Nombre: José Alejandro Rodríguez Porras Carné: 19131 Fecha: 3/09/2020

Laboratorio 6

Link del repositorio: https://github.com/rod19131/electronica_digital1

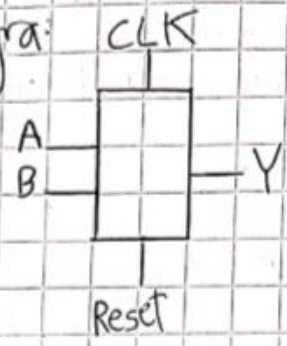
Ejercicio 1

Nombre: José Alejandro Rodríguez Porras 3/09/2020
Carné: 19131

Laboratorio 6

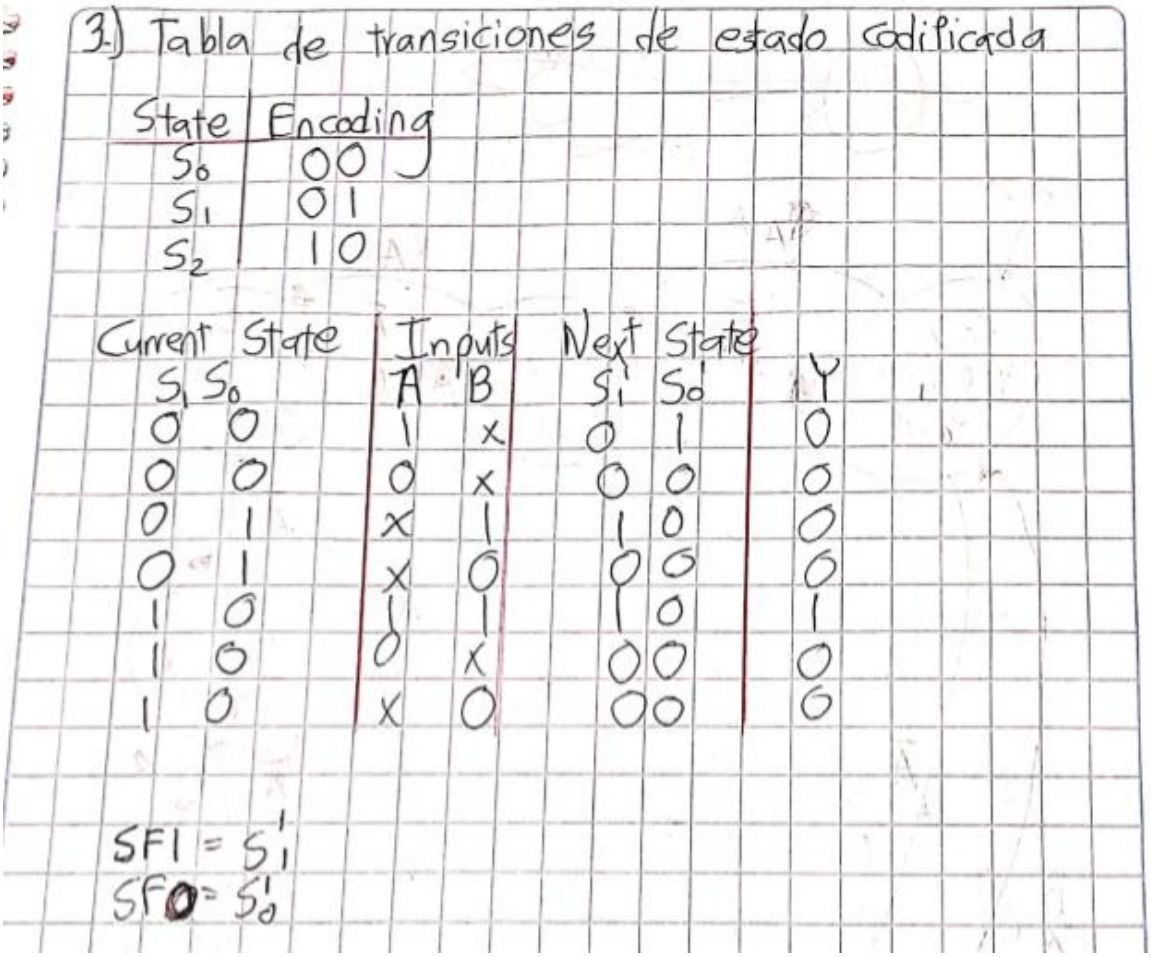
Ejercicio 1

1) Caja Negra:

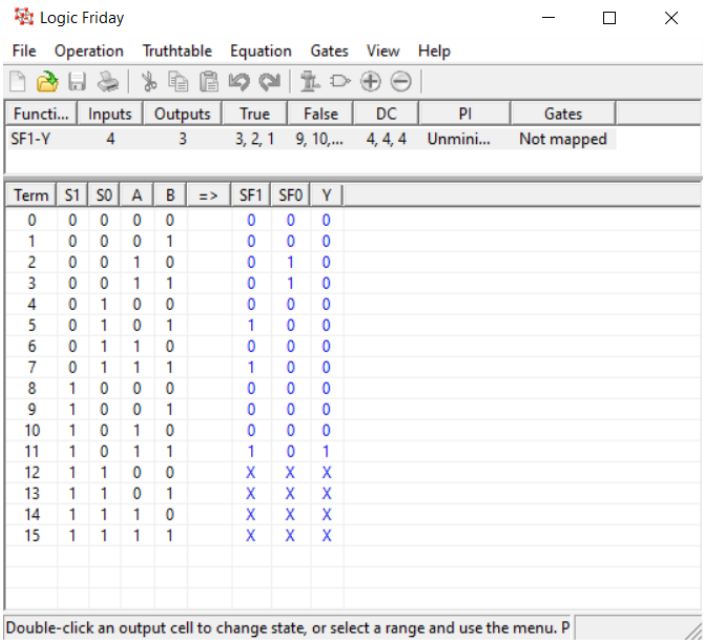


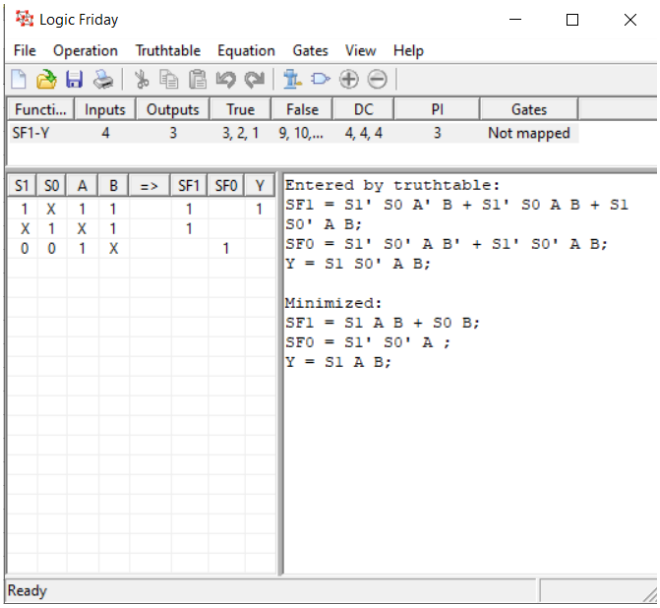
2) Tabla de transiciones de estados sin codificar

Current State	Inputs		Next State	
S	A	B	S'	Y
S_0	1	X	S_1	0
S_0	0	X	S_0	0
S_1	X	1	S_2	0
S_1	X	0	S_0	0
S_2	1	1	S_2	1
S_2	0	X	S_0	0
S_2	X	0	S_0	0



4.) Logic Friday





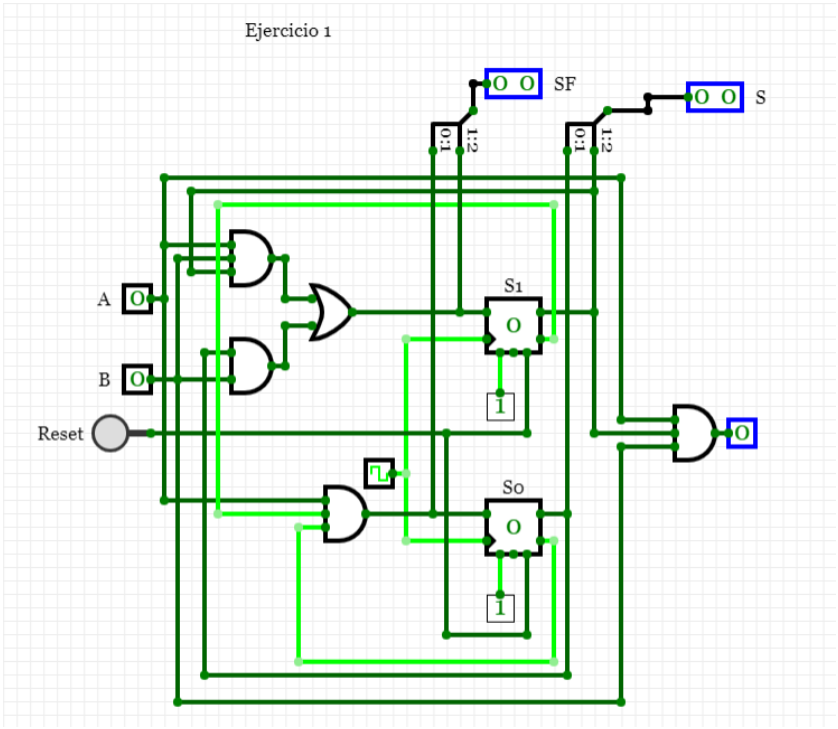
5.) Ecuaciones Booleanas

SF1 = (S1*A*B) + (S0*B)

SF0 = S1'*S0'*A

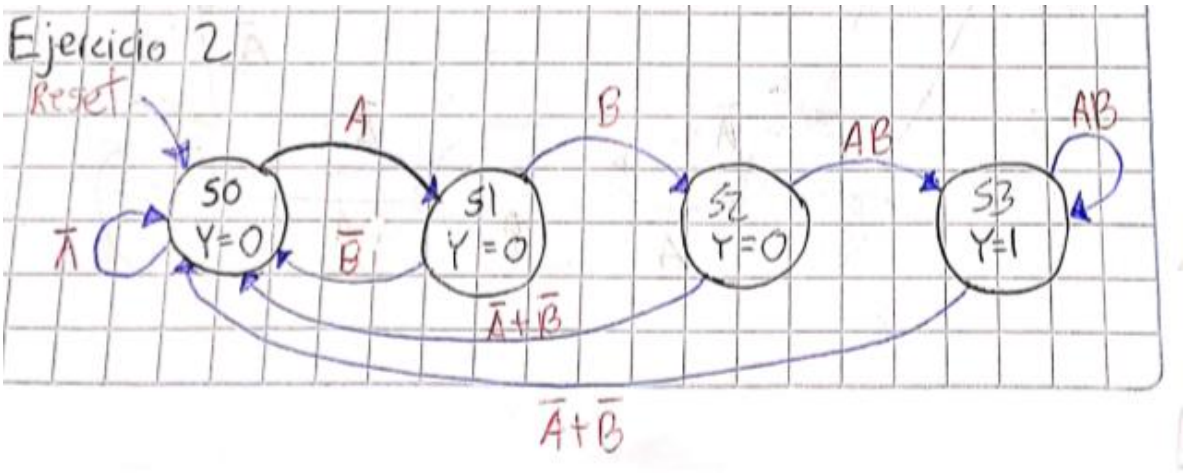
Y = S1*A*B

6.) Implementación en CircuitVerse

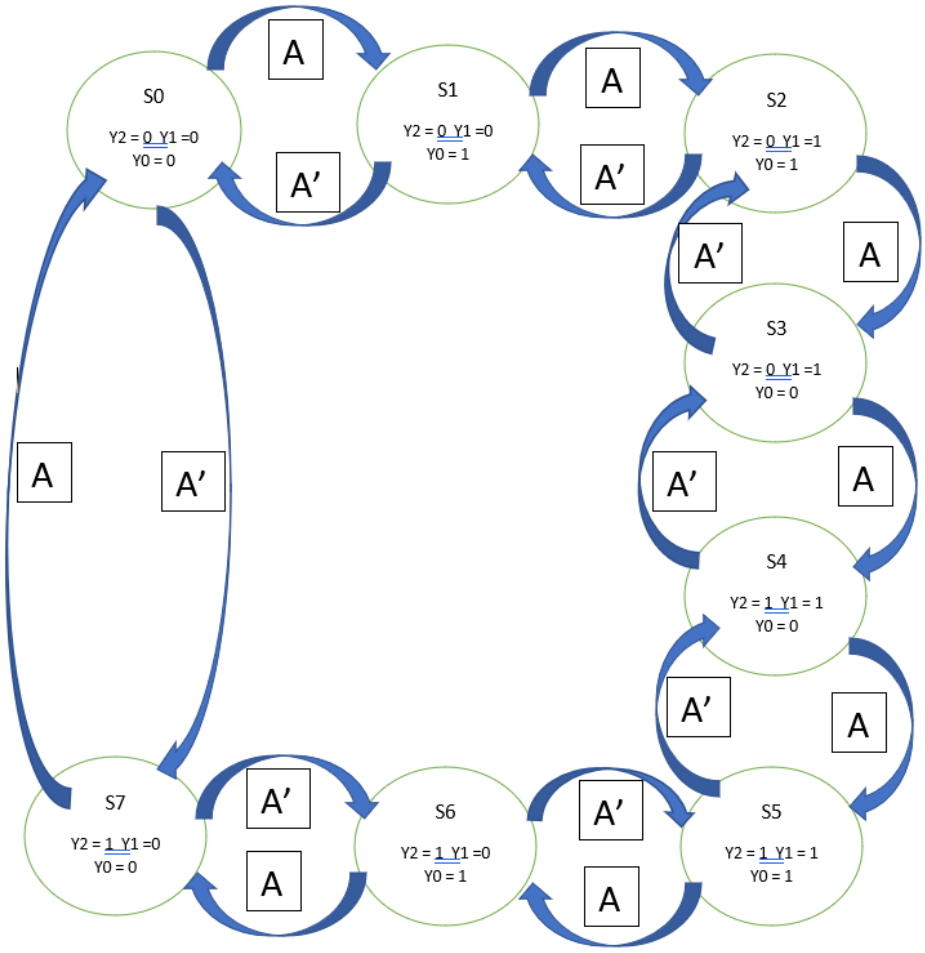


Ejercicio 2

Diagrama FSM de Moore

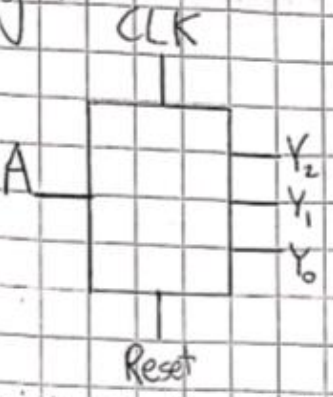


Ejercicio 3



Ejercicio 3.

2) Caja Negra



3.) Tabla de Transiciones de Estado Sin codificar

Current State	Inputs	Next State	Outputs		
S	A	S'	Y_2	Y_1	Y_0
S_0	1	S_1	0	0	0
S_1	1	S_2	0	0	1
S_2	1	S_3	0	1	1
S_3	1	S_4	0	1	0
S_4	1	S_5	0	1	0
S_5	1	S_6	0	1	1
S_6	1	S_7	0	1	1
S_7	1	S_0	0	1	0
S_7	0	S_6	0	1	0
S_6	0	S_5	0	1	1
S_5	0	S_4	0	1	1
S_4	0	S_3	0	1	0
S_3	0	S_2	0	1	0
S_2	0	S_1	0	1	1
S_1	0	S_0	0	0	1
S_0	0	S_7	0	0	0

4.) Tabla de transiciones codificada

Codificación de Estados

State	Encoding
S_0	0 0 0
S_1	0 0 1
S_2	0 1 0
S_3	0 1 1
S_4	1 0 0
S_5	1 0 1
S_6	1 1 0
S_7	1 1 1

Tabla de transiciones codificada

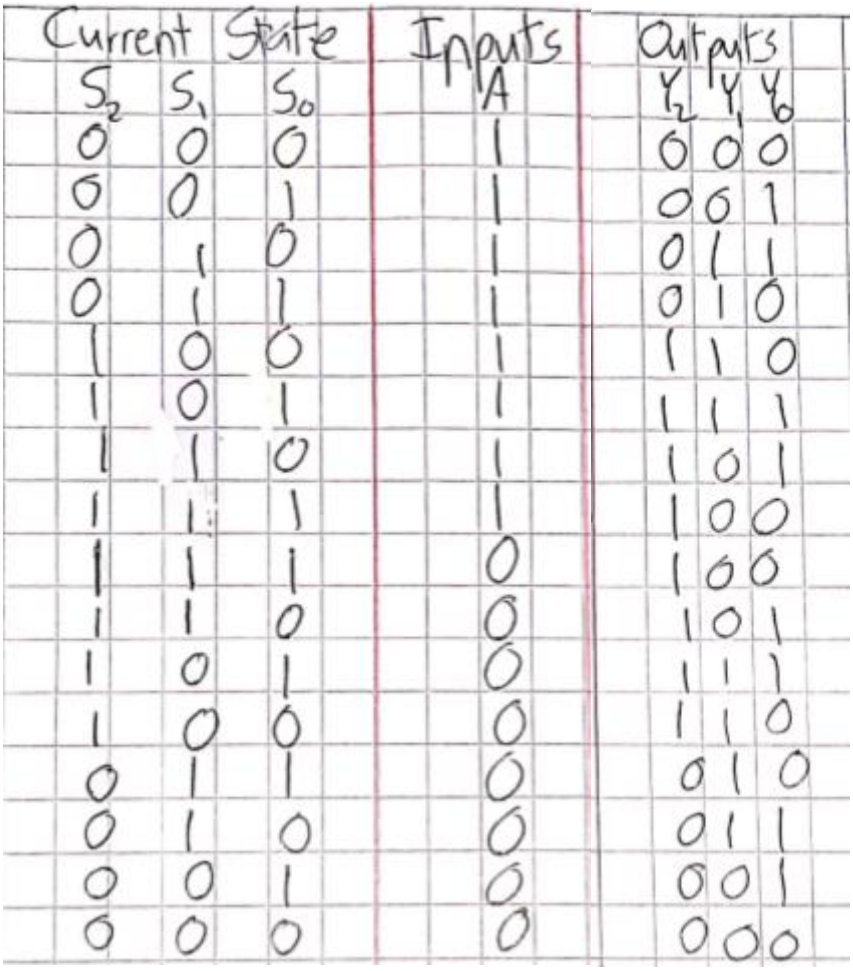
Current State			Inputs	Next State		
S_2	S_1	S_0	A	S_2'	S_1'	S_0'
0	0	0	1	0	0	1
0	0	1	1	0	1	0
0	1	0	1	0	1	1
0	1	1	1	1	0	0
1	0	0	1	1	0	1
1	0	1	1	1	1	0
1	1	0	1	1	1	1
1	1	1	1	0	0	0
1	1	1	0	0	1	0
1	1	0	0	1	0	1
1	0	1	0	1	0	0
1	0	0	0	0	1	1
0	1	1	0	0	1	0
0	1	0	0	0	0	1
0	0	1	0	0	0	0
0	0	0	0	1	1	1

$SF2 = S_2'$

$SF1 = S_1'$

$SF0 = S_0'$

Tabla de Salidas



Current State			Inputs	Outputs		
S_2	S_1	S_0	A	Y_2	Y_1	Y_0
0	0	0	1	0	0	0
0	0	1	1	0	0	1
0	1	0	1	0	1	1
0	1	1	1	0	1	0
1	0	0	1	1	1	0
1	0	1	1	1	1	1
1	1	0	1	1	0	1
1	1	1	1	1	0	0
1	1	1	0	1	0	0
1	1	0	0	1	0	1
1	0	1	0	1	1	1
1	0	0	0	1	1	0
0	1	1	0	0	1	0
0	1	0	0	0	1	1
0	0	1	0	0	0	1
0	0	0	0	0	0	0

5.) Logic Friday

Tabla de transiciones de estado

Logic Friday

File Operation Truthtable Equation Gates View Help

Functi...	Inputs	Outputs	True	False	DC	PI	Gates
SF2-SF0	4	3	8, 8, 8	8, 8, 8	0, 0, 0	Unmini...	Not mapped

Term	S2	S1	S0	A	=>	SF2	SF1	SF0
0	0	0	0	0		1	1	1
1	0	0	0	1		0	0	1
2	0	0	1	0		0	0	0
3	0	0	1	1		0	1	0
4	0	1	0	0		0	0	1
5	0	1	0	1		0	1	1
6	0	1	1	0		0	1	0
7	0	1	1	1		1	0	0
8	1	0	0	0		0	1	1
9	1	0	0	1		1	0	1
10	1	0	1	0		1	0	0
11	1	0	1	1		1	1	0
12	1	1	0	0		1	0	1
13	1	1	0	1		1	1	1
14	1	1	1	0		1	1	0
15	1	1	1	1		0	0	0

Ecuaciones minimizadas de la tabla de transiciones de estado

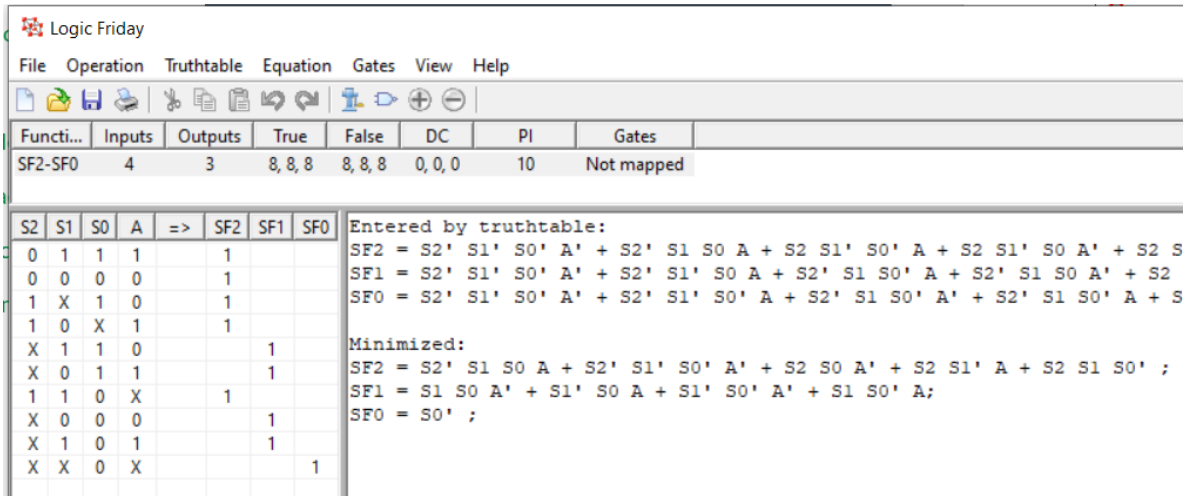
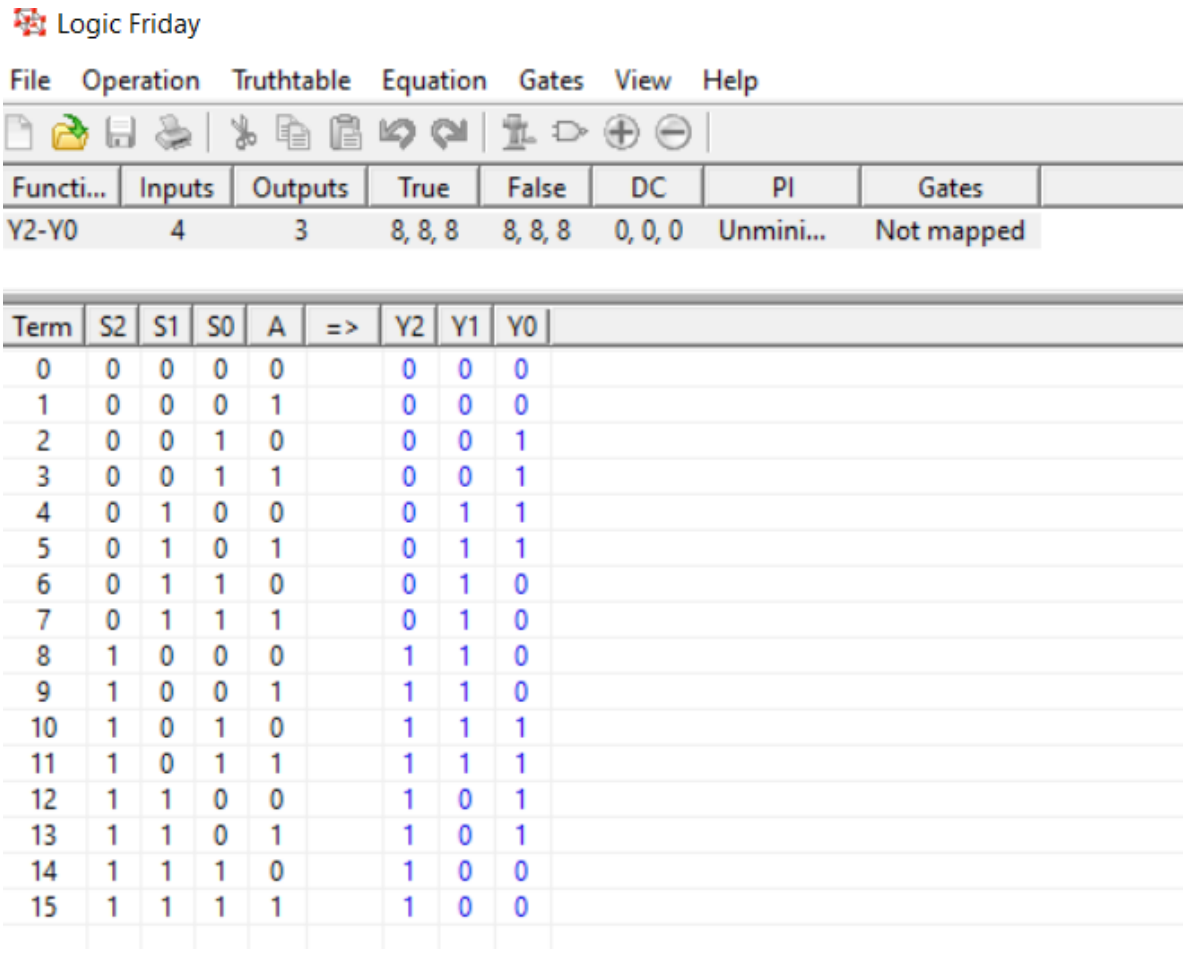
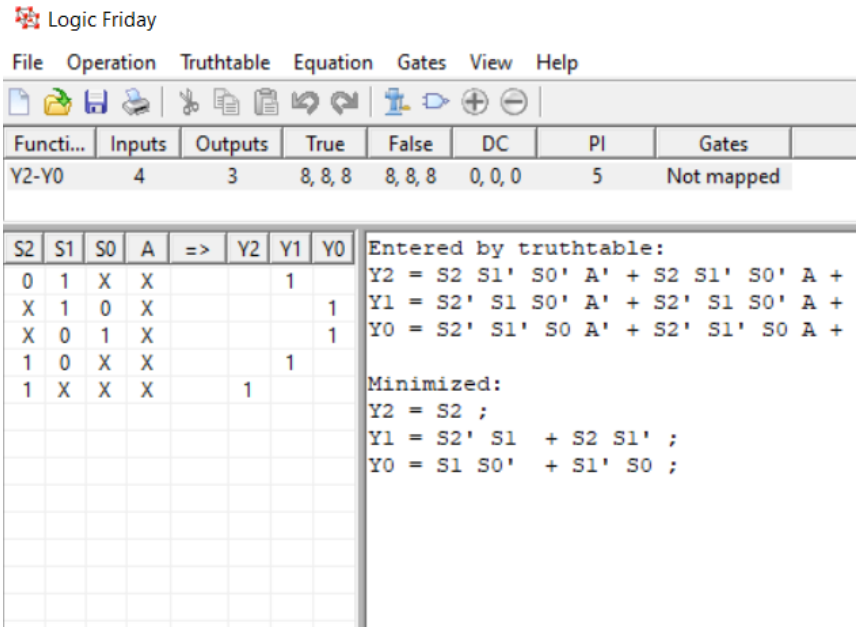


Tabla de salidas



Ecuaciones minimizadas de la tabla de salidas



6.) Ecuaciones booleanas

Estados futuros

SF2 = (S2'*S1*S0*A) + (S2'*S1'*S0'*A') + (S2*S0*A') + (S2*S1'*A) + (S2*S1*S0')

SF1 = (S1*S0*A') + (S1'*S0*A) + (S1'*S0'*A') + (S1*S0'*A)

SF0 = S0'

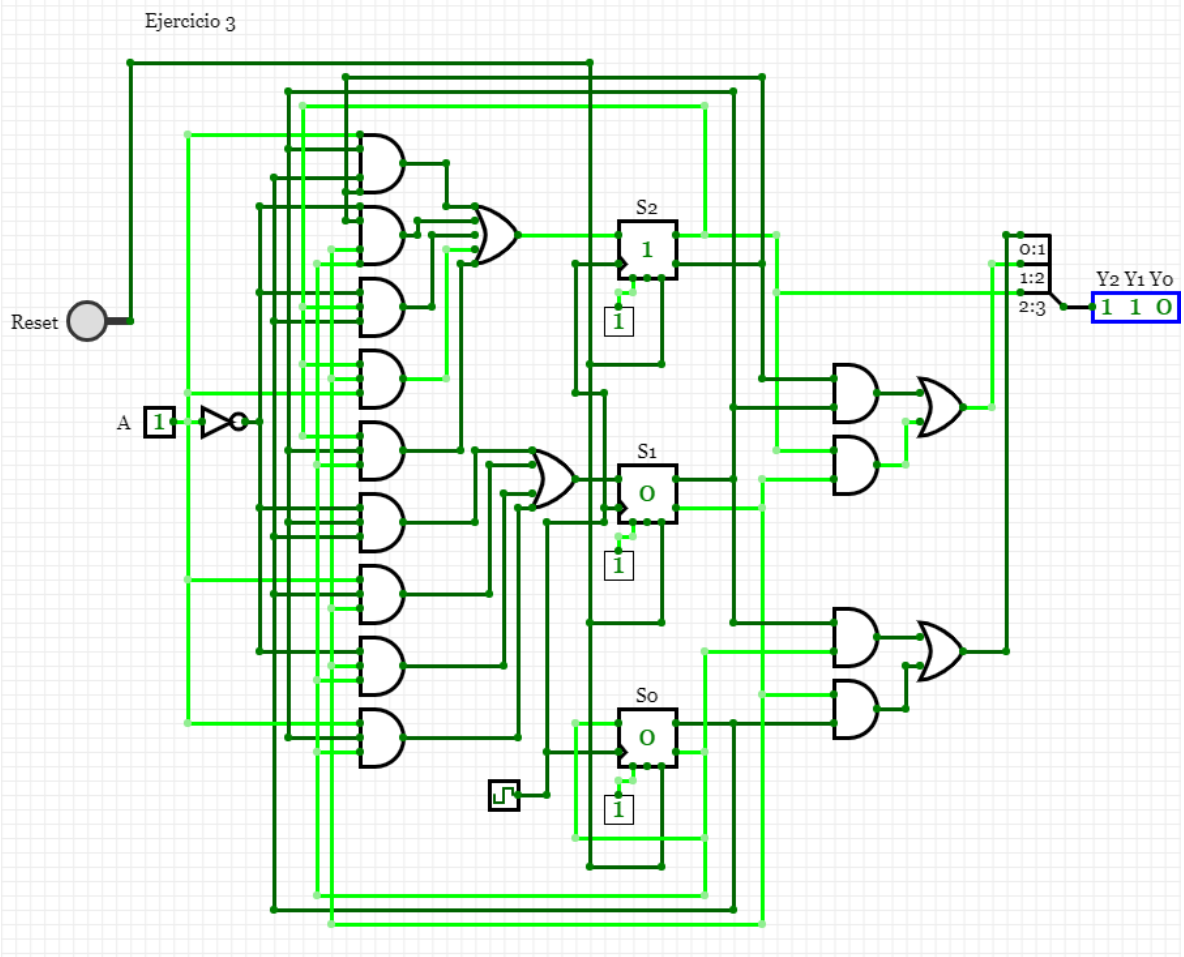
Salidas

Y2 = S2

Y1 = (S2'*S1) + (S2*S1')

Y0 = (S1*S0') + (S1'*S0)

7.) Implementación en CircuitVerse



Ejercicio 4

Un non-blocking assignment es una asignación o actualización del valor de una variable con el valor actual de otra, que ocurre simultáneamente con la asignación de otra variable que también se actualiza. El operador que representa una non-blocking assignment es “<=”, se usa dentro de un bloque “always” y todas las líneas con este operador se actualizan o ejecutan de manera paralela.

La principal diferencia entre un non-blocking assignment (<=) y un blocking assignment (=) es que aunque ambas se ejecuten en un bloque always, las líneas de el blocking assignment se ejecuten una después de la otra, mientras que en el non-blocking assignment las líneas se ejecutan simultáneamente. Debido a la naturaleza de cada tipo de assignment, los blocking assignment resultan más útiles para para lógica combinacional, ya que resultan más convenientes que los non-blocking, debido a que los non-blocking no aportan ningún beneficio extra a los circuitos combinacionales, ya que en los procesos en algunos casos el programa debe evaluar dos veces para llegar al resultado correcto, haciendo la simulación más lenta. Incluso podrían llevar a errores si no se incluyen las variables intermedias

en la lista de sensibilidad. Algunas herramientas de simulación pueden incluso sintetizar el correcto hardware que se desea simular pero con una simulación incorrecta.

En cambio, para realizar lógica secuencial, los non-blocking assignments resultan más apropiados para la tarea, debido a que al actualizarse los valores de manera instantánea, no se da lugar a incorrectas actualizaciones de las variables que solo ralentizan el funcionamiento del circuito secuencial, haciendo de la lógica secuencial un proceso más eficiente.

Ejemplo en código

ejemplos.v

D: > AlejandroDigital > electronica_digital1 > lab06dig > E4 > ejemplos.v

```
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital
3   //Ejercicio 4 Ejemplos
4   //nonblocking assignment
5   //sequential module
6   always_ff @(posedge clk)
7   begin
8       //ambas líneas se ejecutan a la vez
9       n1 <= d;//nonblocking assignment
10      q <= n1;//nonblockiing assignment
11  end
12  //blocking assignment en lógica secuencial
13  assign y = s ? d1 : d0;//blocking assignment
14  always_comb
15      begin
16          p = a^b;//blocking assignment (se ejecuta primero)
17          g = a & b;//blocking assignment (se ejecuta después de la línea anterior)
18          s = p ^cin;
19          cout = g | (p & cin);
20      end
```

Ejercicio 5

Archivo E5.v del módulo flip flop tipo D de 4 bits

E5.v | E1.v | E5_tb.v | ejercicio04_tb.v

D: > AlejandroDigital > electronica_digital1 > lab06dig > E5 > E5.v

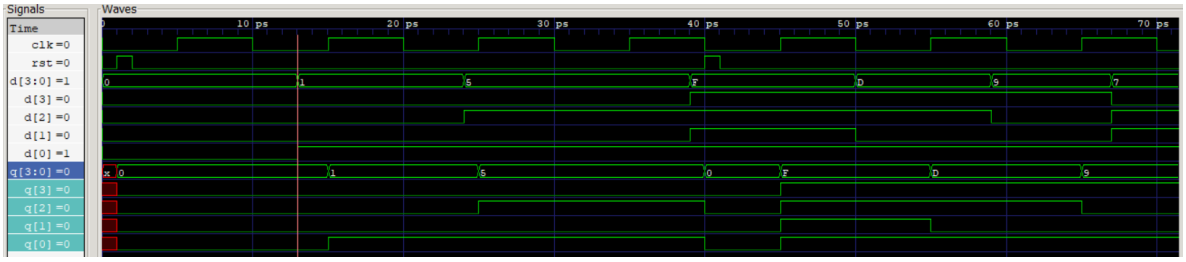
```
1   //José Alejandro Rodríguez Porras 19131
2   //Electrónica Digital
3   //Laboratorio 6
4   //Ejercicio 5: Flip Flop Tipo D de 4 Bits con reset asíncrono y un set síncrono
5   module dff4(input clk, rst, input [3:0] d, output reg [3:0] q);
6
7       always @ (posedge clk or posedge rst) begin
8           if (rst)
9               q <= 4'b0;
10          else
11              q <= d;
12      end
13  endmodule
```

Archivo E5_tb.v del testbench

D: > AlejandroDigital > electronica_digital1 > lab06dig > E5 > E5_tb.v

```
1   //Alejandro Rodríguez 19131
2   //Lab 6
3   //testbench
4   module testbench();
5       reg clk, rst;
6       reg [3:0] d;
7       wire [3:0] q;
8       dff4 FF1(clk, rst, d, q);
9
10      // Prueba del funcionamiento del flip flop tipo D de 4 bits
11      initial begin
12          clk = 0; rst = 0; d = 4'b0000;
13          #1 rst = 1;
14          #1 rst = 0;
15          #11 d = 4'b0001;
16          #11 d = 4'b0101;
17          #15 d = 4'b1111;
18          //Prueba del reset asíncrono en un flanco negativo
19          #1 rst = 1;
20          #1 rst = 0;
21          #9 d = 4'b1101;
22          #9 d = 4'b1001;
23          #8 d = 4'b0111;
24          #100 $finish;
25      end
26      //generación del reloj con período de 10 unidades de tiempo
27      always
28          #5 clk = ~clk;
29      initial begin
30          $dumpfile("E5_tb.vcd");
31          $dumpvars(0,testbench);
32      end
33  endmodule
34
```

Diagrama de timing de pruebas con el tipo Flip Flop tipo D de 4 bits



Ejercicio 6

Archivo E1.v

Código Ejercicio 1

E1.v | E1_tb.v | E3.v | E3_tb.v

D: > AlejandroDigital > electronica_digital1 > lab06dig > E6 > E1 > E1.v

```
1  //José Alejandro Rodríguez Porras 19131
2  //Electrónica Digital
3  //Laboratorio 6
4  //Ejercicio 6: Ejercicio 1
5  // Flip Flop Tipo D con reset asíncrono y
6  module dff(input clk, rst, d, output reg q);
7
8      always @ (posedge clk or posedge rst) begin
9          if (rst)
10             q <= 1'b0;
11         else
12             q <= d;
13     end
14 endmodule
15 //Módulo de la FSM de Mealy del ejercicio 1
16 module E_1(input clk, rst, A, B, output S1, S0, SF1, SF0, output Y);
17     //estaos presentes y futuros
18     wire S1, S0, SF1, SF0;
19     //asignación del estado futuro con sus respectivas ecuaciones minimizadas
20     assign SF1 = (S1 & A & B) | (S0 & B);
21     assign SF0 = (~S1 & ~S0 & A);
22     //creación de cada flip flop para cada estado futuro y su respectiva asignación de variables al módulo
23     dff FF1(clk, rst, SF1, S1);
24     dff FF0(clk, rst, SF0, S0);
25     //asignación de la salida de la máquina con su ecuación minimizada
26     assign Y = (S1 & A & B);
27 endmodule
```

Archivo E1_tb.v

Testbench del ejercicio 1

E1_tb.v E3.v E3_tb.v

D: > AlejandroDigital > electronica_digital1 > lab06dig > E6 > E1 > E1_tb.v

```
1  //Alejandro Rodríguez 19131
2  //Lab 6
3  //testbench
4  //Ejercicio 1
5  module testbench();
6      reg clk, rst, A, B;
7      wire Y, S1, S0, SF1, SF0;
8      E_1 e1(clk, rst, A, B, S1, S0, SF1, SF0, Y);
9      initial begin
10     $display("clk rst A B S1 S0 SF1 SF0 | Y");
11     $display("--------------------------|--");
12     $monitor("%b %b %b %b %b %b %b %b| %b", clk, rst, A, B, S1, S0, SF1, SF0, Y);
13 // Prueba del funcionamiento de la máquina con las distintas combinaciones para que cambie a cada uno de los estados
14         clk = 0; rst = 0; A = 0; B = 0;
15         #1 rst = 1;
16         #1 rst = 0;
17         #3 A = 1; B = 0;
18         #3 A = 0; B = 1;
19         #3 A = 1; B = 1;
20         //Prueba del reset asíncrono en un flanco negativo
21         #1 rst = 1;
22         #1 rst = 0;
23         #3 A = 1; B = 0;
24         #3 A = 0; B = 0;
25         #3 A = 0; B = 0;
26         #3 A = 1; B = 1;
27         #50 $finish;
28     end
29     //generación del reloj con período de 4 unidades de tiempo
30     always
31         #2 clk = ~clk;
32     initial begin
33         $dumpfile("E1_tb.vcd");
34         $dumpvars(0,testbench);
35     end
36 endmodule
```

Diagrama de timing del ejercicio 1

Ejercicio 3

Código Archivo E3.v

E1_tb.v | E3.v | E3_tb.v

D: > AlejandroDigital > electronica_digital1 > lab06dig > E6 > E3 > E3.v

```
1  //José Alejandro Rodríguez Porras 19131
2  //Electrónica Digital
3  //Laboratorio 6
4  //Ejercicio 6: Ejercicio 3
5  // Flip Flop Tipo D con reset asíncrono de un bit
6  module dff(input clk, rst, d, output reg q);
7
8      always @ (posedge clk or posedge rst)
9          if (rst)
10             q <= 1'b0;
11         else begin
12             q <= d;
13     end
14 endmodule
15 //módulo de la FSM de Moore del contador de Gray (Ejercicio 3)
16 module E_3(input clk, rst, A, output S2, S1, S0, SF2, SF1, SF0, Y2, Y1, Y0);
17     //salidas de estados presentes y futuros
18     wire S2, S1, S0, SF2, SF1, SF0;
19     //asignación de cada estado futuro con su ecuación minimizada
20     assign SF2 = (~S2 & S1 & S0 & A) | (~S2 & ~S1 & ~S0 & ~A) | (S2 & S0 & ~A) | (S2 & ~S1 & A) | (S2 & S1 & ~S0);
21     assign SF1 = (S1 & S0 & ~A) | (~S1 & S0 & A) | (~S1 & ~S0 & ~A) | (S1 & ~S0 & A);
22     assign SF0 = ~S0;
23     //creación de los flip flops para cada estado y asignación de variables a cada módulo
24     dff FF2(clk, rst, SF2, S2);
25     dff FF1(clk, rst, SF1, S1);
26     dff FF0(clk, rst, SF0, S0);
27     //asignación de las salidas de la fsm con sus ecuaciones minimizadas
28     assign Y2 = S2;
29     assign Y1 = (~S2 & S1) | (S2 & ~S1);
30     assign Y0 = (S1 & ~S0) | (~S1 & S0);
31 endmodule
```

Testbench Archivo E3_tb.v

E1_tb.v | E3.v | E3_tb.v

D: > AlejandroDigital > electronica_digital1 > lab06dig > E6 > E3 > E3_tb.v

```
1   //Alejandro Rodríguez 19131
2   //Lab 6
3   //testbench
4   //Ejercicio 3
5   module testbench();
6       reg clk, rst, A;
7       wire Y2, Y1, Y0, S2, S1, S0, SF2, SF1, SF0;
8       E_3 e3(clk, rst, A, S2, S1, S0, SF2, SF1, SF0, Y2, Y1, Y0);
9       initial begin
10      $display("S2 S1 S0 SF2 SF1 SF0 | Y2 Y1 Y0");
11      $display("-------------------|----------");
12      $monitor("%b %b %b %b %b %b | %b %b %b", clk, rst, A, S2, S1, S0, SF2, SF1, SF0, Y2, Y1, Y0);
13  // Prueba del funcionamiento de la máquina completa, comienza el contador de Gray
14          clk = 0; rst = 0; A = 0;
15          #1 rst = 1;
16          #1 rst = 0;
17          #3 A = 1;
18          //Prueba del reset asíncrono
19          #31 rst = 1;
20          #1 rst = 0;
21          #1 A = 1;
22          //Se pone en modo reversa
23          #5 A = 0;
24          #100 $finish;
25      end
26      //generación del reloj con período de 4 unidades de tiempo
27      always
28          #2 clk = ~clk;
29          //creación del archivo vcd
30      initial begin
31          $dumpfile("E3_tb.vcd");
32          $dumpvars(0,testbench);
33      end
34  endmodule
```

Diagrama de timing de las salidas Y2 Y1 Y0

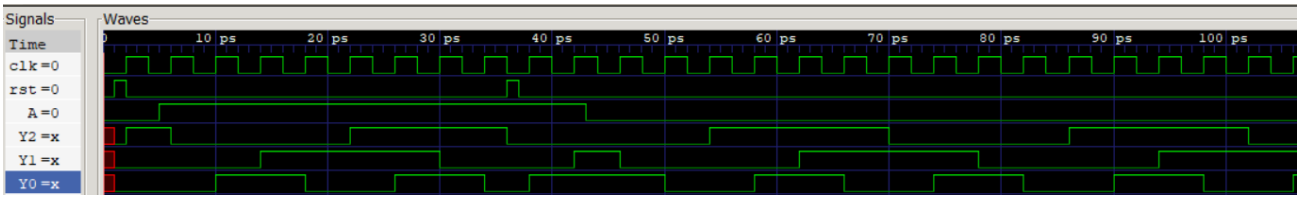
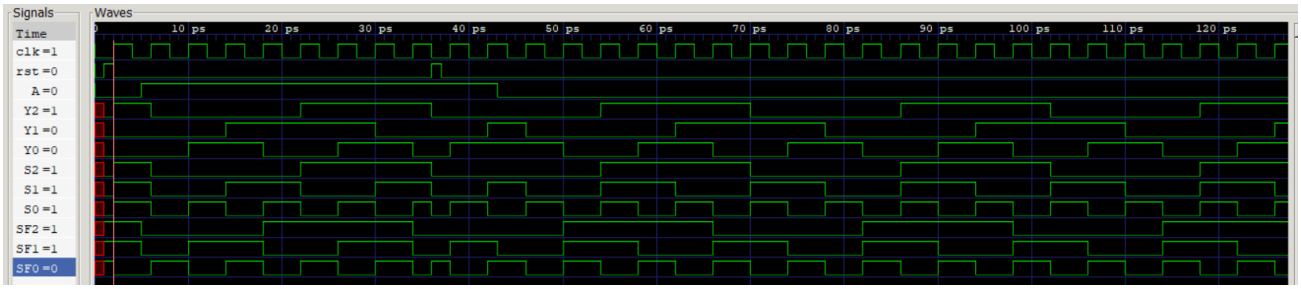


Diagrama de timing de las salidas, estados presentes (S) y estados futuros (SF)



Link del repositorio: https://github.com/rod19131/electronica_digital1