Universidad del Valle de Guatemala
Electrónica Digital 1 Fecha entrega: 05.09.2020
Jonathan Emanuel Pu Aguilera carné 19249

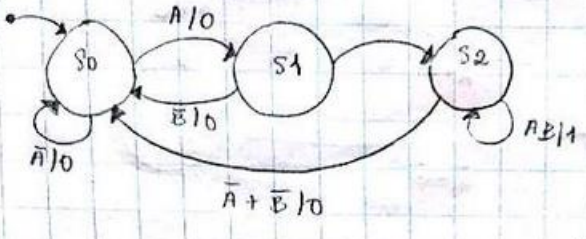
LAB. 06

Jonathan Emanuel Pu Aguilera
Carné 19249

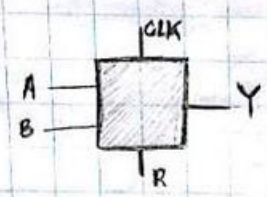
Fecha entrega: 05.09.2020
Lab. sección 11

LABORATORIO #6

Ejercicio 01



1. Caja negra



4. Ecuaciones booleanas

→ $S'_0 = S_1 \cdot B + S_0 \cdot B$
→ $S'_1 = S'_0 \cdot S'_1 \cdot A$
→ $Y = S_0 \cdot B$

2. Tabla de transiciones de estado sin codificar.

Estado actual S	Entradas A	B	Estado futuro S'	Salida Y
S_0	A	X	S_1	$\bar{Y}$
S_0	$\bar{A}$	X	S_0	$\bar{Y}$
S_1	X	B	S_2	$\bar{Y}$
S_1	X	$\bar{B}$	S_0	$\bar{Y}$
S_2	$\bar{A}$	$\bar{B}$	S_0	$\bar{Y}$
S_2	A	B	S_2	Y

3. Tabla codificada $S_0 = 0\ 0$ $S_1 = 0\ 1$ $S_2 = 1\ 0$

Estado actual S	Entradas A	B	Estado futuro S'	Salida Y
0 0	1	X	0 1	0
0 0	0	X	0 0	0
0 1	X	1	1 0	0
0 1	X	0	0 0	0
1 0	0	0	0 0	0
1 0	1	1	1 0	1

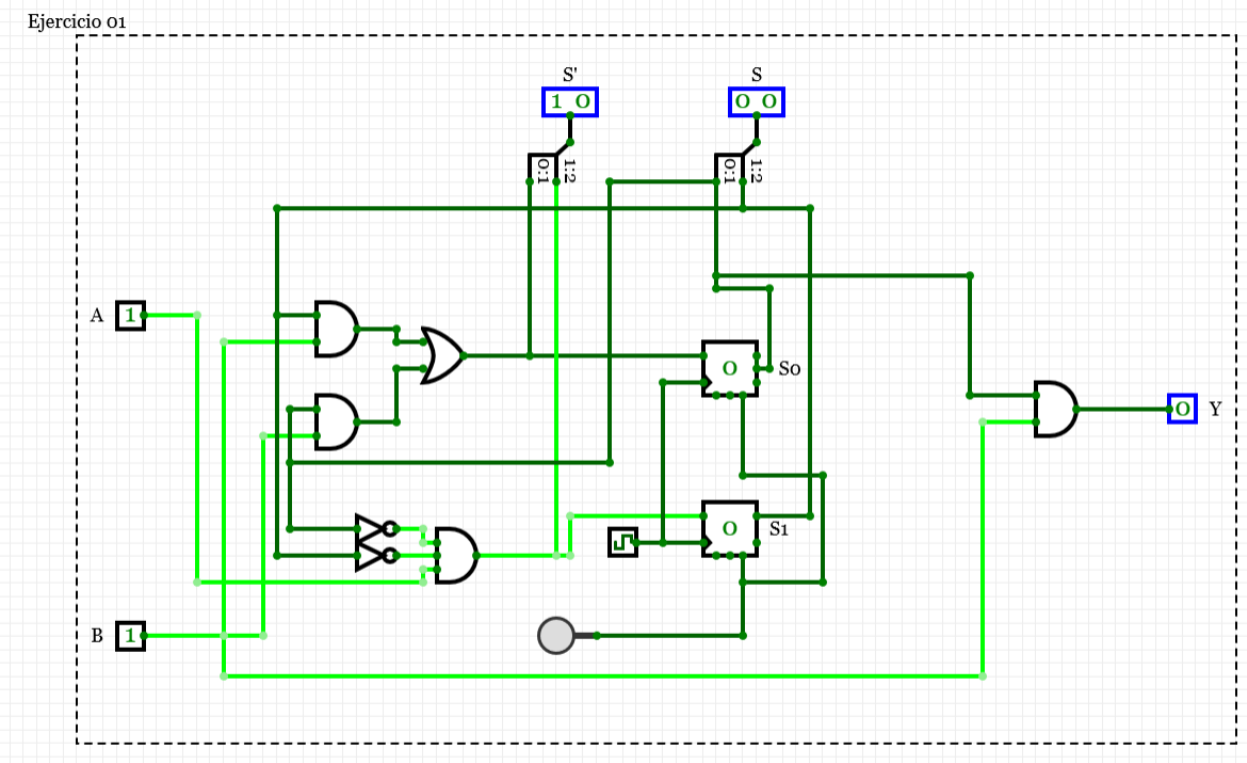
Resolución de la tabla en Logic Friday y simplificación de las ecuaciones

Sa	Sb	A	B	=>	Sf_a	Sf_b	Y
X	1	X	1		1		
0	0	1	X			1	
1	X	X	1		1		1

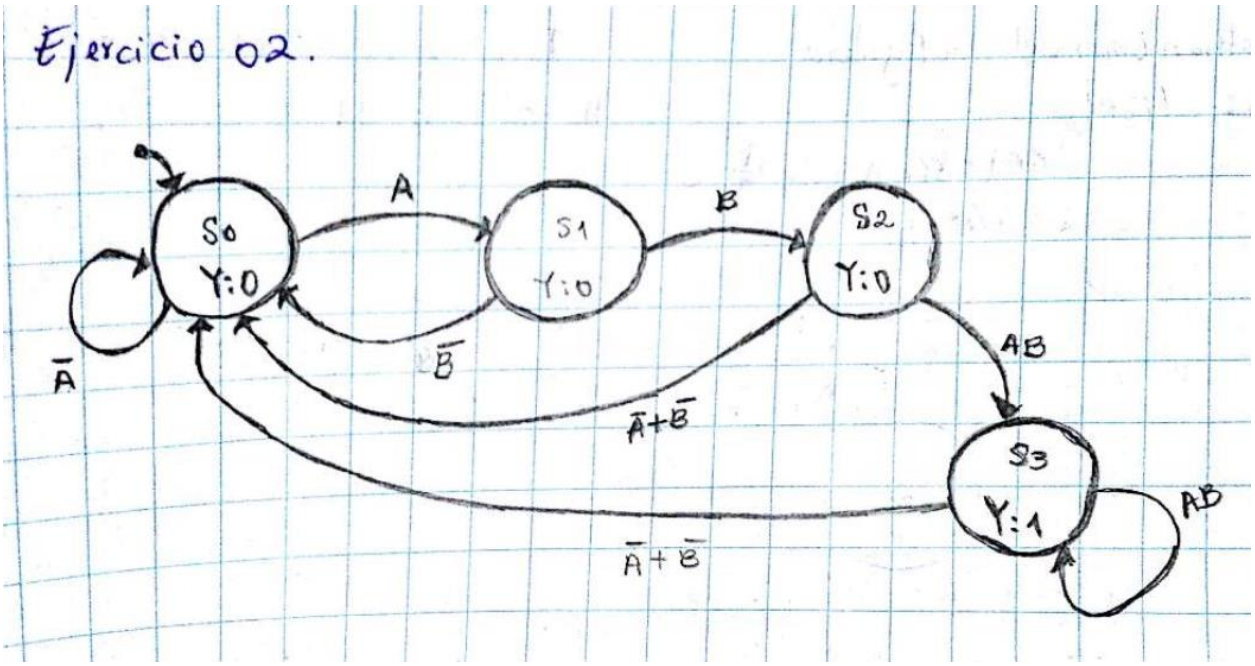
```
Entered by truthtable:
Sf_a = Sa' Sb A' B + Sa' Sb A B + Sa Sb' A B;
Sf_b = Sa' Sb' A B' + Sa' Sb' A B;
Y = Sa Sb' A B;

Minimized:
Sf_a = Sb B + Sa B;
Sf_b = Sa' Sb' A ;
Y = Sa B;
```

Tabla original

Sa	Sb	A	B	=>	Sf_a	Sf_b	Y
0	0	1	0			1	
0	0	1	1			1	
0	1	0	1		1		
0	1	1	1		1		
1	0	0	1		X	X	X
1	0	1	0		X	X	X
1	0	1	1		1		1
1	1	0	0		X	X	X
1	1	0	1		X	X	X
1	1	1	0		X	X	X
1	1	1	1		X	X	X

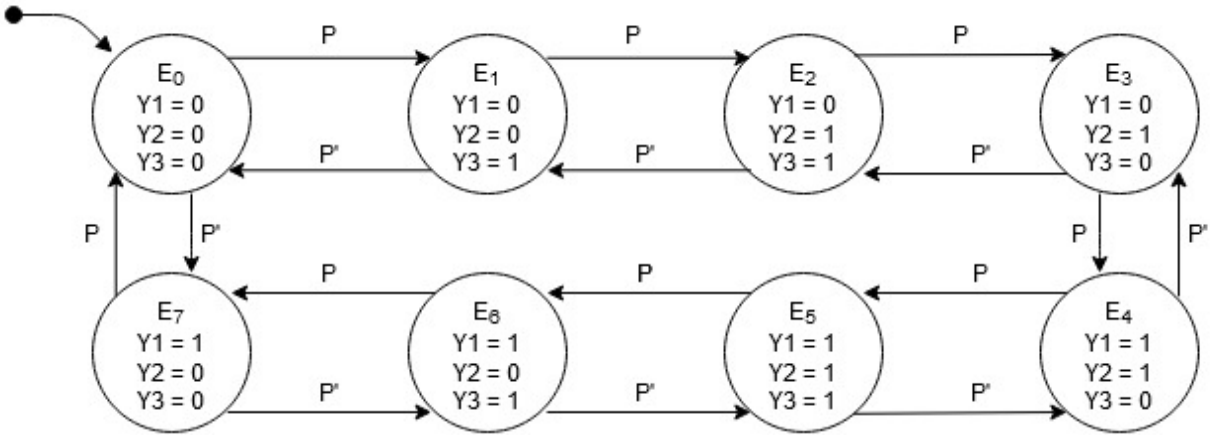


Ejercicio 02.

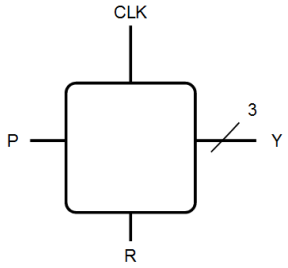


Ejercicio 03.

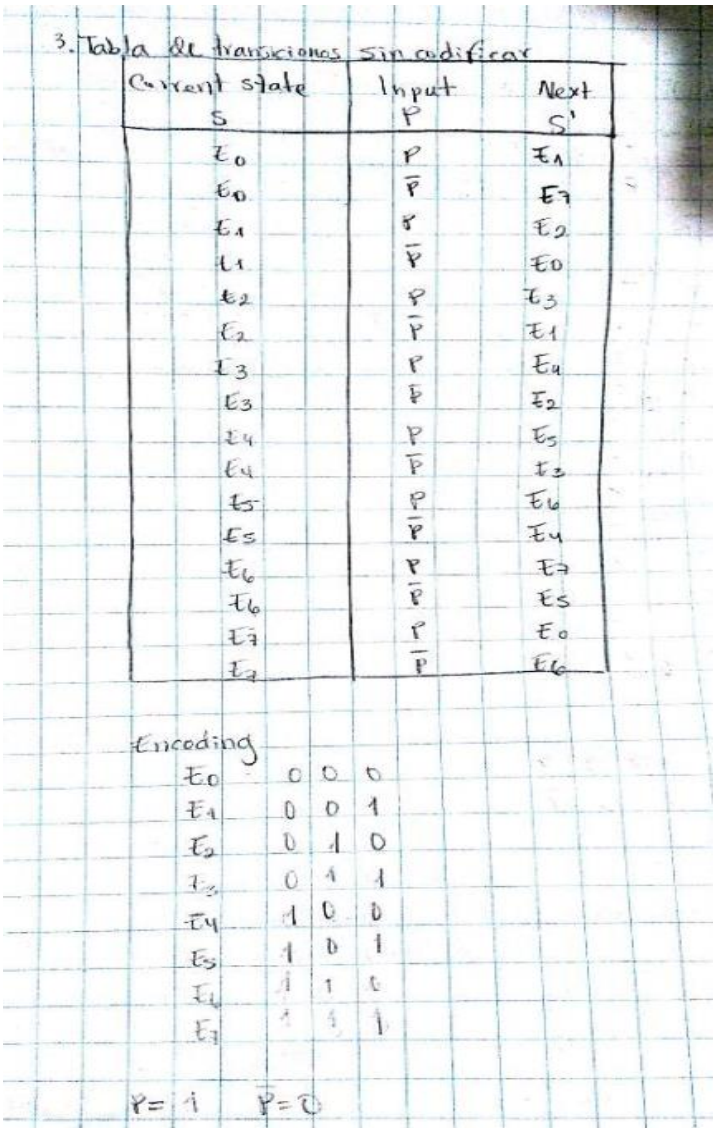
1. Diagrama de transición de estados



2. Caja negra



3. Tablas de transiciones



3. Tabla de transiciones sin codificar

Current state S	Input P	Next S'
E_0	P	E_1
E_0	$\bar{P}$	E_7
E_1	P	E_2
E_1	$\bar{P}$	E_0
E_2	P	E_3
E_2	$\bar{P}$	E_1
E_3	P	E_4
E_3	$\bar{P}$	E_2
E_4	P	E_5
E_4	$\bar{P}$	E_3
E_5	P	E_6
E_5	$\bar{P}$	E_4
E_6	P	E_7
E_6	$\bar{P}$	E_5
E_7	P	E_0
E_7	$\bar{P}$	E_6

Encoding

E_0	0	0	0
E_1	0	0	1
E_2	0	1	0
E_3	0	1	1
E_4	1	0	0
E_5	1	0	1
E_6	1	1	0
E_7	1	1	1

P= 1 $\bar{P}$= 0

4. Tabla de transiciones codificada

Current state S			Input	Next S'		
0	0	0	1	0	0	1
0	0	0	0	1	1	1
0	0	1	1	0	1	0
0	0	1	0	0	0	0
0	1	0	1	0	1	1
0	1	0	0	0	0	1
0	1	1	1	1	0	0
0	1	1	0	0	1	0
1	0	0	1	1	0	1
1	0	0	0	0	1	1
1	0	1	1	1	1	0
1	0	1	0	1	0	0
1	1	0	1	1	1	1
1	1	0	0	1	0	1
1	1	1	1	0	0	0
1	1	1	0	1	1	0

Tabla de salidas

Current State S			Output		
S_2	S_1	S_0	$Y_{[0]}$	$Y_{[1]}$	$Y_{[2]}$
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

4. Ecuaciones minimizadas y tablas resueltas en Logic Friday

```
Minimized:
Sf_0 = S0' S1 S2 P + S0' S1' S2' P' + S0 S2 P' + S0 S1' P + S0 S1 S2' ;
Sf_1 = S1 S2 P' + S1' S2 P + S1' S2' P' + S1 S2' P;
Sf_2 = S2' ;
```

Tabla minimizada

S0	S1	S2	P	=>	Sf_0	Sf_1	Sf_2
0	1	1	1		1		
0	0	0	0		1		
1	X	1	0		1		
1	0	X	1		1		
X	1	1	0			1	
X	0	1	1			1	
1	1	0	X		1		
X	0	0	0			1	
X	1	0	1			1	
X	X	0	X				1

Tabla original

S0	S1	S2	P	=>	Sf_0	Sf_1	Sf_2
0	0	0	0		1	1	1
0	0	0	1				1
0	0	1	1			1	
0	1	0	0				1
0	1	0	1			1	1
0	1	1	0			1	
0	1	1	1		1		
1	0	0	0			1	1
1	0	0	1		1		1
1	0	1	0		1		
1	0	1	1		1	1	
1	1	0	0		1		1
1	1	0	1		1	1	1
1	1	1	0		1	1	

Tabla de salidas simplificada

S0	S1	S2	=>	Y[0]	Y[1]	Y[2]
0	1	X			1	
X	1	0				1
X	0	1				1
1	0	X			1	
1	X	X		1		

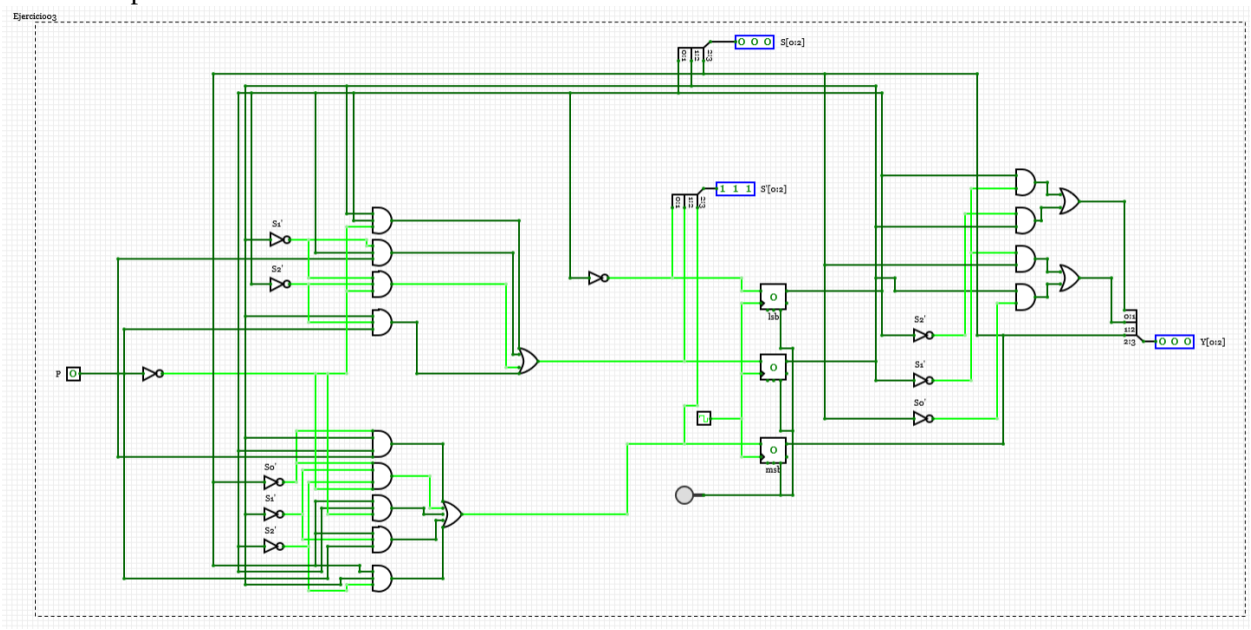
Tabla de salidas originales

S0	S1	S2	=>	Y[0]	Y[1]	Y[2]
0	0	1				1
0	1	0			1	1
0	1	1			1	
1	0	0		1	1	
1	0	1		1	1	1
1	1	0		1		1
1	1	1		1		

Ecuaciones de lógica de salidas simplificada

```
Minimized:
Y[0] = S0 ;
Y[1] = S0' S1  + S0 S1' ;
Y[2] = S1 S2' + S1' S2;
```

- Implementación en CircuitVerse



Ejercicio 04

Non-blocking assignment (<=)
Asigna valores registrando tipos de datos, pero en el proceso no impide que otros assign se lleven a cabo, inicia en un tiempo dado y actualiza sus valores al final de ese tiempo. A diferencia de los **blocking assignments (=)** que no permiten que otras instrucciones se ejecuten al mismo tiempo hasta que se haya dado el valor al assign, no lo interrumpe otra instrucción.

Cuando utilizarlos

Blocking: cuando se hace un bloque always; para modelar lógica combinacional
Non-blocking: para secuencia lógica; latches; en un bloque always que tenga secuenciales y combinacionales

Ejemplo de non-blocking para modelar secuencia lógica síncrona:

```
always_ff @(posedge clk)
        begin
        n1 <= d; // nonblocking
        q <= n1; // nonblocking
end
```

Ejemplo de blocking para lógica combinacional simple:

```
        assign y = s ? d1 : d0;
```

Ejercicio 06

```
3   //Primero se modula el ff_D con asynchronus reset
4   module ff_D1(input wire D, clk, reset, output reg Q);
5       always @ (posedge clk or posedge reset)
6       begin
7           if (reset == 1'b1) begin;
8               Q <= 1'b1;
9           end
10          else begin
11              Q <= D;
12          end
13      end
14  endmodule
15
16  // Ejercicio01
17  module FSM1(
18      input wire clk, inA, inB, reset, output wire out1);
19
20      wire D0, D1; //son los cables que van a entrar como D a mis ff_D
21      wire Q0, Q1; //son los que salen de mis ff_D
22
23      assign D0 = (Q1 & inB) | (Q0 & inB);
24      assign D1 = (~D0 & ~D1 & inA);
25
26      //llamo los dos ff_D que necesito
27      ff_D1 a1(D0, clk, reset, Q0);
28      ff_D1 a2(D1, clk, reset, Q1);
29
30      //ahora ya utilizo estos nuevos cables
31      assign Y = D0 & inB;
32
33  endmodule
```

```
//Jonathan Pu c. 19249

module testbench();
    //Para la primera FSM
    reg clk, reset, A, B;
    wire Y;

    always
        #5 clk = ~clk;
    FSM1 E01(clk, A, B, reset, Y);

    initial begin
        $display("\n");
        $display("clk reset A B | Y");
        $display("----------------------");
        $monitor("%b %b %b %b | %b", clk, reset, A, B, Y);
            //entradas comienzan en 0
            clk = 0; reset = 0; A = 0; B = 0;
        #1 reset = 1;
        #1 reset = 0;
        #10 A = 1; B = 0;
        #10 A = 0; B = 1;
        #10 A = 1; B = 1;
    end

    initial
        #48 $finish;

    //GTK Wave
    initial begin
        $dumpfile("FSM06_tb.vcd"); //nombre del archivo
        $dumpvars(0, testbench); //nombre del modulo
    end

endmodule
```

```
//Ejercicio03
module FSM2 (
    input wire clk, P, reset, output Y0, Y1, Y2);

    //declaro los cables que van a salir de mis ff_D
    wire Q0, Q1, Q2;
    //los cables que entran a mis ff_D
    wire D0, D1, D2;

    //ecuaciones de los estados futuros
    assign D0 = (~D0 & D1 & D2 & P) | (~D0 & ~D1 & ~D2 & ~P)
    | (D0 & D2 & ~P) | (D0 & ~D1 & P) | (D0 & D1 & ~D2);

    assign D1 = (D1 & D2 & ~P) | (~D1 & D2 & P) | (~D1 & ~D2 & ~P) | (D1 & ~D

    assign D2 = ~D2;

    //tres ff_D
    ff_D1 b1(D2, clk, reset, Q2);
    ff_D1 b2(D1, clk, reset, Q1);
    ff_D1 b3(D0, clk, reset, Q0);

    //ecuacion de las salidas
    assign Y0 = Q0;
    assign Y1 = (~Q0 & Q1) | (Q0 & ~Q1);
    assign Y2 = (Q1 & ~Q2) | (~Q1 & Q2);

endmodule
```

```
    reg P;
    wire Y0, Y1, Y2;

    FSM2 E03(clk, P, reset, Y0, Y1, Y2);

    initial begin
        $display("\n");
        $display("clk reset P | Y0 Y1 Y2");
        $display("----------------------");
        $monitor("%b %b %b | %b %b %b", clk, reset, P, Y0, Y1, Y2);
            //entradas comienzan en 0
            clk = 0; reset = 0; P = 0;
        #1 reset = 1;
        #1 reset = 0;
        #10 P = 1;
        #10 P = 0;
    end

    initial
        #48 $finish;

    //GTK Wave
    initial begin
        $dumpfile("FSM06_tb.vcd"); //nombre del archivo
        $dumpvars(0, testbench); //nombre del modulo
    end

endmodule
```

Ejercicio 05

```
1   //Jonathan Pu c. 19249
2
3
4   //Ejercicio05 FF D con R asinc. y S sinc.
5
6   module FF_D (input [0:3]D, CLK,  set, reset, output [0:3]Q);
7
8   //reset asincrono
9   always @ (posedge CLK or posedge reset)
10      if reset == 1'b 1 begin
11          [0:3]Q <= 4'b 0000;
12      end else begin [0:3]Q <= 4'b 0000;
13          [0:3]Q <= [0:3]D;
14      if reset == 0;
15  //set sincrono
16      if set ==1'b begin
17          Q<= 4'b 1111;
18      else
19          [0:3]Q <= [0:3]D;
20      end
21
22  endmodule
```

REPOSITORIO
https://github.com/pu19249/Repositorio-D1-19249.git