

Logica e Reti Logiche

(Episodio 9: Dalla logica ai circuiti)

Francesco Pasquale

29 aprile 2021

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

1 Porte logiche

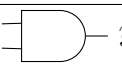
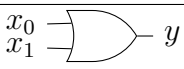

Prima di tutto specifichiamo che quando si parla di circuiti tipicamente si usa una notazione diversa per esprimere gli stessi concetti che abbiamo usato nella logica proposizionale: per esempio, indichiamo con 0 e 1, **False** e **True**, rispettivamente. Nella tabella qui sotto riassumiamo brevemente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

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

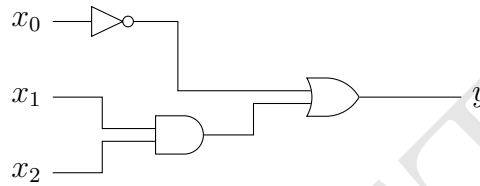
Altri simboli usati in logica non hanno un corrispondente nel “linguaggio” dei circuiti, e viceversa. Per esempio, quando parliamo di circuiti non usiamo mai il simbolo \Rightarrow , ma chiaramente possiamo esprimere $p \Rightarrow q$ con $\bar{p} + q$. D'altra parte, in logica generalmente non si usa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

Esercizio 1. Si osservi 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																																				
																																						
<table><tr><th>x_0</th><th>x_1</th><th>y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x_0	x_1	y	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>x_0</th><th>x_1</th><th>y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x_0	x_1	y	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>x</th><th>y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	y	0	1	1	0
x_0	x_1	y																																				
0	0	0																																				
0	1	0																																				
1	0	0																																				
1	1	1																																				
x_0	x_1	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 $x_0 \Rightarrow (x_1 \wedge x_2)$ 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

$$(p \Rightarrow q \wedge r) \vee (\sim q \Rightarrow \sim p)$$


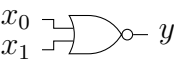
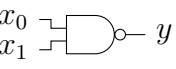
Esercizio 3. Costruire un circuito che implementi la seguente tabella di verità

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

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

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

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																																													
																																															
<table border="1"> <thead> <tr> <th>x_0</th> <th>x_1</th> <th>y</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x_0	x_1	y	0	0	0	0	1	1	1	0	1	1	1	0	<table border="1"> <thead> <tr> <th>x_0</th> <th>x_1</th> <th>y</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x_0	x_1	y	0	0	1	0	1	0	1	0	0	1	1	0	<table border="1"> <thead> <tr> <th>x_0</th> <th>x_1</th> <th>y</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	x_0	x_1	y	0	0	1	0	1	1	1	0	1	1	1	0
x_0	x_1	y																																													
0	0	0																																													
0	1	1																																													
1	0	1																																													
1	1	0																																													
x_0	x_1	y																																													
0	0	1																																													
0	1	0																																													
1	0	0																																													
1	1	0																																													
x_0	x_1	y																																													
0	0	1																																													
0	1	1																																													
1	0	1																																													
1	1	0																																													

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

1. Porte NOR;
2. Porte NAND.

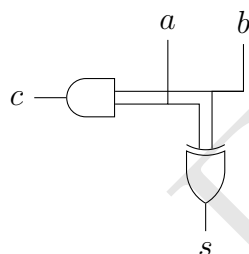
2 Operazioni aritmetiche

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

Esempio. Costruiamo un circuito con tre 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 6. Eseguire la somma $(1011)_2 + (1110)_2$.

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 7 (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 8. Costruire il circuito FULL ADDER dell'esercizio precedente usando due HALF-ADDER e una porta OR.

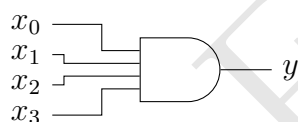
A questo punto, mettendo in sequenza i FULL ADDER, possiamo facilmente costruire un circuito *sommatore*.

Esercizio 9 (4-BIT ADDER). Costruire un circuito che prenda in input otto bit a_3, a_2, a_1, a_0 e b_3, b_2, b_1, b_0 e restituisca 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 sia la somma dei due numeri rappresentati in binario delle due sequenze di 4-bit in input

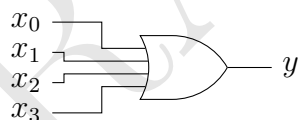
$$(c, s_3, s_2, s_1, s_0)_2 = (a_3, a_2, a_1, a_0)_2 + (b_3, b_2, b_1, b_0)_2$$

Per esempio, se gli otto bit in input sono $(a_3, a_2, a_1, a_0) = (1010)$ e $(b_3, b_2, b_1, b_0) = (1100)$, la sequenza di bit in output deve essere $(c, s_3, s_2, s_1, s_0) = (10111)$.

In aggiunta alle porte logiche elementari per le operazioni binarie, in genere si assume 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.

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

Esercizio 10. 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, $x_1 \oplus x_2 \oplus \dots \oplus x_n$ (indichiamolo con $\oplus_{i=1}^n x_i$)
3. Dimostrare per induzione che, per ogni $n \geq 2$, 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.

Esercizio 11. Mostrare come si possono costruire le porte a più ingressi usando solo le porte a due ingressi.

3 Forme normali e circuiti

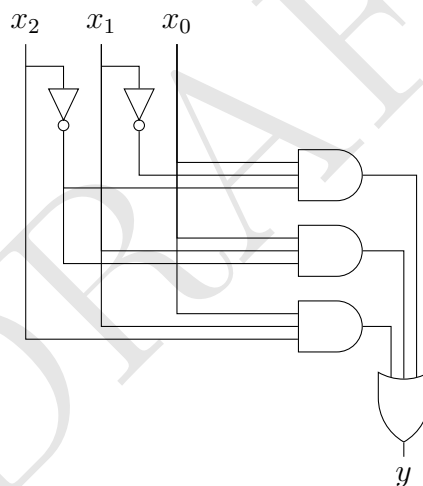
Se abbiamo una formula in *forma normale* è immediato ricavare un circuito e disegnarlo in un modo “standard”. Per esempio, data la seguente tabella di verità

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*

$$y = x_0\bar{x}_1\bar{x}_2 + x_0x_1\bar{x}_2 + x_0x_1x_2 \quad (1)$$

e disegnare un circuito che la implementa così

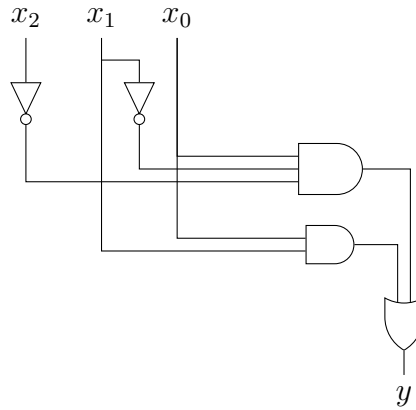


Esercizio 12. Disegnare un circuito nel modo standard per la tabella di verità dell'Esercizio 3.

Osservando con attenzione la formula in (1) vediamo che si può “semplificare” in questo modo

$$\begin{aligned} y &= x_0\bar{x}_1\bar{x}_2 + x_0x_1\bar{x}_2 + x_0x_1x_2 \\ &= x_0\bar{x}_1\bar{x}_2 + x_0x_1(\bar{x}_2 + x_2) \\ &= x_0\bar{x}_1\bar{x}_2 + x_0x_1 \end{aligned}$$

che quindi si traduce in un circuito equivalente che può ancora essere disegnato nel modo standard, ma che è più “efficiente”, perché usa meno porte logiche di quello precedente



Esercizio 13. Verificare se la formula ricavata per l'esercizio precedente si può semplificare per ottenere un circuito più semplice.

4 Conclusioni

In questo episodio abbiamo introdotto le porte logiche e abbiamo visto come implementare le formule della logica proposizionale tramite circuiti. Abbiamo accennato a come è possibile costruire dei circuiti in grado di fare le operazioni aritmetiche, costruendo un circuito che calcola la “somma” di due numeri. Infine abbiamo osservato che è sempre possibile costruire un circuito con una forma “standard” partendo da una formula in forma normale e abbiamo visto che talvolta è possibile “semplificare” le formule, preservandone la *forma normale*, in modo da ottenenere dei circuiti equivalenti costruiti con meno porte.