OPERATORI

~ not
& and ~& nand
^ xor
| or ~| nor

SINTASSI BASE

inizializz. modulo — booleano

```
module nomemodulo (input logic a,b,c
                   output logic y);

  assign y = ~a &b &c |
             a & b & c;

endmodule
```

bus multibit

```
input logic [3:0] a
```

come lo slicing, ma l'inizio è compreso

a[3:0] = a[3], a[2], a[1], a[0]

- ordinati arbitrariamente

ma ord. comuni: little-endian [0:N-1]
big-endian [N-1:0]

assign - operatore di ASSEGNAMENTO CONTINUO
- ricalcola l'uscita ogni volta che gli ingressi cambiano

? : - operatore di ASSEGNAMENTO CONDIZIONALE → Seleziona, sulla base di una prima espressione, la seconda o la terza.

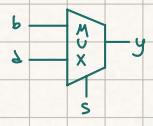
condizione = { 1 → prima esp. ; 0 → seconda esp

MUX DA SOTTO A SOPRA

```
module mux2 (input logic [0:3] a, b,
             input logic       s,
             output logic      y);

  assign y = s ? a : b
             s=1   s=0

endmodule
```



modulo per sommare (o operatore) su tutti i bit

```
module and8 (input logic [0:7]a,
             output logic y);

  assign y = ~a;

endmodule
```

più facile di fare `assign y = a[7] & a[6] & ...`

always @(sensitivity list) istruzione

l'istruzione viene eseguita quando avviene ciò che è scritto nella sens. list

- con gli always si usano
  - <= NON BLOCCANTE - sequenziale
  - = BLOCCANTE - combinatorio

always_ff — @(posedge clk) sensibile al fronte di salita
always_latch
always_comb

FLIP-FLOP D sens. fronte salita

```
module flopd (input logic clk,
              input logic [3:0] d,
              output logic [3:0] q);

  always_ff @(posedge clk)
    q <= d;

endmodule
```

Variabili interne

se bisogna fare molte operazioni, si possono definire variabili interne che non sono né entrate né uscite

```
module grande (input logic a, b, c,
               output logic y);

  assign p = a & b ;   // definite internamente
  assign j = b ^ c;
  assign y = p | j;

endmodule
```

bus concatenati

```
assign y = {c[2:1], {3{d[0]}}, c[0], 3'b101};
```

{3{d[0]}} → 3 copie di d[0]

NUMERI

N'Bvalore
- N = dimensione in bit
- B = lettera che indica base

- 'b - binario
- 'd decimale
- 'o - ottale
- 'h esadecimale

```
module neg (input logic [3:0] a
            output logic [3:0] b);

  always_comb
    y = ~a;

endmodule
```

- **if, else** → sempre dentro un ALWAYS

```
if (istruz) conseguenza;
else if (istruz) conseguenza;
else conseguenza;
```

case (dato) :

Sceglie un'operazione in base al valore di dato
Si può usare default per definire le uscite nei casi non specificati
(es. se un valore compare 3 volte, invece di specificare 3 casi scelgo quello come default e specifico tutti gli altri)
- sempre dentro un always

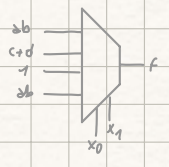
casez (dato)

supporta anche ? come un d

```
casez (a)
  4'b1???: y = b
  4'b0?1 : y = c
```

```
module mux 4-a-1 (input logic a,b,c,
                  input logic [0:1] x
                  output logic    f);

  always_comb
    case (x)
      2'b01 : f = c|d
      2'b10 : f = 1'b1
      default : f = a & b
    endcase
endmodule
```



REGISTRI

- RESET

RESET SINCRONO

```
module flopr(input logic clk,
             input logic reset,
             input logic [3:0] d,
             output logic [3:0]q);

always_ff @(posedge clk)
  if (reset) q<=4'b0;   4 bit di 0 (binario)
  else q<= d;

endmodule
```

RESET ASINCRONO

```
module flopr(input logic clk,
             input logic reset,
             input logic [3:0] d,
             output logic  q);

always_ff @(posedge clk, posedge reset)
  if (reset) q<= 4'b0;
  else q<= d;

endmodule
```

ABILITAZIONE

```
module flopenr(input logic clk,
               input logic reset,
               input logic en,
               input logic [3:0]d,
               output logic [3:0]q);

always_ff @(posedge clk, posedge reset)
  if(reset) q<=4'b0;
  else if(en) q<=d;

endmodule
```

MULTIPLI

```
module sinc(input logic clk,
            input logic d,
            output logic q);

logic n1;

always_ff @(posedge clk)
  begin                  contemporaneamente
    n1 <= d;             d va in n1 e n1 in q
    q <= n1;
  end

endmodule
```

LATCH D

```
module latch(input logic clk,
             input logic d,
             output logic q);

always_latch                valutato ogni volta
  if (clk) q<=d;            che clk o d
                            cambiano
endmodule
```

DOMANDE ESAMI:

① FF T reset asincrono ✓
(2021 02 16 A)

② shift register 8 bit con reset sincrono
(2021 02 16 B)

③ buffer tristate - NON L'ABBIAMO FATTO
(2021 07 12)

④ MUX 8-A-1 ✓
(2022 01 11 A e B)

⑤ contatore mod_{13} ✓
(2022 02 01 A mod_{11} B)

MASSINI

① 2 FF delay (2023 01 18 A e B) ✓

② FF reset asincrono e segnale enable ✓

① FLIP-FLOP T CON RESET ASINCRONO

```
module fftr (input logic clk,
             input logic res,
             input logic t,
             output logic q );

  always_ff @ (posedge clk, posedge res)
     if(res) q <=1'b0;
     else if (t) q<=~q;
     else q <=q;

endmodule
```

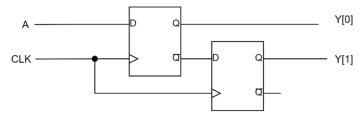
alternativa: `else q <= q ^ t`

XOR, perché

t	q	Q
0	0	0
0	1	1
1	0	1
1	1	0

② MASSINI 2021 18 GEN B

Esercizio 3 (4 punti) Descrivere in SystemVerilog il seguente circuito:




```
module ex3 (input logic clk,
            input logic a,
            output logic [0:1]y);

  always_ff @(posedge clk)
   begin
     y[0] <= a;
     y[1] <= ~y[0]
   end

endmodule
```

③ FF reset asincrono e segnale enable

```
module flop (input logic clk,
             input logic res,
             input logic en,
             input logic d,
             output logic q);

   always_ff @ (posedge clk, posedge res)
      if (res) q <=1'b0;
      else if(en) q<=d;

endmodule
```

④ Shift register 8 bit con reset asincrono

```
module shift (input logic clk,
              input logic res,
              input logic x,
              output logic q);

   logic [7:0] sh_reg;
   always_ff @ (posedge clk, posedge res)
      begin
        if (res) sh_reg <= 8'b0
        else sh_reg <= {sh_reg[6:0], d};
        end
  assign q= sh_reg[7]

endmodule
```

concatenazione: 7 bit del registro + 1 dell'input

l'uscita è solo l'ultimo bit

⑤ contatore mod_{13}

Conta fino a 12 ⟶ 1100

```
module count13 (input logic clk,
                output logic [3:0] y);

  assign y = 4'b0
  always_ff @(posedge clk)
     begin
       if (y == 4'b1100)  y <= 4'b0
       else  y <= y+1
     end

endmodule
```