Logica e Reti Logiche
(Episodio 16: Cenni agli *Hardware Description Languages*)

Francesco Pasquale

18 dicembre 2025

In questo corso abbiamo parlato di circuiti disegnandone lo schema, oppure descrivendone le specifiche funzionali tramite formule o tabelle di verità. Una formula (o una tabella di verità) però non *descrive* un circuito, nel senso che non ci dice come deve essere costruito, ma ne specifica soltanto la funzione; infatti abbiamo visto che data una formula ci sono tanti (infiniti, a voler essere precisi) circuiti equivalenti. L'unico modo che abbiamo visto finora per descrivere esattamente come deve essere costruito un circuito è stato disegnarne lo schema. Potete immaginare che questo può andar bene per circuiti con poche porte logiche, ma per circuiti più grandi abbiamo bisogno di qualche strumento più efficace.

I *linguaggi per la descrizione dell'hardware* (*HDL* - Hardware Description Languages) servono a *descrivere* un circuito come puotrebbe farlo un disegno, usando però semplicemente del testo. Avere il circuito descritto come testo in un formato standard, che quindi può essere facilmente letto da un programma, è utile da un lato per avvalersi di programmi in grado di *simulare* il comportamento del circuito e consentirci quindi di verificare se il circuito svolge esattamente le funzioni che ci si aspetta prima di costruirlo fisicamente, da un altro lato è utile anche per avvalersi di programmi che dalla descrizione di un circuito producono automaticamente lo schema (*sintesi* del circuito).

1 HDL - Verilog

Consideriamo per esempio il semplice circuito combinatorio in Figura 1

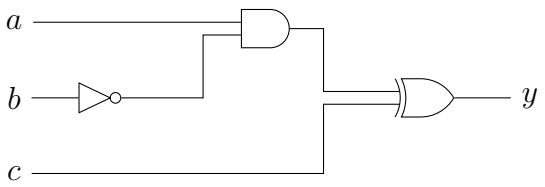


Figura 1: Un circuito combinatorio

Esercizio 1. Avete davanti il disegno del circuito in Figura 1 e dovete descriverlo a parole (senza poter mostrare il disegno) a un vostro collega facendo in modo che disegni esattamente lo stesso circuito. Come glie lo descrivereste?

I linguaggi per la descrizione dell'hardware sono dei veri e propri linguaggi di programmazione che possono anche essere usati per scrivere ed eseguire programmi di ogni tipo. La loro funzione principale tuttavia è quella appunto di "descrivere" l'hardware. I linguaggi di questo tipo più diffusi sono essenzialmente due, *VHDL* e *Verilog*, che differiscono per la sintassi utilizzata. Per gli esempi in questo episodio userò Verilog, di cui potete trovare un compilatore libero qui: `https://github.com/steveicarus/iverilog`.

In un HDL, ogni blocchetto hardware è chiamato *modulo*. Alcuni moduli, come le porte AND, OR, NOT e altre, sono incorporate nella sintassi del linguaggio. A partire da quei moduli possiamo descrivere circuiti più complessi. Per esempio, in Verilog possiamo descrivere il circuito in Figura 1 con il codice in Listing 1.

```
1  module example(a,b,c,y);
2      input a,b,c;
3      output y;
4
5      wire w1,w2;
6
7      not(w1,b);
8      and(w2,a,w1);
9      xor(y,c,w2);
10 endmodule
```

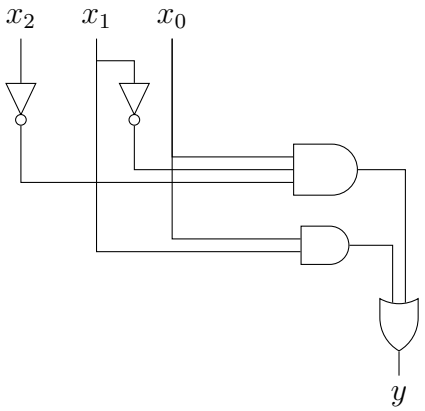
Listing 1: Il circuito in Figura 1 descritto in Verilog

Osservate che alla Linea 1 abbiamo definito il nome del nostro modulo, "example", inserendo come parametri le variabili che indicano gli input e l'output del circuito, e alle Linee 2 e 3 abbiamo specificato quali dei parametri indicano gli input e quale l'output.

Nel circuito in Figura 1 l'output della porta NOT è uno degli input della porta AND e l'output della porta AND è uno degli input della porta XOR. Per specificare ciò senza ambiguità alla Linea 5 definiamo due nuove variabili, $w1$ e $w2$, di tipo WIRE, che serviranno a indicare rispettivamente l'output della porta NOT e l'output della porta AND.

Infine, nelle Linee 7, 8 e 9 specifichiamo quali sono gli input e gli output delle tre porte logiche del circuito. Si noti che il primo dei parametri indica sempre l'output della porta, i successivi parametri gli input.

Esercizio 2. Scrivere un modulo Verilog per il circuito seguente



Una volta definito un modulo, possiamo poi richiamarlo per costruire altri moduli più complessi. Per esempio, nel Listing 2 definiamo prima un HALF ADDER e poi definiamo un FULL ADDER costruendolo con due HALF ADDER e una porta OR.

```
1  module half_adder(a,b,sum,carry);
2      input a,b;
3      output sum,carry;
4
5      xor(sum,a,b);
6      and(carry,a,b);
7  endmodule
8
9  module full_adder(a,b,cin,sum,cout);
10     input a,b,cin;
11     output sum, cout;
12
13     wire s1,c1,c2;
14
15     half_adder ha1(.a(cin), .b(a), .sum(s1), .carry(c1));
16     half_adder ha2(.a(s1), .b(b), .sum(sum), .carry(c2));
17     or(cout,c1,c2);
18 endmodule
```

Listing 2: Full Adder

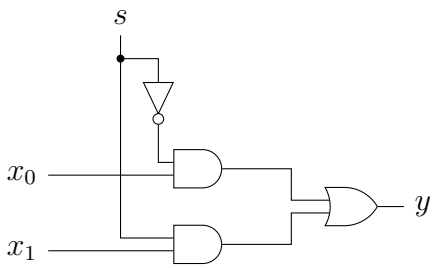
Si noti che quando abbiamo istanziato i due half adder, alle Linee 15 e 16, abbiamo anche specificato a cosa sono collegate le loro porte di input e output (*a*,*b*,*sum*,*carry*), usando la notazione *.nome_porta(espressione)*.

Esercizio 3. Disegnare il circuito corrispondente al modulo "mycirc" descritto nel seguente codice *HDL*

```
1  module blocco(x0, x1, y);
2      input x0, x1;
3      output y;
4
5      wire w1;
6
7      not(w1, x0);
8      and(y, w1, x1);
9  endmodule
10
11
12 module mycirc(in1, in2, in3, out1, out2);
13     input in1, in2, in3;
14     output out1, out2;
15
16     wire w1, w2;
17
18     blocco b1(.x0(in1), .x1(in2), .y(w1));
19     blocco b2(.x0(w1), .x1(in3), .y(out1));
20
21     not(w2, in3);
22     or(out2, in1, w2);
23 endmodule
```

Listing 3: Un circuito descritto in HDL

Le variabili che usiamo all'interno del codice possono anche essere vettori. Per esempio, possiamo descrivere in HDL un MULTIPLEXER 2:1 come in Figura 1




```
1  module mux2to1(x, s, y);
2      input [1:0]x;
3      input s;
4      output y;
5
6      wire [1:0]a;
7      wire n;
8
9      not(n, s);
10     and(a[0], x[0], n);
11     and(a[1], x[1], s);
12     or(y,a[0],a[1]);
13 endmodule
```

Figura 2: MULTIPLEXER 2:1

Il modo di descrivere i circuiti che abbiamo visto finora si chiama *structural*: specifica esattamente come è la struttura di un circuito. Gli HDL consentono anche di descrivere un circuito in modo *behavioral*, ossia specificandone la funzione. Per esempio, avremmo potuto descrivere il multiplexer in Figura 1 anche nel modo seguente.

```
1 module mux2to1(x,s,y);
2     input [1:0]x;
3     input s;
4     output y;
5
6     assign y = s ? x[1]:x[0];
7 endmodule
```

Si noti che con l'istruzione alla Linea 6 stiamo dicendo al programma di assegnare a y il valore di $x[1]$ se $s = 1$ e il valore di $x[0]$ se $s = 0$.

Esercizio 4. Descrivere in Verilog un MULTIPLEXER 4:1, sia in modo *structural* che in modo *behavioral*.