Logica e Reti Logiche
(Episodio 11: Dalla logica ai circuiti)

Francesco Pasquale

19 novembre 2026

In questo episodio introduciamo le *porte logiche* e vediamo come possono essere combinate per ottenere dei circuiti che calcolano funzioni Booleane.

DRAFT

1 Porte logiche

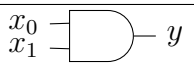
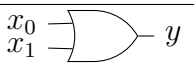
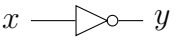
Prima di tutto specifichiamo che quando si parla di circuiti tipicamente si usa una notazione diversa per esprimere gli stessi concetti che abbiamo usato nella logica proposizionale: per esempio, indichiamo con 0 e 1, `False` e `True`, rispettivamente. Nella tabella qui sotto riassumiamo brevemente le notazioni principali.

Logica	Circuiti
`True`	1
`False`	0
$\neg p$	$\bar{p}$
$p \wedge q$	pq
$p \vee q$	$p + q$

Altri simboli usati in logica non hanno un corrispondente nel "linguaggio" dei circuiti, e viceversa. Per esempio, quando parliamo di circuiti non usiamo mai il simbolo $\to$, ma chiaramente possiamo esprimere $p \to q$ con $\bar{p} + q$. D'altra parte, in logica generalmente non si usa un simbolo specifico per lo XOR, mentre quando si parla di circuiti si usa a tale scopo $p \oplus q$.

Esercizio 1. Si verifichi che scrivendo `True` come 1 e `False` come 0, l'AND corrisponde a una moltiplicazione o a un *minimo* (pq è 1 se e solo se p e q sono entrambe 1) e l'OR corrisponde a un *massimo* ($p + q$ è 0 se e soltanto se p e q sono entrambe 0).

In tutta la seconda parte del corso assumeremo di avere a disposizione delle *porte logiche*, che implementano le operazioni logiche elementari, AND, OR e NOT, senza preoccuparci di come sono costruite a partire da componenti elettriche (transistor e diodi) e le disegneremo così

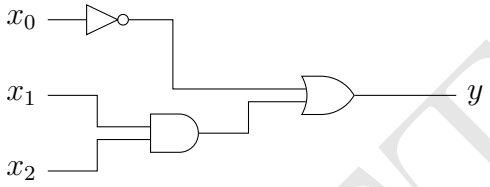
AND	OR	NOT
		

x_1	x_0	y
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_0	y
0	0	0
0	1	1
1	0	1
1	1	1

x	y
0	1
1	0

Data una qualunque formula $\mathcal{F}$ della logica proposizionale, possiamo sempre costruire un circuito che implementi $\mathcal{F}$ usando le porte logiche elementari. Per esempio, la formula $x_0 \to (x_1 \wedge x_2)$ è equivalente alla formula $\bar{x_0} \vee (x_1 \wedge x_2)$, quindi un circuito che la implementa è



Esercizio 2. Costruire un circuito che implementi la formula seguente

$$(p \to q \wedge r) \vee (\neg q \to \neg p)$$

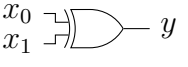
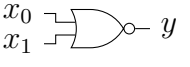
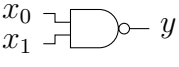
Esercizio 3. Costruire un circuito che implementi la seguente tabella di verità

p	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
q	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
r	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
s	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
?	0	0	0	1	1	0	1	1	0	0	0	0	1	0	1	0

Esercizio 4. Usando soltanto porte AND, OR e NOT, progettare un circuito che implementi la seguente funzione Booleana $f : \{0,1\}^3 \to \{0,1\}$

$$f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$$

In aggiunta alle porte logiche elementari, spesso possiamo assumere di avere anche altre porte logiche che implementano le operazioni binarie più comuni, per esempio

XOR	NOR	NAND
		

x_1	x_0	y
0	0	0
0	1	1
1	0	1
1	1	0

x_1	x_0	y
0	0	1
0	1	0
1	0	0
1	1	0

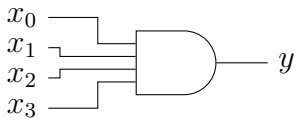
x_1	x_0	y
0	0	1
0	1	1
1	0	1
1	1	0

DRAFT

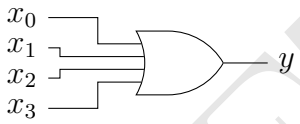
Esercizio 5. Costruire le porte logiche elementari AND, OR e NOT usando solo

1. Porte NOR;
2. Porte NAND.

Le porte logiche elementari sono tipicamente *binarie*: prendono in input due bit e restituiscono in output un bit. In genere si assume anche di avere a disposizione tali porte con un numero arbitrario di ingressi. Per esempio, in una porta AND a quattro ingressi



l'output y è 1 se e solo se *tutti* gli input x_0, x_1, x_2, x_3 sono 1. In una porta OR a quattro ingressi



l'output è 1 se e solo se *almeno uno* degli input è 1.
Naturalmente è sempre possibile costruire porte a più ingressi usando solo porte a due ingressi.

Esercizio 6. Mostrare come si possono costruire le porte a più ingressi usando solo le porte a due ingressi.

Le porte XOR a più ingressi in genere non si usano, ma se vogliamo usarle abbiamo che l'output è 1 se e solo se... Come continuiamo qui? Beh, se non lo sapete è bene che risolviate l'esercizio seguente.

Esercizio 7. La funzione XOR è definita su due bit in questo modo

$$x_1 \oplus x_2 = \begin{cases} 1 & \text{se } x_1 \neq x_2 \\ 0 & \text{altrimenti} \end{cases}$$

1. Dimostrare che la funzione XOR è *associativa* (ossia che per ogni terna di bit x_1, x_2, x_3 vale che $(x_1 \oplus x_2) \oplus x_3 = x_1 \oplus (x_2 \oplus x_3)$)
2. Osservare che, grazie al punto 1, si può definire in modo non ambiguo lo XOR di n bit, $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ (indichiamolo con $\oplus_{i=1}^{n} x_i$)
3. Dimostrare per induzione che, per ogni $n \geqslant 2$, lo XOR di n bit $\oplus_{i=1}^{n} x_i$ è uguale a 1 se e solo se il numero di bit x_i che hanno valore 1 è dispari.

DRAFT

2 Forme normali e circuiti

Una formula si dice in *forma normale disgiuntiva (DNF)*[1] se è una disgiunzione di *clausole congiuntive* $C_1 \vee C_2 \vee \cdots \vee C_n$ dove ogni clausola è una congiunzione di *letterali* $C_i = \ell_{i,1} \wedge \ell_{i,2} \wedge \cdots \wedge \ell_{i,k_i}$ e ogni letterale è una variabile oppure una variabile negata. Per esempio,

$$(p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r) \tag{1}$$

è in forma normale disgiuntiva. Data una formula X esiste sempre una formula Y equivalente[2] a X in forma normale disgiuntiva.

Nel linguaggio che usiamo per i circuiti chiamiamo forma normale *somma di prodotti* la forma normale disgiuntiva. Infatti osservate che se scriviamo la formula in (1) usando la simbologia che abbiamo introdotto per i circuiti otteniamo

$$pq\bar{r} + \bar{p}q\bar{r} + pqr \tag{2}$$

Una formula si dice in *forma normale congiuntiva (CNF)*[3] se è una congiunzione di *clausole disgiuntive* (dette anche semplicemente *clausole*) $D_1 \wedge D_2 \wedge \cdots \wedge D_n$ dove ogni clausola è una disgiunzione di *letterali* $D_i = \ell_{i,1} \vee \ell_{i,2} \vee \cdots \vee \ell_{i,k_i}$ e ogni letterale è una variabile oppure una variabile negata. Per esempio,

$$(p \vee q \vee \neg r) \wedge (\neg p \vee q \vee \neg r)$$

è in forma normale congiuntiva. Data una formula X esiste sempre una formula Y equivalente a X in forma normale congiuntiva.

Nel linguaggio che usiamo per i circuiti chiamiamo forma normale *prodotto di somme* la forma normale congiuntiva.

Esercizio 8. Scrivere una formula in forma normale somma di prodotti e una formula in forma normale prodotto di somme che abbiano entrambe la tabella di verità dell'Esercizio 3.

Se abbiamo una formula in *forma normale* è immediato ricavare un circuito e disegnarlo in un modo "standard". Per esempio, data la seguente tabella di verità

x_0	x_1	x_2	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

possiamo scrivere una formula in forma normale *somma di prodotti*

$$y = x_0\bar{x}_1\bar{x}_2 + x_0x_1\bar{x}_2 + x_0x_1x_2 \tag{3}$$

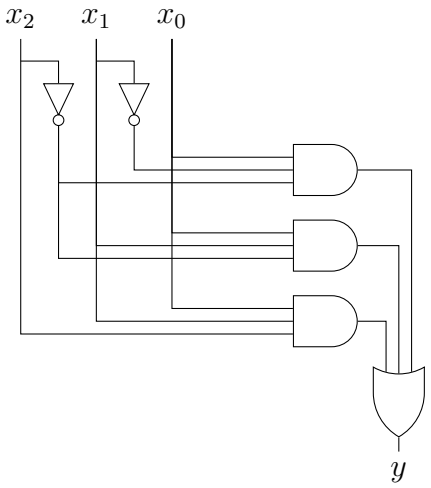
e disegnare un circuito che la implementa così

[1] *Disjunctive Normal Form*

[2] Ricorda che in logica proposizionale due formule X e Y sono *equivalenti* se hanno la stessa tabella di verità

[3] *Conjunctive Normal Form*

DRAFT



Esercizio 9. Disegnare un circuito nel modo standard per la tabella di verità dell'Esercizio 3.

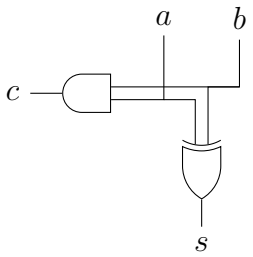
3 Circuiti e operazioni aritmetiche: la somma

Utilizzando opportunamente le porte logiche è facile costruire circuiti in grado di eseguire *operazioni aritmetiche*. Vediamo come.

Esempio. Costruiamo un circuito con due input a, b e due output s e c_{out} con le seguenti tabelle di verità

a	b	s	c_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

La tabella di s è uno XOR mentre quella di c è un AND, quindi possiamo disegnare il circuito così



Si osservi ora che in questo circuito l'output s è proprio la *somma* dei due bit in input, mentre l'output c è il *riporto*. Un tale circuito si chiama HALF ADDER.

Ricordiamoci come facciamo la somma di due numeri espressi in binario.

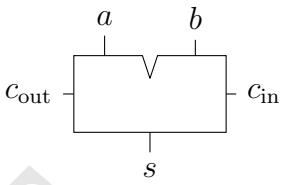
Esercizio 10. Eseguire la somma $(1011)_2 + (1110)_2$.

DRAFT

Osservate che quando facciamo la somma bit a bit in generale su ogni colonna dobbiamo avere la possibilità di sommare tre bit: i due sulla colonna più un eventuale riporto proveniente dalla colonna a destra. Il nostro HALF ADDER non è sufficiente per questo, ma se abbiamo capito il sistema non abbiamo difficoltà a generalizzarlo per ottenere un cosiddetto FULL ADDER.

Esercizio 11 (FULL ADDER)**.** Costruire un circuito con tre input a, b, c_{in} e due output s e c_{out} con le seguenti tabelle di verità

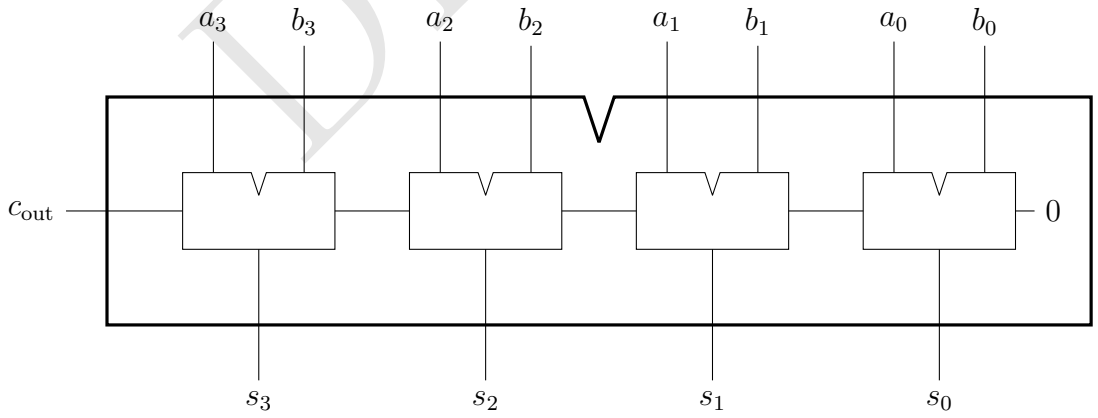
a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Esercizio 12. Costruire il circuito FULL ADDER dell'esercizio precedente usando due HALF-ADDER e una porta OR.

A questo punto, mettendo in sequenza i FULL ADDER, possiamo facilmente costruire un circuito *sommatore*. Per esempio, il circuito qui sotto prende in input otto bit a_3, a_2, a_1, a_0 e b_3, b_2, b_1, b_0 e restituisce in output cinque bit c, s_3, s_2, s_1, s_0 tali che il numero rappresentato in binario dalla sequenza di bit in output è la somma dei due numeri rappresentati in binario delle due sequenze di 4-bit in input

$$(c, s_3, s_2, s_1, s_0)_2 = (a_3, a_2, a_1, a_0)_2 + (b_3, b_2, b_1, b_0)_2$$



Esercizio 13. Progettare un circuito che faccia la differenza fra due numeri espressi in complemento a due.

La soluzione al prossimo esercizio dovrebbe rendere evidente il vantaggio di usare la codifica in *complemento a due*.

DRAFT

Esercizio 14. Progettare un circuito che prenda in input nove bit $a_3, a_2, a_1, a_0, b_3, b_2, b_1, b_0$ e c_{in} e restituisca in output quattro bit s_3, s_2, s_1, s_0. A seconda del valore del bit c_{in}, la sequenza (s_3, s_2, s_1, s_0) deve rappresentare, in complemento a due, la somma o la differenza di (a_3, a_2, a_1, a_0) con (b_3, b_2, b_1, b_0). Più precisamente, il circuito deve fare in modo che

if $c_{\text{in}} = 0$ **then**
$(s_3, s_2, s_1, s_0)_{\bar{2}} = (a_3, a_2, a_1, a_0)_{\bar{2}} + (b_3, b_2, b_1, b_0)_{\bar{2}}$
else
$(s_3, s_2, s_1, s_0)_{\bar{2}} = (a_3, a_2, a_1, a_0)_{\bar{2}} - (b_3, b_2, b_1, b_0)_{\bar{2}}$

4 Conclusioni

In questo episodio abbiamo introdotto le porte logiche e abbiamo visto come implementare le formule della logica proposizionale tramite circuiti. Abbiamo osservato che è sempre possibile costruire un circuito con una forma "standard" partendo da una formula in forma normale. Infine, abbiamo visto come è possibile costruire dei circuiti in grado di fare le operazioni aritmetiche, costruendo un circuito che calcola la "somma" di due numeri.

DRAFT