Introducción a Verilog para lógica combinacional

EL-3307: Diseño Lógico.

Profesor: Pablo Mendoza Ponce.

Asistente: Oscar Fernando Araya, 2018002998.

Segundo Semestre, 2021.

¿Qué es lógica combinacional?

1. Sistema electrónico digital en el que las **salidas solo dependen del valor de las entradas** en un momento específico.

2. Los circuitos combinacionales **carecen de memoria**.

¿Cómo diseñaremos nuestro circuito combinacional?

1. Identificamos las entradas y salidas del sistema.
2. Le damos un valor binario a las entradas y salidas.
3. A partir de estas entradas y salidas escribimos una tabla de verdad que represente, en forma de bits, ambos conjuntos.
4. Reducimos esta tabla de verdad mediante un mapa de Karnaugh, obteniendo la ecuación booleana.
5. Con la ecuación booleana, dibujamos el esquemático del circuito combinacional (opcional, pero lo recomiendo bastante).
6. Codificamos nuestra solución en Verilog.

Ejercicio I (I):

Una empresa lo ha contratado para diseñar la unidad de control de seguridad de su horno industrial. El sistema debe realizar lo siguiente:

1. Si se presiona el botón de emergencia, detener el horno, abrir la puerta del horno y activar la alarma

2. Si se detecta que algo obstruye el cierre de la puerta del horno, detener el horno y abrir la puerta.

3. Si el horno lleva dos horas sin apagarse, detener el horno y activar la alarma.

Ejercicio I (II):

1. Identificamos nuestras entradas y salidas:
   a) Entradas: botón de emergencia, sensor en la puerta, temporizador del horno.
   b) Salidas: horno, puerta del horno, alarma.

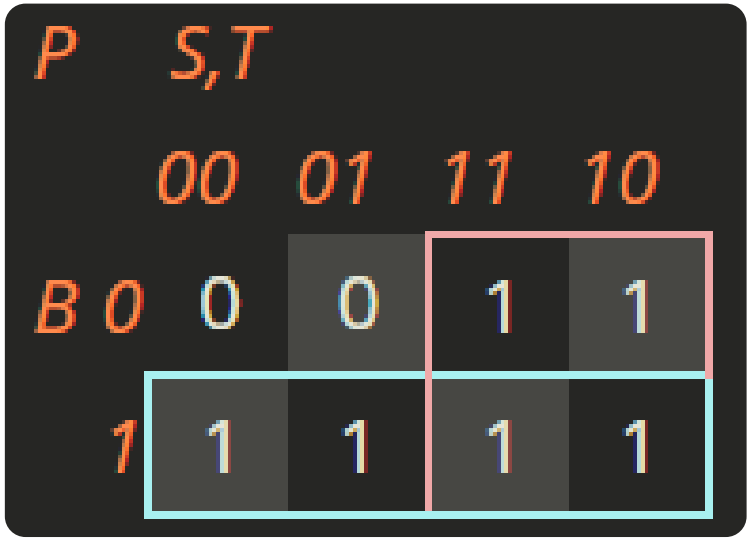
2. Le damos un valor binario a las entradas y salidas:
   a) Botón de emergencia: 1=presionado, 0=sin presionar.
   b) Sensor de la puerta: 1=detecta algo al cerrar, 0=no detecta nada al cerrar.
   c) Temporizador del horno: 1=han pasado dos horas, 0=no han pasado dos horas.
   d) Horno: 1=enciende el horno, 0=apaga el horno.
   e) Puerta del horno: 1=abre la puerta, 0=cierra la puerta.
   f) Alarma: 1=enciende la alarma, 0=apaga la alarma.

ENTRADAS			SALIDAS		
Botón de emergencia	Sensor de la puerta	Temporizador del horno	Horno	Puerta del horno	Alarma
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	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

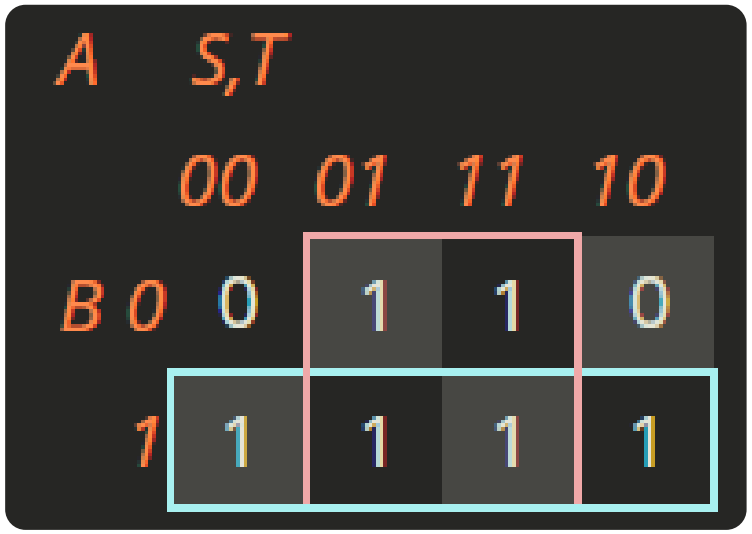
Ejercicio I (III):

3. Escribimos nuestra tabla de verdad (primero sin considerar los valores X)

H(B, S, T) = B'S'T'



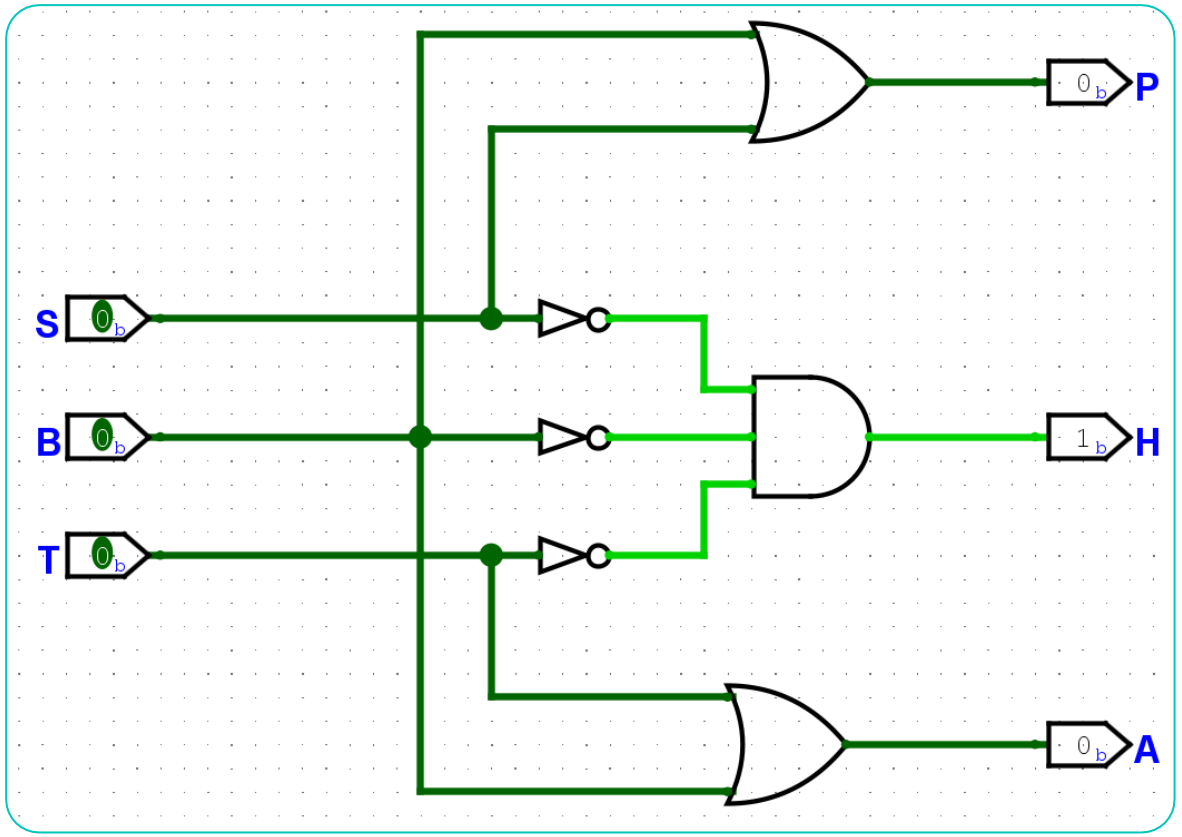
P(B, S, T) = S + B



A(B, S, T) = T + B

Ejercicio I (IV):

4. Se escriben los mapas de Karnaugh y se obtienen las ecuaciones booleanas.



Ejercicio I (V):

5. Dibujamos el esquemático del circuito combinacional.

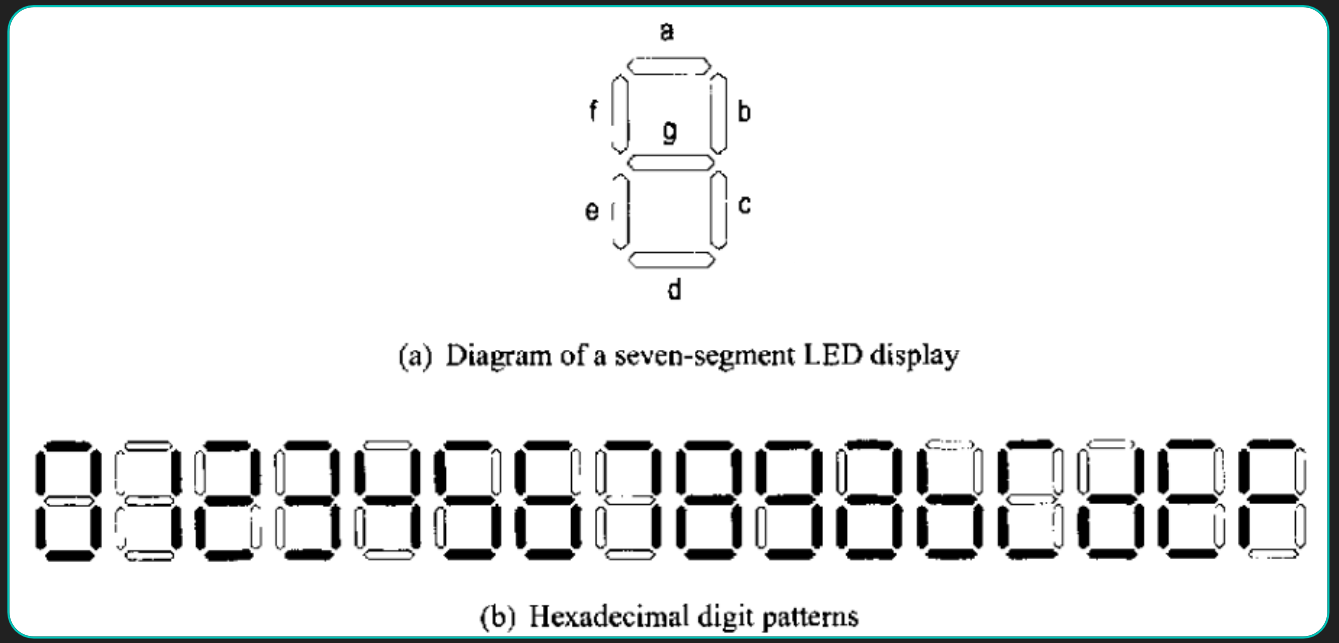
Ejercicio I (VI):

6. Codificamos nuestra solución en Verilog.

```
2
3  Definimos nuestro modulo. Se compone de la siguiente estructura:
4
5  module <nombre del modulo> (<entradas y salidas separadas por coma>):
6
7  */
8
9  module unidadDeControl(S,T,B,H,P,A);
10
11
12
13   // Definimos S como entrada (sensor de la puerta)
14   input S;
15
16   // Definimos B como entrada (botón de emergencia)
17   input T;
18
19   // Definimos T como entrada (temporizador del horno)
20   input B;
21
22
23
24   // Definimos P como salida (puerta del horno)
25   output H;
26
27   // Definimos H como salida (horno)
28   output P;
29
30   // Definimos A como salida (alarma)
31   output A;
32
33
34
35   // Le damos un valor a la salida H, utilizando la ecuación booleana
36   assign H = (~S)&(~T)&(~B);
37
38   // Le damos un valor a la salida P, utilizando la ecuación booleana
39   assign P = S|B;
40
41   // Le damos un valor a la salida A, utilizando la ecuación booleana
42   assign A = T|B;
43
44
45
46 // Finalizamos nuestro modulo
47 endmodule
```

Ejercicio II (I):

Diseñe un circuito que muestre los dígitos hexadecimales (del 0 al F) en un LED de siete segmentos, a partir de una entrada de cuatro bits.



(a) Diagram of a seven-segment LED display

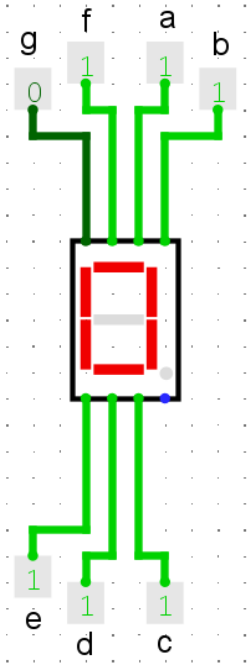
(b) Hexadecimal digit patterns

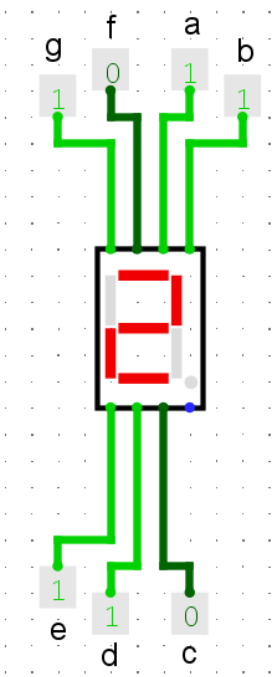
Ejercicio II (II):

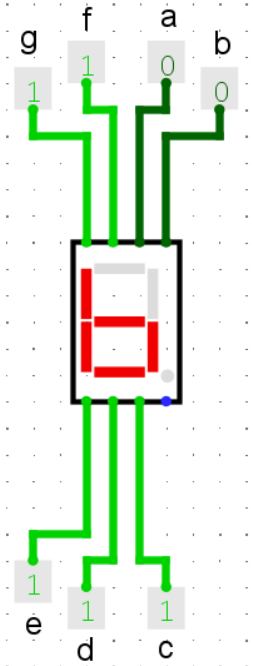
BINARIO	HEXADECIMAL
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Ejercicio II (III):

```
module hexadecimalLED(numeroMostrar,LED);

  // Se define la entrada de tipo wire de 4 bits de ancho, que es el numero a mostrar
  input wire [3:0] numeroMostrar;

  // Se define la salida de tipo reg de 7 bits de ancho, que es cada segmento del display LED
  output reg [6:0] LED;

  // Siempre, cuando alguna entrada cambia, se ejecuta lo siguiente
  always @(*) begin

    // En caso de que el numero a mostrar cambie a alguno de los siguientes valores, se ejecuta lo sig
    case(numeroMostrar)

      // Se va mostrando el numero en el display LED
      4'h0: LED[6:0] = 7'b1111110;
      4'h1: LED[6:0] = 7'b0110000;
      4'h2: LED[6:0] = 7'b1101110;
      4'h3: LED[6:0] = 7'b1111001;
      4'h4: LED[6:0] = 7'b0110011;
      4'h5: LED[6:0] = 7'b1011011;
      4'h6: LED[6:0] = 7'b1011111;
      4'h7: LED[6:0] = 7'b1110000;
      4'h8: LED[6:0] = 7'b1111111;
      4'h9: LED[6:0] = 7'b1111011;
      4'ha: LED[6:0] = 7'b1110111;
      4'hb: LED[6:0] = 7'b0011111;
      4'hc: LED[6:0] = 7'b1001110;
      4'hd: LED[6:0] = 7'b0111101;
      4'he: LED[6:0] = 7'b1001111;

      // El caso por defecto es numeroMostrar = 4'hf
      default: LED[6:0] = 7'b1000111;

    // Cerramos el case
    endcase

  // Cerramos el bloque always
  end

endmodule
```







Ejercicio II (IV):

Solución de la tarea I (I):

1. Expanda en forma canónica SOP (Sum of Products, Suma de Productos) la siguiente expresión:

$$g(x,b) = \overline{b} + x$$

$g(x,b) = \overline{b}(\boldsymbol{x} + \overline{\boldsymbol{x}}) + x(\boldsymbol{b} + \overline{\boldsymbol{b}})$ aplicando idempotencia

$g(x,b) = \overline{b} \cdot x + \overline{b} \cdot \overline{x} + x \cdot b + x \cdot \overline{b}$ aplicando distribución

$g(x,b) = \overline{\boldsymbol{b}} \cdot \boldsymbol{x} + \overline{b} \cdot \overline{x} + x \cdot b$ simplificando $\overline{b} \cdot x + x \cdot \overline{b}$

Solución de la tarea I (II):

2. Reduzca el siguiente polinomio usando mapas de Karnaugh, en forma canónica SOP (Sum of Products, Suma de Productos):

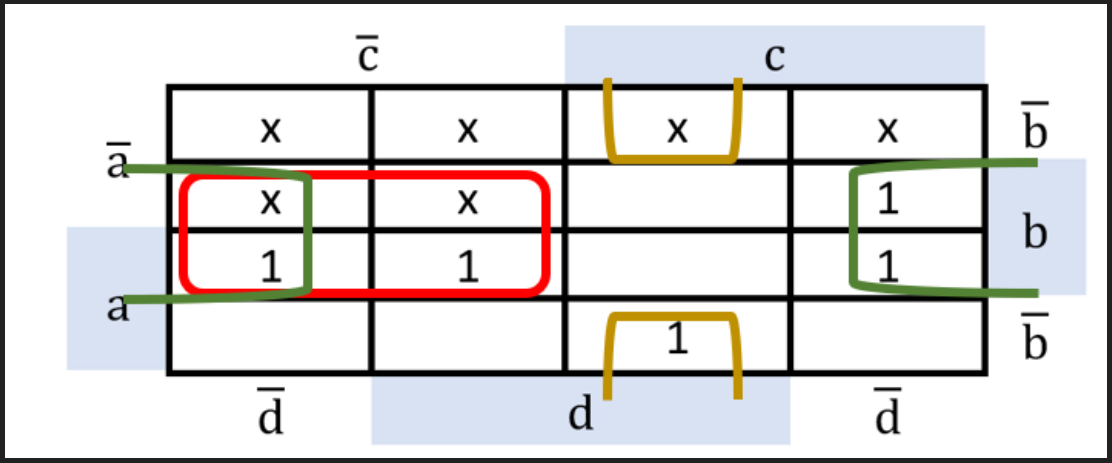
$$H(a,b,c,d) = \Sigma_m(0,2,3,6,7,8,9,10,11)$$

$$H(a,b,c,d) = \overline{a} \cdot c + a \cdot \overline{b} + \overline{b} \cdot \overline{d}$$

Solución de la tarea I (III):

3. En la siguiente función el símbolo ∅ indica aquellas posiciones "no importa". Reduzca usando mapas de Karnaugh (exprese en SOP):

$$Q(a,b,c,d) = \Sigma_m(6,11,12,13,14) + \emptyset(0,1,2,3,4,5)$$

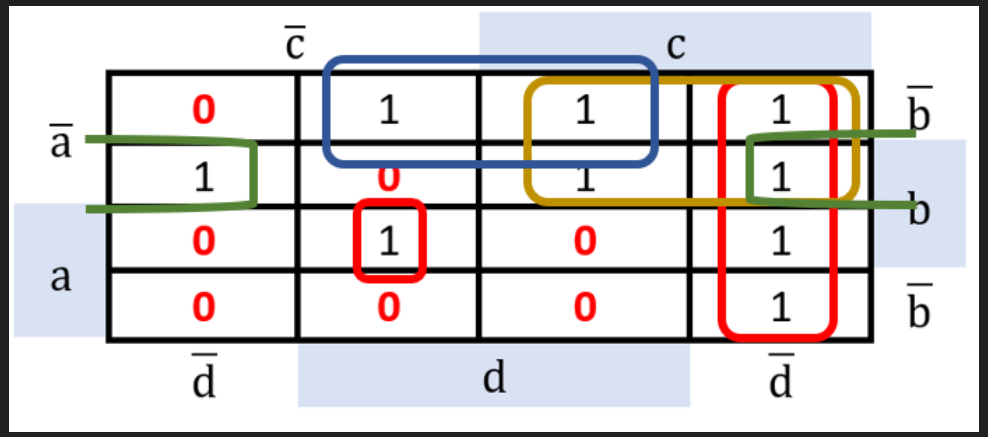


$$Q(a,b,c,d) = b\bar{c} + b\bar{d} + \bar{b}cd$$

Solución de la tarea I (IV):

4. Reduzca el siguiente polinomio de ceros usando mapas de Karnaugh, en forma canónica SOP (Sum of Products, Suma de Productos):

$$Q(v,w,x,y) = \Pi_M(0,5,8,9,11,12,15)$$

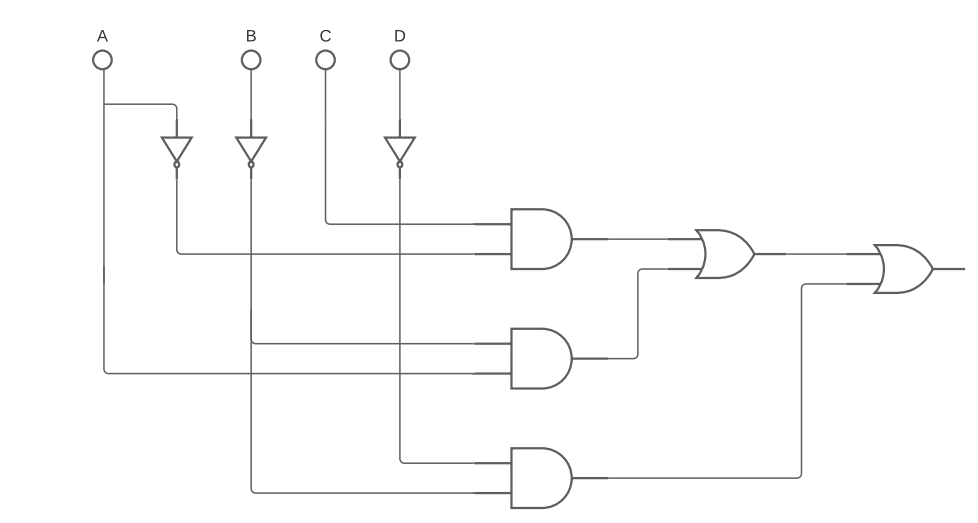


$$Q(v,w,x,y) = \overline{a} \cdot c + c \cdot \overline{d} + \overline{a} \cdot \overline{b} \cdot d + \overline{a} \cdot b \cdot \overline{d} + a \cdot b \cdot \overline{c} \cdot d$$

Solución de la tarea I (V):

5. Dibuje el circuito que se obtiene de la solución del ejercicio 2:

$$H(a,b,c,d) = \overline{a} \cdot c + a \cdot \overline{b} + \overline{b} \cdot \overline{d}$$



Solución de la tarea I (VI):

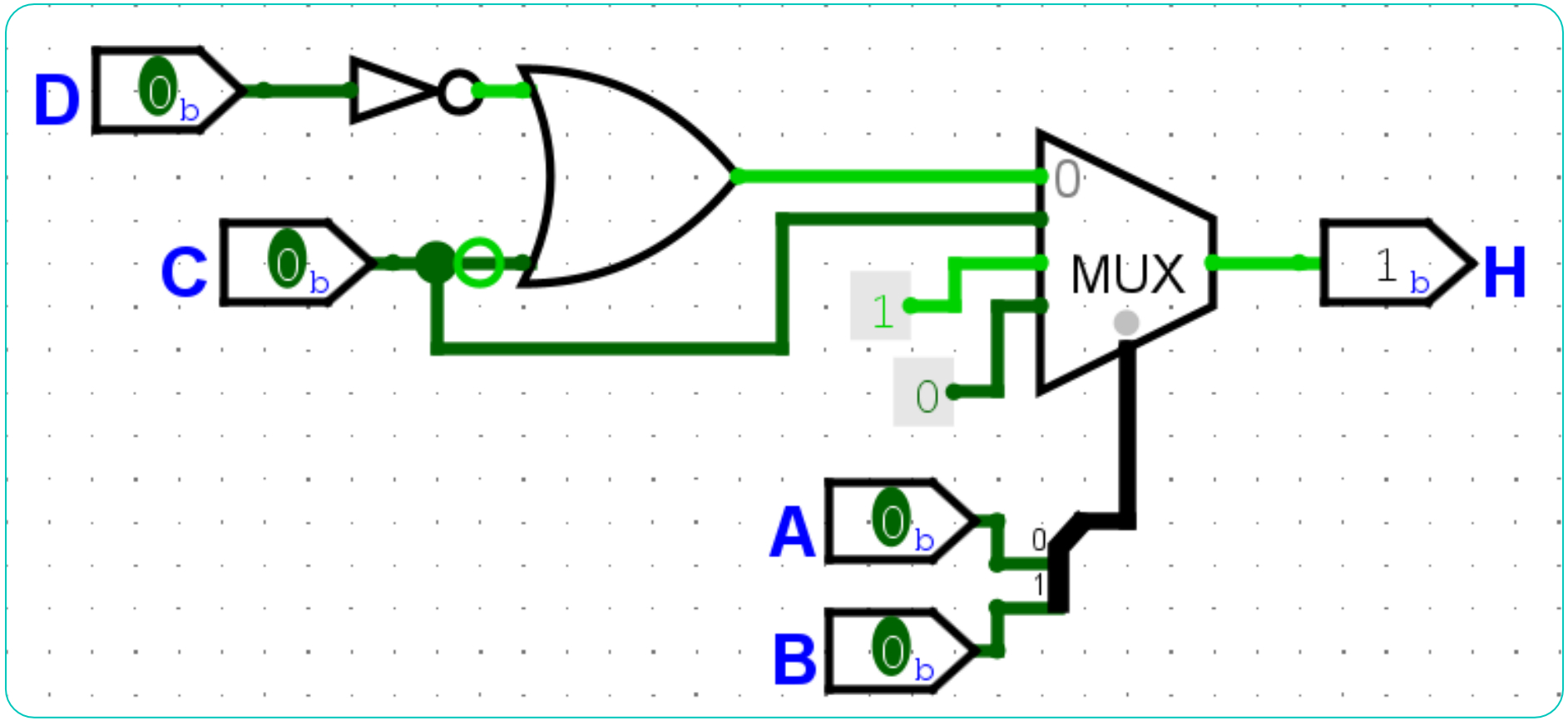
6. Implemente el sistema usado en el ejercicio 2 (sin simplificar) utilizando un multiplexor de 4 a 1:
   a) Un multiplexor 4 a 1 quiere decir que se tienen 4 entradas y 1 salida, por lo que se necesitan 2 bits de selección, que en este caso serán a y b.
   b) Esto parte la tabla de verdad en cuatro "subtablas" de verdad, que deben ser analizadas de forma individual.

$$H(a,b,c,d)\rfloor_{\{a,b\}=00} = c + \overline{d} \qquad H(a,b,c,d)\rfloor_{\{a,b\}=10} = 1$$

$$H(a,b,c,d)\rfloor_{\{a,b\}=01} = c \qquad H(a,b,c,d)\rfloor_{\{a,b\}=11} = 0$$

$$H(a,b,c,d) = \Sigma_m(0,2,3,6,7,8,9,10,11)$$

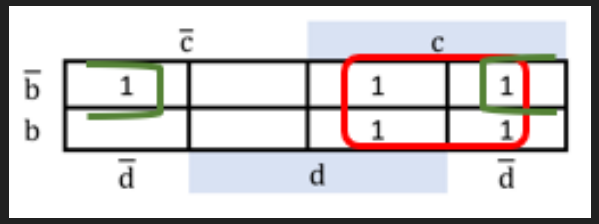
a	b	c	d	H
0	0	0	0	**1**
0	0	0	1	0
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	0
0	1	0	1	0
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**1**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Solución de la tarea I (VII):

Solución de la tarea I (VIII):

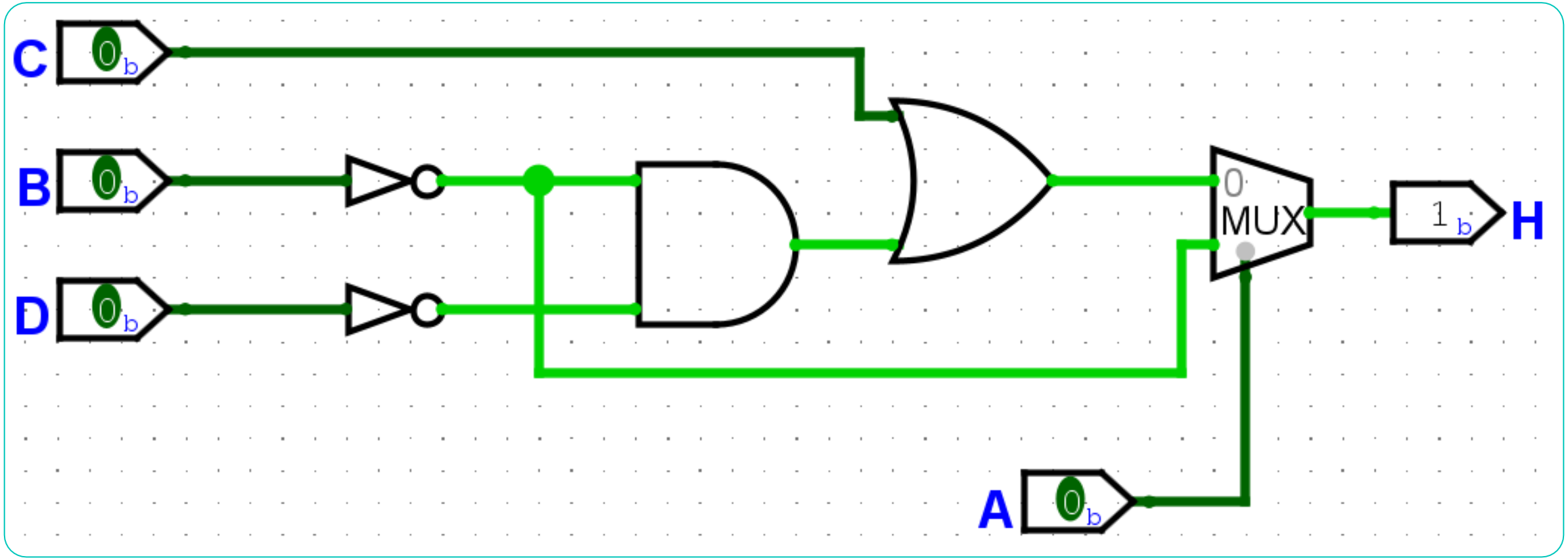
6. Implemente el sistema usado en el ejercicio 2 (sin simplificar) utilizando un multiplexor de 2 a 1:
   a) Un multiplexor 2 a 1 quiere decir que se tienen 2 entradas y 1 salida, por lo que se necesita 1 bit de selección, que en este caso será a.
   b) Esto parte la tabla de verdad en dos "subtablas" de verdad, que deben ser analizadas de forma individual.



$$H(a,b,c,d)\Big|_{\{a\}=1} = \overline{b}$$

$$H(a,b,c,d)\Big|_{\{a\}=0} = c + \overline{b}\cdot\overline{d}$$

a	b	c	d	H
0	0	0	0	**1**
0	0	0	1	0
0	0	1	0	**1**
0	0	1	1	**1**
0	1	0	0	0
0	1	0	1	0
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**1**
1	0	0	1	**1**
1	0	1	0	**1**
1	0	1	1	**1**
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



Solución de la tarea I (IX):

```
`timescale 1ns/1ps

module tb_multiplexor2a1;

  wire salida;
  reg [2:0] sab;

  multiplexor2a1 multiplexor(
    .s(sab[2]),
    .a(sab[1]),
    .b(sab[0]),
    .o(salida)
  );

  initial begin
    sab = 0;
    repeat (7) begin
      #10
      sab = sab+1;
    end
    #10
    $finish;
  end

  initial begin
    $monitor("t=%3ds=%b,a=%b,b=%b,salida=%b",$time,sab[2],sab[1],sab[0],salida);
  end

  initial begin
    $dumpfile("tb_multiplexor2a1.vcd");
    $dumpvars(0,tb_multiplexor2a1);
  end

endmodule
```

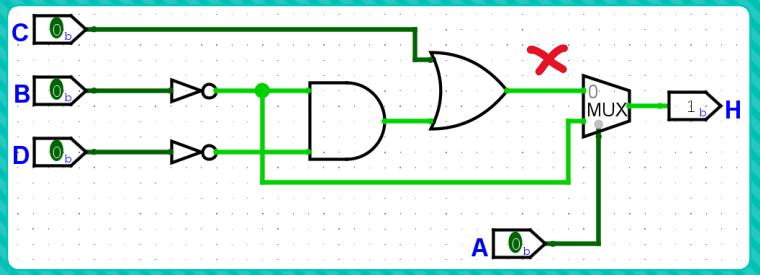
```
`timescale 1ns/1ps

module multiplexor2a1(a,b,s,o);

  // Entradas del multiplexor
  input a;
  input b;

  // Bit de selección
  input s;

  // Salida del multiplexor
  output o;

  // Asignación de la salida. Esto lo podemos leer como "si s, entonces o=b, sino o=a"
  assign o = s?b:a;

endmodule
```

Solución de la tarea I (X):

Escriba el código Verilog que implemente un multiplexor de 2 a 1.

```
`timescale 1ns/1ps

module tb_sistema;

  wire H;
  reg A,B,C,D;

  sistema sistema1(
    .A(A),
    .B(B),
    .C(C),
    .D(D),
    .H(H)
  );

  initial begin
    A=0;
    B=0;
    C=0;
    D=0;
    repeat (15) begin
      #10
      {A,B,C,D} = {A,B,C,D}+1;
    end
    #10
    $finish;
  end

  initial begin
    $display("A B C D | H");
    $monitor("%b %b %b %b | %b",A,B,C,D,H);
  end

  initial begin
    $dumpfile("tb_sistema.vcd");
    $dumpvars(0,tb_sistema);
  end

endmodule
```

```
`timescale 1ns/1ps

module multiplexor2a1(a,b,s,o);

  // Entradas del multiplexor
  input a;
  input b;

  // Bit de selección
  input s;

  // Salida del multiplexor
  output o;

  // Asignación de la salida. Esto lo podemos leer como "si s, entonces o=b, sino o=a"
  assign o = s?b:a;

endmodule



module sistema(input A,B,C,D, output H);
  wire X;
  assign X = C|((~B)&(~D));
  multiplexor2a1(.a(X),.b(~B),.s(A),.o(H));
endmodule

```



Solución de la tarea I (XI):

Utilizando el multiplexor diseñado, escriba en Verilog la solución del ejercicio 6.b:

Recursos:

1. Código en Verilog: https://github.com/OscarAraya18/EL3307.
2. Video del taller: https://www.youtube.com/channel/UCvBTNc-PIOWCkwbR0_A3oYQ.