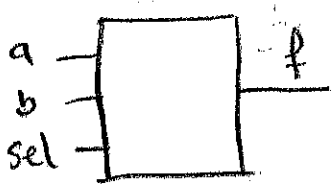
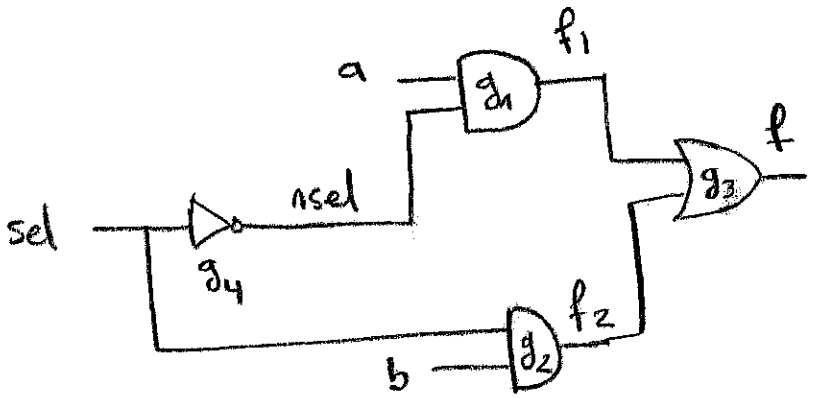
1) a. Obtener el circuito a partir de la descripción en Verilog:

```
module mux (f, a, b, sel);
    input  a, b, sel;
    output f;
wire nsel, f1, f2;
and  g1 (f1, a, nsel);
and  g2 (f2, b, sel);
or   g3 (f, f1, f2);
not  g4 (nsel, sel);
endmodule
```





b- Obtener su descripción funcional

```
module mux (output f, input a, b, sel);
    assign f = (a & nsel) | (b & sel);
endmodule
```

c- obtener su descripción procedimental ← con if / con case / otra

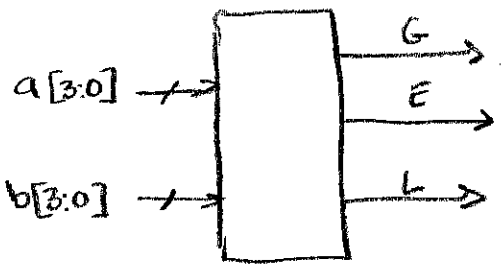
```
module mux (output reg f, input a, b, sel);

always @(a, b, sel)
```

`if (sel == 0)`
`  f = a;`
`else`
`  f = b;`
	`case (sel)`
	`  0: f = a;`
	`  1: f = b;`
	`endcase`
		`f = sel ? b : a;`

↑ OPCIÓN 1	↑ OPCIÓN 2	↑ OPCIÓN 3

```
endmodule
```

2. Describa en Verilog un comparador de 4 bits con salidas G, E, L. ¿Cuál es el tipo de descripción que ha escogido y porqué?



→ estructural no porque tendría que sacar antes el circuito

→ funcional, posible pero tb hay que deducir las funciones lógicas.

→ la más adecuada es la estructural, pues basta saber como se comporta un comparador y será fácilmente adaptable para ≠ anchos de bus.

```
module comp ( input [3:0] a, b, output reg g,e,l);
    always @(G,E,L) begin
        {G,E,L} = 0;
        if (a>b) G=1;
        else if (a<b) L=1;
             else E=1;
    end
endmodule
```

3. Analizar la siguiente descripción para un comparador de 4 bits con salidas GEL y describir los errores.

```
module comp (input [3:0] a, b, output g, e, l);
    g = e = l = 0;
    if (a > b)
        g = 1;
    else if (a < b)
        l = 1;
        else e = 1;
end module
```

→ hay que declarar tipo reg las variables del procedimiento: g, e, l
se puede hacer de dos formas:

1) al declarar las entradas y salidas haciendo "output reg g, e, l"

2) dentro del módulo con "reg gint, eint, lint" y luego "assign g = gint; assign e = eint; assign l = lint;"

→ no se puede poner g = e = l = 0; hay que poner g = 0; e = 0; l = 0; o bien {g, e, l} = 3'b000; ó {g, e, l} = 0

→ falta el always @ (G, E, L) begin ..... end

4. A continuación se muestra la descripción Verilog de un sumador de 4 bits.

a) ¿qué tipo de descripción es?
b) ¿se trata de un circuito modular?
c) modifique la descripción para que sea fácilmente configurable en tamaño

```
module sumador4 ( input [3:0] a, b, input cin,
                  output [3:0] s, output cout );

    reg [4:0] res;
    always @ (a, b, cin)
        res = a + b + cin;
    assign cout = res [4];
    assign s = res [3:0];

endmodule
```

a) es una descripción procedimental.

b) no sabemos qué tipo de circuito es, no hay estructura

c) definiremos una constante para el nº de bits, la llamaré N

```
module sumador ( input [N-1 : 0] a, b, input cin,
                 output [N-1 : 0] s, output cout );

parameter N = 4;
reg [N:0] res;
always @ (a, b, cin)
    res = a + b + cin;
assign cout = res [N];
assign s = res [N-1:0]
end module
```

5. Obtenga la descripcion Verilog de un semisumador a partir del esquema de la figura 5a. A continuación obtenga la descripción del sumador completo instanciando 2 semisumadores (Fig. 5b).

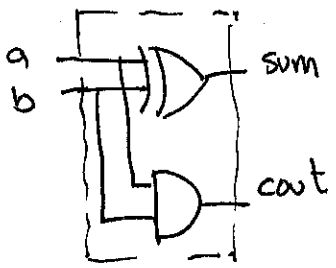
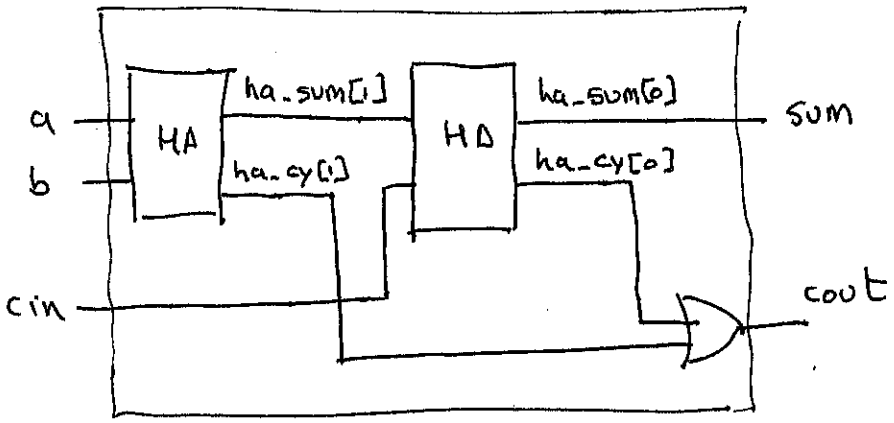


Fig. 5a.



Semisumador

```
module semisumador (input a, b, output sum, cout)
```

`assign sum = a ^ b;`	`xor xor1 (sum, a, b);`
`assign cout = a & b;`	`and and1 (cout, a, b);`

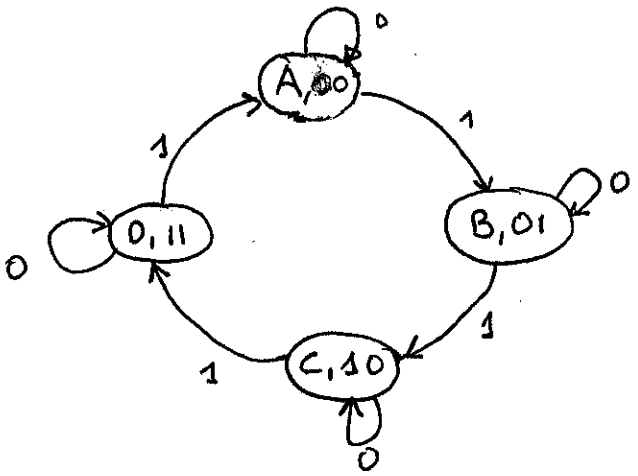
OPCION 1 funcional — OPCION 2 estructural

```
endmodule
```

Sumador completo

```
module sumadorcompleto (input a, b, cin, output sum, cout)

wire [1:0] ha_sum, ha_cy;
semisumador ha1 (.a(a), .b(b), .sum(ha_sum[1]), .cout(ha_cy[1]));
semisumador ha2 (.a(ha_sum[1]), .b(cin), .sum(sum), .cout(ha_cy[0]));
or or1 (cout, ha_cy[0], ha_cy[1]);
endmodule
```

6) Obtener la descripción Verilog del diagrama de estados de un contador módulo 4 con inhibición:



	Q[1]	Q[0]
A	0	0
B	0	1
C	1	0
D	1	1

código

reset

```
module contador (input x, clk, output [1:0] z)

parameter A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;
reg [1:0] current_state, next_state;

always @(posedge clk, posedge reset)
    if (reset)
        current_state <= A;
    else
        current_state <= next_state;

always @(current_state, x)

    case (current_state)
        A: if (x==0)
               next_state = A;
           else
               next_state = B;
        B: if (x==0)
               next_state = B;
           else
               next_state = C;
```

```
C: if (x==0)
     next_state = C;
   else
     next_state = D;
D: if (x==0)
     next_state = D;
   else
     next_state = A;
endcase
assign z = current_state;
```

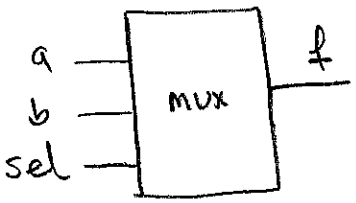
alternativa al case (current_state):

```
case (x)
  0: next_state = current_state;
  1: next_state = current_state + 1;
```

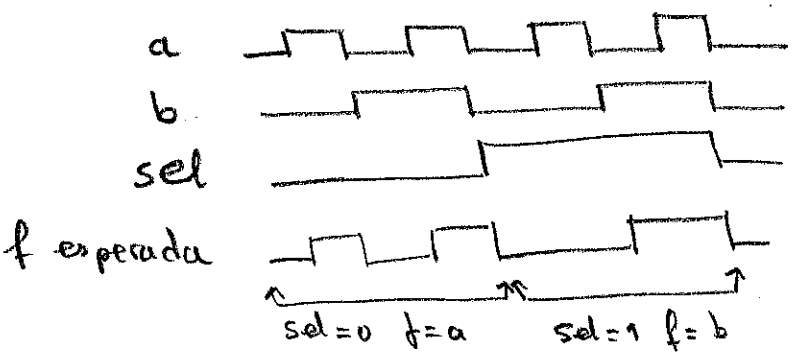
7) Obtener la descripción Verilog del contador del ejemplo anterior. En esta ocasión se pide hacerlo con una descripción procedimental pero sin relacionarla con el diagrama de estados.

```
module contador (input x, ck, reset, output [1:0] z)

reg [1:0] q;
always @ (posedge ck, posedge reset)
    if (reset)
        q <= 0;
    else if (x)
        q <= q+1;
assign z = q;
endmodule
```

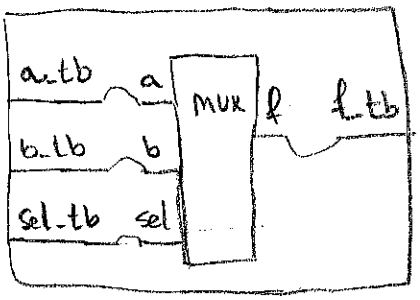
8) Obtener un test bench para el circuito mux del ejemplo 1.



Para probarlo generaremos todas las posibles entradas:



Definimos el módulo test_bench ⇒ mux_tb



mux_tb

```
module mux_tb;
reg a_tb, b_tb, sel_tb;
wire f_tb;
mux mimux (.a(a_tb), .b(b_tb), .sel(sel_tb), .f(f_tb));

initial begin
{a, b, sel} = 0;
# 800;
$ finish;
end
```

Sigue

```
always begin
#100;
{sel_tb, b_tb, a_tb} = {sel_tb, b_tb, a_tb} + 1;
end
endmodule
```

También podíamos haber incluido:

```
initial
$monitor ($time, a, b, sel, f);
```

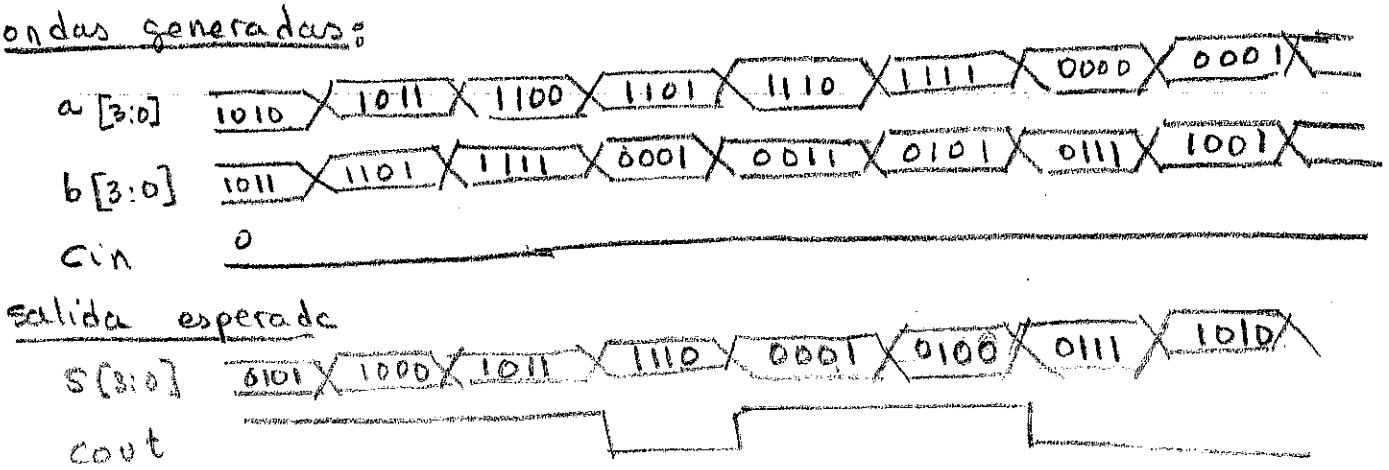
O con formato si lo queremos:

```
initial
$monitor ("%0d ns  %b %b %b  %b", $time, a, b, sel, f);
$monitor ($time, "ns  %b %b %b  %b", a, b, sel, f)
```

9. Realizar un fichero de testbench para el sumador de 4 bits del problema 4.

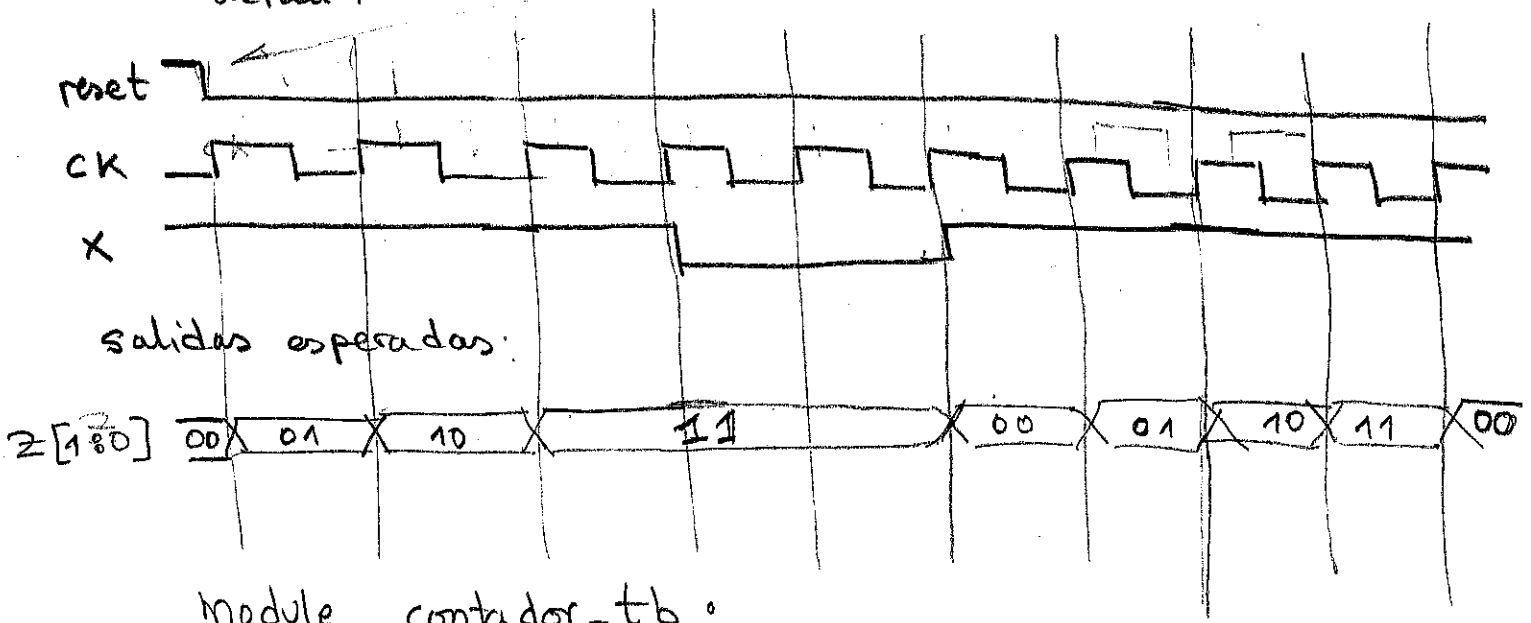
```
module sumador4_tb;

reg [3:0] a, b;
reg cin;
wire [3:0] s;
wire cout;

sumador4 misumador (.a(a), .b(b), .cin(cin), .s(s), .cout(cout));

initial begin
    {a,b,cin} = 9'b1010 1011 0;
    #800;
    $finish;
end

always begin
    #100;
    a = a+1;
    b = b+2;
end
endmodule
```

ondas generadas:



10. Escribir un fichero de testbench para el contador módulo 4 de los problemas 6 y 7.

Generaremos formas de onda que nos permitan comprobar que el contador realiza correctamente la cuenta y que cuando x=0 se inhibe. Haremos un reset inicial. Las ondas generadas serán:




```
module contador_tb;

reg x, ck, reset;
wire [1:0] z;
contador micontador (.x(x), .ck(ck), .reset(reset), .z(z));

initial begin
ck=0; x=1; reset=1;
#50;
reset=0;
#300;
x=0;
#200;
x=1;
#800;
$finish;
end
```

SIGUE ⇒

```
always begin
#50;
ck = ~ck;
end

endmodule
```