Modellazione a livello di porta logica (GATE-LEVEL)

▼ Creatore originale: @Gianbattista Busonera

La modellazione a livello di porta logica rappresenta il circuito come un'interconnessione esplicita di porte elementari (AND, OR, XOR, NAND, flip-flop, ecc.). Ogni istanza di porta è contemporanea, e si ha quindi del parallelismo.

Usi tipici includono:

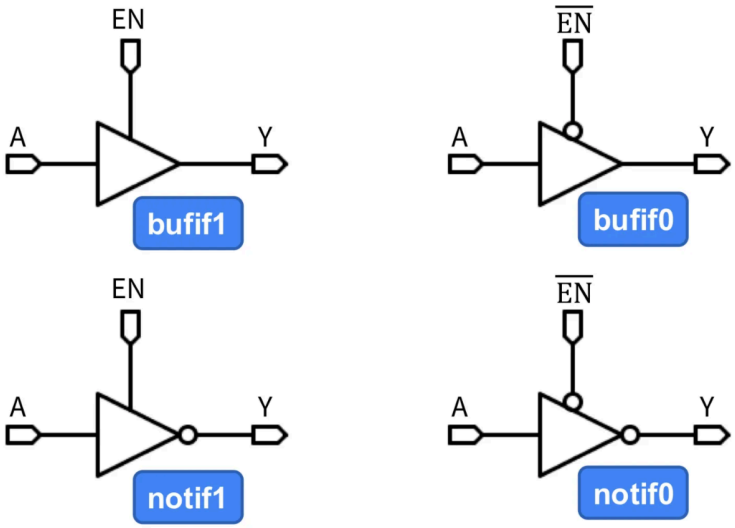
- checks post-sintesi;
- netlist generata dai tool;
- esercizi per piccole logiche.

Simula esattamente la rete fisica, e può includere ritardi di propagazione.

Porte logiche elementari in Verilog

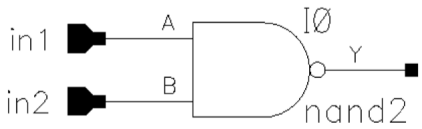
Le porte logiche di base sono:

- `and` ;
- `or` ;
- `not` ;
- `buf` ;
- `nand` ;
- `nor` ;
- `xor` ;
- `xnor` ;
- porte logiche tri-state.
  - `bufif1` , `bufif0` ;
  - `notif1` , `notif0` .

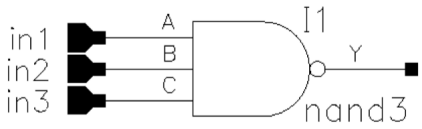


Rappresentazione di porte logiche tri-state

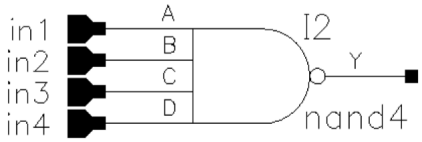
I pin delle porte logiche elementari sopra citate sono espandibili, come mostrato nella figura.




```
nand (y, in1, in2);
```




```
nand (y, in1, in2, in3);
```




```
nand (y, in1, in2, in3, in4);
```

1 uscita

N ingressi

Espansione dei pin delle porte logiche

Valori di uscita delle porte logiche elementari

AND	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

OR	0	1	X	Z
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

XOR	0	1	X	Z
0	0	1	X	X
1	1	0	X	X
X	X	X	X	X
Z	X	X	X	X

NAND	0	1	X	Z
0	1	1	1	1
1	1	0	X	X
X	1	X	X	X
Z	1	X	X	X

NOR	0	1	X	Z
0	1	0	X	X
1	0	0	0	0
X	X	0	X	X
Z	X	0	X	X

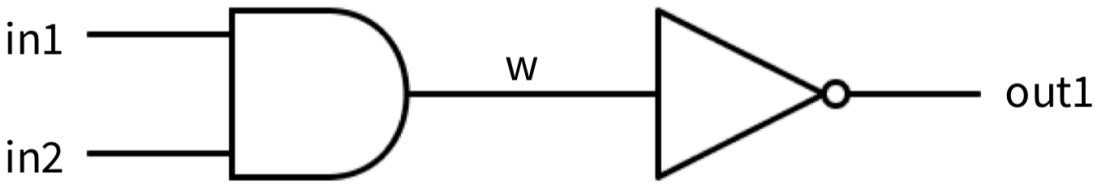
XNOR	0	1	X	Z
0	1	0	X	X
1	0	1	X	X
X	X	X	X	X
Z	X	X	X	X

BUF	
0	0
1	1
X	X
Z	X

NOT	
0	1
1	0
X	X
Z	X

Tabelle che rappresentano i valori di uscita delle porte logiche elementari

▼ Esempio - Modellazione a livello di porta logica



Circuito di riferimento da simulare: una porta NAND

```
module my_nand (out1, in1, in2);
   // inizio interfaccia
   output out1;
   input in1, in2; // input inseriti nella stessa definizione,
                              // in quanto sono entrambi da 1 bit
   // fine interfaccia

   // visto che devo collegare la porta AND e la porta NOT, mi serve un wire "w
   wire w;
   and(w, in1, in2); // l'uscita della porta AND è il wire "w"
   not(out1, w);     // l'uscita della porta not è out1, l'ingresso è w
endmodule
```

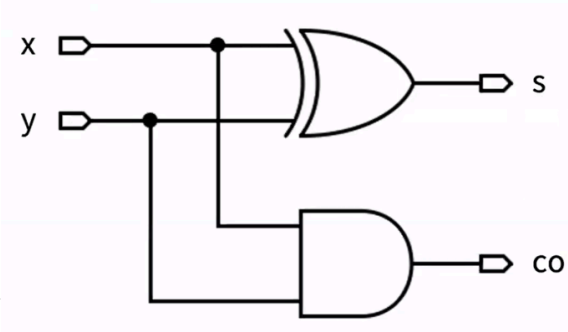
▼ Esempio - Semisommatore con porte logiche

Si realizzi l'Half Adder in figura, tale che

`s = x+y` e `co` sia il Carry Out.

In tal senso, occorre ricordare che vogliamo fare in modo che `s` sia:

0+0 = 0	1+0 = 1	0+1 = 1



Visualizzazione del modulo Half Adder

Questo si può ottenere facendo lo XOR tra x e y, mentre il Carry Out è ottenibile dall'AND di x e y.

```
module hadd(
    s, co, // uscite
    x, y   // ingressi
);
    input x, y;
    output s, co;
    // FINE INTERFACCIA

  xor(s, x, y);     // somma
  and(carry, x, y); // riporto
endmodule
```

Ritardi di porte logiche

I ritardi in Verilog sono preceduti da un cancelletto (`#`). Tipicamente, un ritardo associato a una porta logica è dichiarato come segue:

$$\#(\text{ritardo di propagazione low} \rightarrow \text{high, ritardo di propagazione h}\rightarrow\text{l}) = \#(t_{\text{p}}^{\text{L}\rightarrow\text{H}}, t_{\text{p}}^{\text{H}\rightarrow\text{L}})$$

Si può essere ancora più specifici, fornendo il range dei ritardi minimi, tipici e massimi. Sui valori di $t_{\text{p}}^{\text{L}\rightarrow\text{H}}$ o di $t_{\text{p}}^{\text{H}\rightarrow\text{L}}$ possiamo definire i ritardi `minimi:tipici:massimi`.

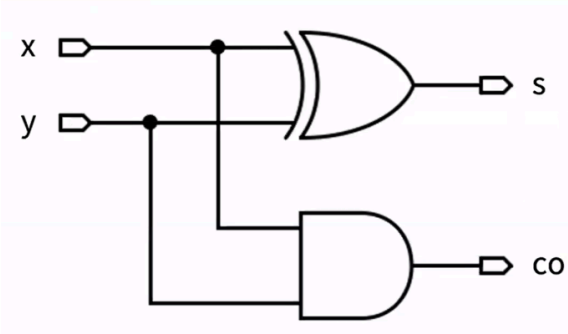
È inoltre indispensabile specificare l'unità di misura di tali ritardi tramite la direttiva:

```
`timescale <unità_di_tempo>/<precisione_temporale>
// solitamente si specifica ad inizio file
```

⚠️ Si noti che tali ritardi sono utili solo nella simulazione comportamentale del circuito. Non hanno, infatti, alcun significato nella sintesi e nella realizzazione effettiva del circuito.

▼ Esempio - Semisommatore con porte logiche, con ritardi

Si realizzi un Half Adder del tipo in figura, tale che `s = x+y` e `co` sia il Carry Out.



Si sa che:

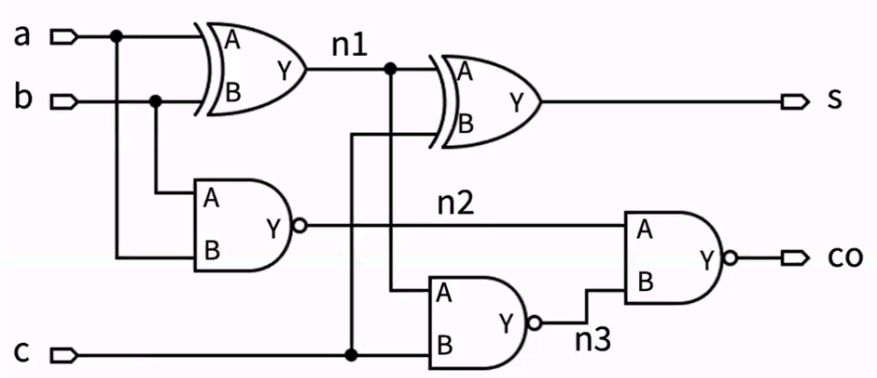
- $t^{\mathrm{L}\to\mathrm{H}}_{\mathrm{p_{XOR}},\mathrm{min}} = 2\ \mathrm{ns}$;
- $t^{\mathrm{L}\to\mathrm{H}}_{\mathrm{p_{XOR}},\mathrm{max}} = 4\ \mathrm{ns}$;
- $t^{\mathrm{H}\to\mathrm{L}}_{\mathrm{p_{XOR}}} = 5\ \mathrm{ns}$;
- $t^{\mathrm{L}\to\mathrm{H}}_{\mathrm{p_{AND}}} = t^{\mathrm{H}\to\mathrm{L}}_{\mathrm{p_{AND}}} = 3.6\ \mathrm{ns}$.

```
// dico che l'unita di tempo è 1 ns e che la precisione temporale
// è pari a 100 ps = 0.1 ns
`timescale 1ns/100ps
...
module hadd(
    s, co, // uscite
    x, y   // ingressi
);
    input x,y;
    output s, co;
    // FINE INTERFACCIA

  xor # // definizione del ritardo
        (2:3:4,     // definizione del tempo minimo:tipico:medio del tp l→h XOR
          5)        // definizione del tempo tipico del tp h→l XOR
         (s, x, y);  // somma
    // per intero, andrebbe scritto:
    //     xor #(2:3:4, 5) (s,x,y);

  and #(3.567) // definizione del ritardo della porta AND
  // NOTA BENE: visto che la precisione è di 100 ps = 0.1 ns,
  // 3.567 ns viene arrotondato a 3.6 ns
           (co, x, y);
```

```
    // and #(3.6) (co,x,y);
endmodule
```

▼ Esempio - Utilizzo di sottomoduli

L’obiettivo è realizzare un Full Adder a 3 ingressi tramite porte logiche elementari.

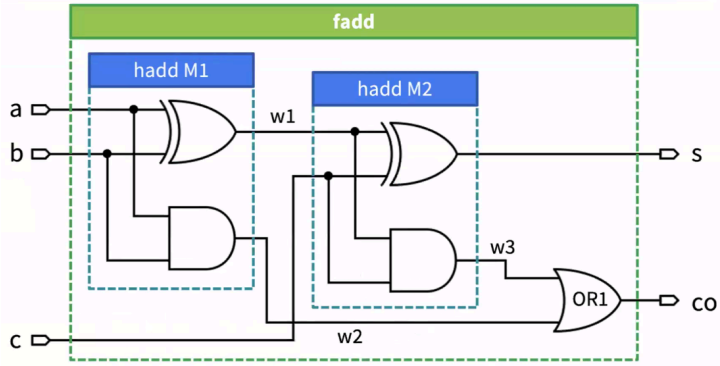


Schema circuitale

```
module fadd(s, co, a,b,c); // come di consuetudine: prima gli output, poi gli inp
    output co, s;
    input a,b,c;
    // FINE INTERFACCIA

    wire n1, n2, n3;
    // FINE NET

    // l'ordine dei comandi sotto è stato volutamente "sparso" per rimarcare ch
    // con questa metodologia di programmazione, l'ordine delle istruzioni
    // NON è importante
    nand(n3,n1,c);
    xor(n1,a,b);
    nand(n2,a,b);
    xor(s,n1,c);
    nand(co,n2,n3);
endmodule
```

E’ anche vero, però, che possiamo realizzare un Full Adder come interconnessione di Half Adder!

Quindi, avendo prima creato il modulo hadd, possiamo istanziarlo due volte per creare lo schema in figura.

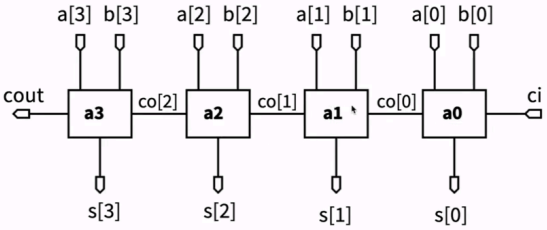


Schema Full Adder definito come interconnessione di Half Adder

```
module fadd(co,s,a,b,c);
   output co,s;
   input a,b,c;

   wire w1, w2, w3;
   // si istanzia il modulo hadd dandogli un nome!
   // hadd (nome modulo) M1 (nome istanza) (componenti)
   hadd M1 (w1, w2, a,b):  // la somma va su w1, il carry su w2
   hadd M2 (s, w3, w1, c); // la somma va su s, il carry su w3
   // N.B. serve avere un modulo hadd in un altro file!
   or OR1 (co, w3, w2);
endmodule
```

▼ Esempio - Sommatore a quattro bit con sottomoduli

- Implementazione gerarchica del modulo add4
  - Quattro moduli fadd
  - Ogni fadd consiste di
    - Due moduli hadd
    - Una porta logica elementare OR




```
1  module add4 (s, cout, ci, a, b);
2
3      input  [3:0] a, b;
4      input        ci;
5      output [3:0] s;
6      output       cout;
7
8      wire   [2:0] co;
9
10     fadd a0 (co[0], s[0], a[0], b[0], ci);
11     fadd a1 (co[1], s[1], a[1], b[1], co[0]);
12     fadd a2 (co[2], s[2], a[2], b[2], co[1]);
13     fadd a3 (cout, s[3], a[3], b[3], co[2]);
14
15 endmodule
```

Definizione del sommatore a quattro bit con sottomoduli