Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Fundamentos de Arquitectura de Computadores Grupo #1
Estudiante: Kevin Josué Ruiz Rodríguez
Carné: 2018170538

Bitácora

20/3/2025

Se realizó la tabla de verdad 1 para interpretar las entradas, en la cual se le asigna un valor a cada una de las 4 combinaciones posibles.

Cuadro 1: Tabla de verdad de las entradas.

A	B	C	D	X_1	X_0
1	0	0	0	0	1
1	1	0	0	1	0
1	1	1	0	1	1
1	1	1	1	0	0

Teniendo estos valores asignados, se encontraron las siguientes ecuaciones booleanas para cada dígito utilizando los mintérminos para encontrar la suma de productos.

$$X_1 = AB\overline{CD} + ABC\overline{D}$$

$$X_0 A\overline{BCD} + ABC\overline{D}$$

Estas ecuaciones se simplificaonr utilizando álgebra booleana para facilitar la implementación del circuito.

$$X_1 = AB\overline{D}$$

$$X_0 A\overline{D}\overline{(B \oplus C)}$$

Además se realizó la tabla para el decodificador de suma

Cuadro 2: Tabla de verdad del decodificador de suma.

X_1	X_0	Y_1	Y_0	S_1	S_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	1	0

Para simplificar la tabla 2 se realizaron los siguientes mapas de Karnaugh

Cuadro 3: Mapa K para S_1.

$X_1X_0 \vert Y_1Y_0$	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	1	0	1	0
10	1	1	0	0

Cuadro 4: Mapa K para S_0.

$X_1X_0 \vert Y_1Y_0$	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	1	0	0	1
10	0	1	1	0

30/3/2025

Del mapa K 3 se obtiene la ecuación

$$S_1 = \overline{X_1X_0}Y_1 + \overline{X_1}X_0\overline{Y_1}Y_0 + \overline{X_1}Y_1\overline{Y_0} + X_1\overline{Y_1Y_0} + X_1X_0Y_1Y_0 + X_1\overline{X_0Y_1}$$

Esta ecuación para Z_1 requiere muchas compuertas en su implementación, por lo que se puede simplificar en la siguiente ecuación.

$$S_1 = (X_1 \oplus Y_1) \oplus X_0Y_0$$

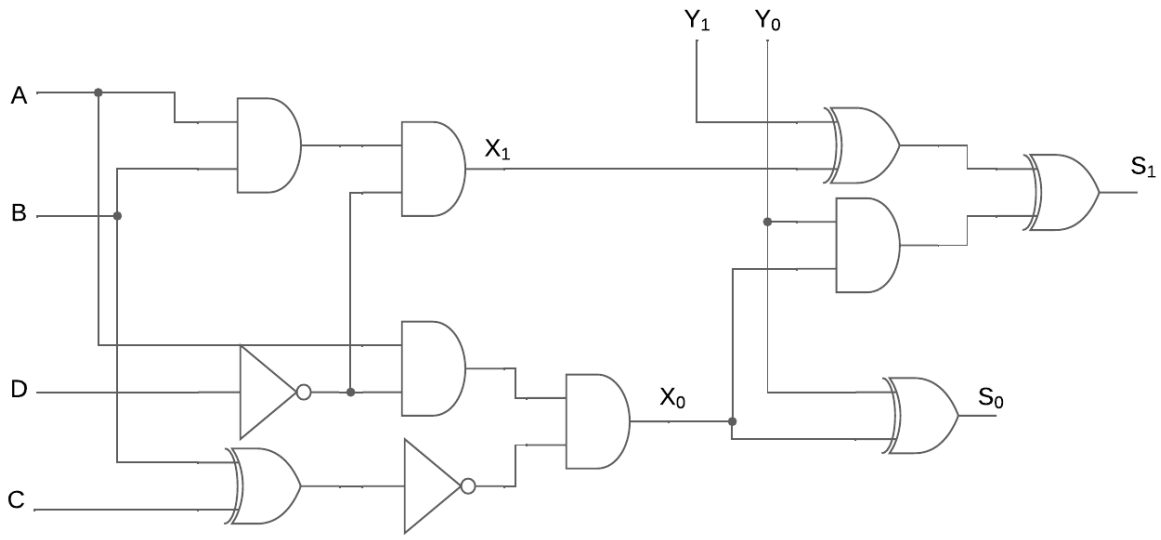

Figura 1: Circuito propuesto para los decodificadores de número y suma.

De la misma manera, con el mapa K 4 se obtiene una ecuación para este dígito

$$S_0 = X_0\overline{Y_0} + \overline{X_0}Y_0$$

Se puede observar que esta ecuación corresponde a la operación de o exclusivo, o XOR

$$S_0 = X_0 \oplus Y_0$$

Con estas ecuaciones para encontrar el número a sumar, y la suma de los 2 números se propone el circuito de la figura

31/3/2025

Basándose en el circuito de la figura 1 se montó el circuito utilizando un circuito integrado 74ls04 para 4 compuertas NOT, la conexión de este puede ver en la figura 2, 74ls86 para 4 compuertas XOR, cuya conexión se puede ver en la figura 3 y 2 74ls08 para 8 compuertas AND, el cual se conecta según la figura 4, además de un 74ls48 como BCD, para cual se usó como guía el pinout de la figura 5, el cual envía la señal a un display de 7 segmentos 5161AS, el cual se conecta según la figura 6.
El resultado se puede observar en la figura 7, sin embargo, no se pudo comprobar el funcionamiento debido a que no se recibió respuesta.

8/4/2025

Ya que no se pudo realizar la implementación en protoboard, se realizó una simulación y pruebas en SystemVerilog, para verificar la validez del diseño planteado.

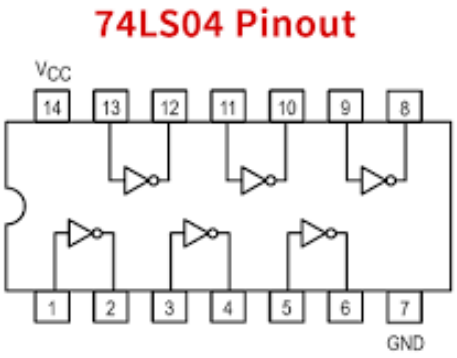

Figura 2: 74ls04 pinout.

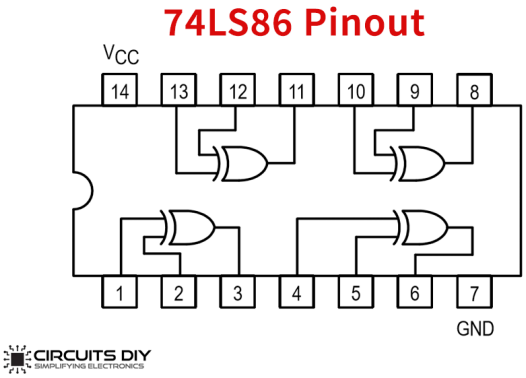

Figura 3: 74ls86 pinout.

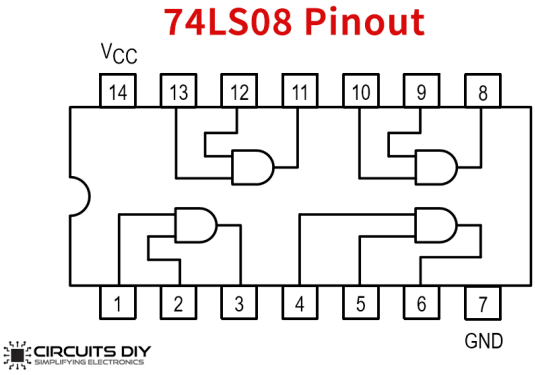

Figura 4: 74ls08 pinout.

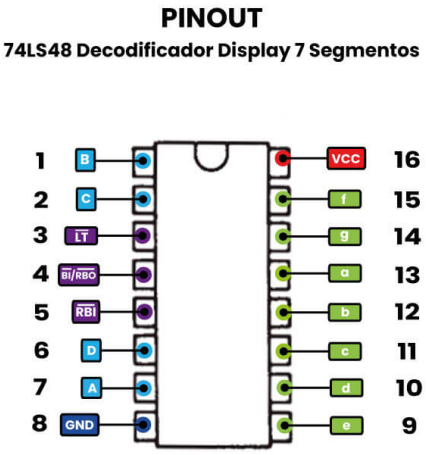

Figura 5: 74ls48 pinout.

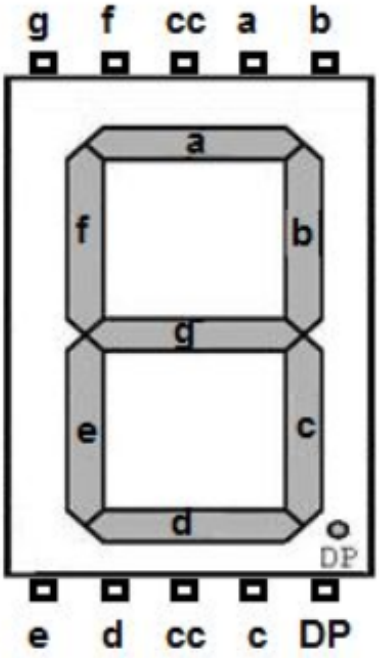

Figura 6: 5161AS pinout.

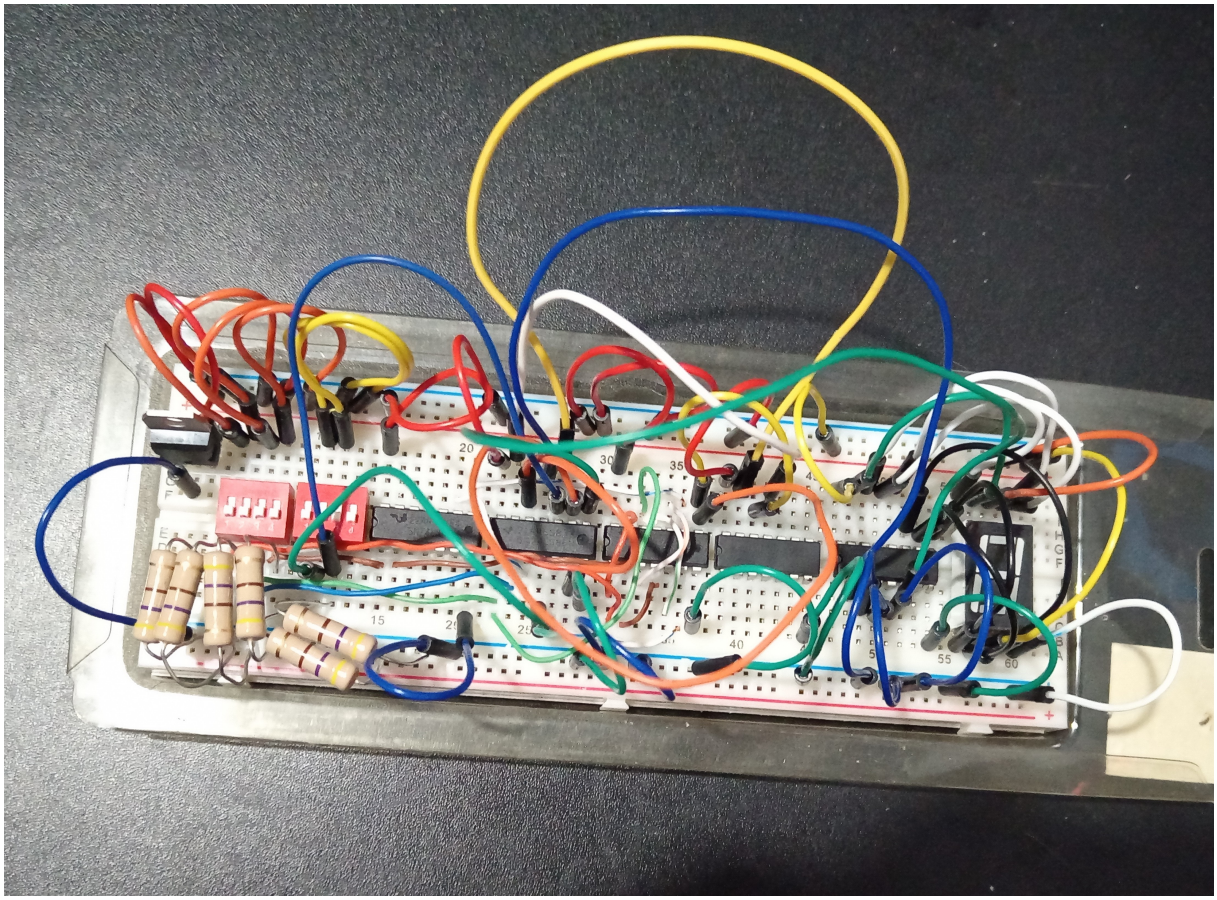

Figura 7: Circuito montado en Protoboard.

```
1  module inDecoder (input logic [3: 0] in,
2                    output logic [1: 0] x);
3                     //a      //d      //b      //c
4      assign x[0] = in[3] & ~in[0] & ~(in[2] ^ in[1]);
5                     //a      //b      //d
6      assign x[1] = in[3] & in[2] & ~in[0];
7
8  endmodule
```

Figura 8: Implemtación del codificador de entrada en SystemVerilog.

```
1  module inDecoder_tb;
2
3     logic [3: 0] in;
4     logic [1: 0] x;
5
6     inDecoder indec (in, x);
7
8     initial begin
9        in = 4'b1000; #10;
10       assert (x == 2'b01) else $error("failed 1");
11       in = 4'b1100; #10;
12       assert (x == 2'b10) else $error("failed 2");
13       in = 4'b1110; #10;
14       assert (x == 2'b11) else $error("failed 3");
15       in = 4'b1111; #10;
16       assert (x == 2'b00) else $error("failed 4");
17    end
18 endmodule
19
```

Figura 9: Testbench del codificador de entrada en SystemVerilog.

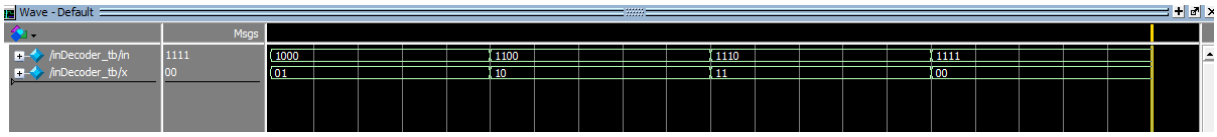

Figura 10: Resultado del testbench del codificador de entrada en ModelSim.

```
1  module sumDecoder(input logic [1: 0] x, y,
2                    output logic [1: 0]s);
3
4     assign s[0] = x[0] ^ y[0];
5     assign s[1] = (x[1] ^ y[1]) ^ x[0] & y[0];
6
7  endmodule
8
```

Figura 11: Implemtación del codificador de suma en SystemVerilog.

```
1  module sumDecoder_tb;
2
3     logic [1: 0] x, y, s;
4
5     sumDecoder sumdec (x, y, s);
6
7     initial begin
8        y = 2'b00;
9        x = 2'b00; #10;
10       assert (s === 2'b00) else $error ("failed 0+0");
11       x = 2'b01; #10;
12       assert (s === 2'b01) else $error ("failed 1+0");
13       x = 2'b10; #10;
14       assert (s === 2'b10) else $error ("failed 2+0");
15       x = 2'b11; #10;
16       assert (s === 2'b11) else $error ("failed 3+0");
17       y = 2'b01;
18       x = 2'b00; #10;
19       assert (s === 2'b01) else $error ("failed 0+1");
20       x = 2'b01; #10;
21       assert (s === 2'b10) else $error ("failed 1+1");
22       x = 2'b10; #10;
23       assert (s === 2'b11) else $error ("failed 2+1");
24       x = 2'b11; #10;
25       assert (s === 2'b00) else $error ("failed 3+1");
26       y = 2'b10;
27       x = 2'b00; #10;
28       assert (s === 2'b10) else $error ("failed 0+2");
29       x = 2'b01; #10;
30       assert (s === 2'b11) else $error ("failed 1+2");
31       x = 2'b10; #10;
32       assert (s === 2'b00) else $error ("failed 2+2");
33       x = 2'b11; #10;
34       assert (s === 2'b01) else $error ("failed 3+2");
35       y = 2'b11;
36       x = 2'b00; #10;
37       assert (s === 2'b11) else $error ("failed 0+3");
38       x = 2'b01; #10;
39       assert (s === 2'b00) else $error ("failed 1+3");
40       x = 2'b10; #10;
41       assert (s === 2'b01) else $error ("failed 2+3");
42       x = 2'b11; #10;
43       assert (s === 2'b10) else $error ("failed 3+3");
44    end
45 endmodule
46
```

Figura 12: Testbench del codificador de suma en SystemVerilog.

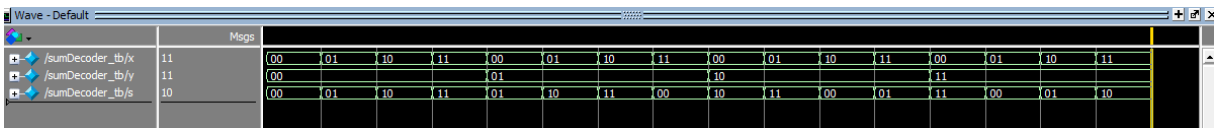

Figura 13: Resultado del testbench del codificador de suma en ModelSim.

```
1   module register (input logic rst, en,
2                    input logic [1: 0] data,
3                    output logic [1: 0] q);
4
5      always_ff @(posedge en or posedge rst)
6         if (rst) q = 2'b00;
7         else
8            if (en) q = data;
9
10  endmodule
11
```

Figura 14: Módulo de registro en SystemVerilog.

```
1   module comparator (input [1: 0] a,
2                      output equal);
3
4      assign equal = (a[0] & a[1]);
5
6   endmodule
7
```

Figura 15: Módulo de comparador en SystemVerilog.

```
1   module main (input logic rst, en,
2                input logic [3: 0] in,
3                output logic motor_on,
4                output logic [1: 0] acc);
5
6      logic [1: 0] x, res;
7
8      inDecoder indec (in, x);
9
10     sumDecoder sum (x, acc, res);
11
12     register res_reg (rst, en, res, acc);
13
14     comparator cmp (acc, motor_on);
15
16  endmodule
17
```

Figura 16: Implementación del circuito completo en SystemVerilog.

```
 1  module main_tb;
 2
 3      logic rst, en;
 4      logic [3: 0] in;
 5      logic motor_on;
 6      logic [1: 0] acc;
 7
 8      main testing (rst, en,
 9                    in, motor_on, acc);
10
11      always #5 en = ~en;
12
13      initial begin
14          en = 1'b0;
15          rst = 1'b1; #10;
16          rst = 1'b0;
17          in = 4'b1000; #40;
18          in = 4'b1100; #40;
19          in = 4'b1110; #40;
20          in = 4'b1111; #40;
21          rst = 1'b1;
22          #20; $stop;
23      end
24  endmodule
25
```

Figura 17: Testbench para el circuito completo.

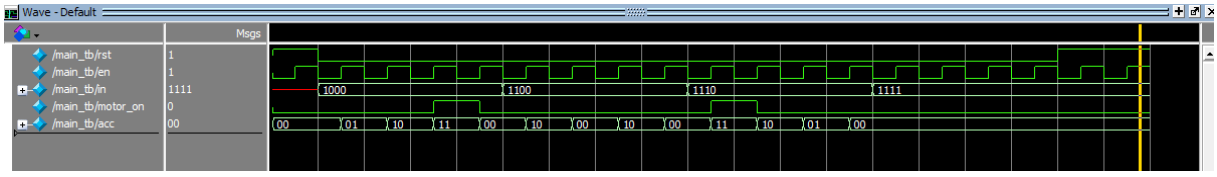

Figura 18: Resultado del testbench del circuito completo.