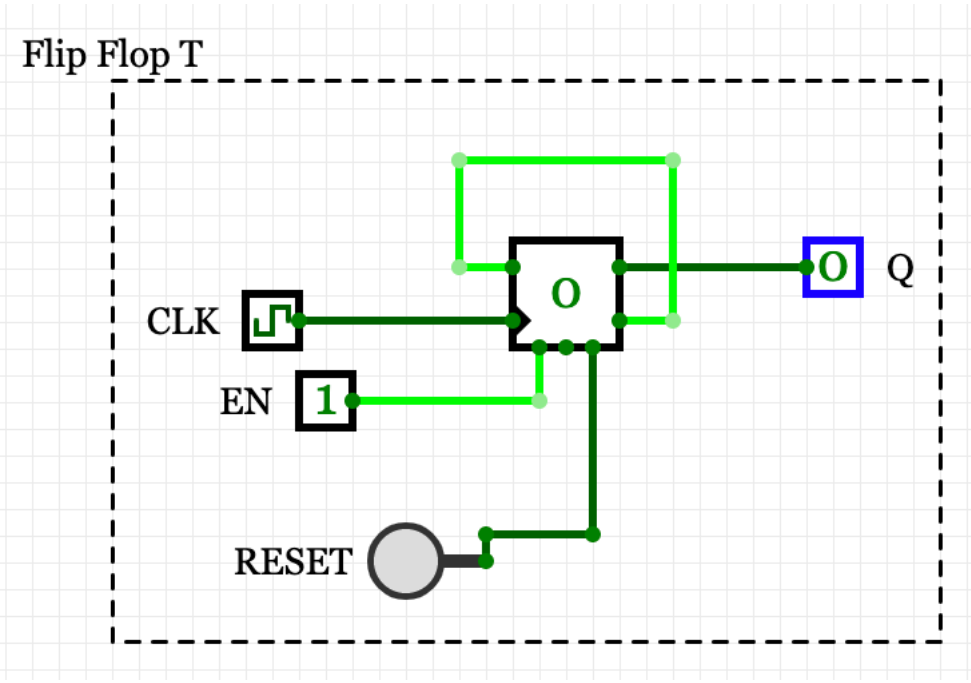


En este ejercicio se realizaron 3 Flip Flops, empezando con un bit, el cual se implemento en un flip flop para volverlo de 2 bits y la misma mecanica para volverlo en 4 bits, en el 3 Flip flop se implementaron 2 Flilp Flops de 2 bits.
Se puede decir que la estructura es la misma, las variaciones estan en el numero de entradas y salidas, y solo el primer FF poseia la salida “reg”.

Ejercicio 2

```
module flipD(input wire CLK, RESET, EN, D, output reg Q);
  always @ (posedge CLK or posedge RESET)begin
    if(RESET)
      Q <= 1'b0;
    else if (EN)
      Q <= D;
  end
endmodule

module flipT(input wire CLK, RESET, EN, output wire Q);
flipD FT(CLK, RESET, EN, ~Q, Q);
endmodule
```

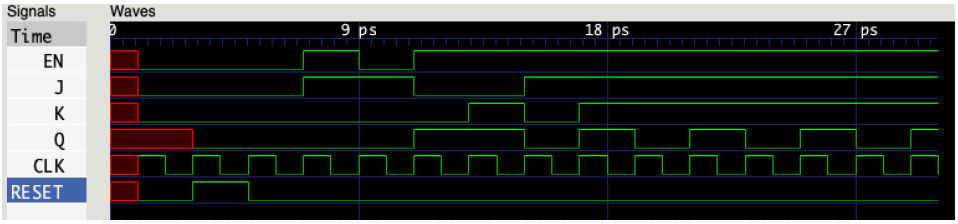


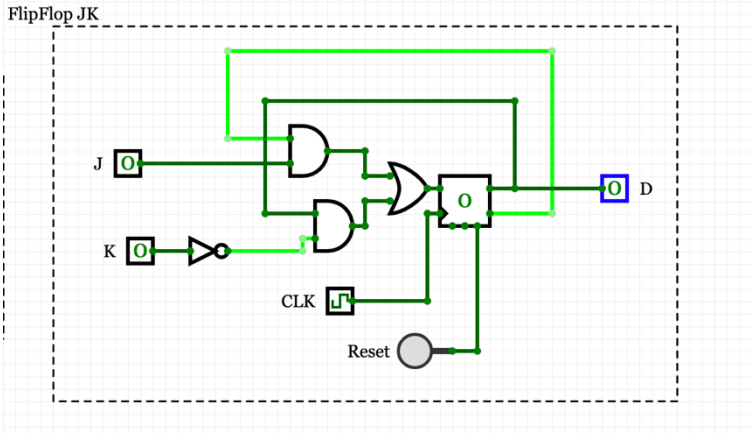


En el ejercicio 2 Se realizo un FF tipo T el cual provien de un FF tipo D, la unica variacion que contiene este es que la entrad es la inversa de la salida, ademas del CLK y el enable. Primero se realizo el moduelo del FF tipo D y se implemento en el FF tipo T pero en lugar de la entrada D se coloco ~Q(inversa de la salida).

Ejercicio 3

```
module flipfD(input wire CLK, RESET, EN, D, output reg Q);
  always @ (posedge CLK or posedge RESET)begin
    if(RESET)
      Q <= 1'b0;
    else if (EN)
      Q <= D;
  end
endmodule


module flipJK(input wire CLK, RESET, EN, J, K, output wire Q);
  wire NJ, NK, s1, s2, q;

  not(NQ, Q);
  not(NK, K);
  and(s1, J, NQ);
  and(s2, NK, Q);
  or(q, s1, s2);

  flipfD FJK(CLK, RESET, EN, q, Q);

endmodule
```

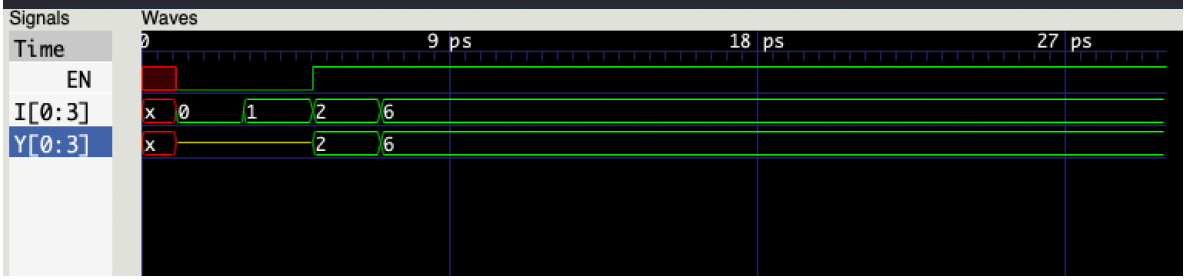




En el ejercicio 3 se utilizo un FF tipo D para poder construir un FF tipo JK, el cual define una sere de combinaciones de JK y convierte la salida Q en otra entrada para el FF. Se baso en una tabla de verdad para poder hacer la nube combinacional que se observa en circuit verse.

Ejercicio 4

```
module BT_4B (input wire EN, input wire [0:3]I, output wire [0:3]Y);

assign Y = (EN) ? I:4'bz;
endmodule
```

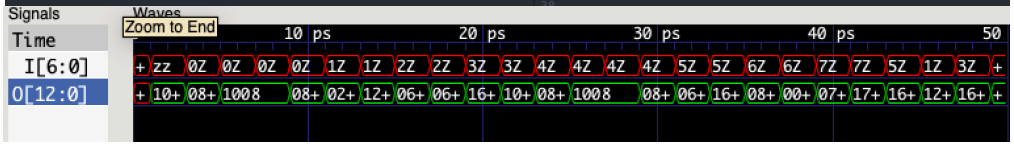


En este ejercicio se contruyó un Buffer Tri-estado, el cual requiere unicamente de 2 entradas, y cuando el enable esta apagado sus salidas estan en alta impedancia o “z”.

Ejercicio 5

```
module ROM_5(input wire [6:0]I, output reg [12:0]O);
reg [12:0] ROM [0:127];
  always @ (I)begin

    casez(I)
        7'b??????0: O = 13'b1000000001000;//
        7'b0000??1: O = 13'b0100000001000;//
        7'b0000??1: O = 13'b1000000001000;//
        7'b0001??1: O = 13'b0100000001000;//
        7'b0001??1: O = 13'b0100000001000;//
        7'b0010??1: O = 13'b0001001000010;//
        7'b0011??1: O = 13'b1001001100000;//
        7'b0100??1: O = 13'b0011010000010;//
        7'b0101??1: O = 13'b0011010000100;//
        7'b0110??1: O = 13'b1011010100000;//
        7'b0111??1: O = 13'b1000000111000;//
        7'b1000?11: O = 13'b0100000001000;//
        7'b1000?01: O = 13'b1000000001000;//
        7'b1001?11: O = 13'b1000000001000;//
        7'b1001?01: O = 13'b0100000001000;//
        7'b1010??1: O = 13'b0001011000010;//
        7'b1011??1: O = 13'b1011011100000;//
        7'b1100??1: O = 13'b0100000001000;//
        7'b1101??1: O = 13'b0000000001001;//
        7'b1110??1: O = 13'b0011100000010;//
        7'b1111??1: O = 13'b1011100100000;//
        default: O = 7'b???????0;// si no se le asigna valor queda en
    endcase
  end
endmodule
```

```
module testbench();

reg [6:0]I;
wire [12:0]O;

ROM_5 MROM(I, O);

initial begin
#1
$display("      IN      | Funcion de la ROM ");
$monitor("      %b              %b", I, O);
I = 7'b??????0;
#2 I = 7'b0000??1;
#2 I = 7'b0000??1;
#2 I = 7'b0001??1;
#2 I = 7'b0001??1;
#2 I = 7'b0010??1;
#2 I = 7'b0011??1;
#2 I = 7'b0100??1;
#2 I = 7'b0101??1;
#2 I = 7'b0110??1;
#2 I = 7'b0111??1;
#2 I = 7'b1000?11;
#2 I = 7'b1000?01;
#2 I = 7'b1001?11;
#2 I = 7'b1001?01;
#2 I = 7'b1010??1;
#2 I = 7'b1011??1;
#2 I = 7'b1100??1;
#2 I = 7'b1101??1;
#2 I = 7'b1110??1;
#2 I = 7'b1111??1;
#2 I = 7'b1011??1;
#2 I = 7'b0011??1;
#2 I = 7'b0110?01;
#2 I = 7'b0111??1;
#2 I = 7'b1010101;
```



En este ejercicio se armo una memoria capaz de guardar todos los comando que contiene un procesador. En este ejercicio se elabora una ROM de 7 bits de entrada y 13 bits de salida. Esto se realizon con un casez donde se guardaron los posibles cambios de la memoria, se probaron las 21 posibilidades y 5 con datos cambiandoles los don´t cares por 0`s o 1`s.