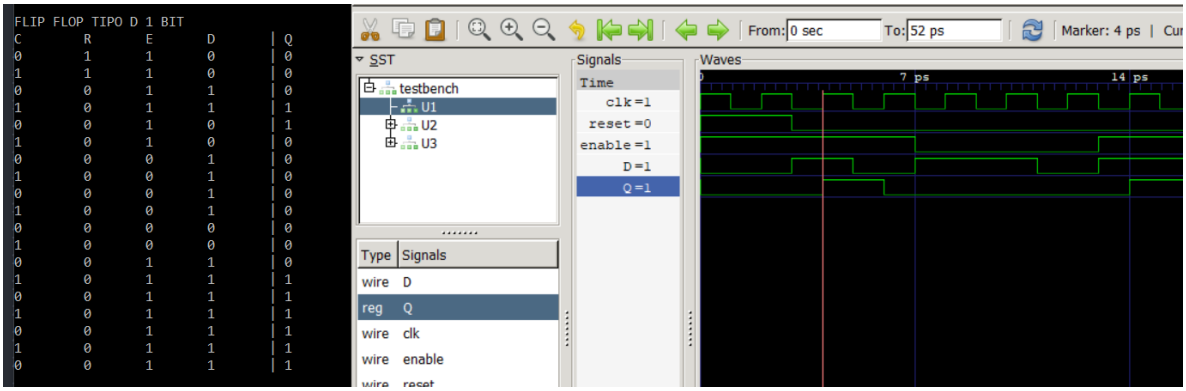
Ejercicio #1

Flip Flop D 1 Bit con Reset y Enable.

```
6   module flipflopD (input wire clk, reset, enable, D,
7                     output reg Q);
8
9       always @ ( posedge clk, posedge reset ) begin
10        if (reset) begin
11          Q <= 1'b0;
12        end
13        else if (enable) begin //SI EL ENABLE == 1 DEJA PASAR EL VALOR DE D A LA SALIDA Q
14          Q <= D;
15        end
16      end
17  endmodule // flipflopD
```

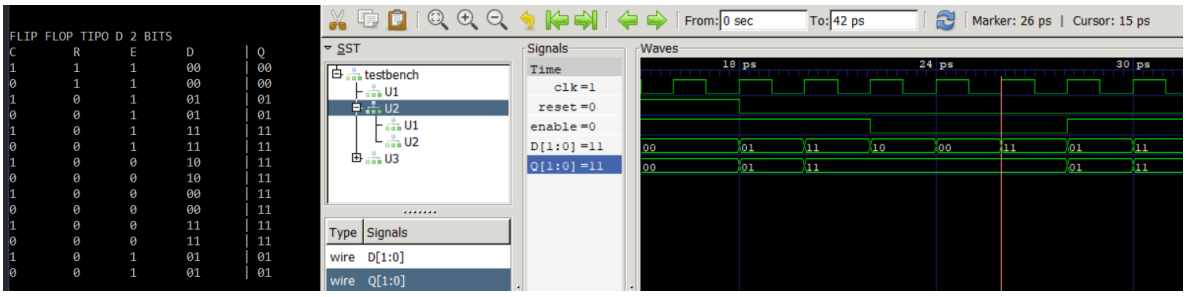


```
FLIP FLOP TIPO D 1 BIT
C       R       E       D       | Q
0       1       1       0       | 0
1       1       1       0       | 0
0       0       1       1       | 0
1       0       1       1       | 1
0       0       1       0       | 1
1       0       1       0       | 0
0       0       0       1       | 0
1       0       0       1       | 0
0       0       0       1       | 0
1       0       0       1       | 0
0       0       0       0       | 0
1       0       0       0       | 0
0       0       1       1       | 0
1       0       1       1       | 1
0       0       1       1       | 1
1       0       1       1       | 1
0       0       1       1       | 1
1       0       1       1       | 1
0       0       1       1       | 1
```

Cuando el enable = 1 la señal de D es la misma que Q, luego de un cambio positivo en el flanco de reloj, si el enable = 0, ya no deja pasar la señal de D a Q, y Q continua con el ultimo valor que obtuvo.

Flip Flop D 2 Bits con Reset y Enable.

Se utilizó el módulo del Flip Flop D de un bit para cada una de las salidas del flip flop de 2 bits. Y el funcionamiento es el mismo solo que con 2 bits de entrada y 2 de salida.

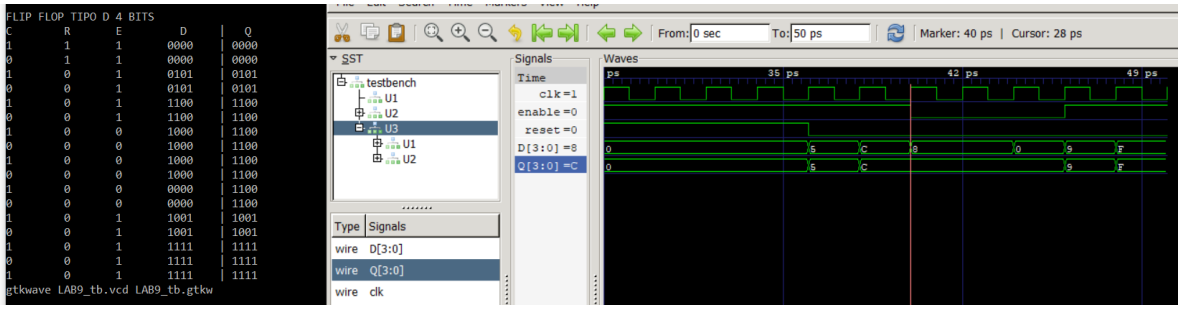


```
FLIP FLOP TIPO D 2 BITS
C       R       E       D       | Q
1       1       1       00      | 00
0       1       1       00      | 00
1       0       1       01      | 01
0       0       1       01      | 01
1       0       1       11      | 11
0       0       1       11      | 11
1       0       0       10      | 11
0       0       0       10      | 11
1       0       0       00      | 11
0       0       0       00      | 11
1       0       0       11      | 11
0       0       0       11      | 11
1       0       1       01      | 01
0       0       1       01      | 01
```

```
19  module flipflop2D (input wire clk, reset, enable,
20                     input wire [1:0]D,
21                     output wire [1:0]Q);
22      flipflopD U1(clk,reset, enable,D[0],Q[0]);//LLAMAMOS AL MODULO flipflopD
23      flipflopD U2(clk,reset, enable,D[1],Q[1]);//LLAMAMOS AL MODULO flipflopD
24
25  endmodule // flipflop2D
```

Flip Flop D 4 Bits con Reset y Enable.

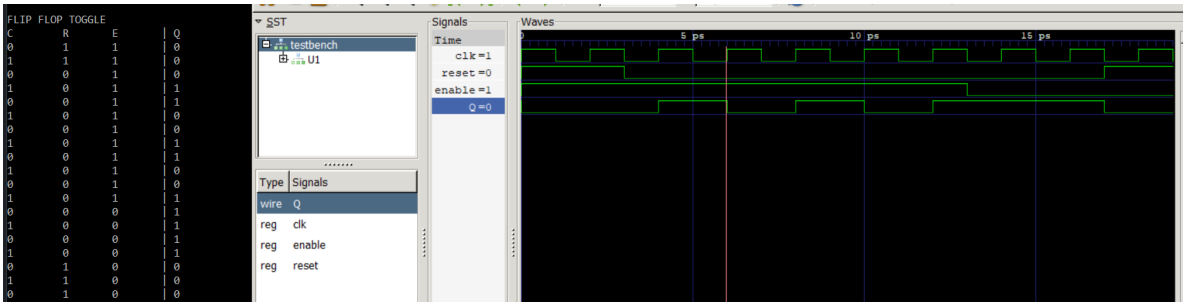
Se utilizó el módulo del Flip Flop D de 2 bits para cada dos entradas y dos de las salidas del flip flop de 4 bits. Y el funcionamiento es el mismo solo que con 4 bits de entrada y 4 de salida.

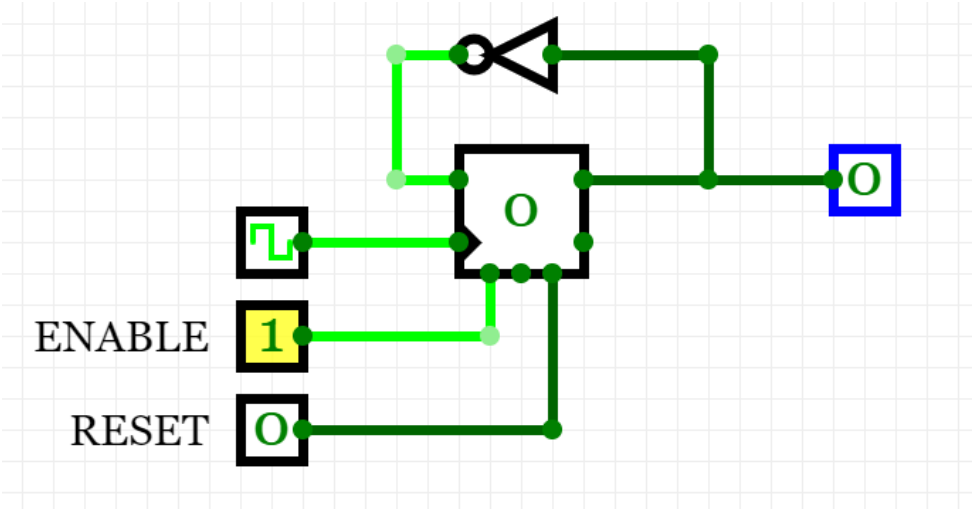



```
27  module flipflop4D (input wire clk, reset, enable,
28                     input wire [3:0]D,
29                     output wire [3:0]Q);
30      flipflop2D U1(clk, reset, enable,D[1:0], Q[1:0]);//LLAMAMOS AL MODULO FLIPFLOP 2 BITS
31      flipflop2D U2(clk, reset, enable,D[3:2], Q[3:2]);//LLAMAMOS AL MODULO FLIPFLOP 2 BITS
32
33  endmodule // flipflop4D
34
```

Ejercicio #2 Flip Flop T

```
19  module flipflopJK (input wire clk, reset, enable, J, K,
20                     output wire Q);
21     wire Qn,Kn,w1,w2, w3;
22     not U1(Qn, Q);
23     not U2(Kn, K);
24     nand U3(w1,Kn, Q);
25     nand U4(w2,J, Qn);
26     nand U5(w3, w1, w2);
27
28     flipflopD U6(clk,reset,enable,w3,Q);
29     endmodule//flipflopJK
30
```





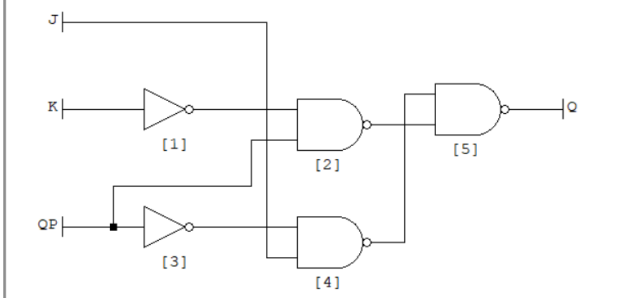
El funcionamiento del flip flop T es el siguiente, se toma el valor de la salida y lo invierte en cada periodo de cambio de reloj, se utilizo un flip flop D, en donde en la entrada tiene conectada la salida invertida.

Ejercicio #3 Flip Flop JK

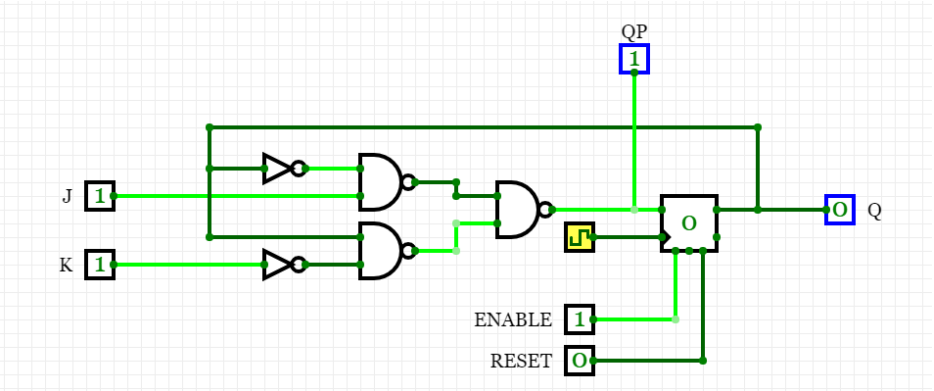
Tabla, ecuación reducida y Mapeo en Logic Friday

Term	J	K	QP	=>	Q
0	0	0	0		0
1	0	0	1		1
2	0	1	0		0
3	0	1	1		0
4	1	0	0		1
5	1	0	1		1
6	1	1	0		1
7	1	1	1		0

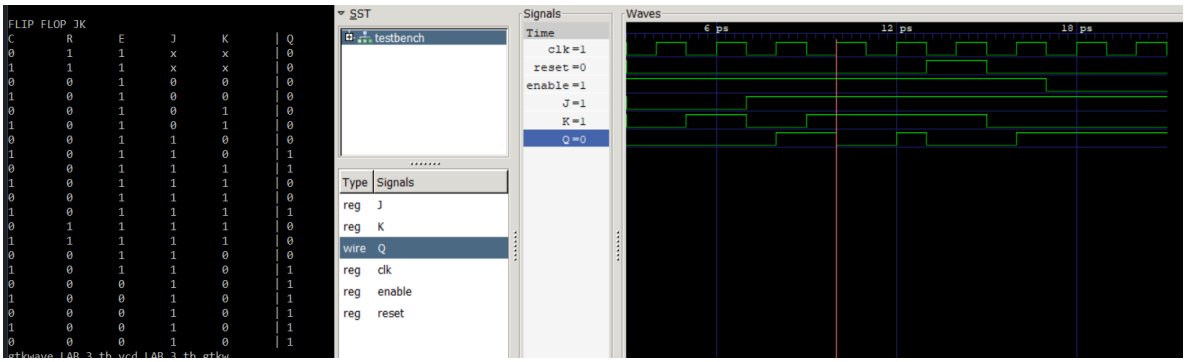
```
Minimized:
Q = K' QP + J QP';
```



Simulación En Circuit Verse



```
19  module flipflopJK (input wire clk, reset, enable, J, K,
20                     output wire Q);
21     wire Qn,Kn,w1,w2, w3;
22     not U1(Qn, Q);
23     not U2(Kn, K);
24     nand U3(w1,Kn, Q);
25     nand U4(w2,J, Qn);
26     nand U5(w3, w1, w2);
27
28     flipflopD U6(clk,reset,enable,w3,Q);
29     endmodule//flipflopJK
30
```

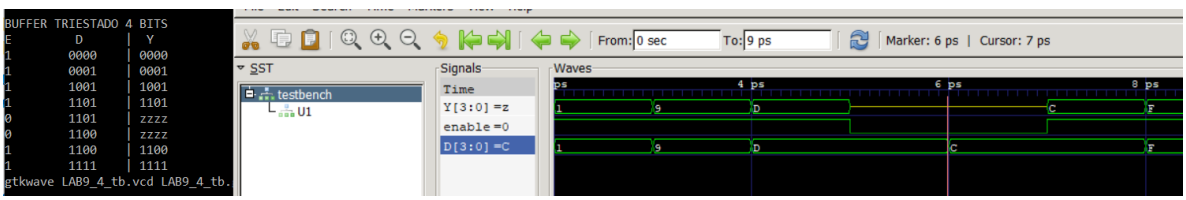


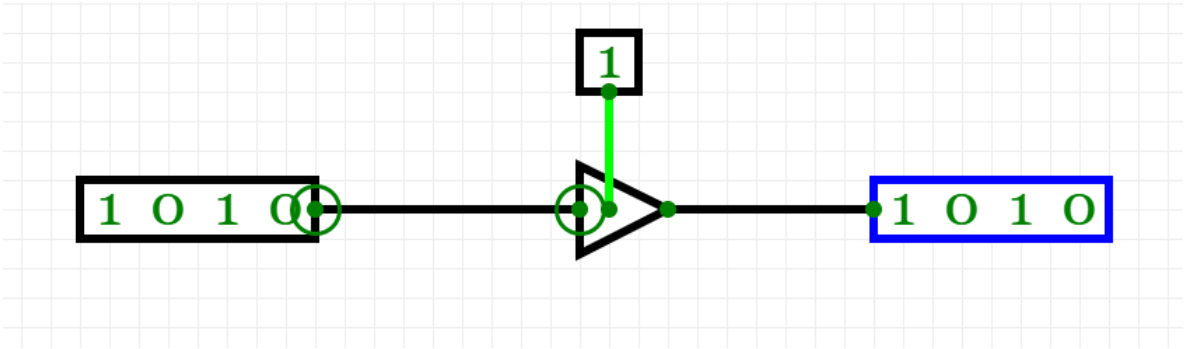
Se puede observar el funcionamiento del flip flop JK, cuando se coloca J y K = 0, la salida continua con su mismo valor. Cuando J = 0 y K = 1 la salida es 0. Cuando J = 1 y K = 0 la salida es 1. Cuando J y K = 1 la salida toma el valor que tenía anteriormente.

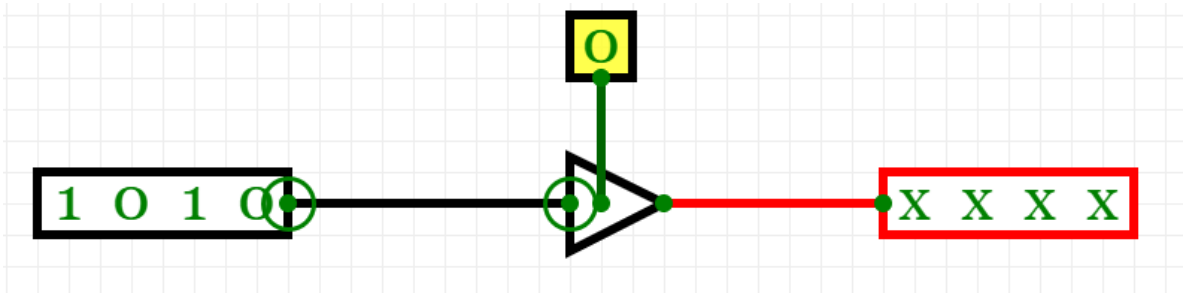
Ejercicio #4 Buffer triestado de 4 bits

Si enable = 1 deja pasar el dato de D, sino en coloca la salida en alta impedancia (valores con z).

```
6   module buffer4 (input wire [3:0]D,
7                   input wire enable,
8                   output wire [3:0]Y);
9     assign Y = enable ? D : 4'bz;
10   //si enable = 1 deja pasar el dato de D, sino en alta impedancia
11  endmodule // buffer4
12
```







Ejercicio #5

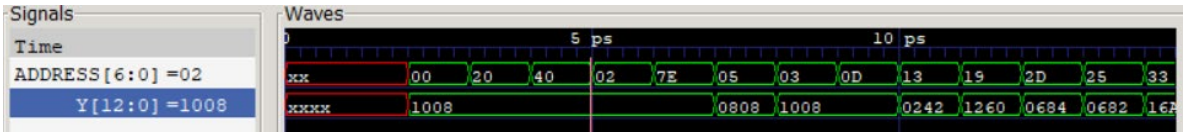
Tabla y Prueba de Direcciones de la ROM

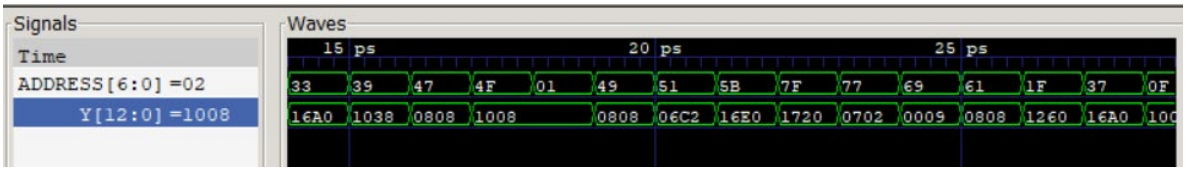
```
MEMORIA ROM
ADDRESS          |  Y
xxxxxxx          | xxxxxxxxxxxxx
0000000          | 1000000001000
0100000          | 1000000001000
1000000          | 1000000001000
0000010          | 1000000001000
1111110          | 1000000001000
0000101          | 0100000001000
0000011          | 1000000001000
0001101          | 1000000001000
0010011          | 0001001000010
0011001          | 1001001100000
0101101          | 0011010000100
0100101          | 0011010000010
0110011          | 1011010100000
0111001          | 1000000111000
1000111          | 0100000001000
1001111          | 1000000001000
0000001          | 1000000001000
1001001          | 0100000001000
1010001          | 0011011000010
1011011          | 1011011100000
1111111          | 1011100100000
1110111          | 0011100000010
1101001          | 0000000001001
1100001          | 0100000001000
0011111          | 1001001100000
0110111          | 1011010100000
0001111          | 1000000001000
1001101          | 0100000001000
1110011          | 0011100000010
1111101          | 1011100100000
1111100          | 1000000001000
```

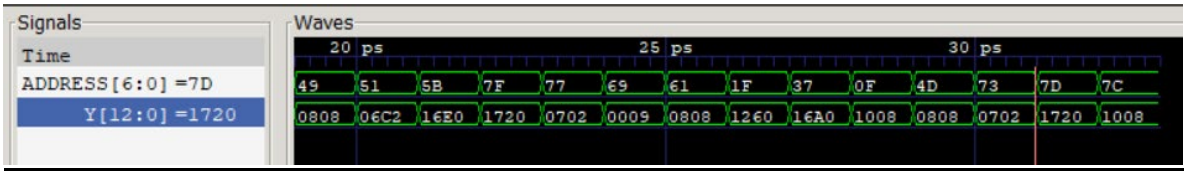
Inputs							Outputs											
x	x	x	x	x	x	0	1	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	x	1	0	1	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	x	1	1	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	1	x	1	1	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	x	1	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	x	x	1	0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	1	x	x	1	1	0	0	1	0	0	1	1	0	0	0	0
0	1	0	0	x	x	1	0	0	1	1	0	1	0	0	0	0	1	0
0	1	0	1	x	x	1	0	0	1	1	0	1	0	0	0	0	1	0
0	1	1	0	x	x	1	1	0	1	1	0	1	0	1	0	0	0	0
0	1	1	1	x	x	1	1	0	0	0	0	0	0	1	1	1	0	0
1	0	0	0	x	1	1	0	1	0	0	0	0	0	0	1	0	0	0
1	0	0	0	x	0	1	1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	1	x	1	1	1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	1	x	0	1	0	1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	x	x	1	0	0	1	1	0	1	1	0	0	0	1	0
1	0	1	1	x	x	1	1	0	1	1	0	1	1	1	0	0	0	0
1	1	0	0	x	x	1	0	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	x	x	1	0	0	0	0	0	0	0	0	1	0	0	1
1	1	1	0	x	x	1	0	0	1	1	1	0	0	0	0	0	1	0
1	1	1	1	x	x	1	1	0	1	1	1	0	0	1	0	0	0	0

Figure 1: Tabla de Verdad Ejercicio 05

Diagrama de Timing







Codigo.v

```
module ROM (input wire [6:0]Address,
            output reg [12:0]Y);
  always @(Address) begin

    case (Address)
      7'b0000000: Y = 13'b1000000001000 ;//primer caso con Don't cares
      7'b1000000: Y = 13'b1000000001000 ;
      7'b0100000: Y = 13'b1000000001000 ;
      7'b0010000: Y = 13'b1000000001000 ;
      7'b0001000: Y = 13'b1000000001000 ;
      7'b0000100: Y = 13'b1000000001000 ;
      7'b0000010: Y = 13'b1000000001000 ;
      7'b1111110: Y = 13'b1000000001000 ;
      7'b1111100: Y = 13'b1000000001000 ;
      7'b1111000: Y = 13'b1000000001000 ;
      7'b1110000: Y = 13'b1000000001000 ;
      7'b1100000: Y = 13'b1000000001000 ;
      7'b0111110: Y = 13'b1000000001000 ;
      7'b0011110: Y = 13'b1000000001000 ;
      7'b0001110: Y = 13'b1000000001000 ;
      7'b0000110: Y = 13'b1000000001000 ;
      7'b1011110: Y = 13'b1000000001000 ;
      7'b1001110: Y = 13'b1000000001000 ;
      7'b1000110: Y = 13'b1000000001000 ;
      7'b1000010: Y = 13'b1000000001000 ;
      7'b1101110: Y = 13'b1000000001000 ;
```

```
      7'b1100110: Y = 13'b1000000001000 ;
      7'b1100010: Y = 13'b1000000001000 ;
      7'b1110110: Y = 13'b1000000001000 ;
      7'b1110010: Y = 13'b1000000001000 ;
      7'b1111010: Y = 13'b1000000001000 ;
      7'b1111100: Y = 13'b1000000001000 ;//27 opciones para el primer caso
      7'b0000101: Y = 13'b0100000001000 ;//caso 2
      7'b0000111: Y = 13'b0100000001000 ;
      7'b0000001: Y = 13'b1000000001000 ;//caso 3
      7'b0000011: Y = 13'b1000000001000 ;
      7'b0001101: Y = 13'b1000000001000 ;//caso 4
      7'b0001111: Y = 13'b1000000001000 ;
      7'b0001001: Y = 13'b0100000001000 ;//caso 5
      7'b0001011: Y = 13'b0100000001000 ;
      7'b0010001: Y = 13'b0001001000010 ;//caso 6
      7'b0010011: Y = 13'b0001001000010 ;
      7'b0010101: Y = 13'b0001001000010 ;
      7'b0010111: Y = 13'b0001001000010 ;
      7'b0011001: Y = 13'b1001001100000 ;//caso 7
      7'b0011011: Y = 13'b1001001100000 ;
      7'b0011101: Y = 13'b1001001100000 ;
      7'b0011111: Y = 13'b1001001100000 ;
      7'b0100001: Y = 13'b0011010000010 ;//caso8
      7'b0100011: Y = 13'b0011010000010 ;
      7'b0100101: Y = 13'b0011010000010 ;
      7'b0100111: Y = 13'b0011010000010 ;
      7'b0101001: Y = 13'b0011010000100 ;//Caso 9
      7'b0101011: Y = 13'b0011010000100 ;
```

```
        7'b0101101: Y = 13'b0011010000100 ;
        7'b0101111: Y = 13'b0011010000100 ;
        7'b0110001: Y = 13'b1011010100000 ;//caso 10
        7'b0110011: Y = 13'b1011010100000 ;
        7'b0110101: Y = 13'b1011010100000 ;
        7'b0110111: Y = 13'b1011010100000 ;
        7'b0111001: Y = 13'b1000000111000 ;//caso 11
        7'b0111011: Y = 13'b1000000111000 ;
        7'b0111101: Y = 13'b1000000111000 ;
        7'b0111111: Y = 13'b1000000111000 ;
        7'b1000011: Y = 13'b0100000001000 ;//caso 12
        7'b1000111: Y = 13'b0100000001000 ;
        7'b1000001: Y = 13'b1000000001000 ;//caso 13
        7'b1000101: Y = 13'b1000000001000 ;
        7'b1001011: Y = 13'b1000000001000 ;//caso 14
        7'b1001111: Y = 13'b1000000001000 ;
        7'b1001001: Y = 13'b0100000001000 ;//caso 15
        7'b1001101: Y = 13'b0100000001000 ;
        7'b1010001: Y = 13'b0011011000010 ;//caso 16
        7'b1010011: Y = 13'b0011011000010 ;
        7'b1010101: Y = 13'b0011011000010 ;
        7'b1010111: Y = 13'b0011011000010 ;
        7'b1011001: Y = 13'b1011011100000 ;//caso 17
        7'b1011011: Y = 13'b1011011100000 ;
        7'b1011101: Y = 13'b1011011100000 ;
        7'b1011111: Y = 13'b1011011100000 ;
        7'b1100001: Y = 13'b0100000001000 ;//caso 18
        7'b1100011: Y = 13'b0100000001000 ;
```

```
        7'b1100101: Y = 13'b0100000001000 ;
        7'b1100111: Y = 13'b0100000001000 ;
        7'b1101001: Y = 13'b0000000001001 ;//caso 19
        7'b1101011: Y = 13'b0000000001001 ;
        7'b1101101: Y = 13'b0000000001001 ;
        7'b1101111: Y = 13'b0000000001001 ;
        7'b1110001: Y = 13'b0011100000010 ;//caso 20
        7'b1110011: Y = 13'b0011100000010 ;
        7'b1110101: Y = 13'b0011100000010 ;
        7'b1110111: Y = 13'b0011100000010 ;
        7'b1111001: Y = 13'b1011100100000 ;//caso21
        7'b1111011: Y = 13'b1011100100000 ;
        7'b1111101: Y = 13'b1011100100000 ;
        7'b1111111: Y = 13'b1011100100000 ;
        default:    Y = 13'bxxxxxxxxxxxxx ;
      endcase
    end


endmodule // ROM
```

Codigo Testbench Prueba de las direcciones

```
#1 ADDRESS = 7'b0000000;//caso1
#1 ADDRESS = 7'b0100000;
#1 ADDRESS = 7'b1000000;
#1 ADDRESS = 7'b0000010;
#1 ADDRESS = 7'b1111110;
#1 ADDRESS = 7'b0000101;//caso 2
#1 ADDRESS = 7'b0000011;//caso 3
#1 ADDRESS = 7'b0001101;//caso 4
#1 ADDRESS = 7'b0010011;//caso 6
#1 ADDRESS = 7'b0011001;//caso 7
#1 ADDRESS = 7'b0101101;//caso 9
#1 ADDRESS = 7'b0100101;//caso 8
#1 ADDRESS = 7'b0110011;//caso 10
#1 ADDRESS = 7'b0111001;//caso 11
#1 ADDRESS = 7'b1000111;//caso 12
#1 ADDRESS = 7'b1001111;//caso 14
#1 ADDRESS = 7'b0000001;//caso 13
#1 ADDRESS = 7'b1001001;//caso 15
#1 ADDRESS = 7'b1010001;//caso 16
#1 ADDRESS = 7'b1011011;//caso 17
#1 ADDRESS = 7'b1111111;//caso 21
#1 ADDRESS = 7'b1110111;//caso 20
#1 ADDRESS = 7'b1101001;//caso 19
#1 ADDRESS = 7'b1100001;//caso 18
#1 ADDRESS = 7'b0011111;//caso 7
#1 ADDRESS = 7'b0110111;//caso 10
#1 ADDRESS = 7'b0001111;//caso 4
#1 ADDRESS = 7'b1001101;//caso 15
```