

Palermo

Sistemi di Elaborazione

Wakerly – DDPP
EE121

Capitoli 1 - 5

Capitolo 1: Gli elementi basilari

Elementi fondamentali di ogni circuito digitale sono le porte logiche o *gates* elementari. Esse costituiscono lo stadio più basso dell'architettura elaborativa, essendo i mattoni con cui sono costruiti tutti i dispositivi logici che stanno alla base della moderna elettronica digitale. Collegate opportunamente fra loro, secondo i dettami delle leggi dell'*algebra booleana*, consentono di implementare funzioni di varia complessità, costituendo fra loro delle reti logiche.

La porta logica è la realizzazione pratica dell'astrazione di operatore logico, la cui simbologia viene adottata per la descrizione delle porte stesse. Ad una porta logica vanno applicati in ingresso degli impulsi, che possono avere livello logico "0" o "1" (altrimenti detto rispettivamente "basso"/"low" e "alto"/"high"), e si ottiene in uscita un segnale che è funzione degli impulsi di input. E' importante tener presente che esistono soltanto 2 livelli logici.

Attraverso la porta logica si realizza una funzione che *mappa*, per specifici valori di ingresso, specifici ed univoci valori di uscita. Tale *mappatura* viene riassunta in una tabella detta "tavola della verità" che è uno dei metodi di rappresentazione della funzione medesima.

Nel dettaglio, le porte logiche basilari sono tre e corrispondono agli operatori logici di congiunzione (AND), disgiunzione (OR) e negazione logica (NOT).

La **porta AND**, dati "n" ingressi a cui è data facoltà di assumere valori logici alti o bassi (in figura sono due e sono indicati con "X" e "Y"), risponde in uscita con un segnale che risulta essere alto se e solo se entrambi gli ingressi sono alti; ciò significa, come evincibile dalla tabella della verità, che se uno solo dei segnali d'ingresso è allo stato alto o se entrambi si trovano allo stato basso sull'output della porta avremo uno stato logico basso. La simbologia per indicare l'operazione di AND fra due o più variabili è:

$$X \text{ AND } Y \quad \text{oppure} \quad X \wedge Y$$

ma in elettronica digitale si preferisce usare:

$$X \cdot Y$$

e definire l'operazione di AND come prodotto logico fra "n" variabili. Il simbolo caratteristico della porta AND è una figura avente tre lati simili ad un rettangolo ed uno sostituito da un arco convesso. Gli ingressi sono disegnati entranti sul lato opposto all'arco stesso da cui esce, invece, l'output della porta.

La **porta OR**, dati "n" ingressi a cui è stata data facoltà di assumere valori logici alti o bassi (indicati anch'essi nella relativa figura con "X" e "Y"), risponde in uscita con un segnale che risulta essere alto se almeno uno degli ingressi è alto; ciò significa che dà un livello basso solo se entrambi gli input sono, al loro volta, bassi. Nei restanti casi l'uscita è sempre alta. E' importante notare che se sia X che Y sono alti anche l'uscita lo sarà in quanto questa operazione di OR non è esclusiva. La simbologia per la scrittura può assumere anch'essa diverse forme come:

$$X \text{ OR } Y \quad \text{oppure} \quad X \vee Y \quad \text{oppure} \quad X + Y$$

e l'operazione di OR viene indicata come somma logica fra "n" variabili. Il simbolo caratteristico è simile a quello



X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1



X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

dell'AND, ma il lato degli ingressi è sostituito da un altro arco, in genere con minore freccia, ma concavo.

N.B.: Per scrivere operazioni di AND fra più variabili si usa la forma:

$$X \cdot Y \cdot Z \cdot K$$

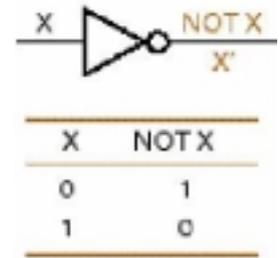
mentre per varie OR si usa:

$$X + Y + Z + K$$

Inoltre le porte logiche AND ed OR hanno un numero di ingressi $n > 1$ ed una sola uscita, a differenza del terzo tipo di porta elementare, la NOT, che dispone di un solo ingresso ed una sola uscita.

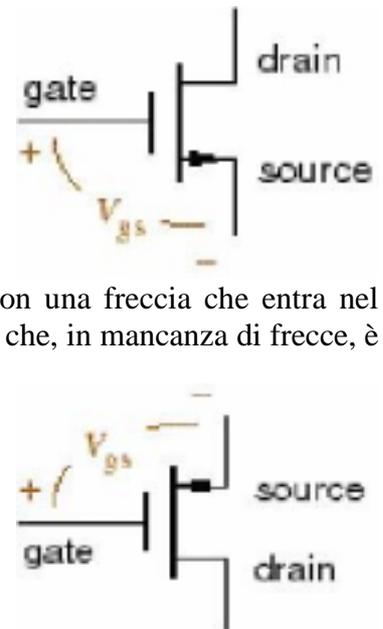
La **porta NOT**, infatti, ha il solo scopo di effettuare una inversione logica, cioè di convertire un ingresso alto in una uscita bassa e viceversa. La simbologia utilizzata per il NOT è anch'essa multipla:

$$\neg X \quad \text{oppure} \quad X' \quad \text{oppure} \quad \overline{X}$$



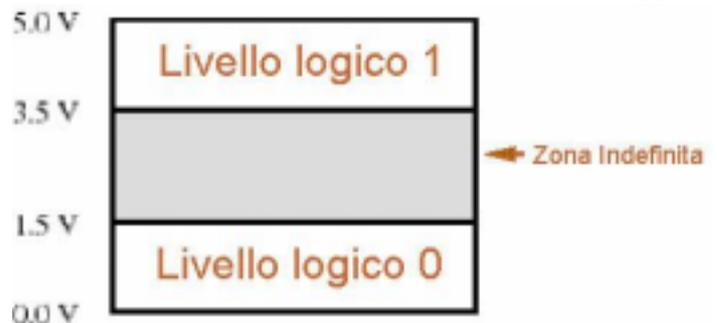
Riguardo al simbolo grafico di questa porta è necessario fare un'osservazione molto importante: la rappresentazione classica è quella realizzata con un triangolino con un ingresso alla base e una pallina (*bubble*) sul vertice opposto da cui si preleva l'uscita; in realtà il simbolo di inversione viene determinato dalla sola bubble, ed il triangolino rappresenta una *transmission gate* in grado di rigenerare elettricamente il segnale senza apportare alcuna modifica al suo stato logico. Per questo motivo ovunque si trovi la bubble, anche disegnata su altri simboli, avviene una inversione logica.

Le porte logiche possono essere realizzate con diverse **tecnologie** come *TTL* e *CMOS*, per citarne due; quest'ultima è quella di cui ci occuperemo. **CMOS** è acronimo di "*Complementary Metal Oxide Semiconductors*". Complementare perché si utilizzano le due famiglie *PMOS* e *NMOS* per avere il kit completo di semiconduttori necessari a coprire la gamma di esigenze conduttive. Nella fattispecie, per la realizzazione delle porte si usano transistor MOS *P-Channel* ed *N-Channel*. La simbologia per indicarli è quella indicata in figura con una freccia che entra nel *source* nel caso degli N-Channel, e che ne esce nel caso dei P-Channel che, in mancanza di frecce, è caratterizzato da una bubble posta sull'ingresso gate. Le differenze elettriche dei transistor a canale P ed N sono relativi alla loro polarizzazione ma, semplificando, possiamo dire che i P-Channel conducono se al gate è applicata una tensione "0" ed è isolato per una tensione prossima a V_{cc} (che è la tensione di alimentazione), mentre gli N-Channel si comportano esattamente in modo inverso, cioè conducono per tensioni al gate prossime a V_{cc} e si interdicono per tensioni prossime allo "0". Dicendo che conduce o si interdice si intende dire che la *resistenza ohmica* del transistor è nell'ordine di poche centinaia di Ω (in caso di conduzione) oppure



nell'ordine dei $M\Omega$ (in caso di interdizione).

La specifica dei **livelli di tensione CMOS** ha la caratteristica di essere simmetrica, ossia le zone a livello logico alto e quelle a livello logico basso hanno una soglia di rumore (*noise margin*) abbastanza alta, che copre per ogni livello



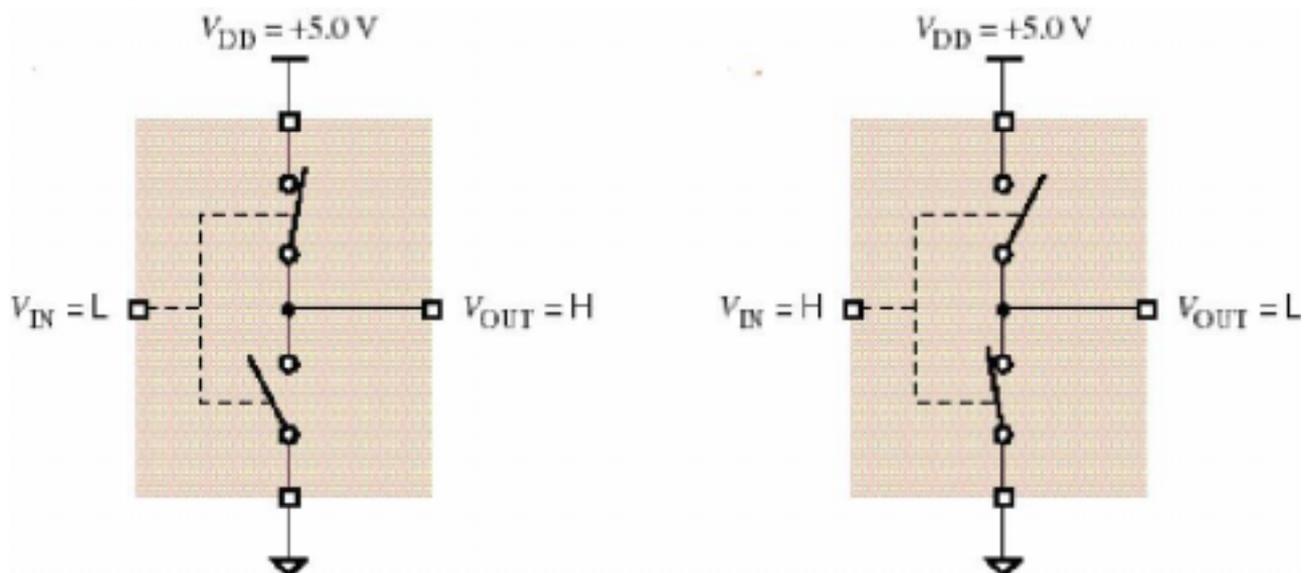
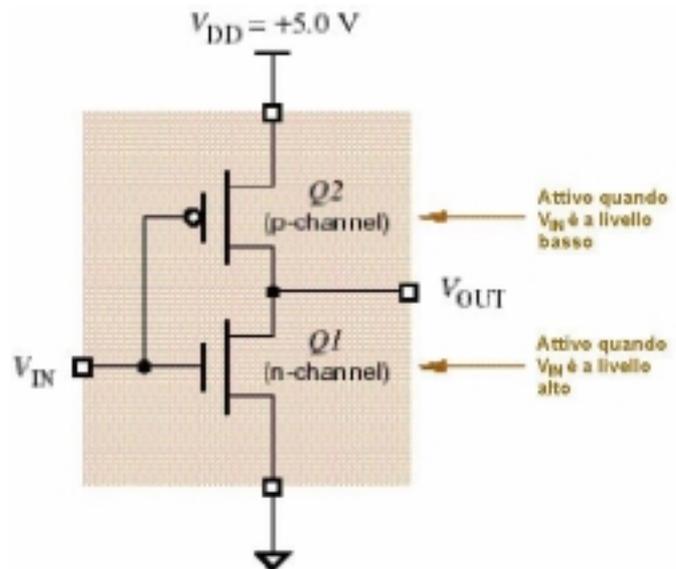
il 30% circa dell'intera escursione di tensione. Essendo V_{cc} la tensione di alimentazione, il livello logico si attesta su "basso" se la tensione dell'impulso di input è compreso tra lo 0% ed il 30% (nella figura si assume $V_{cc}=+5V$) ed "alto" se supera il 70%. L'intervallo compreso fra il 30% ed il 70% è una zona indefinita; impulsi di tensione ricadenti in questa fascia danno risultati anomali e comunque non funzionali. In questa zona il dispositivo ha una caratteristica analogica del tutto estranea alle necessità imposte dal contratto digitale. A tal fine è sempre indispensabile che il circuito sia ben calcolato e che il segnale abbia sempre un livello opportuno, venendo rigenerato se necessario.

Per **contratto digitale** si intende la specifica secondo cui a specifici livelli logici corrispondono specifici livelli elettrici; segnali al di fuori del contratto digitale non sono né utilizzabili né consentiti.

Costruttivamente le porte logiche sono fatte da blocchi in tecnologia CMOS (nell'illustrazione un *inverter*) posti in contrapposizione in maniera che una serie si abiliti quando l'uscita deve essere a livello logico alto (*pull-up*) e un'altra si abiliti per portare l'uscita a livello basso (*pull-down*). In questo caso se V_{IN} è a livello basso, "Q2" (il pull-up) si attiva mentre "Q1" è interdetto; viceversa, se V_{IN} è a livello alto attiva "Q1" portando V_{OUT} a livello basso e interdicendo "Q2". E' molto importante che il pull-up e il pull-down non conducano contemporaneamente, nel qual caso si avrebbe un *corto circuito* che potrebbe distruggere la porta; per questo motivo vengono usati insieme transistor a canale P ed N.

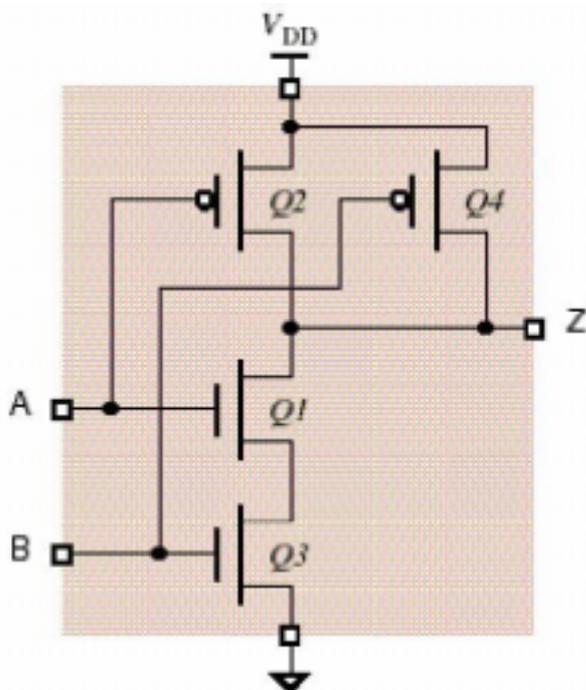
E' da notare che, in tecnologia CMOS, a parità di superficie di silicio, i transistor a canale N sono più efficienti, per cui si preferiscono porte costruite con pull-down di questo tipo; si vedrà infatti che si privilegeranno le porte di tipo NAND.

I transistor si comportano come degli interruttori; quando uno è attivato non lo è l'altro e viceversa, come mostrato nello schema:



Capitolo 2: Porte logiche CMOS, cenni sulle TTL, elettronica e tempistica

Spingendosi più a fondo fin al cuore delle porte logiche, si vede che è possibile realizzarle con diverse tecnologie, come ad esempio la TTL (acronimo di “*Transistor Transistor Logic*”), o la CMOS, che è oggetto del nostro studio immediato. Le porte logiche sono costituite da transistor che, opportunamente collegati fra loro, ne determinano la caratteristica. In questa sede non considereremo il comportamento analogico dei *transistors* ma solo le loro caratteristiche di saturazione ed interdizione. Considereremo quindi un transistor saturo come un interruttore chiuso (cioè in grado di condurre) e uno in interdizione come un interruttore aperto (cioè incapace di condurre). Lo stato di conduzione o meno di un transistor è determinato dal livello di tensione applicato al suo terminale di controllo o *gate*.



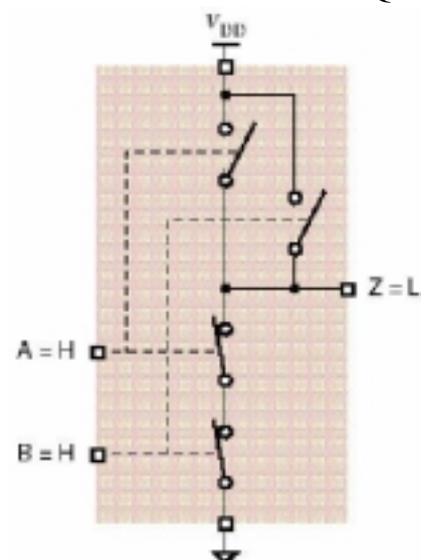
A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	H
H	L	on	off	off	on	H
H	H	on	off	on	off	L



Qui sopra una NAND in tecnologia CMOS, se ne osservi il comportamento: “A” e “B” sono gli ingressi e “Z” l’uscita della porta. “A” è connesso sul terminale di controllo dei transistor “Q2” (a canale P) e “Q1” (a canale N), mentre “B” è connesso sul terminale di controllo dei transistor “Q4” (a canale P) e “Q3” (a canale N). I transistor Q2 e Q4 sono saturi (cioè conducono) se sui relativi terminali di controllo è presente una tensione di 0 V (corrispondente ad un livello logico basso) ed interdetti (cioè isolati) in presenza di una tensione pari a V_{DD} (corrispondente ad un livello logico alto). I transistor Q1 e Q3, viceversa, conducono se controllati da una tensione pari a V_{DD} e si interdicono se controllati da una tensione di 0 V. Nella figura sovrastante, come nelle successive, si indica con “L” una tensione di 0 V e con “H” una tensione pari a V_{DD} ; inoltre è indicato con “ON” un transistor che conduce e con “OFF” uno isolato.

Si analizzino i casi verificabili:

- A=L e B=L: sono interdetti Q1 e Q3 e conducono Q2 e Q4, attraverso i quali V_{DD} giunge al terminale Z, che è quindi a



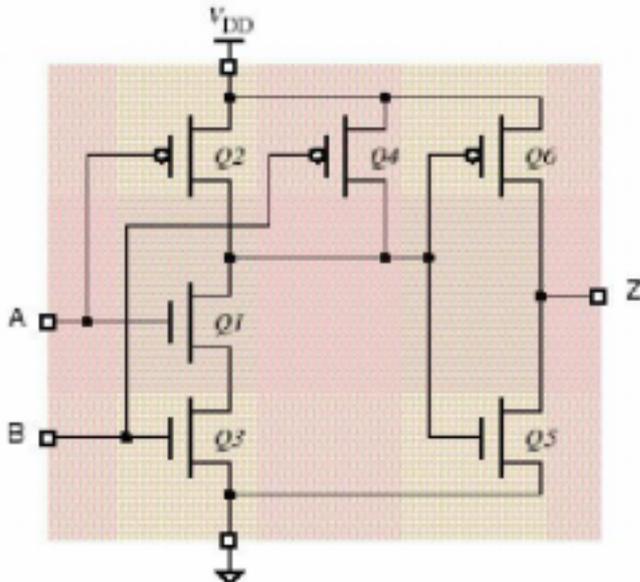
livello H;

- A=L e B=H: sono interdetti Q1 e Q4 e conducono Q2 e Q3, V_{DD} arriva a Z attraverso Q2 mentre la tensione "0" non è in grado di arrivarci perché bloccata da Q1 che è aperto; quindi, anche in questo caso, $Z=H$;
- A=H e B=L: sono interdetti Q2 e Q3 e conducono Q1 e Q4; V_{DD} giunge a Z attraverso Q4 mentre la tensione "0" è bloccata da Q3 che è aperto, quindi $Z=H$;
- A=H e B=H: sono interdetti Q2 e Q4, quindi V_{DD} non può giungere a Z, e conducono Q1 e Q3, attraverso cui la tensione pari a 0 V giunge a Z; in questo caso $Z=L$

Si vede da quanto detto che il complesso Q2 e Q4, quando opportunamente abilitato, serve a far giungere V_{DD} su Z ed è per ciò detto **pull-up**; viceversa il complesso Q1 e Q3 è in grado di far giungere sull'uscita la tensione di 0 V ed è per tale motivo detto **pull-down**.

N.B.: Ogni porta logica deve disporre di un pull-up e di un pull-down che, opportunamente pilotati, portino l'uscita al livello elettrico (e quindi logico) corretto e desiderato.

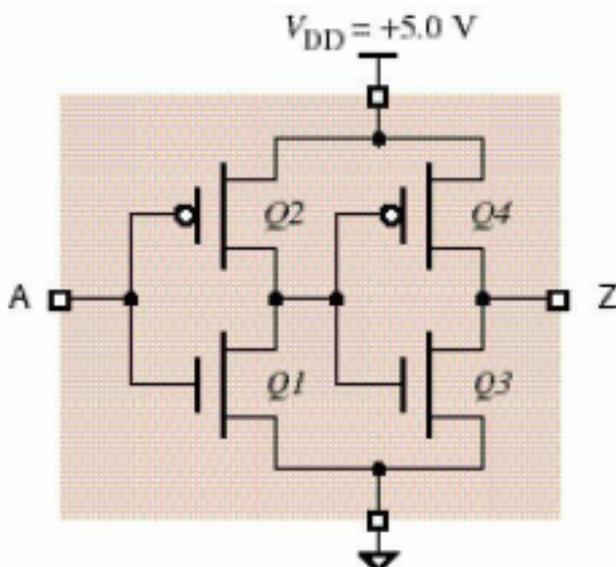
In CMOS una porta **AND** è costituita da una NAND seguita da un inverter; per motivi che saranno meglio compresi più avanti, la porta NAND è il mattone costruttivo nella logica realizzata in questa tecnologia.



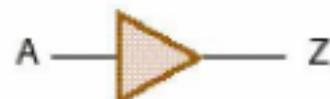
A	B	Q1	Q2	Q3	Q4	Q5	Q6	Z
L	L	off	on	off	on	on	off	L
L	H	off	on	on	off	on	off	L
H	L	on	off	off	on	on	off	L
H	H	on	off	on	off	off	on	H



Analogamente a quanto succede per la AND, anche un **buffer** è costituito da un primo inverter seguito da un secondo (il primo costituito dai transistor "Q1" e "Q2" ed il secondo da "Q3")



A	Q1	Q2	Q3	Q4	Z
L	off	on	on	off	L
H	on	off	off	on	H



e “Q4”).

Gli unici **due stadi** possibili si comportano evidentemente in maniera complementare:

- A=L: è interdetto “Q1” ed attivo “Q2”, che porta V_{DD} sul pilotaggio di “Q4” e “Q3”, ma abilitando solo quest’ultimo che fa transitare la tensione “0” verso Z, quindi Z=L;
- A=H: Q2 è interdetto e Q1 è attivo; sul pilotaggio di Q3 e Q4 arrivano 0 volts e quindi Q4 si porta in conduzione facendo giungere V_{DD} su Z che si porta a stato H.

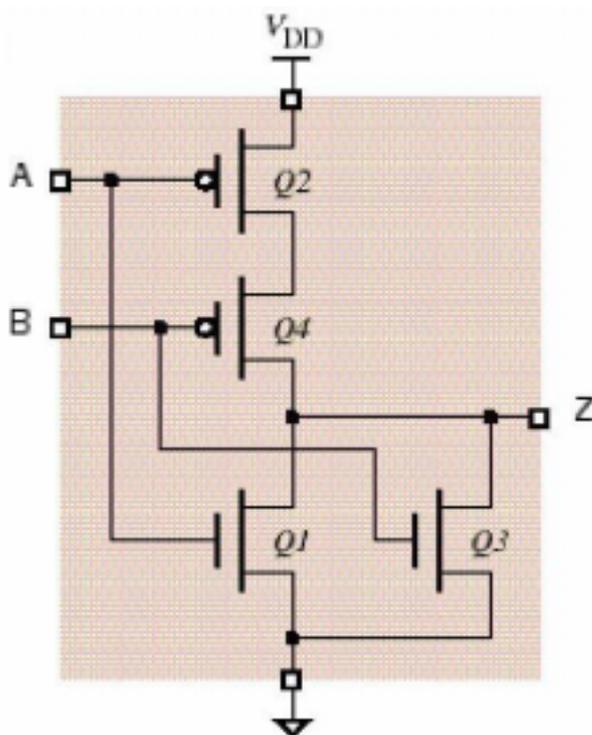
E’ interessante notare, a questo punto, che il **numero di transistor** necessari in una porta con uscita negata è pari a due volte il numero degli ingressi secondo la formula:

$$T_{out} = 2 \cdot T_{in}$$

Per le porte non invertite questa cifra v’è incrementata di due unità, in quanto bisogna implementare un inverter d’uscita che restituisca un segnale non invertito; i transistor necessari sono quindi:

$$T_{out} = 2 \cdot T_{in} + 2$$

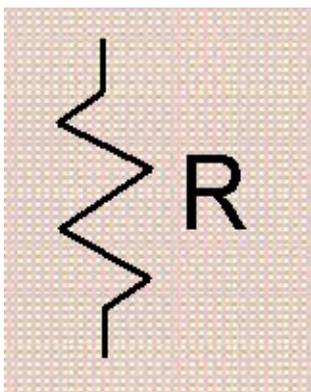
Come si vede qui di seguito anche la porta **NOR** segue la regola.



A	B	Q1	Q2	Q3	Q4	Z
L	L	off	on	off	on	H
L	H	off	on	on	off	L
H	L	on	off	off	on	L
H	H	on	off	on	off	L



Seguiranno semplici **concetti di elettrotecnica** relativi al comportamento dei *circuiti resistivi*, a cui possiamo assimilare l’astrazione delle nostre porte logiche, avendone fissati gli ingressi. In altre parole, ogni configurazione d’ingresso della porta logica è assimilabile ad un preciso circuito resistivo, cioè composto da sole resistenze appositamente collegate.



Una *resistenza*, la cui unità di misura è l’Ohm (Ω), è un dispositivo in grado di opporsi al passaggio della corrente elettrica, ossia il flusso elettronico viene “frenato” e limitato passando attraverso di esso. In un circuito resistivo circola una corrente elettrica (misurata in Amperes, simbolo A) direttamente proporzionale alla tensione di alimentazione ed inversamente proporzionale alla resistenza del circuito stesso, secondo la relazione:

$$\text{Corrente} = \frac{\text{Tensione}}{\text{Resistenza}} \quad [A] = \frac{[V]}{[\Omega]}$$

rappresentante la **prima legge di Ohm**.

Utili sono al nostro scopo anche le formule che determinano la **resistenza equivalente** (R_{EQ}) di serie e paralleli di resistenze:

•Resistenze in serie: $R_{EQ} = R1 + R2 + \dots + Rn$

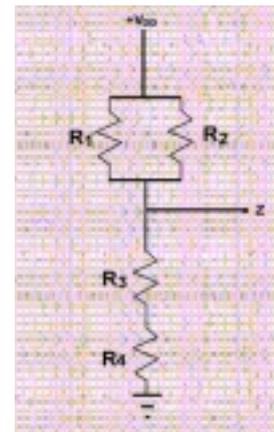
•Resistenze in parallelo: $R_{EQ} = \left[\frac{1}{R1} + \frac{1}{R2} + \dots + \frac{1}{Rn} \right]^{-1}$

Nel caso particolare in cui si hanno due sole resistenze in parallelo, la formula diventa:

$$R_{EQ} = \frac{R1 \cdot R2}{R1 + R2}$$

Analizziamo, per esempio, il **comportamento di una porta NAND a due ingressi**, le cui configurazioni dei transistor sono pari a $T_{out} = 2 \cdot T_{in}$, cioè 4:

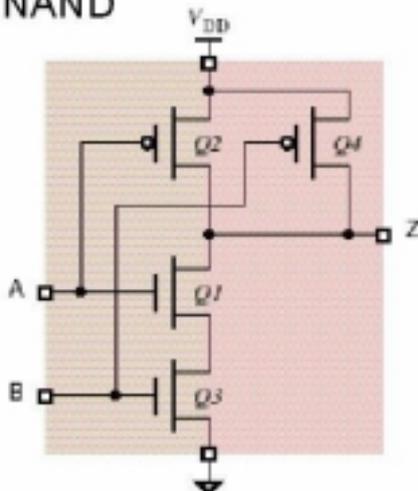
- A=0, B=0: le resistenze R1 ed R2 sono circa 100Ω ciascuna, mentre R3 e R4 sono circa 1MΩ. $[R1 + R2]_P = 50\Omega$ quindi V_{DD} incontra una resistenza bassa per giungere a Z, mentre $[R3 + R4]_S = 2M\Omega$ quindi i 0 volts incontrano un'alta resistenza per raggiungere l'uscita; in questo caso Z=H;
- A=0, B=1: R1 e R4 sono circa 100Ω ciascuna mentre R3 e R2 circa 1MΩ. $[R1 + R2]_P \cong 99,9\Omega$ quindi V_{DD} incontra una resistenza bassa per giungere a Z, mentre $[R3 + R4]_S = 1,0001M\Omega$ e quindi gli 0 volts incontrano un'alta resistenza per raggiungere l'uscita; in questo caso Z=H;
- A=1, B=0: R2 e R3 sono circa 100Ω ciascuna mentre R1 e R4 sono circa 1MΩ. $[R1 + R2]_P \cong 99,9\Omega$ quindi V_{DD} incontra una resistenza bassa per giungere a Z, mentre $[R3 + R4]_S = 1,0001M\Omega$ e quindi gli 0 volts incontrano un'alta resistenza per raggiungere l'uscita; in questo caso Z=H;
- A=1, B=1: R3 e R4 sono circa 100Ω ciascuna mentre R1 e R2 circa 1MΩ. $[R1 + R2]_P = 0,5M\Omega$ quindi V_{DD} incontra una resistenza alta per giungere a Z, mentre $[R3 + R4]_S = 200\Omega$ e quindi lo 0 incontra un'alta resistenza per raggiungere l'uscita; in questo caso Z=L.



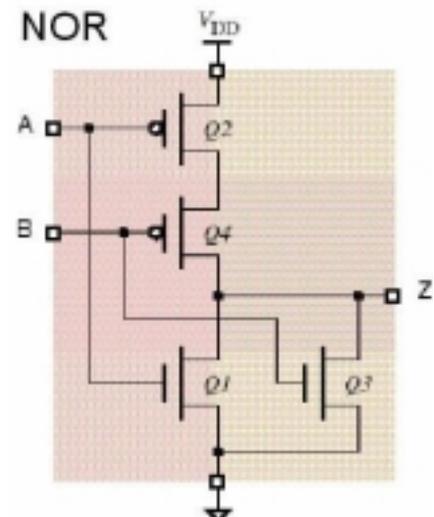
N.B.: è importante notare che gli stadi (pull-up o pull-down) realizzati con resistenze in parallelo riescono ad erogare o assorbire molta più corrente (2-4 volte) di quella degli omologhi in serie. In questo modo si spiega il fatto che in tecnologia CMOS, a parità di superficie di silicio, i transistor a canale P hanno un rendimento inferiore a confronto di quelli a canale N.

Dalle due considerazioni precedenti si deduce la seguente regola: per controbilanciare l'inefficienza dello stadio in serie è meglio utilizzare transistor a rendimento più elevato. Ciò significa che l'optimum è utilizzare serie di transistor a canale N, ma l'unica porta in cui si ritrova questa condizione è la NAND, che la utilizza per lo stato di pull-down. Per questo motivo il tipo di porta che si preferisce usare in tecnologia CMOS è proprio la NAND.

NAND

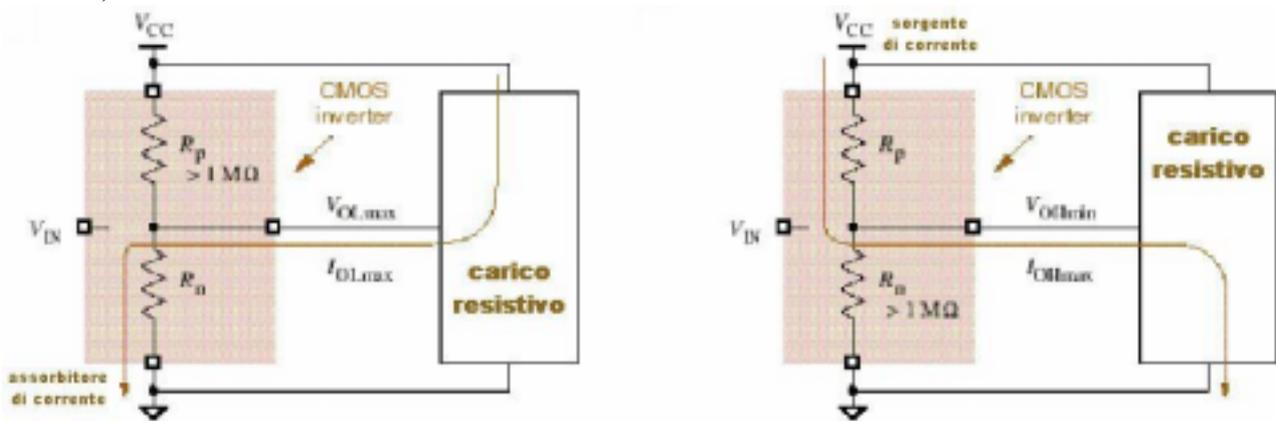


NOR



I transistor, di cui sono composte le porte logiche, hanno **specifiche ben precise**, necessitano cioè di ben specifici livelli di corrente/tensione per portarsi in conduzione o interdizione. Da ciò segue che ogni stadio deve essere perfettamente bilanciato a recepire i segnali a lui destinati ed essere in grado di fornire un'uscita adeguata a pilotare gli stadi successivi. Le caratteristiche degli elementi CMOS possono essere influenzate da fattori esterni quali la tensione di alimentazione, la temperatura, la qualità del segnale d'ingresso ed il carico d'uscita. Per questo è sempre importantissimo tener conto di questi parametri in fase di progettazione e non esulare dalle specifiche di fanin e fanout della porta. In seguito si vedrà che oltre a questi due fattori si dovrà, al fine del corretto funzionamento del circuito logico, tener sempre ben presenti anche altre due componenti che sono i tempi di setup e di hold.

Il *fanin* è una caratteristica di ingresso che definisce la minima qualità del segnale di input; si anticipa che il fanin di una porta deve avere una caratteristica più estesa rispetto al fanout di quella pilotante. Per il *fanout* necessita invece fare un discorso ben più articolato; lo definiremo in prima istanza come la capacità di carico di uscita di una porta. Ogni porta pilotata assorbe dalla pilotante una certa quantità di corrente che, seppur piccola, si fa rilevante se la pilotante pilota tante porte. Se ipotizziamo la somma dei fanin delle porte pilotate come un carico resistivo connesso alla porta pilotante possiamo trovarci di fronte ad una delle seguenti situazioni (nell'esempio un inverter):

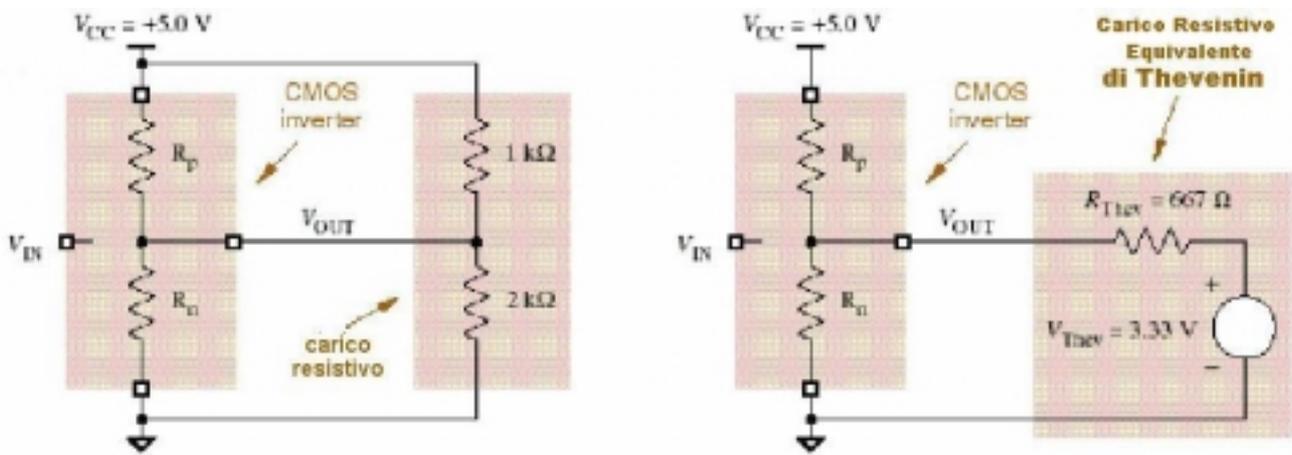


R_p ed R_n rappresentano rispettivamente il pull-up ed il pull-down della porta pilotante mentre il carico resistivo è la somma vettoriale dei fanin di tutti gli stadi pilotati. Il caso della figura di sinistra si verifica quando l'uscita della porta è $Z=L$ (cioè bassa), situazione nella quale il dispositivo è costretto ad “assorbire” (*sink*) la corrente erogata dall'insieme di quelli pilotati. Viceversa, a destra, il dispositivo ha uscita $Z=H$ (cioè alta) e deve quindi “erogare” (*source*) corrente verso quelli pilotati. Da quanto detto si capisce che la porta pilota deve essere in grado sia di assorbire la corrente (eventualmente erogata dagli altri dispositivi messi in cascata) o eventualmente di erogarla. Abbiamo però visto che gli stadi di pull-up è pull-down di una porta non hanno la stessa “forza”, non sono cioè in grado di erogare o assorbire la stessa corrente; pertanto il fanout va valutato in funzione dello stadio più debole.

Spingendosi oltre in questa valutazione possiamo sostituire al carico resistivo un *circuito equivalente di Thevenin*. Il **teorema di Thevenin** (applicazione del *teorema di Kirchhoff*) recita testualmente:

“una rete elettrica, considerata tra i punti “A” e “B”, si comporta, rispetto al carico che gli sta a valle, come un generatore che fornisce la tensione a vuoto fra “A” e “B” e che mostra una impedenza interna pari all'impedenza che è possibile misurare fra “A” e “B” cortocircuitando tutti i generatori di tensione ed aprendo tutti quelli di corrente”

In altre parole, un carico resistivo si comporta, rispetto all'uscita della porta, come un generatore erogante tensione V_{Thev} con in serie una resistenza R_{Thev} .



A titolo di esempio si passa allo svolgimento dell'**esercizio** in questione; nella figura sono già indicati V_{Thev} ed R_{Thev} .

a) Calcolo di V_{Thev} :

La corrente che passa attraverso le resistenze del carico resistivo, secondo la legge di Ohm, è pari al rapporto tra la tensione V_{CC} e la R_{EQ} della serie della resistenza di $1K\Omega$ e quella da $2K\Omega$, quindi:

$$I_{RC} = \frac{V_{CC}}{R_{EQ}} = \frac{V_{CC}}{[1K\Omega + 2K\Omega]_s} \cong 0.001666 \text{ A}$$

la tensione ai capi della resistenza da $2K\Omega$ è pari a:

$$V_{Thev} = I_{RC} \cdot 2\Omega \cong 3.33 \text{ V}$$

b) Calcolo di R_{Thev} :

$$R_U = [R_p + 1K\Omega]_p \cong 999 \Omega$$

$$R_D = [R_n + 2K\Omega]_p \cong 95.2 \Omega$$

$$[R_U + R_D]_s \cong 994.2 \Omega$$

la corrente che transita attraverso l'intero circuito è:

$$I = \frac{V_{CC}}{[R_U + R_D]_s} \cong 0.0503 \text{ A}$$

la tensione ai capi è:

$$V_{RD} = I \cdot R_D \cong 0.4.5 \text{ V}$$

ma la stessa:

$$V_{RD} = V_{Thev} \cdot \left(\frac{R_n}{[R_n + R_{Thev}]_s} \right)$$

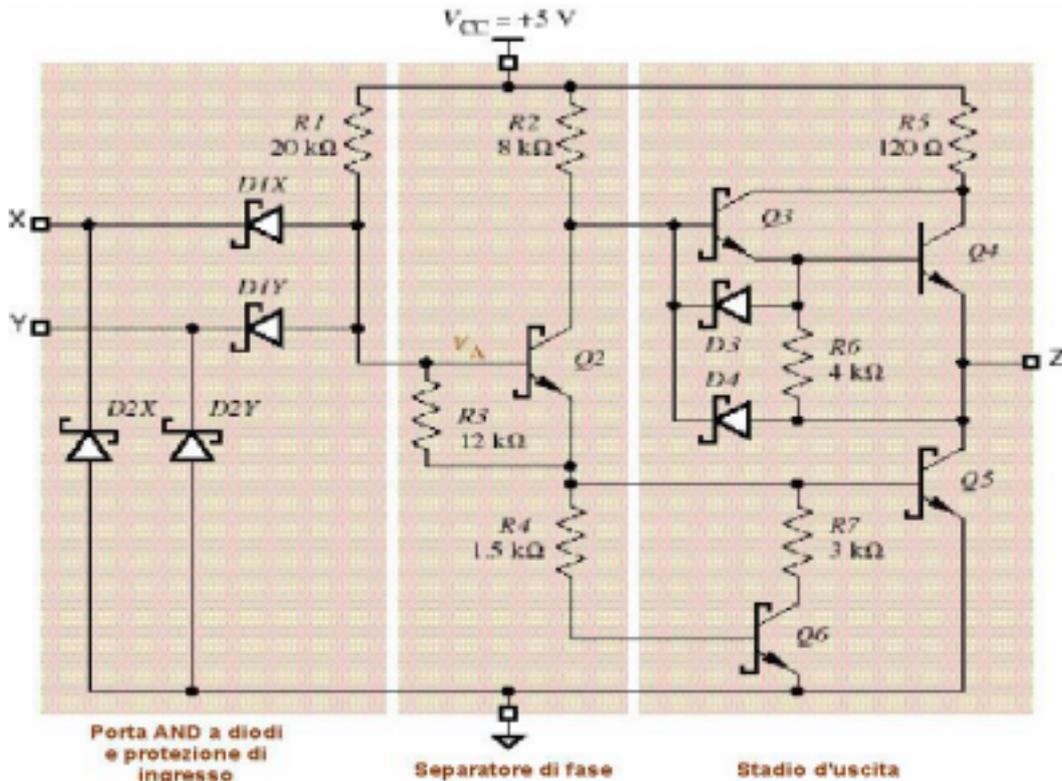
da cui segue che:

$$R_{Thev} = \left(\frac{V_{Thev} \cdot R_n}{V_{RD}} \right) - R_n \cong 667 \Omega$$

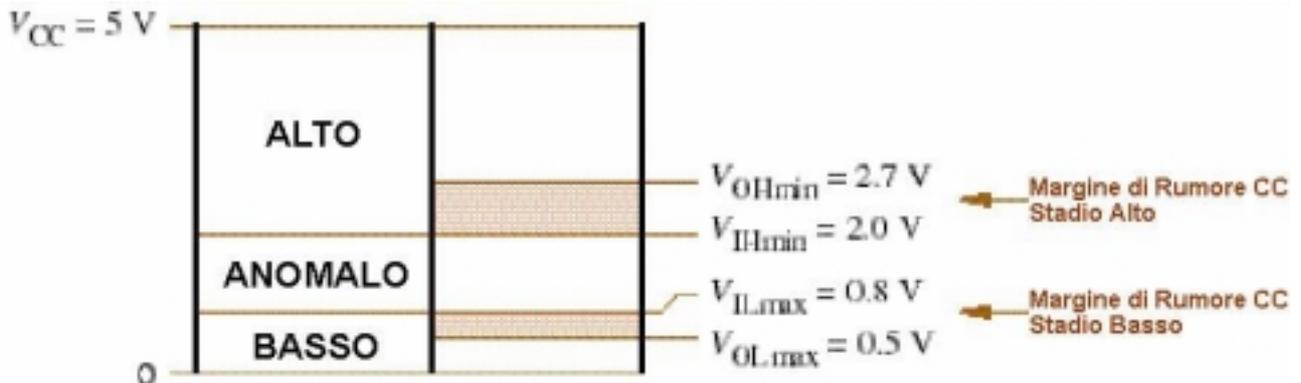
Come visto precedentemente, i livelli elettrici devono rispettare il contratto digitale e se, a causa dei problemi appena visti, per carichi eccessivi, i livelli di tensioni passano nella zona

indefinita, questo è il caso di un errato progetto che non ha tenuto conto di fanin e fanout delle porte.

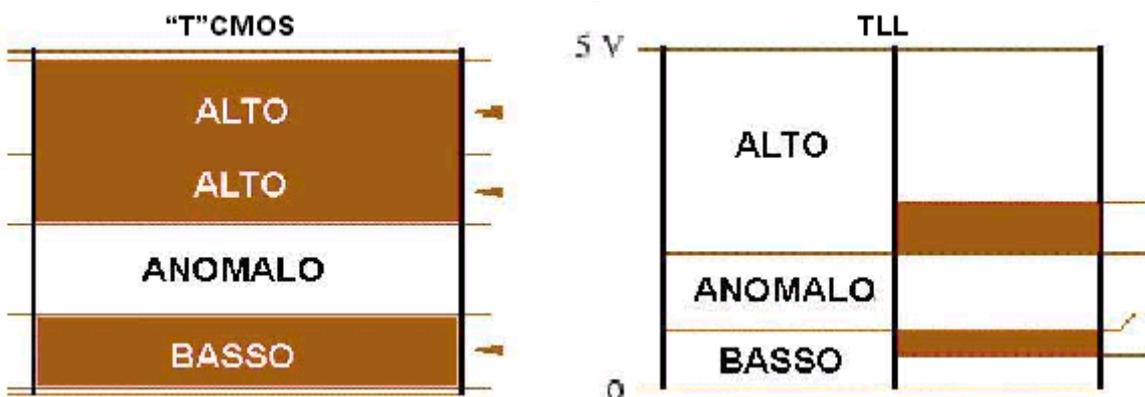
Si esamineranno adesso le caratteristiche elettriche delle famiglie TTL.



Le tensioni associate ai livelli logici ed i relativi noise margin hanno peculiarità assolutamente dissimili da quelle delle famiglie CMOS, come mostra la figura sottostante:



Come si vede, i livelli hanno una caratteristica assolutamente asimmetrica; un noise margin molto ristretto ed una escursione valida del livello logico basso molto ristretta; per contro una molto estesa variabilità del livello elettrico associato al livello logico alto.



Questa discrepanza tra le caratteristiche delle due famiglie fa sì che ci sia una completa *incompatibilità*, a meno di usare delle famiglie di transizione ed adattamento. In CMOS sono nate le "T"CMOS (HCT, FCT, VHCT) con caratteristiche modificate. Altre differenze relative a TTL sono:

- rilevante corrente di input allo stato BASSO, dispersione di corrente allo stato ALTO;
- l'output richiede molta più corrente allo stato BASSO (transistori saturi);
- l'output può fornire una limitata corrente allo stato ALTO.

Capitolo 3: Algebra Booleana & analisi dei circuiti combinatori

Gli elaboratori digitali, per rappresentare le informazioni al proprio interno, utilizzano il codice binario costituito dai simboli “0” ed “1”. Questi prendono il nome di bit, dalla contrazione di “*binary digit*”. Il motivo per cui il sistema binario ha avuto tanta importanza nei sistemi di elaborazione é dovuto al contributo di *George Boole*, il quale dimostrò come la logica possa essere ridotta ad un sistema algebrico molto semplice, del tutto elementare, che utilizza un codice binario (0 e 1, vero e falso). Il *codice binario* fu trovato particolarmente utile nella teoria della commutazione per descrivere il comportamento dei circuiti digitali (1=acceso, 0=spento). Successivamente *Claude Shannon* definì un metodo per rappresentare un qualsiasi circuito costituito da una combinazione di interruttori (e/o *rélais*) mediante un insieme di espressioni matematiche, basate sulle regole dell’algebra Booleana. L’eccezionale contributo di Boole fu quello di “dare espressione alle leggi fondamentali del ragionamento nel linguaggio simbolico del Calcolo”.

Boole suppone infatti di avere:

- un sistema di simboli arbitrari (x, y, z, \dots) i quali rappresentano gli oggetti che devono essere discussi;
- due operazioni designate ($+$ e \cdot) che operano sui simboli producendo ulteriori simboli giacché, se “ x ” ed “ y ” sono simboli, lo sono anche “ $x+y$ ” e “ $x \cdot y$ ”;
- un’operazione di identità “ $=$ ”;
- svariate regole che governano il comportamento delle operazioni, ad esempio:

$$x + y = y + x; \quad x \cdot y = y \cdot x; \quad x \cdot (y + z) = x \cdot y + x \cdot z$$

Queste regole sono le stesse dell’algebra classica, con una sola eccezione:

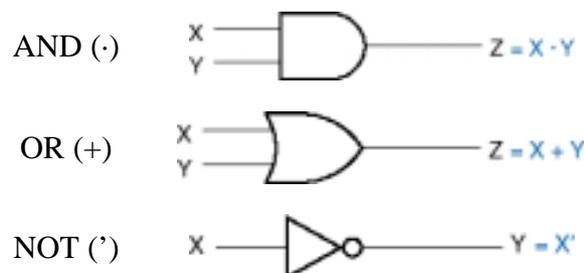
$$x + x = x$$

$$x \cdot x = x$$

Gli *operatori booleani* utilizzati sono:

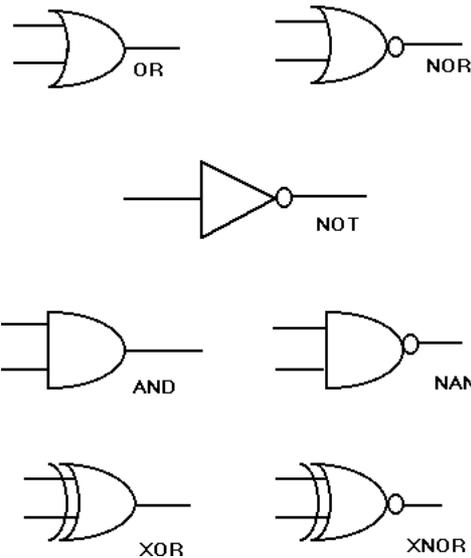
X	Y	X AND Y	X	Y	X OR Y	X	NOT X
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Le **porte logiche** utilizzate nel funzionamento dei circuiti digitali sono:



In figura (pagina successiva), sono riportati i simboli grafici usati per rappresentare i 7 tipi di porte logiche elementari. Il simbolo \oplus (XOR) é usato da alcuni come simbolo dell’OR.

N.B.: La pallina presente sulla porta logica indica negazione.



Due importanti teoremi dell'algebra booleana furono introdotti da **DeMorgan**:

$$\overline{(x + y)} = \bar{x} \cdot \bar{y}$$

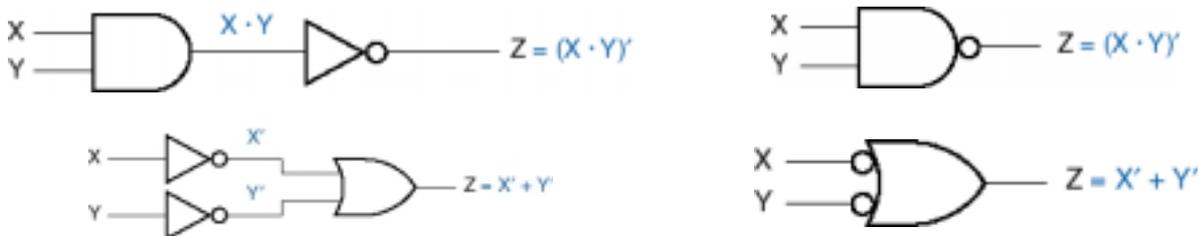
$$\overline{(x \cdot y)} = \bar{x} + \bar{y}$$

Grazie a questi teoremi è possibile ricavare il complemento di un'espressione booleana o parte di essa. Per ottenere il *complemento*, secondo questi due teoremi, si compiono i seguenti due passi:

1. i simboli di somma vengono sostituiti dai simboli di moltiplicazione e viceversa;
2. ognuno dei termini dell'espressione viene complementato.

I teoremi di DeMorgan esprimono il *principio di dualità* dell'algebra di Boole: ogni espressione booleana (ad esempio $x+y+z$) ha una sua duale ($x \cdot y \cdot z$).

A questo punto è possibile passare da una porta logica ad un'altra applicando DeMorgan:



Adesso vediamo come si passa dalla tabella della verità all'espressione Booleana; si studieranno quindi le **forme canoniche** congiuntive e disgiuntive. Consideriamo un circuito combinatorio con "n" variabili di ingresso (X_1, \dots, X_n). Denotiamo ogni singola variabile, sia in forma semplice (X_i) che complementata (\bar{X}_i) col nome di *letterale*. Ad ogni riga della tabella di verità, costituita da una sequenza di "n" valori booleani (b_{n-1}, \dots, b_0), associamo una sequenza di letterali nel seguente modo:

- se $b_i=0$, facciamo corrispondere a b_i il letterale \bar{X}_i ;
- se $b_i=1$, facciamo corrispondere a b_i il letterale X_i ;

il prodotto (AND) degli "n" letterali così ricavato si chiama *minterm*. Ad esempio, alla stringa "001" corrisponde il minterm $\bar{X}_2 \bar{X}_1 X_0$.

Viceversa, se ad ogni riga della tabella di verità associamo una sequenza di letterali nel seguente modo:

- se $b_i=0$, facciamo corrispondere a b_i il letterale X_i ;

- se $b_i=1$, facciamo corrispondere a b_i il letterale $\overline{X_i}$;

la somma (OR) degli “n” letterali così ottenuta si chiama *maxterm*. Ad esempio, alla stringa “001” corrisponde il maxterm $X_2+X_1+\overline{X_0}$.

Il prodotto (AND) dei maxterm corrispondenti alle righe della tabella della verità in cui la variabile booleana di uscita è “0” prende il nome di *prima forma canonica* o *forma canonica congiuntiva* della funzione booleana del circuito. La somma (OR) dei minterm corrispondenti alle righe della tabella in cui la variabile booleana di uscita è “1” prende il nome di *seconda forma canonica* o *forma canonica disgiuntiva* della funzione booleana del circuito.

Esempio: data la seguente tabella di verità

$X_2X_1X_0$	Y
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

ricaviamo i minterm per le righe “001”, “101” e “111”.

Da ultimare

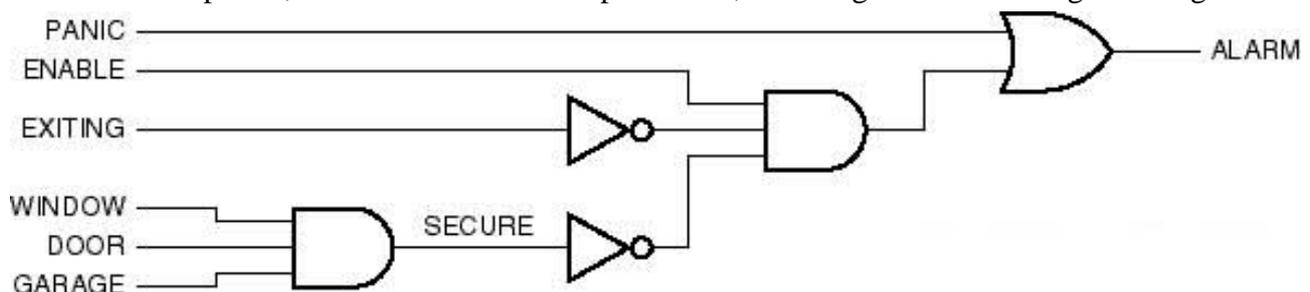
Capitolo 4: Sintesi dei circuiti combinatori

Ci si accinge adesso all'analisi dei circuiti logici e della molteplicità delle combinazioni possibili per ottenere, dati gli inputs, come output i valori della definita funzione booleana corrispondente.

La descrizione di un circuito logico combinatorio può avvenire in tre modi:

- 1) *esaustivo* (per mezzo cioè della sua tabella di verità);
- 2) *funzionale* (previo l'esplicitazione tramite l'associata funzione Booleana);
- 3) *per simulazione o prova* (seguendo ed attenendosi ai seguenti items):
 - I) scrittura della descrizione funzionale in *HDL* (Hardware Description Language);
 - II) definizione delle condizioni di prova;
 - III) test dei vettori per i casi limite;
 - IV) confronto degli outputs con quelli previsti con il metodo funzionale;
 - V) riconduzione del test dal punto III con valori d'ingresso casuali.

In altre parole, lasciando stare la terza possibilità, si immagini il circuito logico di seguito:



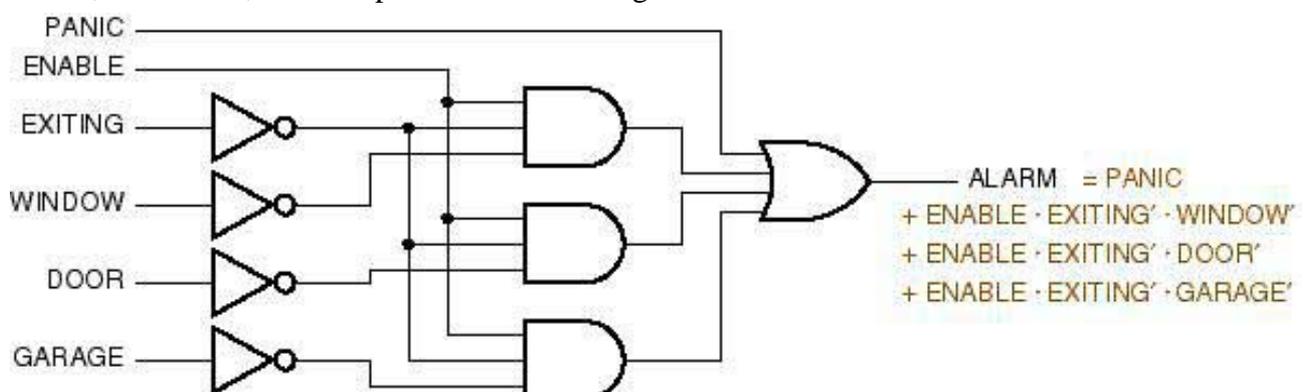
la cui funzione logica è:

$$\text{ALARM} = \text{PANIC} + \text{ENABLE} \cdot \text{EXITING}' \cdot (\text{WINDOW} \cdot \text{DOOR} \cdot \text{GARAGE})'$$

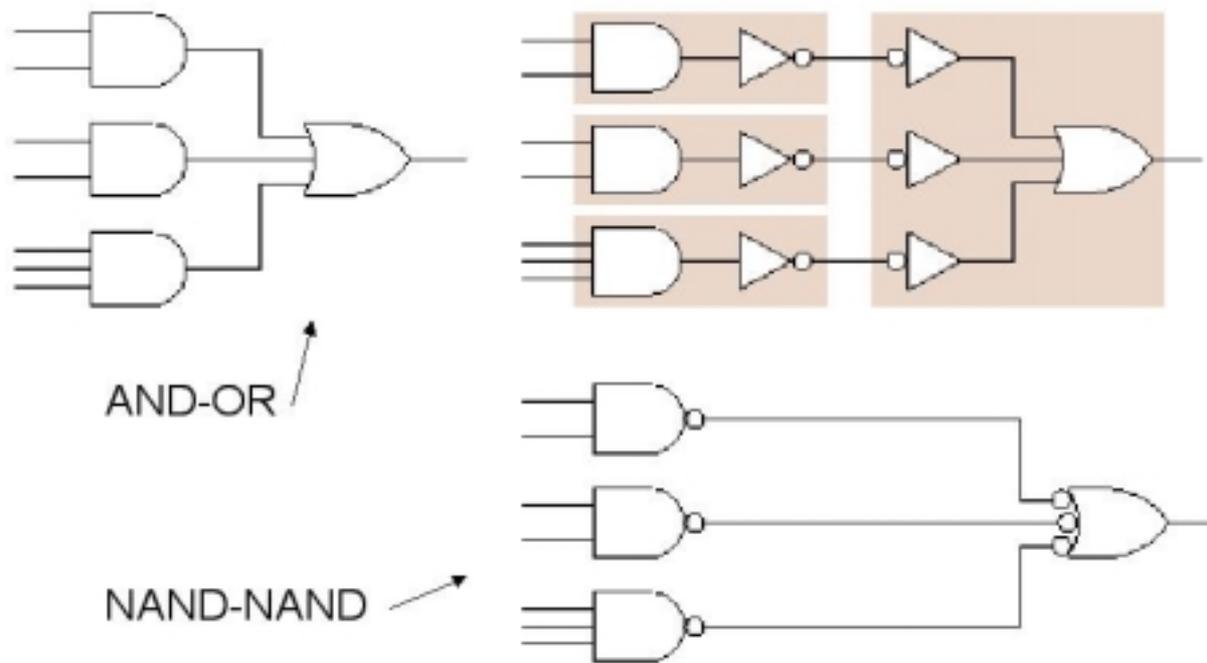
la quale, per la *distributivity property*, diventa:

$$\text{ALARM} = \text{PANIC} + \text{ENABLE} \cdot \text{EXITING}' \cdot \text{WINDOW}' + \text{ENABLE} \cdot \text{EXITING}' \cdot \text{DOOR}' + \text{ENABLE} \cdot \text{EXITING}' \cdot \text{GARAGE}'$$

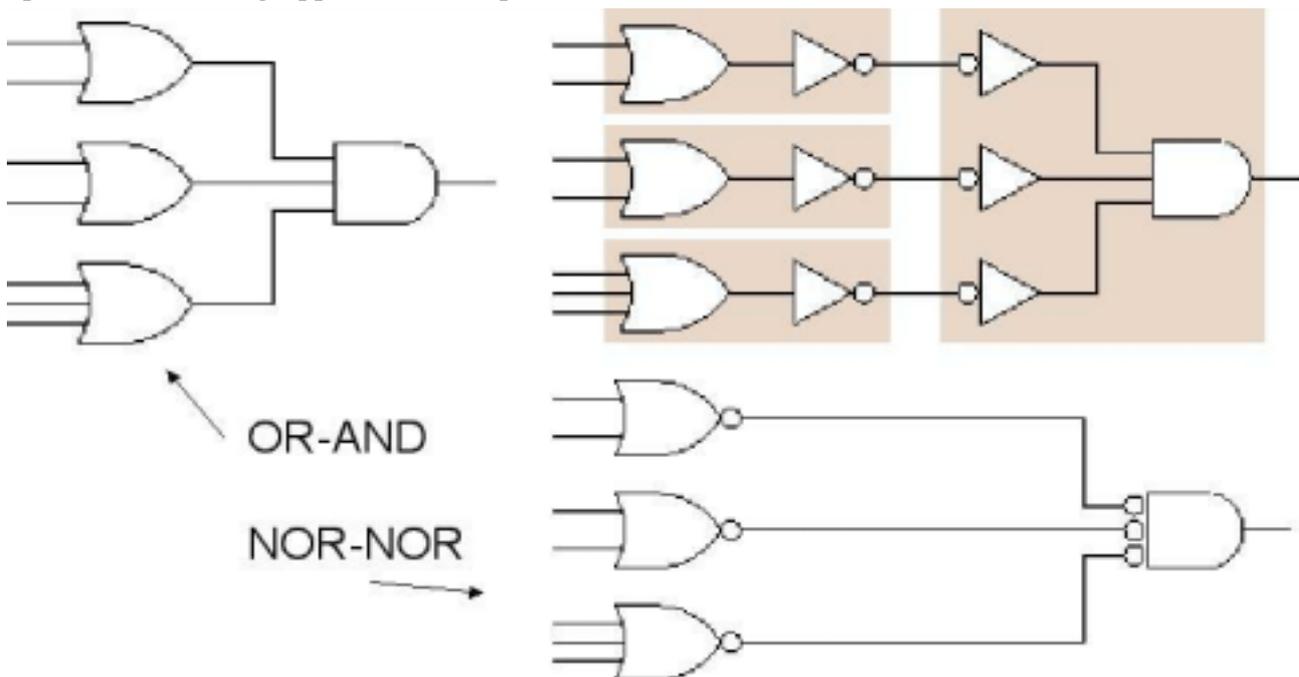
che ha, a sua volta, un corrispondente circuito logico:



La forma di quest'ultima funzione è detta *somma di prodotti* ed opera un "OR" a gruppi di elementi posti in "AND":



Da questa forma, come facilmente intuibile, esiste anche il duale detto *prodotto di somme*, che opera un “AND” a gruppi di elementi posti in “OR”:



N.B.: quando una funzione è descritta come somma di tutti i prodotti possibili si dice espressa nella prima forma canonica:

$$\sum_{i=0}^{2^n-1} (\alpha_i \cdot P_i) \text{ con } \begin{cases} n = \text{numero delle variabili;} \\ i = \text{indice del minterm;} \\ \alpha_i = \text{presenza o meno del minterm;} \\ P_i = \text{prodotto standard (minterm);} \end{cases}$$

mentre se è descritta come prodotto di tutte le somme possibili è detta espressa in **seconda forma canonica**:

$$\prod_{i=0}^{2^n-1} (\beta_i + S_i) \text{ con } \begin{cases} n = \text{numero delle variabili;} \\ i = \text{indice del maxterm;} \\ \beta_i = \text{presenza o meno del maxterm;} \\ S_i = \text{somma standard (maxterm);} \end{cases}$$

Passiamo ora alla **progettazione** dei circuiti logici combinatori. Per far ciò si costruisce la tabella della verità, e si vede quindi come gli “n-bit” vengono mappati nella singola uscita. Questa è la prima fase di ogni metodo utilizzabile, di cui si fornisce di seguito una panoramica ordinata in base al numero di variabili utilizzate nel computo.

Volendo implementare un circuito che verifichi se un numero è primo o meno, si ha che:

Riga	N_3	N_2	N_1	N_0	F
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	0	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

► Progettazione Brute-force (utilizzata didatticamente)

Fasi successive alla prima:

2) generazione della funzione nella prima forma canonica:

individuazione dell'*on-set* sotto forma di $\sum N_{n-1} \cdot N_{n-2} \cdot N_{n-3} \cdot \dots \cdot N_1 \cdot N_0$:

i	N_3	N_2	N_1	N_0	α	ON	P
0	0	0	0	0	0	N	N'_3 N'_2 N'_1 N'_0
1	0	0	0	1	1	Y	N'_3 N'_2 N'_1 N_0
2	0	0	1	0	1	Y	N'_3 N'_2 N_1 N'_0
3	0	0	1	1	1	Y	N'_3 N'_2 N_1 N_0
4	0	1	0	0	0	N	N'_3 N_2 N'_1 N'_0
5	0	1	0	1	1	Y	N'_3 N_2 N'_1 N_0
6	0	1	1	0	0	N	N_3 N'_2 N'_1 N'_0
7	0	1	1	1	1	Y	N'_3 N_2 N_1 N_0
8	1	0	0	0	0	N	N_3 N'_2 N'_1 N'_0
9	1	0	0	1	0	N	N_3 N'_2 N'_1 N_0
10	1	0	1	0	0	N	N_3 N'_2 N_1 N'_0
11	0	0	1	1	1	Y	N'_3 N'_2 N_1 N_0
12	1	1	0	0	0	N	N_3 N_2 N'_1 N'_0
13	1	1	0	1	1	Y	N_3 N_2 N'_1 N_0
14	1	1	1	0	0	N	N_3 N_2 N_1 N'_0
15	1	1	1	1	0	N	N_3 N_2 N_1 N_0

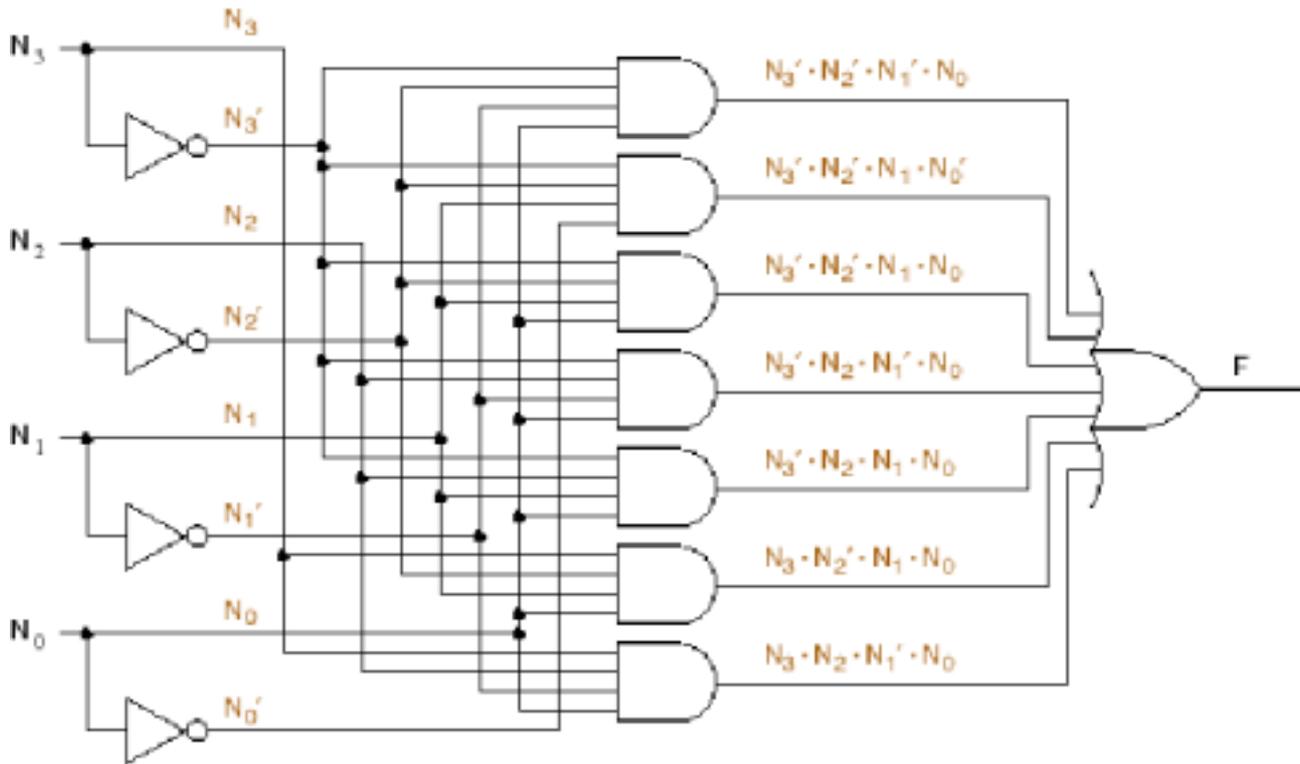
e l'*on-set* è: {1,2,3,5,7,11,13}

$$F = 0 \cdot N_3' \cdot N_2' \cdot N_1' \cdot N_0' + 1 \cdot N_3' \cdot N_2' \cdot N_1' \cdot N_0 + 1 \cdot N_3' \cdot N_2' \cdot N_1 \cdot N_0' + 1 \cdot N_3' \cdot N_2' \cdot N_1 \cdot N_0 + 0 \cdot N_3' \cdot N_2 \cdot N_1' \cdot N_0' + 1 \cdot N_3' \cdot N_2 \cdot N_1' \cdot N_0 + 0 \cdot N_3 \cdot N_2' \cdot N_1' \cdot N_0' + 1 \cdot N_3 \cdot N_2' \cdot N_1' \cdot N_0 + 0 \cdot N_3 \cdot N_2' \cdot N_1 \cdot N_0' + 0 \cdot N_3 \cdot N_2' \cdot N_1 \cdot N_0 + 0 \cdot N_3 \cdot N_2 \cdot N_1' \cdot N_0' + 1 \cdot N_3 \cdot N_2 \cdot N_1' \cdot N_0 + 0 \cdot N_3 \cdot N_2 \cdot N_1 \cdot N_0' + 1 \cdot N_3 \cdot N_2 \cdot N_1 \cdot N_0$$

cioè:

$$\begin{aligned} F &= \Sigma_{N_3, N_2, N_1, N_0} (1, 2, 3, 5, 7, 11, 13) \\ &= N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0' + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2 \cdot N_1' \cdot N_0 \\ &\quad + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + N_3 \cdot N_2' \cdot N_1' \cdot N_0 + N_3 \cdot N_2 \cdot N_1' \cdot N_0 \end{aligned}$$

L funzione canonica così ottenuta ha però un corrispondente circuitale molto complesso:



che necessita di essere semplificato al fine di evitare sprechi nell'utilizzo della porte logiche e, di conseguenza, contenere i costi. Si procede quindi verso il prossimo step;

- 3) riduzione della funzione a mezzo di teoremi e proprietà:
grazie al teorema T8 (che esprime la proprietà distributiva) che recita:

$$X \cdot Y + X \cdot Y' = X$$

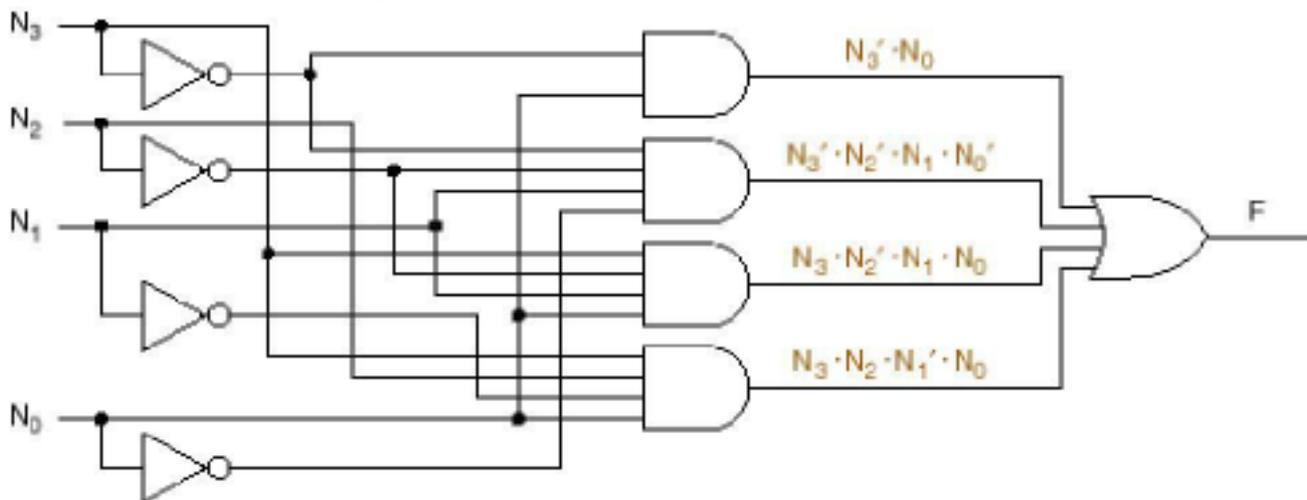
la funzione precedentemente trovata si può ridurre in:

$$\begin{aligned} F &= \Sigma_{N_3, N_2, N_1, N_0} (1, 3, 5, 7, 2, 11, 13) \\ &= N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0' + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2 \cdot N_1' \cdot N_0 + \dots \\ &= (N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0) + (\cdot N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0) + \dots \\ &= N_3' \cdot N_2' \cdot N_0 + N_3' \cdot N_2 \cdot N_0 + \dots \end{aligned}$$

fino ad ottenere:

$$F = N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0' + N_3 \cdot N_2' \cdot N_1 \cdot N_0 + N_3 \cdot N_2 \cdot N_1' \cdot N_0$$

il cui circuito corrispondente è:



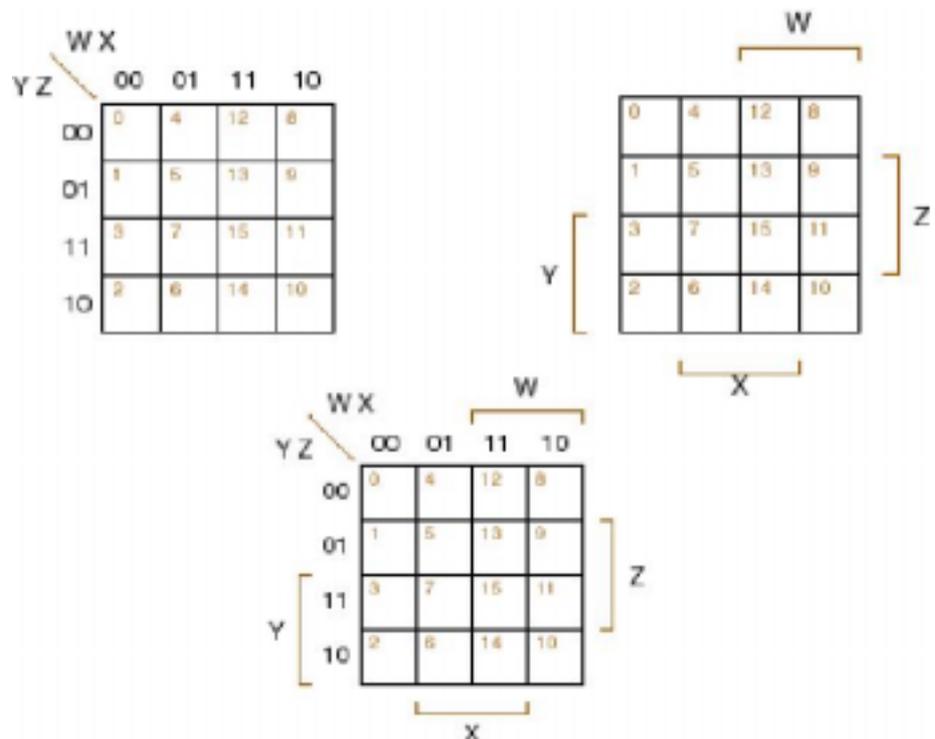
che, oltre ad essere molto semplice ha, come si vedrà più avanti, anche la capacità di essere di facile implementazione con matrici di porte logiche programmabili.

► Progettazione minimale con mappe di Karnaugh (utilizzabile validamente fino a 6 variabili)

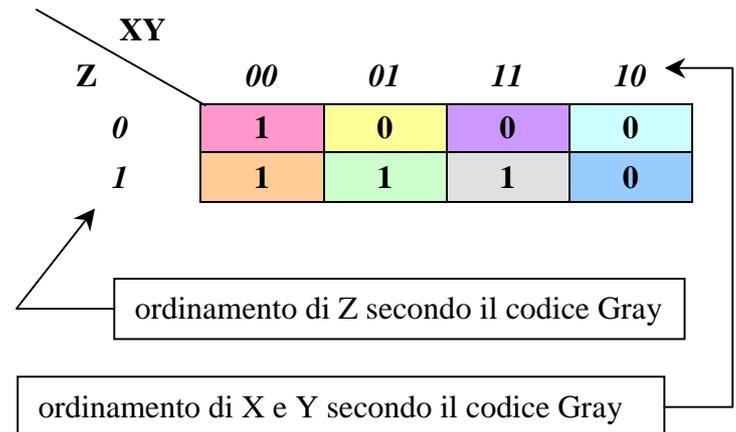
Questo metodo si avvale delle *mappe di Karnaugh*, alle quali bisogna accennare prima di proseguire. Una mappa di questo tipo è costituita da righe e colonne ordinate secondo il *codice Gray*, per il quale due cifre sono adiacenti solo se si differenziano di un solo bit. Sugli assi di righe e colonne possono essere alloggiati fino ad un massimo di due variabili ciascuno, ciò significa che su una singola mappa di Karnaugh si possono rappresentare fino a 4 variabili. Per superare questo limite si raddoppiano le mappe per ogni variabile in più delle 4 alloggiabili; si deduce quindi che per minimizzare una funzione di 6 variabili sono necessarie ben 8 mappe, sulle quali controllare le adiacenze diventa veramente una impresa difficile se pur non impossibile.

Come appena anticipato, il metodo consiste nel trovare il “minimo numero di più grandi adiacenze” e poi “incravattarle” (se le adiacenze sono a loro volta adiacenti). Quella che definiamo *adiacenza* è un sottoinsieme dell’on-set della tabella della verità che, opportunamente trascritto sulla mappa, risulti su essa raggruppato in forma rettangolare o quadrata e contenente un numero di elementi rappresentabili come potenze di 2.

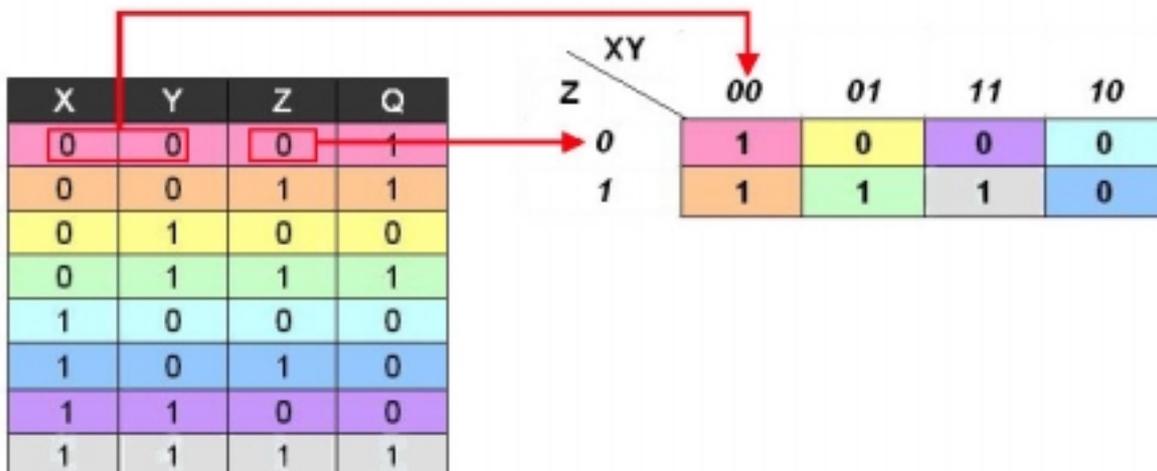
Il sistema migliore per capire come i dati di una tabella della verità possano passare sulla mappa di Karnaugh è quello di fare un **esempio** e commentarlo:



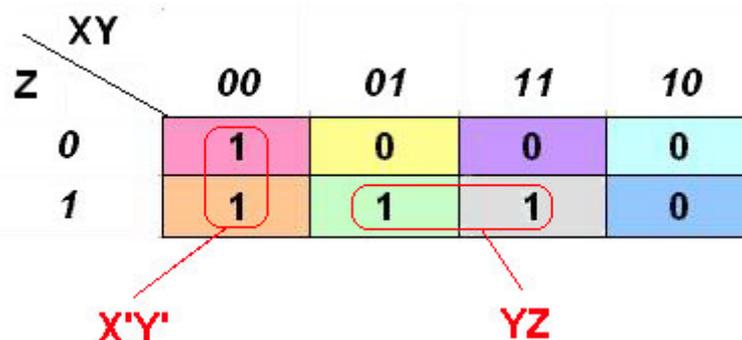
X	Y	Z	Q
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Il valore di ogni “Q” va posto nella tabella di Karnaugh nella cella che corrisponde agli appositi valori delle variabili X,Y e Z come indicato nell’illustrazione sotto. Per maggior chiarezza, ogni riga della tabella della verità ha lo stesso colore della cella della mappa di Karnaugh che gli corrisponde:



Il passo successivo è l’individuazione delle adiacenze, dette anche *sottocubi*, e l’eventuale incravattamento dei sottocubi vicini. Si cominci con il ricercare i sottocubi:



Una volta individuati i sottocubi (in questo caso due), da ognuno di essi bisogna estrarre un prodotto semplificato; più precisamente vanno mantenute le variabili presenti in tutte le caselle del sottocubo con il medesimo valore (“X” e “Y” sono sempre “0” nel sottocubo di sinistra) e scartare le variabili che all’interno del sottocubo cambiano valore (“Z” varia da “0” a “1”). Inoltre, se il valore delle variabili non varianti all’interno del sottocubo è discorde con quello di “Q” (cioè 1) allora la variabile va indicata ma negata.

Seguiamo quanto accade nell’esempio:

a) sottocubo di sinistra:

“X” è costantemente “0”, sia nella casella sopra che in quella sotto, ma essendo discorde con “Q”(=1) va indicata come negata, quindi X’;
 per “Y” vale lo stesso ragionamento fatto per “X”, quindi va aggiunto anche Y’;
 “Z”, invece, nella casella sopra ha valore “0” ed in quella sotto valore “1”, quindi cambia valore all’interno del sottocubo ed in questo caso non va indicata;

b) sottocubo di destra:

“X” nella casella di sinistra ha valore “0” e a destra valore “1”, essendo variante non va indicata;

“Y” è “1” sia nella casella di sinistra che in quella di destra ed è concorde con “Q”, quindi va indicata senza negazione: Y;

“Z” è anch’esso “1” in entrambe le caselle, quindi anch’esso concorde e va indicato: Z;

Ma il lavoro non è ancora terminato; adesso bisogna trovare, se possibile, gli *incravattamenti*, che si hanno in caso di sottocubi adiacenti. Nel nostro caso bisogna cercare un sottocubo che usi elementi del primo e del secondo sottocubo:

XY		Z			
		00	01	11	10
Z	0	1	0	0	0
	1	1	1	1	0

X'Y'
X'Z
YZ

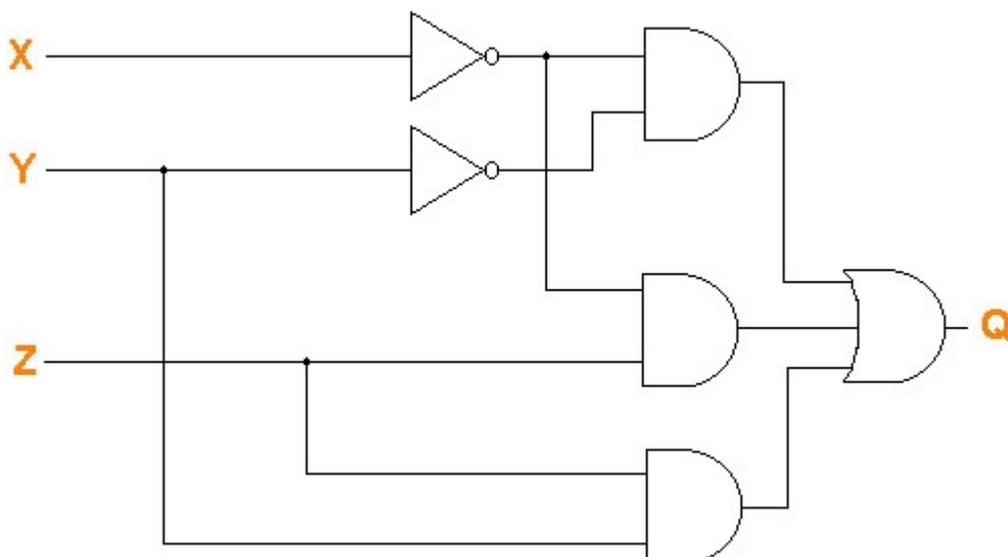
Tale occorrenza è stata individuata come mostrato e, seguendo ragionamenti fin qui seguiti, se ne può facilmente estrarre la funzione:

$$X'Z$$

A questo punto la funzione complessiva non sarà altro che la somma dei prodotti ottenuti:

$$F = X'Y' + X'Z + YZ$$

che ha come circuito logico il seguente:

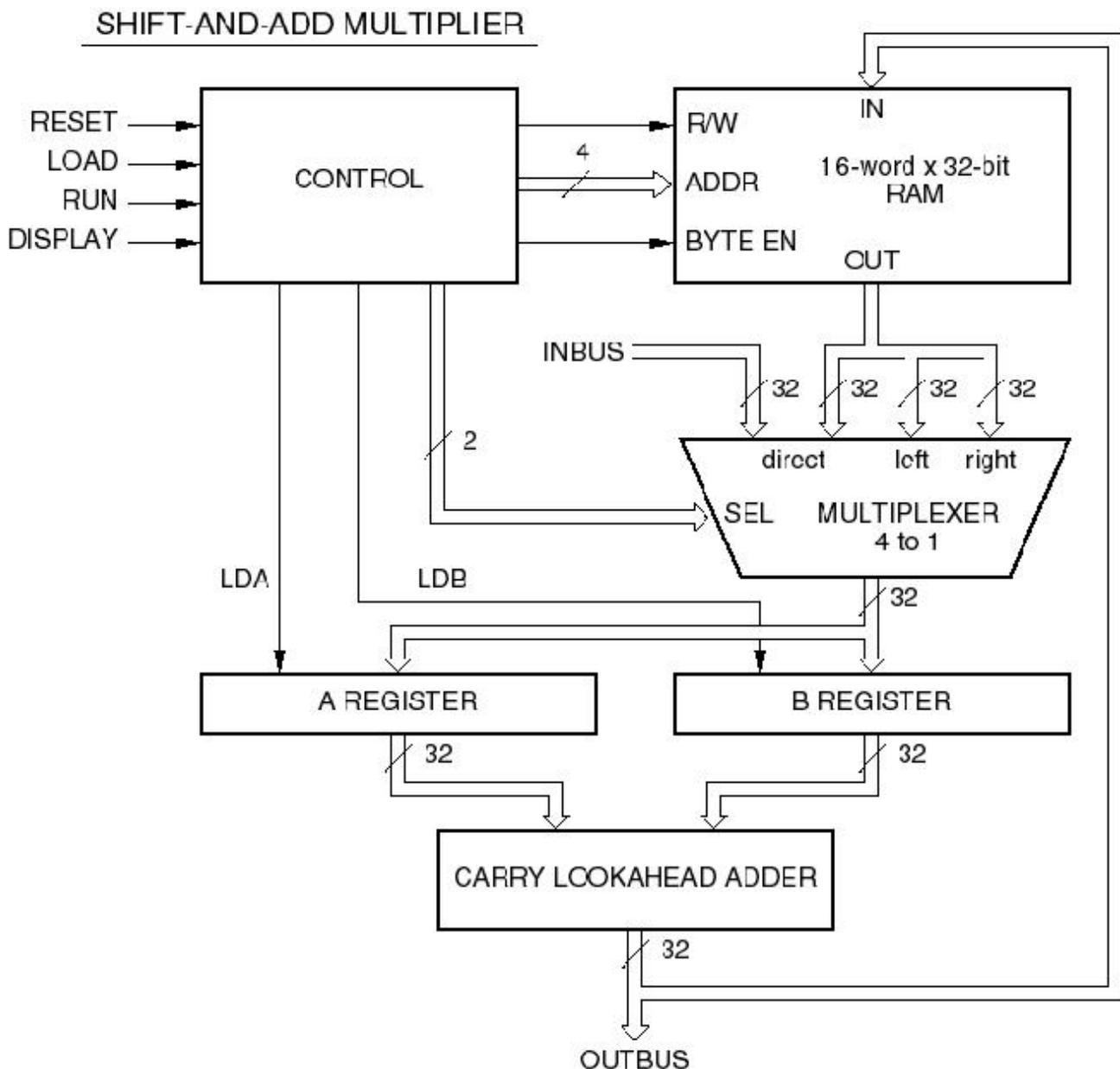


Perché l’incravattamento? L’incravattamento è una pratica necessaria al fine di eliminare i rischi di malfunzionamenti: le mappe di Karnaugh individuano degli *ipercubi* che hanno per vertici le configurazioni di input del circuito logico. Con la scelta dei sottocubi si opera la

selezione di un sottoinsieme di tutte le possibili funzioni e si potrebbe verificare il caso che per transitare tra due configurazioni adiacenti, ma appartenenti a due sottocubi diversi, si possa avere una momentanea anomalia di output. In altre parole, passando da un sottocubo ad un altro c'è il rischio, detto appunto *alea*, che l'output assuma, durante il passaggio, un valore indesiderato. Se questo conta molto poco per le transizioni di output di tipo "α" (1→0) e "β" (0→1), ha invece un'importanza vitale nelle pseudo-transizioni "0" (0→0) e "1" (1→1), durante le quali si potrebbe verificare un *glitch*, cioè una momentanea inversione dell'output, la quale è assolutamente imprevedibile essendo un effetto elettrico e non logico. A maggior puntualizzazione si fa presente che le funzioni logiche senza l'incravattamento sono perfettamente valide, ma nell'applicazione a sistemi elettronici è fortemente consigliabile evitare le alee, in modo da avere un circuito sicuramente più stabile ed indubbiamente funzionante.

Capitolo 5: Documentazione standard, PLD e Decoders

Il trasporto è l'intelleggibilità degli schemi digitali presuppone una standardizzazione del formato grafico di rappresentazione. L'idea è quella di generare un piano di facile lettura che rappresenti, su un certo *livello di dettaglio*, i diversi blocchi costituenti un circuito logico completo. Come avviene per le carte topografiche, per i circuiti digitali complessi esiste una corografia (consistente del *diagramma schematico a blocchi*) in cui vengono delineati, con opportuna simbologia, i blocchi funzionali del sistema. Ogni blocco può essere a sua volta esploso in un elaborato a se stante che, via via, ne aumenti il dettaglio e quindi approssimi la rappresentazione al livello elettrico passando per quelli funzionale e logico. Un diagramma schematico a blocchi ha la seguente forma:

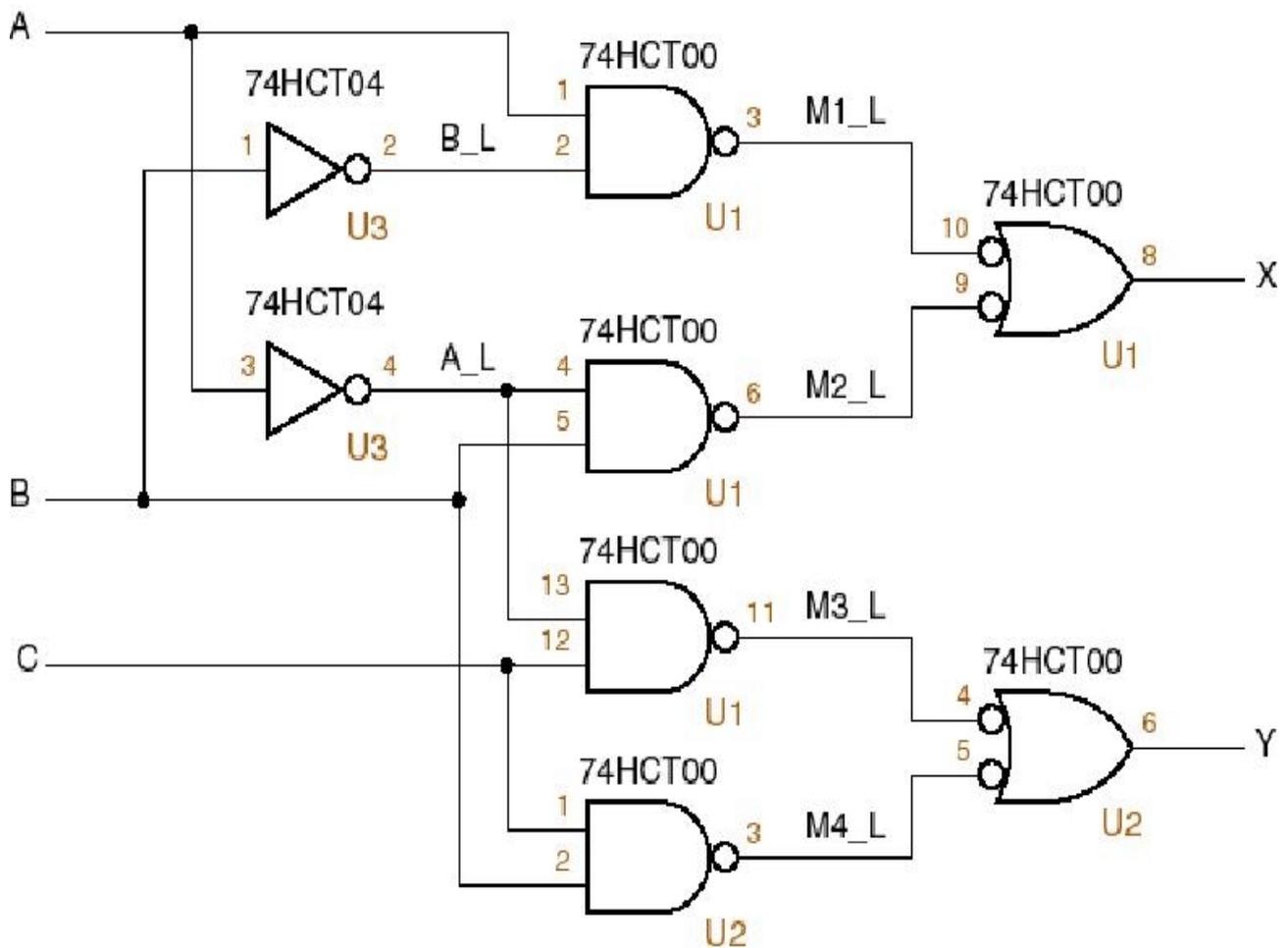


e deve constare delle seguenti indicazioni:

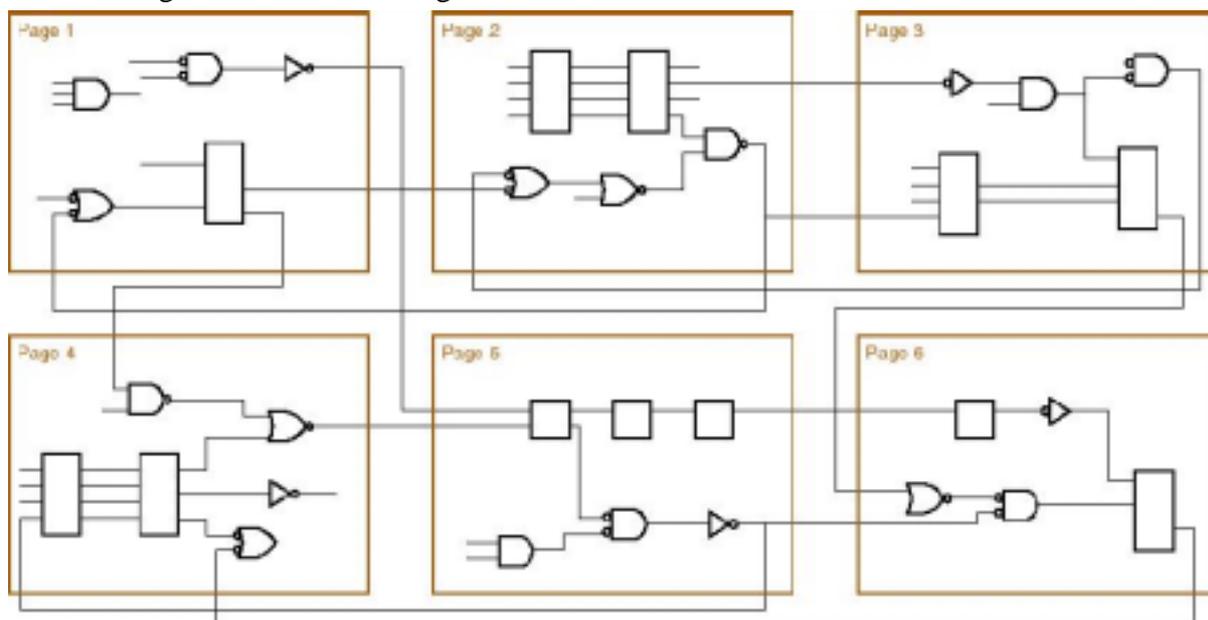
- dettaglio dei componenti di input, output ed interconnessioni;
- delineatori di riferimento;
- piedini (pins) con relativo numero e funzione;

- titolo o nome dei blocchi;
- nome di tutti i segnali digitali;
- connessioni verso altre parti dello schema site in altre pagine dell'elaborato.

Come già detto, ogni blocco può avere un maggiore a mezzo dello schema del blocco:

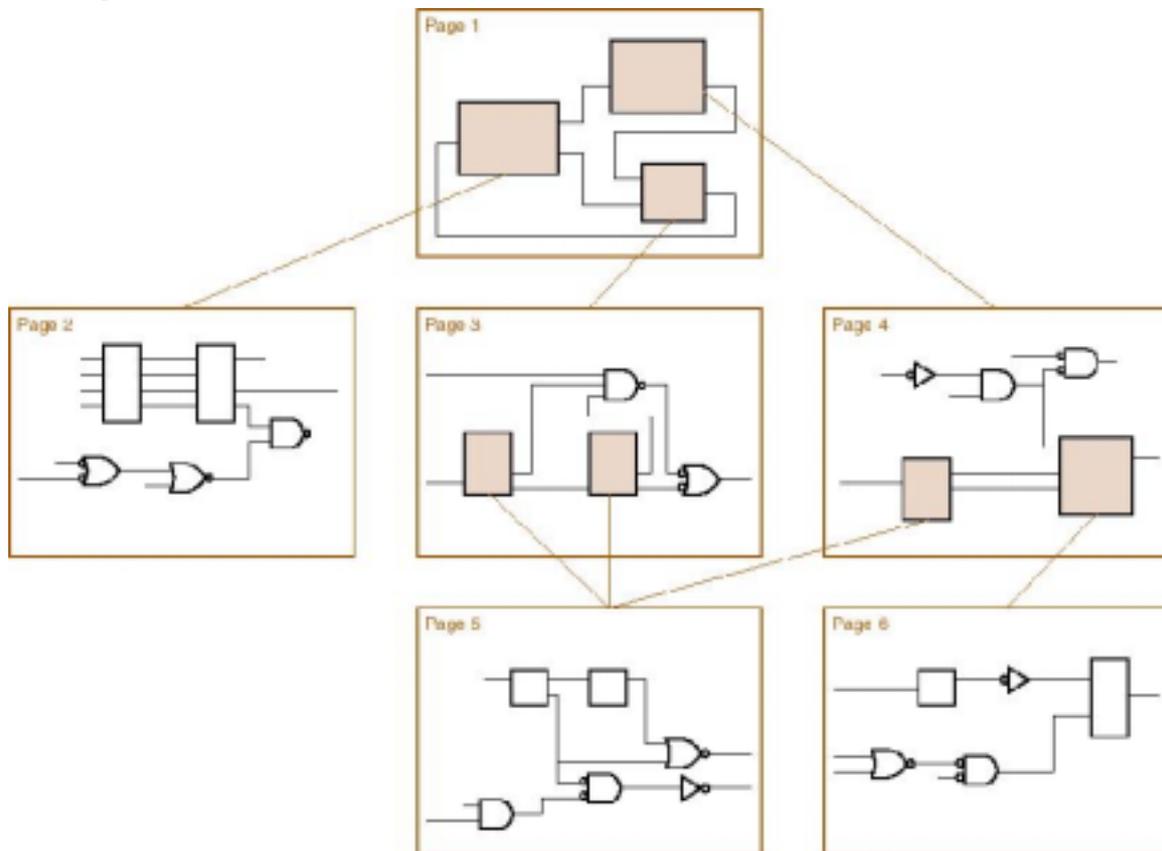


il quale deve riportare anche la specificazione di ogni circuito integrato (o parte di esso) e rispettivi piedini di collegamento utilizzati negli items dello schema.



Nel caso il circuito risulti particolarmente complesso e composto da innumerevoli blocchi si può

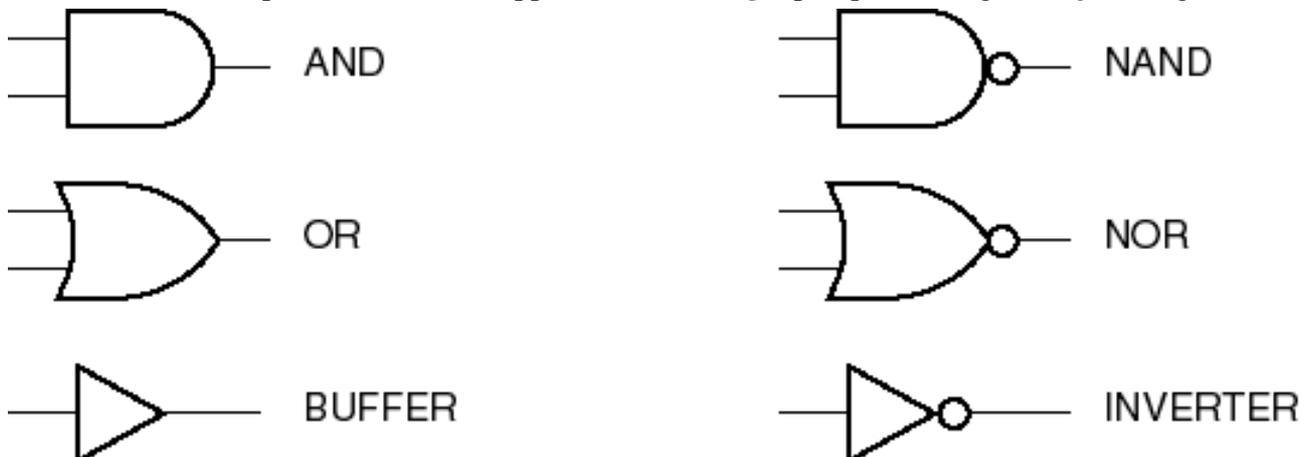
rendere necessario un ulteriore livello, una corografia ad ancor minor dettaglio, che rappresenti un indice per districarsi tra le notevoli pagine dell'elaborato. In questo caso vengono usati i *Flat Schematic Structure Diagram* (FSSD) che rinvii alla tavola dell'elaborato corrispondente allo schema o al sottoblocco cercato, e gli *Hierarchical Schematic Structure* (HSS) che danno una indicazione gerarchica del blocco cercato:



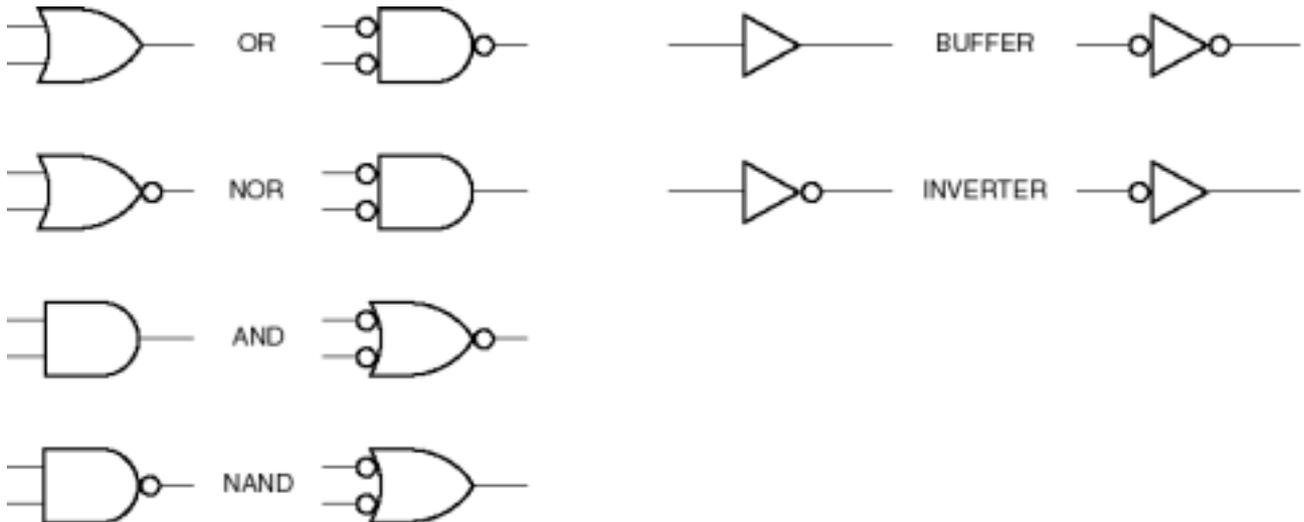
Oltre agli schemi, però, viene sovente diffusa altra documentazione riguardante:

- *diagrammi temporali*
 - ✓ output dei simulatori;
 - ✓ strumenti specifici di compilazione dei diagrammi temporali;
- *descrizioni circuitali*
 - ✓ testi;
 - ✓ E102E;
 - ✓ possono risultare grandi come libri (come la descrizione del Cisco ASICS);
 - ✓ solitamente incorporano altri elementi (diagrammi a blocchi, diagrammi temporali, ecc.);

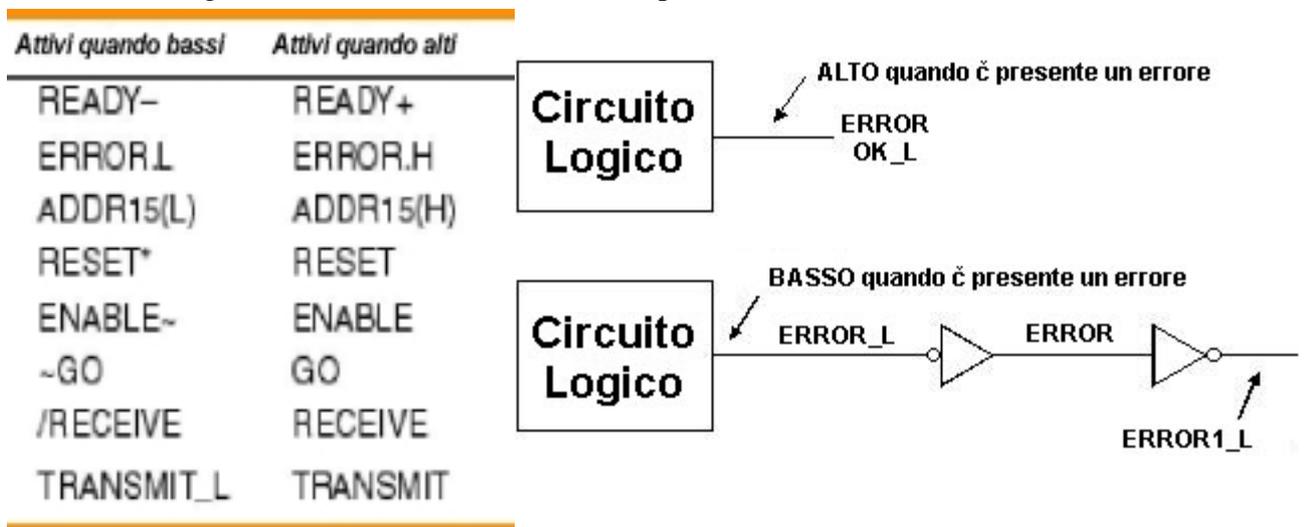
Lo standard prevede anche un'opportuna **simbologia** per quel che riguarda *gates* digitali:



compresi, ovviamente, i *simboli equivalenti di DeMorgan*:



e quelli per i segnali digitali dai quali dipende la scelta dei simboli di rappresentazione delle gates da utilizzare negli schemi, come visibile nell'esempio (a destra):

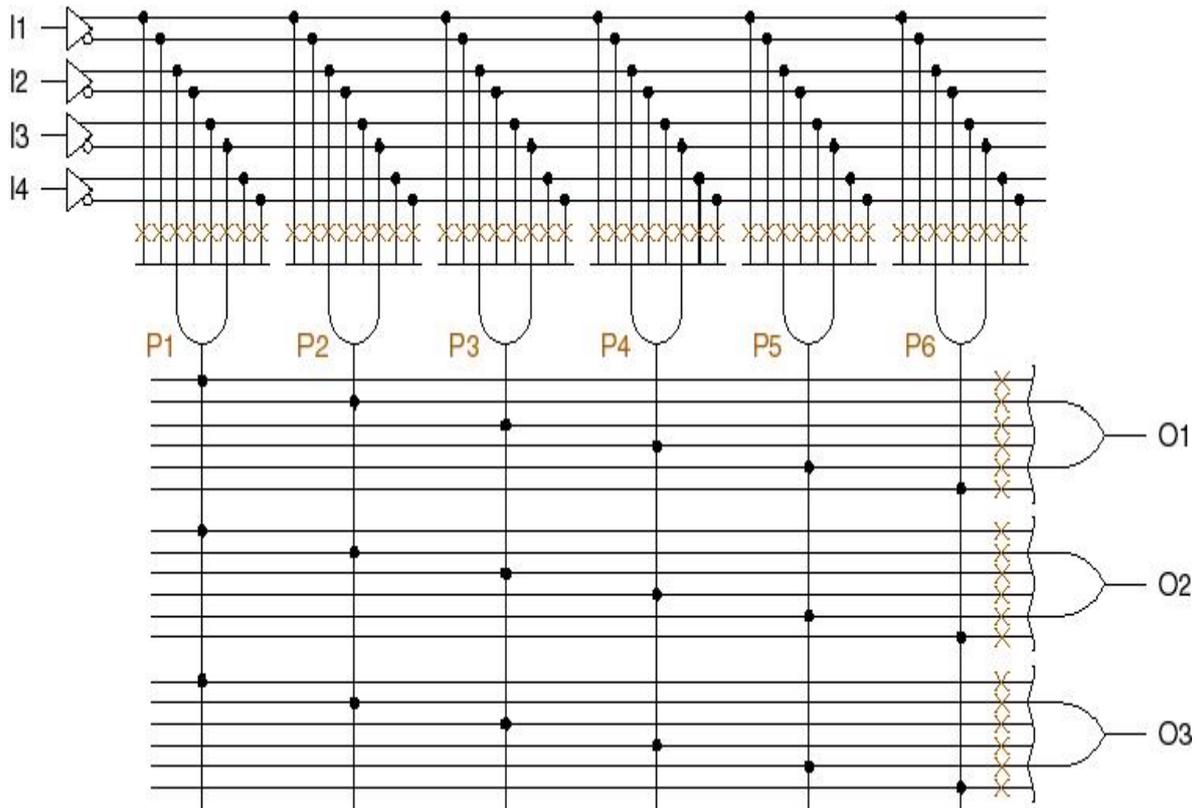


Nel IV capitolo si è visto come qualsiasi funzione logica possa essere sintetizzata usando prodotti di somme o, ancor meglio, somme di prodotti logici. Sfruttando questa caratteristica sono stati messi a punto **dispositivi programmabili** (utili in fase di prototipaggio, test e piccolissima produzione) in grado di poter riprodurre qualunque funzione logica di opportuna complessità. La programmazione avviene inviando un apposito *bitstream*, generato dal software per computer tipo compilatori HDL o digital CAD/digital workbench, i quali approntano sul dispositivo una *fusemap*. Questi dispositivi sono detti *PLDs*, acronimo di Programmable logic devices, ed a questa famiglia appartengono le classi di:

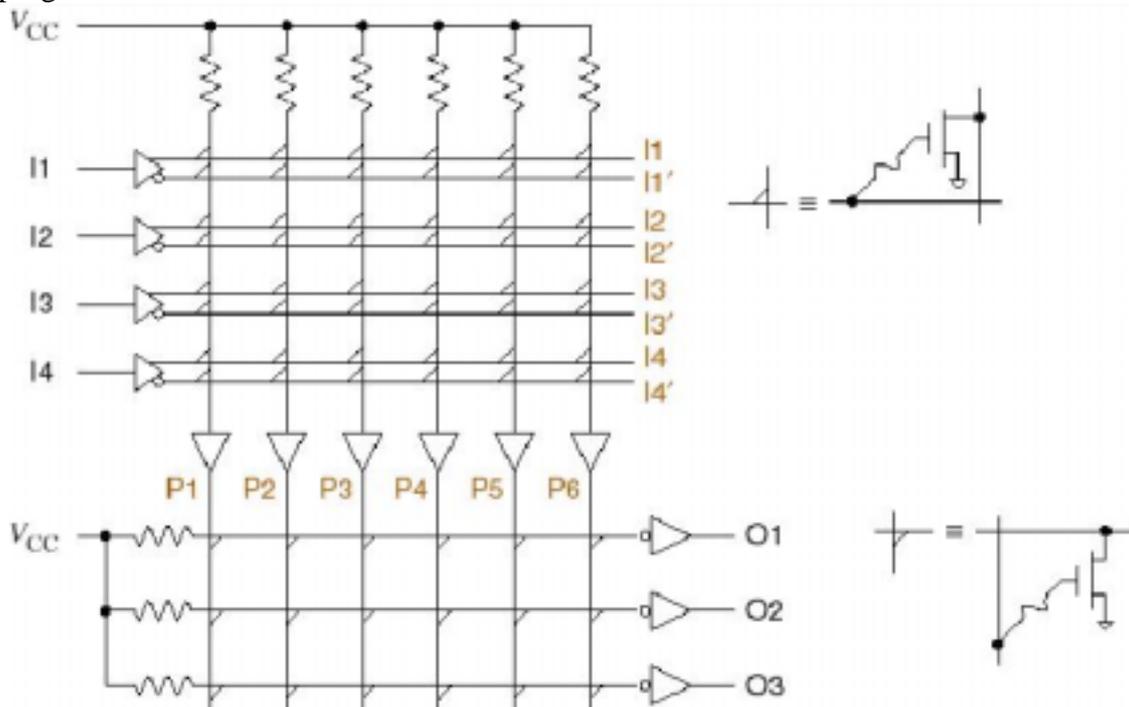
- PLAs
- PALs
- FPGA
- CPLD

Ecco una rapida carrellata sui componenti di questa famiglia.

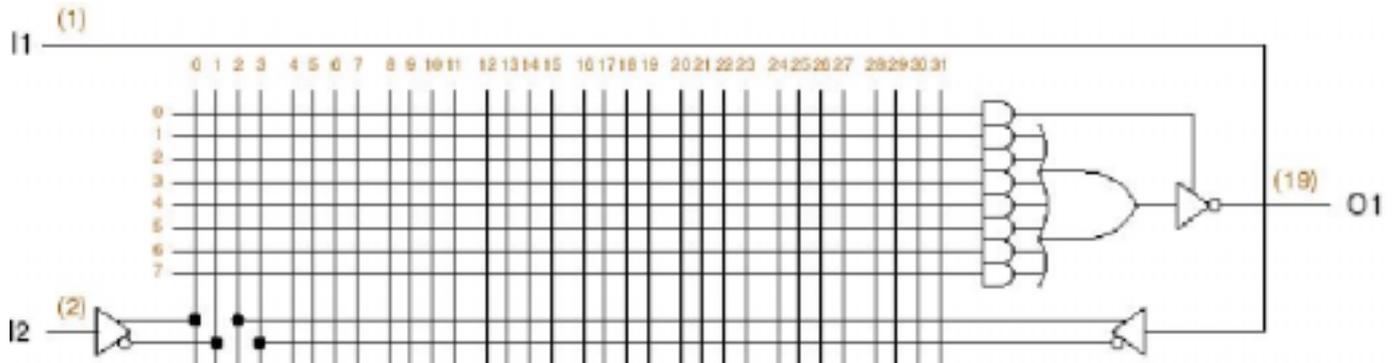
PLAs, acronimo di Programmable Logic Arrays, è costituito da abbondanti matrici di porte AND-OR. Chiamando “ n ” il numero degli ingressi, “ m ” è il numero delle uscite pilotate da altrettante OR in grado, ognuna, di accettare “ n ” ingressi. Ad ogni AND gate sono applicati tutti gli “ n ” ingressi e i relativi “ n ” complementari (in tutto $2n$) attraverso dei *fuses* in grado di essere interrotti in fase di programmazione. Successivamente le uscite delle AND sono collegate, sempre tramite *fuses*, alle porte OR. In tale modo è possibile programmare sia quali ingressi debbano incidere su ogni AND gate sia quali prodotti standard debbano incidere su ogni OR gate. Grazie a questa peculiarità questi devices sono molto flessibili.



In maggior dettaglio si possono vedere nelle seguente figura la postura ed il funzionamento dei *fuses* programmabili:



PALs (© by AMD), acronimo di Programmable Arrays Logic, sono concettualmente simili ai *PLAs* ma hanno però l'uscita della matrice AND connesso direttamente all'OR. La programmazione agisce quindi soltanto all'ingresso delle AND, mentre la OR è fissa. Peculiarità è quella di disporre della possibilità di riportare in ingresso alcune uscite. La flessibilità di questi dispositivi è ridotta rispetto a quella dei *PLAs*.

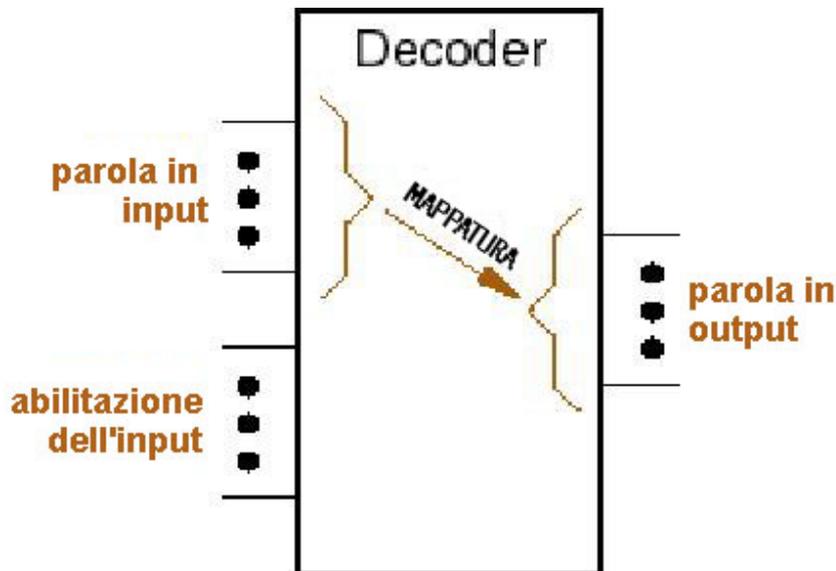


FPGAs, acronimo di Field Programmable Gates Arrays, sono dispositivi di prototipaggio che dispongono di una consistente quantità di diversi tipi di porte logiche, le cui interconnessioni possono essere programmate, tramite un bitstream, generato analogamente ai casi relativi ai *PLAs* e ai *PALs*. La flessibilità è massima ma il costo è molto elevato. Possono essere programmati svariate volte, e ciò fa di essi un valido strumento di sviluppo e perfezionamento per la messa a punto dei circuiti logici per i quali si intende avviare una produzione di massa.

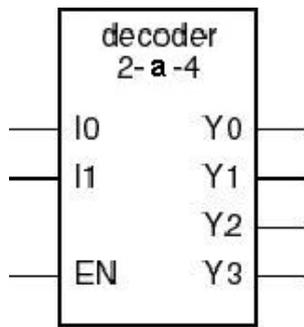
CPLDs, acronimo di Complex Programmable Logic Devices.

Ci si occuperà adesso dei circuiti logici combinatori che hanno funzione di instradare i segnali e modificarne la codifica. Questa categoria è detta **steering logic** e comprende decoders, encoders, multiplexer e demultiplexer.

La funzione del **decoder** è quella di trasformare una parola, codificata secondo una determinata *collating sequenze*, in un'altra, codificata secondo una *collating sequenze* differente.



Tipicamente un decoder ha “n” ingressi e “2ⁿ” uscite e dispone di una sezione di abilitazione.

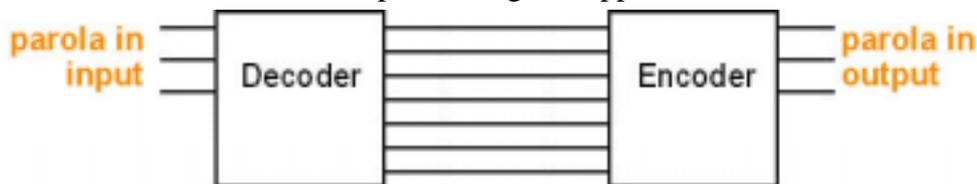


Inputs			Outputs			
EN	I1	I0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

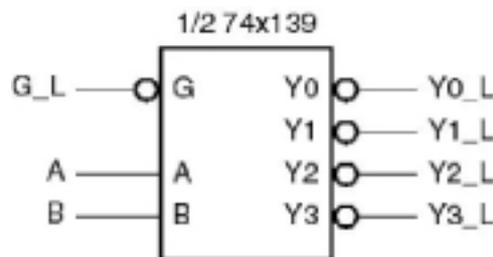
Nota: "x" è la notazione per i termini *don't care*.

L'**encoder** svolge una funzione complementare, ossia svolge il cambio di codifica ma, viceversa, dispone generalmente di "2ⁿ" ingressi e "n" uscite.

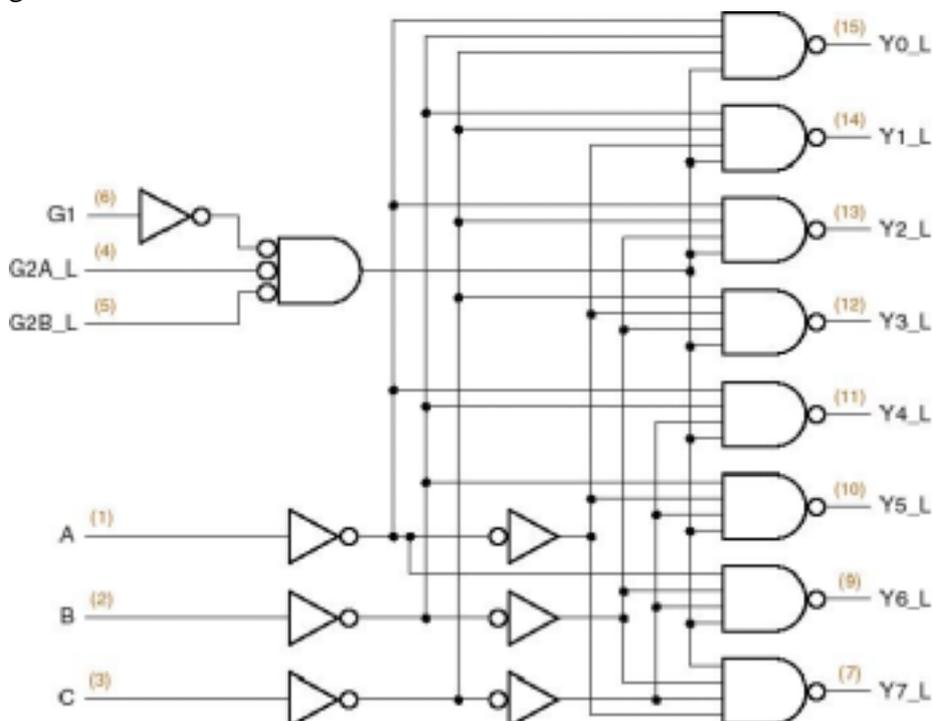
Una parola, passando prima per un decoder e poi per un encoder, subisce una *transcodifica* e l'oggetto ottenuto dall'unione dei due dispositivi logici è appunto un *transcoder*:



Di seguito è riportato un esempio di corretta notazione grafica relativa ai decoders:



e lo schema logico relativo ad un decoder 3 a 8:



Da ultimare

Traduzione delle slides di Wakerly - DDPP

EE121 Capitoli 1-5

Università di Palermo

D.U. in Informatica

Docente: Prof. Ivan Angelo

Autori: Dadà & Diemme

e-mail: dadadiemmegigio@supereva.it

URL: <http://web.ddpp.com/ee121/>

URL: <http://utenti.tripod.it/dada/>