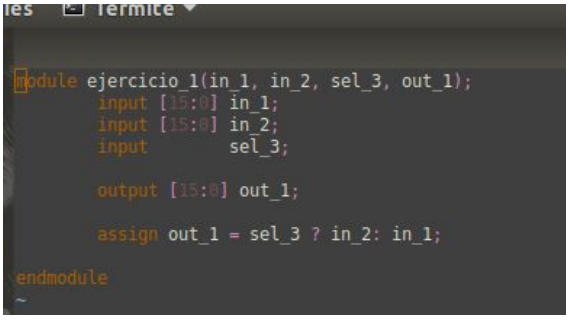
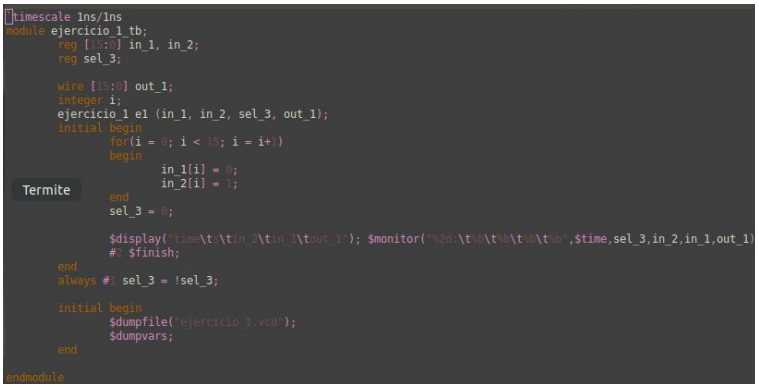
Ejercicios

1. Implementar un MUX 2:1 de 16-bits.

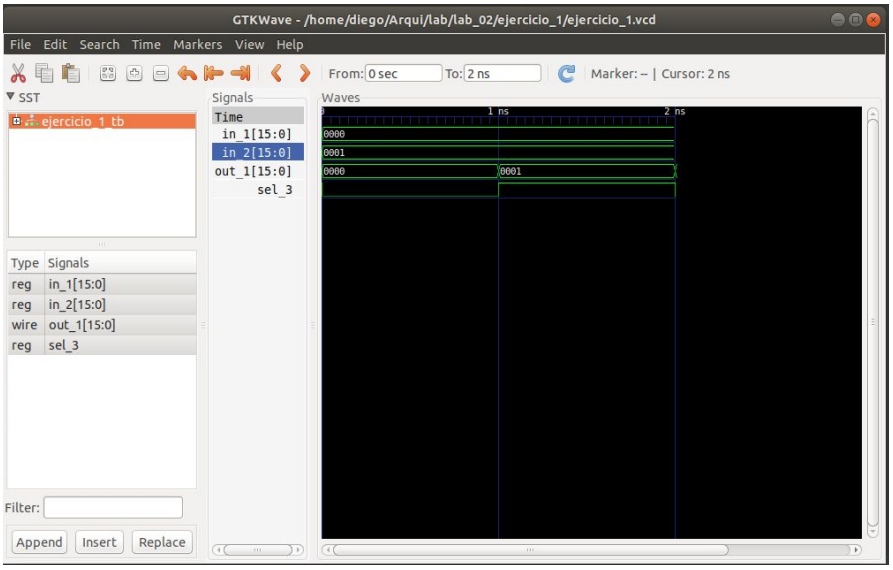
Lo que hice fue implementar un MUX 2:1 con un operador ternario (si sel es 1 out es in_2 de lo contrario es in_1) pero con los dos inputs y output como arrays [15:0] bits



Puse al input 1 como 16 ceros y al 2 como 16 unos mediante un for loop



Estos son los resultados:



2. Implementar los siguientes MUX :a) MUX 2:1 de 16 bits, b) MUX 8 : 1 de 16-bits, c) MUX 16:1 de 16-bits. TIP: Usar un módulo small (repetir) dentro del top.

a) se encuentra en el ejercicio 1

b) Hice uso de 7 MUX 2:1 para crear un MUX 8:1

```
module ejercicio_2_b(in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8, sel_1, sel_2, sel_3, out_1);

        input [15:0] in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8;
        input sel_1, sel_2, sel_3;
        wire [15:0] wi_1, wi_2, wi_3, wi_4, wi_5, wi_6;

        output [15:0] out_1;

        ejercicio_1 a1_0(in_1, in_2, sel_1, wi_1);
        ejercicio_1 a1_1(in_3, in_4, sel_1, wi_2);
        ejercicio_1 a1_2(in_5, in_6, sel_1, wi_3);
        ejercicio_1 a1_3(in_7, in_8, sel_1, wi_4);

        ejercicio_1 a2_0(wi_1, wi_2, sel_2, wi_5);
        ejercicio_1 a2_1(wi_3, wi_4, sel_2, wi_6);

        ejercicio_1 a3_0(wi_5, wi_6, sel_3, out_1);

endmodule
```

Termite

testbench:

```
`timescale 1ns/1ns
module ejercicio_2_b_tb;
        reg [15:0] in_1, in_2,in_3, in_4, in_5, in_6, in_7, in_8;
        reg sel_1, sel_2,sel_3;

        wire [15:0] out_1;

       ejercicio_2_b e2_b(in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8, sel_1, sel_2, sel_3, out_1);
        initial begin
                in_1 <= 0;
                in_2 <= 1;
                in_3 <= 1;
                in_4 <= 1;
                in_5 <= 1;
                in_6 <= 1;
                in_7 <= 1;
                in_8 <= 1;
                sel_1 = 0;
                sel_2 = 0;
                sel_3 = 0;

                $display("time\tsel_1\tsel_2\tsel_3\tout_1"); $monitor("%2d\t%b\t%b\t%b\t%b",$time, sel_1, sel_2, sel_3, out_1);
                #2 $finish;
        end

        always #1 sel_1 = sel_1;
        always #1 sel_2 = sel_2;
        always #1 sel_3 = ~sel_3;

        always #1 sel_1 = sel_1;
        always #1 sel_2 = ~sel_2;
        always #1 sel_3 = sel_3;

        always #1 sel_1 = sel_1;
        always #1 sel_2 = sel_2;
        always #1 sel_3 = sel_3;

        always #1 sel_1 = ~sel_1;
        always #1 sel_2 = sel_2;
        always #1 sel_3 = sel_3;

        always #1 sel_1 = sel_1;
        always #1 sel_2 = sel_2;
        always #1 sel_3 = sel_3;
```

ejercicio_2_b_tb.v

(estuve como demente 2 días intentando que los selectores del MUX sigan un orden creciente y la única manera en la que logré conseguirlo fue así, sigo sin entender el por qué de ello)

```
    always #1 sel_1 = sel_1;
    always #1 sel_2 = sel_2;
    always #1 sel_3 = ~sel_3;

    always #1 sel_1 = sel_1;
    always #1 sel_2 = ~sel_2;
    always #1 sel_3 = sel_3;

    always #1 sel_1 = sel_1;
    always #1 sel_2 = sel_2;
    always #1 sel_3 = sel_3;

    always #1 sel_1 = ~sel_1;
    always #1 sel_2 = sel_2;
    always #1 sel_3 = sel_3;

    always #1 sel_1 = sel_1;
    always #1 sel_2 = sel_2;
    always #1 sel_3 = sel_3;

    always #1 sel_1 = sel_1;
    always #1 sel_2 = sel_2;
    always #1 sel_3 = sel_3;

    always #1 sel_1 = sel_1;
    always #1 sel_2 = sel_2;
    always #1 sel_3 = sel_3;
```

```
VCD info: dumpfile ejercicio_2_b.vcd opened for output.
 0:     0       0       0       0000000000000000
 1:     0       0       1       0000000000000100
 2:     0       1       0       0000000000000010
 3:     0       1       1       0000000000000110
 4:     1       0       0       0000000000000001
 5:     1       0       1       0000000000000101
 6:     1       1       0       0000000000000011
 7:     1       1       1       0000000000000111
 8:     0       0       0       0000000000000000
```

c) Y para crear el MUX 16:1 use dos MUX 8:1 y un MUX 2:1

```
module ejercicio_2_c(in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8, in_9, in_10, in_11, in_12, in_13, in_14, in_15, in_16, sel_1, sel_2, sel_3, sel_4, out_1);

    input [15:0] in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8, in_9, in_10, in_11, in_12, in_13, in_14, in_15, in_16;
    input sel_1, sel_2, sel_3, sel_4;
    wire [15:0] wi_1, wi_2;

    output [15:0] out_1;

    ejercicio_2_b a1_0(in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8, sel_1, sel_2, sel_3, wi_1);
    ejercicio_2_b a1_1(in_9, in_10, in_11, in_12, in_13, in_14, in_15, in_16, sel_1, sel_2, sel_3, wi_2);

    ejercicio_1 a2_0(wi_1, wi_2, sel_4, out_1);

Termite
```

testbench y resultados:

```
`timescale 1ns/1ns
module ejercicio_2_b_tb;
	reg [15:0] in_1, in_2,in_3, in_4, in_5, in_6, in_7, in_8, in_9, in_10, in_11, in_12, in_13, in_14, in_15, in_16;
	reg sel_1, sel_2,sel_3, sel_4;

	wire [15:0] out_1;

	ejercicio_2_c e2_b(in_1, in_2, in_3, in_4, in_5, in_6, in_7, in_8, in_9, in_10, in_11, in_12, in_13, in_14, in_15, in_16, sel_1, sel_2, sel_3, sel_4, out_1);

	initial begin
		in_1 <= 0;
		in_2 <= 1;
		in_3 <= 2;
		in_4 <= 3;
		in_5 <= 4;
		in_6 <= 5;
		in_7 <= 6;
		in_8 <= 7;
		in_9 <= 8;
		in_10 <= 9;
		in_11 <= 10;
		in_12 <= 11;
		in_13 <= 12;
		in_14 <= 13;
		in_15 <= 14;
		in_16 <= 15;

		sel_1 = 0;
		sel_2 = 0;
		sel_3 = 0;
		sel_4 = 0;

		$display("time\tsel_1\tsel_2\tsel_3\tsel_4\tout_1"); $monitor("%2d:\t%b\t%b\t%b\t%b\t%b",$time, sel_1, sel_2, sel_3, sel_4, out_1);
		#15 $finish;
	end

	always #1 sel_1 = sel_1;
	always #1 sel_2 = sel_2;
	always #1 sel_3 = sel_3;
	always #1 sel_4 = ~sel_4;

	always #2 sel_1 = sel_1;
	always #2 sel_2 = sel_2;
	always #2 sel_3 = ~sel_3;
	always #2 sel_4 = sel_4;
```

```
VCD info: dumpfile ejercicio_2_c.vcd opened for output.
 0:     0       0       0       0       0000000000000000
 1:     0       0       0       1       0000000000001000
 2:     0       0       1       0       0000000000000100
 3:     0       0       1       1       0000000000001100
 4:     0       1       0       0       0000000000000010
 5:     0       1       0       1       0000000000001010
 6:     0       1       1       0       0000000000000110
 7:     0       1       1       1       0000000000001110
 8:     1       0       0       0       0000000000000001
 9:     1       0       0       1       0000000000001001
10:     1       0       1       0       0000000000000101
11:     1       0       1       1       0000000000001101
12:     1       1       0       0       0000000000000011
13:     1       1       0       1       0000000000001011
14:     1       1       1       0       0000000000000111
15:     1       1       1       1       0000000000001111
➜  ejercicio_2
```

3. Implementar los siguientes DEMUX (operación inversa al MUX):a) DE-MUX 1:2 de 16 bits, b) MUX 1 : 8 de 16-bits, c) MUX 1:16 de 16-bits.
TIP: Usar un módulo small (repetir) dentro del top.

a) Según lo que entendí como demux recibe un valor y este decide según el selector donde va a dejar salir ese valor.
Con esa idea hice un operador ternario que hace lo dicho antes.

```
module ejercicio_3_a(in_1,sel_1, out_1, out_2);
	input [15:0] in_1;
	input sel_1;
	output [15:0] out_1, out_2;

	assign out_1 = (in_1 && ~sel_1)?in_1:0;

	assign out_2 = (in_1 && sel_1)? in_1:0;

endmodule
```

testbench y resultados:

```
`timescale 1ns/1ns
module ejercicio_3_a_tb;
        reg [15:0] in_1;
        reg sel_1;

        wire [15:0] out_1, out_2;

       ejercicio_3_a e3_a(in_1, sel_1, out_1, out_2);
        initial begin
                in_1 <= 2;
                sel_1 = 0;

                $display("time\tsel_1\t\tout_1\tout_2"); $monitor("%2d:\t%b\t\t%b\t%b",$time, sel_1, out_1, out_2);
                #2 $finish;
        end

        always #1 sel_1 = !sel_1;

        initial begin
                $dumpfile("ejercicio_3_a.vcd");
                $dumpvars;
        end

endmodule
```

Termite

```
time    sel_1           out_1   out_2
VCD info: dumpfile ejercicio_3_a.vcd opened for output.
 0:     0       0000000000000010        0000000000000000
 1:     1       0000000000000000        0000000000000010
 2:     0       0000000000000010        0000000000000000
→  ejercicio_3
```

b) Y como en el MUX 8:1 aquí hice uso de 7 DEMUX 1:2

vim ejercicio_3_b.v

```
module ejercicio_3_b(in_1, sel_1, sel_2, sel_3, out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8);
        input [15:0] in_1;
        input sel_1, sel_2, sel_3;
        output [15:0] out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8;
        wire [15:0] wi_1, wi_2, wi_3, wi_4, wi_5, wi_6;

        ejercicio_3_a a1(in_1, sel_3, wi_1, wi_2);

        ejercicio_3_a a2_0(wi_1, sel_2, wi_3, wi_4);
        ejercicio_3_a a2_1(wi_2, sel_2, wi_5, wi_6);

        ejercicio_3_a a3_0(wi_3, sel_1, out_1, out_2);
        ejercicio_3_a a3_1(wi_4, sel_1, out_3, out_4);
        ejercicio_3_a a3_2(wi_5, sel_1, out_5, out_6);
        ejercicio_3_a a3_3(wi_6, sel_1, out_7, out_8);


endmodule
```

Termite

testbench y resultados:

```
`timescale 1ns/1ns
module ejercicio_3_b_tb;
        reg [15:0] in_1;
        reg sel_1, sel_2, sel_3;

        wire [15:0] out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8;

       ejercicio_3_b e3_a(in_1, sel_1, sel_2, sel_3, out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8);
        initial begin
                in_1 <= 1;
                sel_1 = 0;
                sel_2 = 0;
                sel_3 = 0;

                $display("time\t\tsel_1\tsel_2\tsel_3\tout_1\tout_2\tout_3\tout_4\tout_5\tout_6\tout_7\tout_8"); $monitor("%2d\t\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b",$time, sel_1,
sel_2, sel_3, out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8);
                #7 $finish;
        end

        always #1 sel_1 = sel_1;
        always #1 sel_2 = sel_2;
        always #1 sel_3 = ~sel_3;

        always #2 sel_1 = sel_1;
        always #2 sel_2 = ~sel_2;
        always #2 sel_3 = sel_3;

        always #3 sel_1 = sel_1;
        always #3 sel_2 = sel_2;
        always #3 sel_3 = sel_3;

        always #4 sel_1 = ~sel_1;
        always #4 sel_2 = sel_2;
        always #4 sel_3 = sel_3;

        always #5 sel_1 = sel_1;
        always #5 sel_2 = sel_2;
        always #5 sel_3 = sel_3;

        always #6 sel_1 = sel_1;
        always #6 sel_2 = sel_2;
        always #6 sel_3 = sel_3;

        always #7 sel_1 = sel_1;
        always #7 sel_2 = sel_2;
```

Termite

(lamento el formato pero al parecer los tabs desvian los valores muy a la derecha)

```
VCD info: dumpfile ejercicio_3_b.vcd opened for output.
 0:      0       0       0       0000000000000001        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        000000000
0000000 0000000000000000
 1:      0       0       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000001        0000000000000000        000000000
0000000 0000000000000000
 2:      0       1       0       0000000000000000        0000000000000000        0000000000000001        0000000000000000        0000000000000000        0000000000000000        000000000
0000000 0000000000000000
 3:      0       1       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        000000000
0000001 0000000000000000
 4:      1       0       0       0000000000000000        0000000000000001        0000000000000000        0000000000000000        0000000000000000        0000000000000000        000000000
0000000 0000000000000000
 5:      1       0       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000001        000000000
0000000 0000000000000000
 6:      1       1       0       0000000000000000        0000000000000000        0000000000000000        0000000000000001        0000000000000000        0000000000000000        000000000
0000000 0000000000000000
 7:      1       1       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        000000000
0000000 0000000000000001
➜  ejercicio_3
```

c) Lo mismo aquí use dos DEMUX 1:8 y un DEMUX 1:2

```
module ejercicio_3_c(in_1, sel_1, sel_2, sel_3, sel_4, out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8, out_9, out_10, out_11, out_12, out_13, out_14, out_15, out_16);
        input [15:0] in_1;
        input sel_1, sel_2, sel_3, sel_4;
        output [15:0] out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8, out_9, out_10, out_11, out_12, out_13, out_14, out_15, out_16;
        wire [15:0] wi_1, wi_2;

        ejercicio_3_a a1(in_1, sel_4, wi_1, wi_2);

        ejercicio_3_b a2_0(wi_1, sel_1, sel_2, sel_3, out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8);
        ejercicio_3_b a2_1(wi_2, sel_1, sel_2, sel_3, out_9, out_10, out_11, out_12, out_13, out_14, out_15, out_16);



endmodule
```

testbench y resultados

```
`timescale 1ns/1ns
module ejercicio_3_c_tb;
        reg [15:0] in_1;
        reg sel_1, sel_2,sel_3, sel_4;

        wire [15:0] out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8, out_9, out_10, out_11, out_12, out_13, out_14, out_15, out_16;

        ejercicio_3_c e3_b(in_1, sel_1, sel_2, sel_3, sel_4, out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8, out_9, out_10, out_11, out_12, out_13, out_14, out_15, out_16);

        initial begin
                in_1 <= 8;

Termite         sel_1 = 0;
                sel_2 = 0;
                sel_3 = 0;
                sel_4 = 0;

                $display("time\tsel_1\tsel_2\tsel_3\tsel_4\tout_1\tout_2\tout_3\tout_4\tout_5\tout_6\tout_7\tout_8\tout_9\tout_10\tout_11\tout_12\tout_13\tout_14\tout_15\tout_16"); $mon
itor("%2d:\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b\t%b",$time, sel_1, sel_2, sel_3,sel_4, out_1, out_2, out_3, out_4, out_5, out_6, out_7, out_8, out
_9, out_10, out_11, out_12, out_13, out_14, out_15, out_16);
                #15 $finish;
        end

                always #1 sel_1 = sel_1;
        always #1 sel_2 = sel_2;
        always #1 sel_3 = sel_3;
        always #1 sel_4 = ~sel_4;

        always #1 sel_1 = sel_1;
        always #1 sel_2 = sel_2;
        always #1 sel_3 = ~sel_3;
        always #1 sel_4 = sel_4;

        always #1 sel_1 = sel_1;
        always #1 sel_2 = sel_2;
        always #1 sel_3 = sel_3;
        always #1 sel_4 = sel_4;

        always #1 sel_1 = sel_1;
        always #1 sel_2 = ~sel_2;
        always #1 sel_3 = sel_3;
        always #1 sel_4 = sel_4;

        always #1 sel_1 = sel_1;
        always #1 sel_2 = sel_2;
```

```
0000000000000000 0000000000000000
 1:     0       0       0       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000001       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
 2:     0       0       1       0       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000001        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
 3:     0       0       1       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000001        0000000000000000
0000000000000000 0000000000000000
 4:     0       1       0       0       0000000000000000        0000000000000000        0000000000000000        0000000000000001        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
 5:     0       1       0       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000001        0000000000000000        0000000000000000
0000000000000000 0000000000000000
 6:     0       1       1       0       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000001 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
 7:     0       1       1       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000001 0000000000000000
 8:     1       0       0       0       0000000000000000        0000000000000001        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
 9:     1       0       0       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000001        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
10:     1       0       1       0       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000001
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
11:     1       0       1       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000001
0000000000000000 0000000000000000
12:     1       1       0       0       0000000000000000        0000000000000000        0000000000000000        0000000000000001        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
13:     1       1       0       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000001        0000000000000000        0000000000000000
0000000000000000 0000000000000000
14:     1       1       1       0       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000001       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000
15:     1       1       1       1       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000000       0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000        0000000000000000
0000000000000000 0000000000000001
```

4. Implementar un decoder 3-8 (m − 2 m). Asumir 1-bit input enable, m-bit input n[m − 1 : 0], y 2 m -bit output d[2 m − 1 : 0. Si enable=1, luego 11-bit d[n] = 1, los otros bits son 0; si enable=0, todos los bits del output d[2 m − 1 : 0 son 0. Tip: definir las funciones booleanas de Fig. 1 y analizar la Fig. 2.

Gracias a a Fig. 1 pude darme cuenta que el output prendía uno de sus bits dependiendo del los valores del input y el enable, la manera como lo implemente es como las anteriores.. con un operador ternario, pero esta vez dependiendo del input retorna el enable o 0 y en caso de que el enable sea 0 el output siempre va a ser 00000000.

```
module ejercicio_4 (in_1, en, out_1);
        input [2:0] in_1;
        input en;
        output [7:0] out_1;

        assign out_1[0] = in_1 == 0? en : 0;
        assign out_1[1] = in_1 == 1? en : 0;
        assign out_1[2] = in_1 == 2? en : 0;
        assign out_1[3] = in_1 == 3? en : 0;
        assign out_1[4] = in_1 == 4? en : 0;
        assign out_1[5] = in_1 == 5? en : 0;
        assign out_1[6] = in_1 == 6? en : 0;
        assign out_1[7] = in_1 == 7? en : 0;
endmodule
~
~
```

testbench y resultados

```
timescale 1ns/1ns
module ejercicio_4_tb;
       reg [2:0] in_1;
       reg en;
       wire [7:0] out_1, out_2;

      ejercicio_4 e4 (in_1, en, out_1);

     initial begin
              in_1 = 0;
              en = 1;

              $display("time\ten\tin_1\tout_1"); $monitor("%2d:\t%b\t%b\t%b",$time,
en, in_1, out_1);
              #9 $finish;
       end

       always #1 in_1 = in_1 +1;
       always #2 in_1 = in_1;
       always #3 in_1 = in_1;
       always #4 in_1 = in_1;
       always #5 in_1 = in_1;
       always #6 in_1 = in_1;
       always #7 in_1 = in_1;

       always #8 en = ~en;


       initial begin
              $dumpfile("ejercicio_4.vcd");
              $dumpvars;
       end

endmodule
```

Termite

```
time    en      in_1    out_1
VCD info: dumpfile ejercicio_4.vcd opened for output.
 0:     1       000     00000001
 1:     1       001     00000010
 2:     1       010     00000100
 3:     1       011     00001000
 4:     1       100     00010000
 5:     1       101     00100000
 6:     1       110     01000000
 7:     1       111     10000000
 8:     0       000     00000000
 9:     0       001     00000000
→  ejercicio_4
```

5. Implementar un encoder 8-3 con prioridad (operación inversa del decoder). Tiene un máximo de 2 m inputs y m outputs. Cada input tiene una prioridad asignada. En nuestro priority encoder, ver Fig. 3), el input d[7] tiene la más alta prioridad y d[0] la más baja prioridad. El valor x es ”don’t care”, pudiendo ser 1 o 0, sin efecto sobre el output. La tabla de verdad lista todas las combinaciones de inputs d[7:0] y enable. Tip: definir las funciones booleanas y analizar el output esperado (Fig. 4). Usar casex.

En mi enconder se verifica primero si el enable no es 0, en caso este sea 0 el valor del output siempre va a ser 0, en caso contrario dependiendo del bit con más prioridad del input se genera un output.

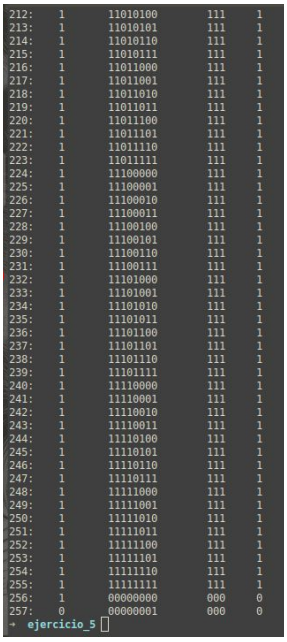
Analizando la tabla de verdad de la Fig. 3 me di cuenta que el valor de g depende del en y del input, con esa info hice un operador ternario que recibe a los anteriores valores para generar un output de g.

```
module ejercicio_5(in_1, en, out_1, g);
        input wire[7:0] in_1;
        input en;
        output reg[2:0] out_1;
        output g;

        always @*
        begin
                if(!en)
                        out_1 = 3'o0;
                else
                        casex(in_1)
                                8'b00000001 : out_1 = 3'o0;
                                8'b0000001x : out_1 = 3'o1;
                                8'b000001xx : out_1 = 3'o2;
                                8'b00001xxx : out_1 = 3'o3;
                                8'b0001xxxx : out_1 = 3'o4;
                                8'b001xxxxx : out_1 = 3'o5;
                                8'b01xxxxxx : out_1 = 3'o6;
                                8'b1xxxxxxx : out_1 = 3'o7;

                                default: begin
                                        out_1 = 3'o0;
                                end
                        endcase
        end
        assign g = (en && in_1)? 1:0;

endmodule
```

testbench y resultados

A diferencia de la Fig. 4 lo que yo elegí como output fue probar todos los 257 casos posibles, como se puede ver en la imagen debajo de esta, debido a que es una cantidad enorme de outputs no podre ponerlos todos aquí, pero usted es libre de probar.

```
`timescale 1ns/1ns
module ejercicio_5_tb;
        reg [7:0] in_1;
        reg en;
        wire [2:0] out_1;
        wire g;

      ejercicio_5 e5 (in_1, en, out_1, g);

     initial begin
                in_1 = 0;
                en = 1;
                $display("time\ten\tin_1\tout_1"); $monitor("%2d\t%b\t%b\t%b\t%b",$time, en, in_1, out_1, g);
                #257 $finish;
        end

        always #1 in_1 = in_1 +1;


        always #257 en = ~en;


        initial begin
                $dumpfile("ejercicio_5.vcd");
                $dumpvars;
        end

endmodule
```



6. Implementar un barrel shifter de 32-bit input hacia izquierda o derecha de 0 a 31 bits, basado en el right input y 5-bit sa input (cantidad). Usar un input arithmetic que indica si se debe realizar un logical shift o un arithmetic shift cuando right es 1. Un logical shift right inserta ceros en las posiciones de bits vacías, y un arithmetic shift right replica el sign bit en las posiciones de bits vacías. Ejemplo:

• Data d: 11111111 00000000 00000000 11111111

• Shift d hacia la izquierda de 8 bits: 00000000 00000000 11111111 00000000

(no me alcanzó el tiempo)