Appunti sulle Reti Combinatorie

versione per studenti

Giovanni Stea

Ultima modifica: 08/10/2026

Sommario

1 Le reti logiche come modello astratto di sistemi fisici 5
1.1 Limiti del modello 7
1.1.1 Transizione dei segnali 8
1.1.2 Contemporaneità 9
2 Generalità sulle Reti Combinatorie 11
2.1.1 Tempo di attraversamento 12
2.2 Modalità di descrizione delle reti combinatorie 12
2.2.1 Esempio 13
2.3 Reti combinatorie elementari 14
2.3.1 Reti a zero ingressi 14
2.3.2 Reti ad un ingresso 15
2.3.3 Reti a due ingressi 16
2.4 And ed Or a più ingressi 17
2.4.1 Esempio: OR ad 8 ingressi 18
2.4.2 Esercizi (per casa) 19
3 Algebra di Boole 20
3.1 Proprietà degli operatori booleani 21
3.1.1 Teoremi di De Morgan per N variabili logiche 22
3.1.2 Equivalenza tra espressioni dell'algebra di Boole e reti combinatorie 23
3.2 Esercizi (per casa) 24
4 Reti combinatorie significative 25
4.1 Decoder 25
4.1.1 Esempio: Decoder 2 to 4 25
4.1.2 Esempio: Realizzazione di decoder 1 to 2 26
4.2 Decoder con enabler (espandibile) 27
4.2.1 Esempio: costruzione di un decoder 4 to 16 da decoder 2 to 4 28
4.3 Demultiplexer 30
4.4 Multiplexer 31
4.4.1 Il multiplexer come rete combinatoria universale 32
4.4.2 Esercizio: realizzazione di una RC ad N ingressi con un MUX ad N-1 variabili di comando 33
5 Modello strutturale universale per reti combinatorie 36
5.1.1 Esercizio (per casa) 39

6 Sintesi di reti in forma SP a costo minimo 40
6.1 Alcune note aggiuntive 44
6.1.1 Notazione compatta 44
6.1.2 Metodo di Quine-McCluskey 44
6.2 Mappe di Karnaugh 45
6.2.1 Algoritmo di ricerca dei sottocubi principali mediante mappe di Karnaugh 49
6.2.2 Ricerca delle liste di copertura non ridondanti 50
6.2.3 Riepilogo – procedura per la sintesi a costo minimo SP 56
6.2.4 Riepilogo – definizione e classificazione di implicanti e sottocubi 57
6.2.5 Esercizio – sintesi di leggi non completamente specificate - decodificatore BCD a 7 segmenti 58
6.2.6 Esercizio (per casa) 60
6.2.7 Esercizio (per casa) 60
6.2.8 Esercizio (per casa) 61
7 Sintesi di reti in formato PS 62
7.1 Approfondimento: Sintesi PS per via algebrica 66
7.2 Esercizio (per casa) 67
8 Porte logiche universali 69
8.1 Sintesi a porte NAND 69
8.2 Sintesi a porte NOR 70
8.3 Esercizio d'esame 71
8.3.1 Sintesi a porte NAND 71
8.3.2 Sintesi a porte NOR 73
9 Porte tri-state 75
10 Circuiti di ritardo e formatori di impulsi 79
11 Soluzioni degli esercizi per casa 82
11.1 Esercizio 3.2 82
11.2 Esercizio 5.1.1 83
11.3 Esercizio 6.2.6 84
11.4 Esercizio 6.2.7 84
11.5 Esercizio 6.2.8 85
11.6 Esercizio 7.2 87
12 Esercizi di riepilogo 88
12.1 Esercizio (Sintesi PS-SP a costo minimo) 88

12.2 Esercizio (sintesi a costo minimo) 89
13 Appendice 91
13.1 Domande a risposta multipla 91
13.2 Risposte 93

Version history

12/09/18: prima versione

22/10/18: corretti piccoli errori su cose dette a lezione.

22/10/21: corretto diagramma del circuito di ritardo sul fronte di salita.

08/02/23: corretta la definizione di IP semplicemente eliminabile a pag. 56.

07/02/24: corretto un refuso nella definizione di involutività a pag. 21

27/06/25: aggiunta appendice con domande a risposta multipla.

10/10/25: riformulato l'esercizio sul multiplexer a pag. 33.

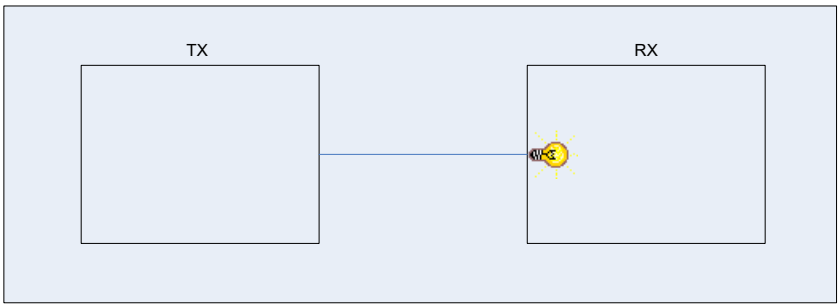
16/10/25: modifiche cosmetiche alla soluzione dell'esercizio 6.2.6.

04/09/26: aggiunto il paragrafo 6.1.1 sulla notazione compatta.

08/10/26: modifiche lessicali a pag. 43-44.

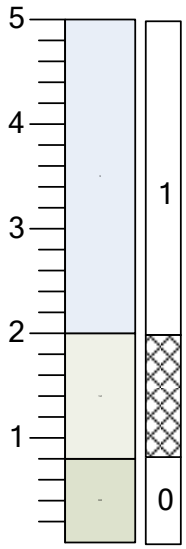
1 Le reti logiche come modello astratto di sistemi fisici

Supponiamo di avere il seguente sistema **fisico** (cioè, realizzato in pratica):

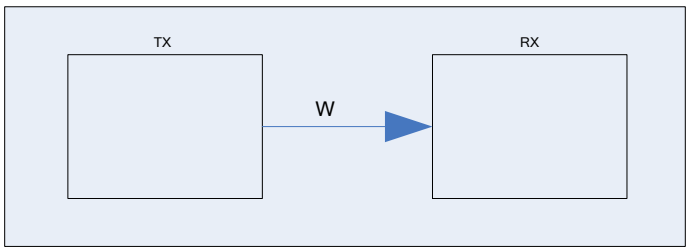


Le due scatolette comunicano tramite un filo elettrico. Nella scatoletta di sinistra ho la possibilità di impostare una **tensione** tra 0V e 5Vs, **variandola nel tempo**, e nella scatoletta di destra ho la possibilità di **osservare** una lampadina. In particolare, supponiamo che un osservatore sia in grado di decidere che:

- se la tensione impostata sul filo è tra **0V e 0.8V**, allora la lampada è **spenta;**
- se la tensione è tra **2V e 5V**, allora la lampada è **accesa**;
- se la tensione è **nel mezzo** tra i due intervalli, l'osservatore qualche volta decide che è spenta e qualche volta che è accesa (l'osservatore prende **sempre** una decisione).



Supponiamo che di questo sistema non ci interessi descrivere altro aspetto che questo.



Questo sistema **fisico**, cioè reale, è modellabile (descrivibile, rappresentabile) tramite un **modello astratto** chiamato **rete logica**. Una rete logica nella quale esistono:

- due **sottoreti**, dette Tx ed Rx, la cui composizione interna non mi interessa (in questo momento)
- una **variabile logica** *w* che le connette (di uscita rispetto a Tx, di ingresso rispetto ad Rx). Una *variabile logica* è una variabile che può assumere **due valori**, distinti tra loro. I due valori che una variabile logica assume vengono, **per consuetudine**, codificati con i **simboli '0' ed '1',** detti anche **bit** o cifre binarie.

Più in generale, una rete logica è un **modello astratto** di un **sistema fisico**, quest'ultimo costituito da **dispositivi tra loro interconnessi**. Tali dispositivi si scambiano **informazioni codificate**. Le in-

formazioni sono codificate tramite **fenomeni fisici** che si presentano ad un osservatore in **due aspetti distinti**. Ad esempio:

- corrente forte, corrente debole
- tensione alta, tensione bassa
- perforazione di una zona di un foglio di carta, assenza di perforazione
- magnetizzazione positiva/negativa di un'areola di materiale ferromagnetico
- …

In particolare, durante il corso useremo questo tipo di modello per descrivere i **circuiti elettronici** che formano i sistemi di elaborazione dell'informazione. Tali circuiti infatti verificano la definizione data sopra.

Vediamo di dare una descrizione più completa del modello "rete logica":



Una rete logica è caratterizzata da:

1) un insieme di N variabili logiche di ingresso. L'insieme degli N valori assunti all'istante t dalle variabili di ingresso si chiama **stato di ingresso** presente all'istante t. L'insieme di tutti i possibili stati di ingresso (2^N) verrà indicato con $\boldsymbol{X}$. $\boldsymbol{X} = \{(x_{N-1}x_{N-2}\dots x_1x_0)\}$.
2) un insieme di M variabili logiche di uscita. L'insieme degli M valori assunti all'istante t dalle variabili di uscita si chiama **stato di uscita** presente all'istante t. L'insieme di tutti i possibili stati di uscita (2^M) verrà indicato con $\boldsymbol{Z}$. $\boldsymbol{Z} = \{(z_{M-1}z_{M-2}\dots z_1z_0)\}$.
3) una **legge di evoluzione nel tempo**, che dice come le uscite evolvono in funzione degli ingressi.

Le reti logiche si classificano in base a **due criteri**, entrambi riguardanti il tipo di legge di evoluzione nel tempo

a) **presenza/assenza di memoria:**

- reti **combinatorie**: sono quelle reti in cui lo stato di uscita dipende soltanto dallo stato di ingresso. Ad un certo stato di ingresso corrisponde uno ed un solo stato di uscita. Analogia con il concetto di **funzione matematica**.
- reti **sequenziali**: in cui lo stato di uscita dipende dalla **storia** degli stati di ingresso precedenti. Ad un certo stato di ingresso può corrispondere uno stato di uscita od un altro, a se-

conda della storia passata. Queste ultime sono **reti con memoria**, in quanto per decidere quale sia l'uscita hanno bisogno di **ricordare** il passato.

b) **temporizzazione della legge di evoluzione**: la legge che fa corrispondere le uscite agli ingressi può essere resa operativa **in ogni istante**, oppure messa in pratica **ad istanti discreti nel tempo**. Nel primo caso, le uscite sono continuamente adeguate agli ingressi. Nel secondo caso, la rete "dorme", e "si sveglia" soltanto in certi istanti, adeguando lo stato di uscita allo stato di ingresso presente in quei dati istanti. Si distinguono a tale proposito:

- reti **asincrone**, in cui l'aggiornamento delle uscite avviene continuamente nel tempo
- reti **sincronizzate**, in cui l'aggiornamento delle uscite avviene ad istanti (di sincronizzazione) separati nel tempo

	asincrono	**sincronizzato**
combinatorio	1) Reti combinatorie	(non hanno un nome, sottocaso del caso 3)
sequenziale	2) Reti sequenziali asincrone	3) Reti sequenziali sincronizzate

Una rete logica è inserita all'interno di un mondo in cui qualcuno imposta dei valori logici sulle variabili di ingresso e qualcun altro legge i valori logici delle variabili d'uscita. Quale sia **l'interpretazione** che viene data a tali variabili è una cosa che non ci interessa a questo livello, al pari del fenomeno fisico con il quale l'informazione viene scambiata. Il modello consente quindi di astrarre dalla *fisica* del sistema, e dal *contorno* del sistema.

1.1 Limiti del modello

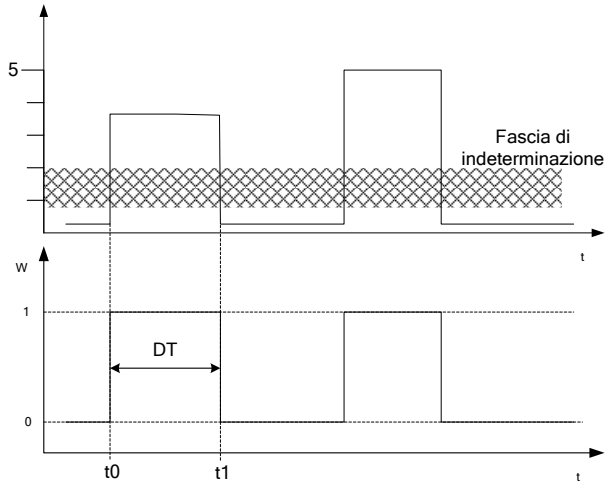
Vorrei richiamare l'attenzione sulla parola **modello**. Un modello è un modo, normalmente **semplificato**, di descrivere alcuni aspetti di una realtà complessa.

- Esempio preso dalla fisica: il **punto materiale** è un modello (semplice) di un sistema fisico (complesso), utile per descrivere alcune proprietà del moto del sistema stesso.
- Esempio preso dall'elettrotecnica: **un circuito a parametri concentrati** è un modello nel quale si suppone che gli effetti resistivi, capacitivi, induttivi, siano localizzati in alcuni componenti invece che diffusi su tutti i conduttori.

In particolare, tutte le volte che si modella qualcosa, si **trascurano** alcuni aspetti della realtà. Torniamo un attimo all'esempio iniziale.

1.1.1 Transizione dei segnali

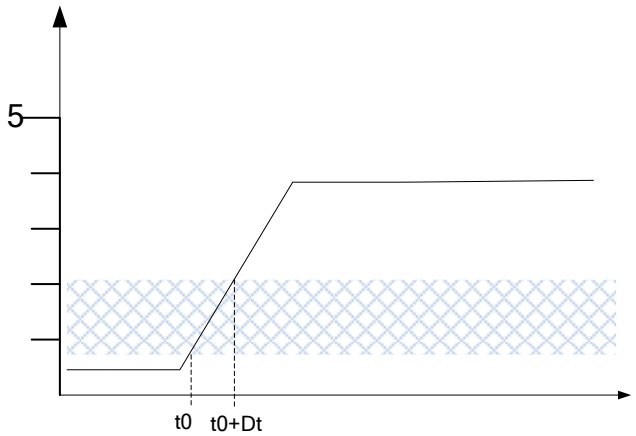
Disegniamo una possibile **evoluzione nel tempo** della variabile logica. La variabile logica può stare soltanto in due stati, 0 ed 1, e quindi le sue transizioni sono istantanee. Disegniamo anche la corrispondente variazione della tensione ad un capo del filo.



Diremo che:

- all'istante t_0 la variabile logica w **si setta**, oppure transisce ad 1, etc.
- all'istante t_1 la variabile logica w **si resetta**, oppure transisce a 0, etc.

Ora, mentre in un modello astratto possiamo pensare che una variabile logica cambi **istantaneamente ad un dato istante**, una tensione non cambia istantaneamente. Le grandezze fisiche cambiano in modo **continuo**. In realtà quello che succede nel sistema fisico è questo:



I sistemi che modelliamo hanno comunque transizioni secche se la grandezza fisica varia in modo monotono.

La grandezza fisica, per un certo tempo Δt, rimane nella **fascia di indeterminazione**, in quella fascia cioè in cui non appartiene a nessuno dei due intervalli di tensione associati ad uno dei due valori logici. Quindi, a rigor di logica, l'istante in cui la variabile logica cambia valore è noto con una certa incertezza.

Nei sistemi fisici che intendiamo modellare come reti logiche, il tempo per il quale le grandezze fisiche che modelliamo come variabili logiche variano è **molto minore** del tempo nel quale restano a re-

gime. Sotto questa ipotesi, un modello in cui le variabili logiche cambiano stato in maniera istantanea, pur se approssimato, è comunque un buon modello.

Parlando di reti logiche, supporremo sempre che le transizioni fisiche avvengano in un tempo $\Delta t \ll \Delta T$, quest'ultimo essendo il tempo per cui le variabili logiche mantengono il proprio **valore di regime**.

Sotto l'ipotesi che $\Delta t \ll \Delta T$, posso dire (con un leggero abuso di lessico), che all'istante t0 la variabile logica si è settata. In realtà, ciò avviene in un *qualunque istante tra* t0 e t0+Δt, ma non me ne preoccupo.

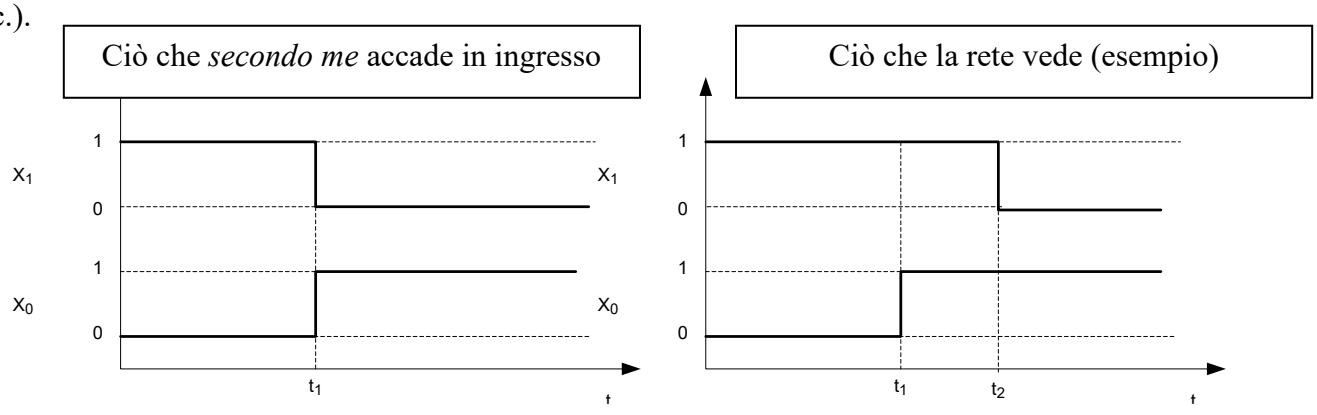
1.1.2 Contemporaneità

Abbiamo detto che il fatto che una variabile logica cambi istantaneamente è abbastanza plausibile, purché non si scenda troppo in basso con la scala dei tempi (ma noi non lo faremo). Consideriamo il seguente esempio:



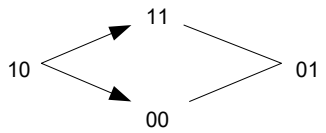
Questa è una rete logica a due ingressi ed un'uscita, quindi $\boldsymbol{X} = \{(00), (10), (01), (11)\}$, $\boldsymbol{Z} = \{0,1\}$. Una rete che non sappiamo cosa faccia, ma in questo momento non ci interessa. Supponiamo che prima dell'istante t_1 lo stato di ingresso presente sia, ad esempio, (1,0), e che *all'istante* t_1 lo stato di ingresso diventi (0,1).

Il problema è che **non sono in grado di garantire** che due grandezze **fisiche** varino contemporaneamente. Abbiamo appena visto che le variazioni di una grandezza non sono istantanee, e quindi se – per assurdo – suppongo di poter iniziare allo stesso istante la variazione di due grandezze, in ogni caso la rete "si accorgerà" in tempi diversi della variazione. Dipende da come sono fatti i contatti elettrici, da come è fatta la rete dentro (a livello di componenti elettrici, di drogaggio del silicio, etc.).



Se, quindi, baso un ragionamento sull'ipotesi che due variabili di ingresso varino contemporaneamente, poi non sarò mai in grado di costruire un sistema fisico che verifichi questa ipotesi.

Pertanto, **devo evitare di supporre che due variabili di ingresso varino contemporaneamente**.



Non è possibile che in una realizzazione fisica di un sistema si presentino in sequenza due stati di ingresso che differiscono tra loro per più di un bit. Se prima dell'istante t1 è presente lo stato di ingresso (10), e dopo è presente lo stato di ingresso (01), vuol dire **necessariamente** che, per un certo intervallo di tempo, **la rete ha visto in ingresso uno stato di ingresso intermedio** (11 o 00).

Questo può avere delle implicazioni. In particolare:

- se la rete è di tipo **sincronizzato**, e siamo sufficientemente distanti da un istante di sincronizzazione, non ci sono problemi di sorta.
- Se, invece la rete è di tipo **asincrono** (ad esempio, è una rete combinatoria o sequenziale asincrona), devo tenere conto **di cosa potrebbe presentarsi in uscita** non solo in conseguenza degli stati di ingresso 10 e 01, ma anche degli *altri* possibili stati di ingresso "transitori" 11 e 00. Addirittura, se la rete che sto modellando è **sequenziale**, una sua realizzazione fisica potrebbe evolvere in un modo o in un altro a seconda dell'ordine con cui cambiano i due ingressi.

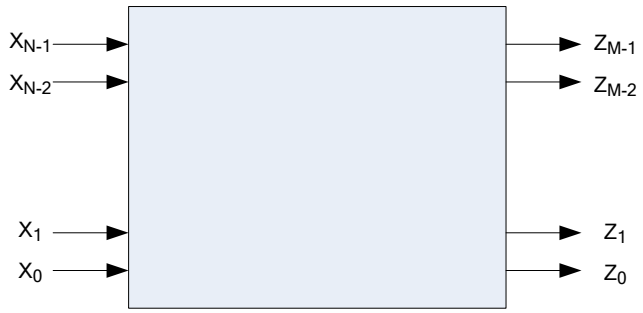
Nel descrivere l'evoluzione temporale di una rete **asincrona**, dovremo quindi preoccuparci del fatto che gli stati di ingresso siano vincolati a cambiare **un bit alla volta.** In altre parole si dovrà sempre supporre che **stati di ingresso consecutivi siano adiacenti**. (adiacenti significa che differiscono di un solo bit).

La stessa cosa, inoltre, potrebbe succedere se vario i due ingressi in modo **molto ravvicinato nel tempo.** Può darsi che, a causa delle variazioni di impedenza dei contatti, ed a causa del fatto che le reti non sono oggetti ideali, due transizioni troppo ravvicinate vengano risentite dalla rete **nell'ordine opposto** rispetto a quello che io penso di aver imposto.

2 Generalità sulle Reti Combinatorie

Per adesso tratteremo reti **combinatorie**, quindi reti in cui:

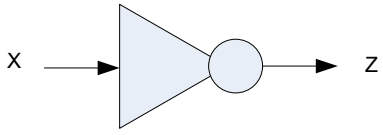
a) l'uscita viene aggiornata continuamente. Ciò vuol dire che la rete è sempre **sensibile** a ciò che succede in ingresso.
b) l'uscita dipende **soltanto** dallo stato di ingresso presente in quel momento.



Una rete combinatoria è caratterizzata da:

1) un insieme di N variabili logiche di ingresso.
2) un insieme di M variabili logiche di uscita.
3) Una **descrizione funzionale** del tipo $F : \boldsymbol{X} \rightarrow \boldsymbol{Z}$, che mappa quindi uno stato di ingresso in uno stato di uscita. Questa legge stabilisce il comportamento della rete, quindi dare questa legge significa dare la specifica di cosa faccia la rete. Ci soffermeremo a lungo sui modi per farlo.
4) Una **legge di evoluzione nel tempo** che dice: **continuamente,** detto X lo stato di ingresso, imposta lo stato di uscita ad $F(X)$. Una rete combinatoria lavora **continuamente**, cioè è sempre in evoluzione (è infatti una rete di tipo asincrono). Lo stato di ingresso può cambiare in qualunque momento.

<u>Esempio</u> di rete combinatoria: l'invertitore

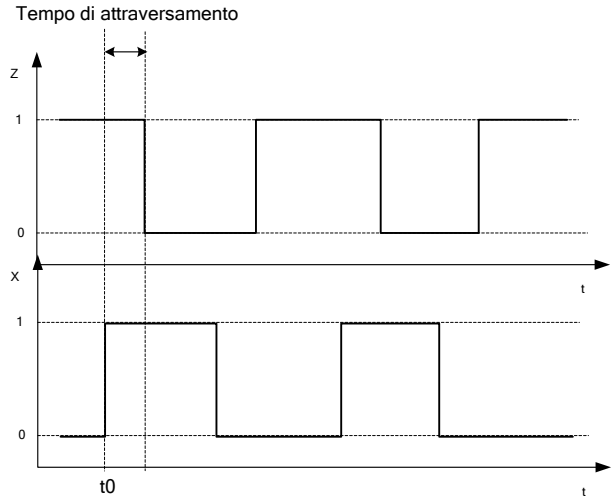


Particolare rete logica combinatoria con un ingresso ed un'uscita (N=1, M=1), quindi: $\boldsymbol{X} = \{0,1\}$, $\boldsymbol{Z} = \{0,1\}$. La legge di corrispondenza, data a parole, è: "Se l'ingresso è zero, l'uscita è uno e viceversa".

Quello è il simbolo con il quale si disegna un invertitore. Ogni rete logica elementare ha il proprio.

2.1.1 Tempo di attraversamento

Abbiamo scritto che una rete combinatoria **continuamente** elabora lo stato di ingresso per produrre uno stato di uscita. In ogni sistema fisico (ad esempio, nei dispositivi elettronici che formano i sistemi di elaborazione), una variazione negli ingressi viene a manifestarsi in uscita dopo un tempo *finito* (es: **carica di un condensatore**). Parleremo di questo tempo dicendo che una rete logica ha un **tempo di attraversamento (di accesso, di risposta).** Dire che una rete logica ha un tempo di attraversamento di 20ns significa dire che lo stato di uscita si sarà adeguato ad un cambiamento dello stato di ingresso **dopo** 20ns che lo stato di ingresso sarà stato cambiato.



Si dice, in tal caso, che la rete logica va **a regime** dopo 20 ns. Quando una rete è a regime, è pronta ad assorbire una nuova variazione dello stato di ingresso. In particolare, chi pilota questa rete deve **evitare di variare** lo stato di ingresso più velocemente del tempo di risposta della rete.

Una rete in cui gli ingressi cambiano in modo tale che la rete riesca ad andare a regime dopo ogni cambiamento dello stato di ingresso si dice **pilotata in modo fondamentale.**

2.2 Modalità di descrizione delle reti combinatorie

Una rete combinatoria si descrive dicendo:

- quanti sono gli ingressi;
- quante sono le uscite;
- qual è la funzione *F*, cioè la descrizione funzionale.

In particolare, la descrizione funzionale può essere data in diversi modi.

- Un modo è **a parole**, come visto prima.

- Esistono **linguaggi di descrizione** (vedrete il **Verilog**), che hanno una propria sintassi, e che servono ad assistere il progettista di sistemi di elaborazione.
- Il modo più immediato è utilizzare **tabelle di verità**.

Una **tabella di verità** è una tabella che reca a sinistra l'insieme dei possibili stati di ingresso e a destra l'insieme degli stati di uscita corrispondenti.

2.2.1 Esempio



x2	x1	x0	z1	z0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

La presenza di una riga in questa tabella significa che, per lo stato di ingresso (100), in uscita è presente – a regime – lo stato di uscita (11). Si dice che la variabile di uscita z1 **riconosce** gli stati di ingresso (010), (011), (100), (101), e che la variabile di uscita z0 riconosce (001), (100), (101).
Ovviamente, nella parte sinistra della tabella ci devono essere **tutti** i possibili stati di ingresso.
In alcuni casi (e.g., se sto progettando una rete combinatoria che dovrò poi sintetizzare), potrebbe **non interessarmi** quale valore assuma una variabile di uscita in corrispondenza di un particolare stato di ingresso. Magari sono sicuro (perché so dove la mia rete verrà inserita) che un certo stato di ingresso non si potrà presentare. In questi casi, **evito di specificare** un valore per le variabili di uscita nei casi suddetti, **inserendo un trattino** in corrispondenza del valore di uscita che non voglio specificare.

x2	x1	x0	z1	z0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	-	0
1	0	0	1	-
1	0	1	1	1
1	1	0	0	-
1	1	1	-	-

Il simbolo indicato dal trattino si chiama "non specificato", e **non è un valore logico**. Non specificato significa che **non mi interessa – adesso – decidere quale valore assumerà la variabile logica di uscita**. Resta inteso che assumerà comunque valore 0 o 1.

In molti casi (che vedremo), evitare di specificare il valore di una variabile di uscita rende più semplice **sintetizzare** la rete, cioè realizzarla con un sistema fisico.

È bene iniziare a fare caso fin d'ora alla differenza tra **descrizione e sintesi** di una rete.

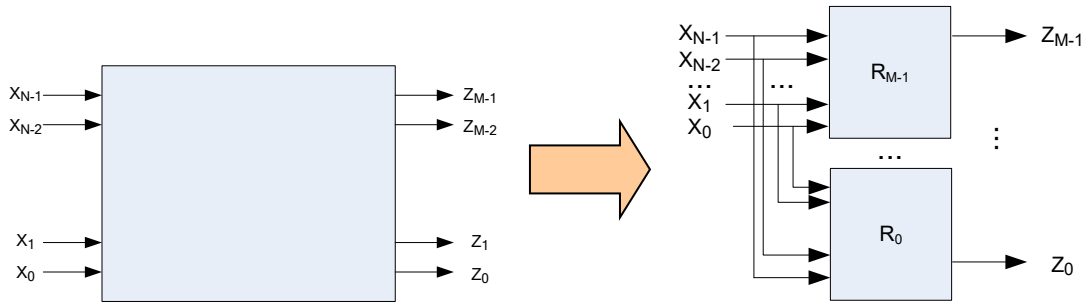
- La **descrizione** di una rete è un modo **formale** per dire **che cosa fa** quella rete, qual è cioè il suo comportamento osservabile. Una tabella di verità, ad esempio, spiega quale valore si osserva in uscita in corrispondenza di uno stato di ingresso.
- La **sintesi** di una rete è il progetto della realizzazione fisica di una rete, cioè la decisione di quali "scatolette" vanno messe, e connesse come, in modo da far sì che la rete si comporti come indicato nella descrizione.

Durante il corso, vedremo diversi tipi di reti logiche, e per ciascuna indicheremo modalità di descrizione e sintesi

2.3 Reti combinatorie elementari

Diamo ora descrizione di tutte le reti combinatorie significative. Come prima cosa osserviamo la seguente proprietà:

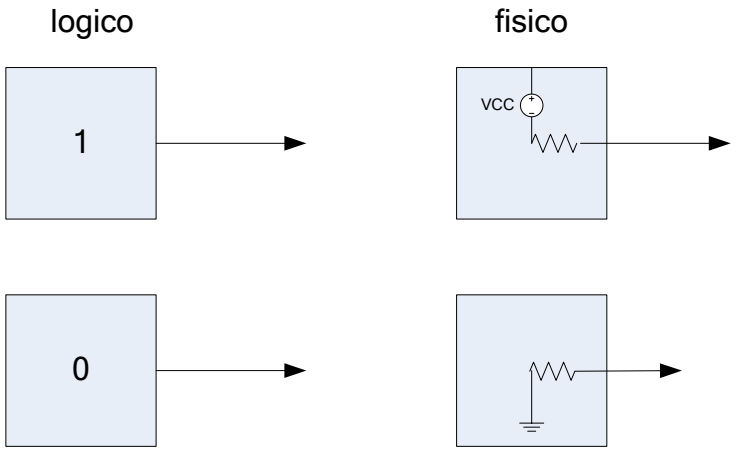
"**Una rete combinatoria ad *N* ingressi ed *M* uscite può essere realizzata interconnettendo *M* reti combinatorie ad *N* ingressi ed una uscita**".



Data questa proprietà, possiamo limitarci a considerare **soltanto reti con un'uscita**. Tanto, qualunque rete con più uscite può essere scomposta. Questo ci offre un primo esempio del fatto che reti logiche **più complesse** possono essere realizzate mettendo insieme sottosistemi più semplici. Non è detto che quello a destra **sia il miglior modo possibile** di realizzare una rete ad *N* ingressi ed *M* uscite, però si può fare.

2.3.1 Reti a zero ingressi

Dette anche **generatori di costante**, sono un caso degenere.

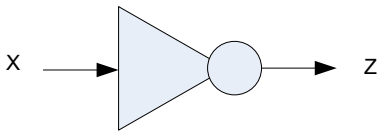


Dal punto di vista fisico, si realizzano connettendo l'uscita alla tensione di riferimento VCC (1) o a massa (0). Quindi sono "reti" per modo di dire. **Basta un filo** connesso ad una tensione.

2.3.2 Reti ad un ingresso

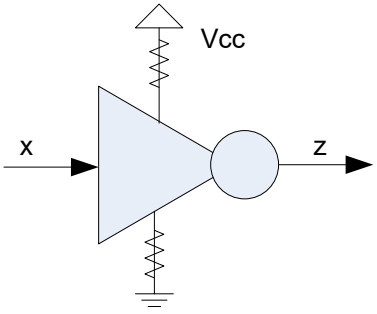
Ne abbiamo già vista una, l'**invertitore**. Descriviamo tutte quelle possibili.

Invertitore

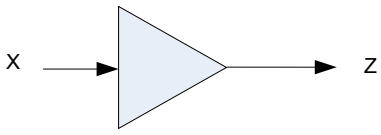


x	z
0	1
1	0

In realtà le reti sono connesse all'alimentazione e alla massa.



Elemento neutro



x	z
0	0
1	1

A cosa serve una rete del genere?

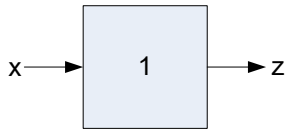
- a generare ritardo (utile per le temporizzazioni)
- a **rigenerare** i segnali elettrici degradati. Infatti, una rete logica (qualunque) correttamente funzionante:

  a) interpreta qualunque stato di ingresso (se qualche variabile è nella fascia di indeterminazione, non siamo sicuri di *come* lo interpreta, negli altri casi siamo sicuri che l'interpretazione è corretta), ma:

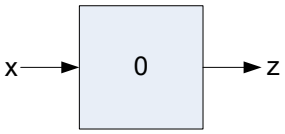
  b) presenta stati di uscita in cui le variabili di uscita sono **molto vicine** al fondo scala, quindi *migliora* (o *rigenera*) le proprietà elettriche del segnale.

Ci sono inoltre due casi degeneri:

Generatori di costante



x	z
0	1
1	1

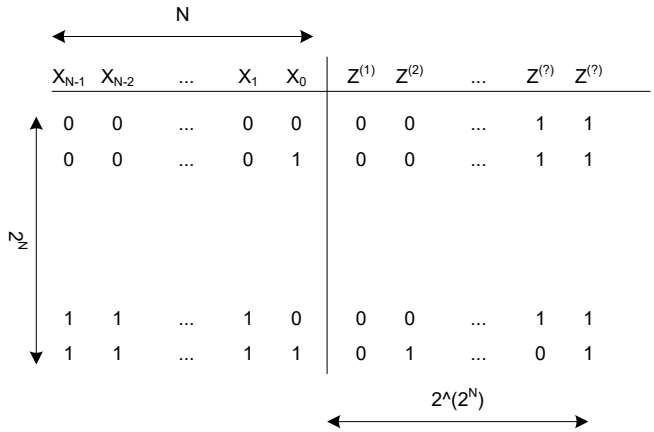


x	z
0	0
1	0

2.3.3 Reti a due ingressi

Prima di andare avanti, poniamoci una domanda: **quante sono** le possibili reti combinatorie diverse ad *N* ingressi ed un'uscita?

Per rispondere, dovrei poter dire quante sono le **possibili tabelle di verità** differenti per reti ad *N* ingressi. Ciascuna di queste tabelle di verità ha $\mathbf{2}^{N}$ righe, tante quanti sono i possibili stati di ingresso.

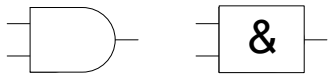


La risposta corretta è $\mathbf{2^{\wedge}(2^N)}$. Quindi, le possibili reti con 2 ingressi ed un'uscita sono in tutto 16. Vediamole:

x_1	x_0	$z^{(0)}$	$z^{(1)}$	$z^{(2)}$	$z^{(3)}$	$z^{(4)}$	$z^{(5)}$	$z^{(6)}$	$z^{(7)}$	$z^{(8)}$	$z^{(9)}$	$z^{(10)}$	$z^{(11)}$	$z^{(12)}$	$z^{(13)}$	$z^{(14)}$	$z^{(15)}$
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		(*)	(A)		(+)		(+)	(B)	(C)	(D)	(E)	(-)		(-)		(F)	(*)

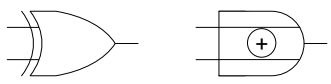
Quelle contrassegnate da una lettera hanno un nome:

A) porta **AND**: $z = 1 \Leftrightarrow x_0 = x_1 = 1$



L'uscita vale 1 solo se entrambi gli ingressi valgono 1.

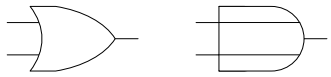
B) porta **XOR**: $z = 1 \Leftrightarrow x_0 \neq x_1$



L'uscita vale 1 se gli ingressi sono diversi.

C) porta **OR**: $z = 0 \Leftrightarrow x_0 = x_1 = 0$

NB: Per evitare problemi di interpretazione, nel seguito disegnerò le porte come scatolette con sopra il nome della funzione logica



L'uscita vale 1 se almeno un ingresso vale 1.

D) porta **NOR**: $z = 1 \Leftrightarrow x_0 = x_1 = 0$



L'uscita vale 0 se almeno un ingresso vale 1. La legge di corrispondenza è **equivalente** a quella che si ottiene inserendo un invertitore in cascata ad una porta **OR**.

E) porta **XNOR**: $z = 1 \Leftrightarrow x_0 = x_1$



L'uscita vale 1 se gli ingressi sono uguali. La legge di corrispondenza è **equivalente** a quella che si ottiene inserendo un invertitore in cascata ad una porta **XOR**.

F) porta **NAND**: $z = 0 \Leftrightarrow x_0 = x_1 = 1$



L'uscita vale 0 solo se entrambi gli ingressi valgono 1. La legge di corrispondenza è **equivalente** a quella che si ottiene inserendo un invertitore in cascata ad una porta **AND**.

Le sei porte scritte sopra si trovano **in commercio** come componentistica elettronica (ad esempio, **www.digchip.com**).

La prima e l'ultima (*) sono generatori di costante, quindi casi degeneri. Altre ripetono semplicemente una delle variabili di ingresso diretta (+) o negata (-), e quindi sono anch'esse casi semi-degeneri. Per le altre, comunque possibili, non esistono nomi particolari.

Si può dimostrare che **qualunque rete combinatoria** può essere implementata mediante combinazione di un sottoinsieme delle porte sopra descritte, come vedremo.

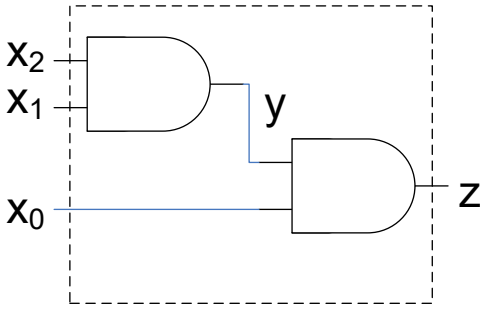
2.4 And ed Or a più ingressi

Posso pensare di **estendere** la funzione logica realizzata dalle porte AND e OR al caso di *N* ingressi.

- AND ad *N* ingressi: l'uscita vale 1 se e solo se **tutti gli ingressi valgono 1**
- OR ad *N* ingressi: l'uscita vale 1 se e solo se **almeno un ingresso vale** 1

Detto questo, posso **realizzare** reti di questo genere per mezzo di porte AND ed OR a 2 ingressi. Vediamo come si fa.

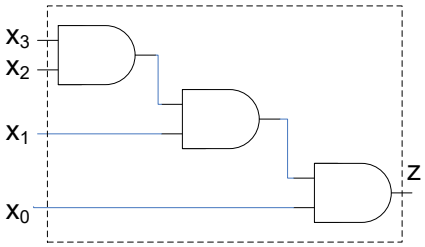
Esempio: AND a più ingressi



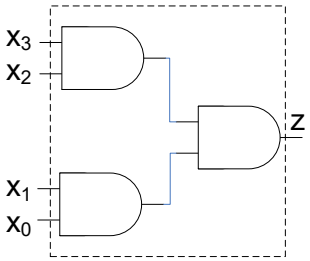
x_2	x_1	x_0	y	z
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

È immediato rendersi conto che questa rete combinatoria ha in uscita 1 solo se tutti gli ingressi sono ad 1.

Per un AND a 4 ingressi, potrei pensare di fare:

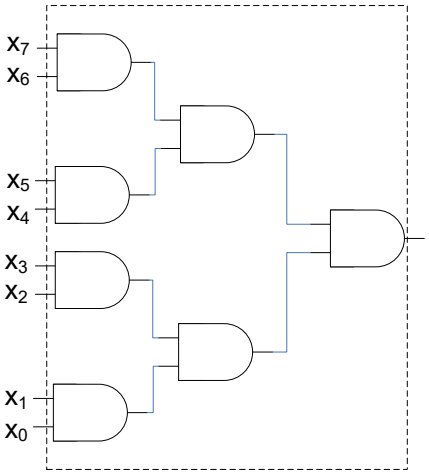


In realtà esiste un modo **migliore**:



Il numero di porte AND utilizzato è lo stesso (3), ma nel secondo caso il segnale di ingresso deve attraversare **2 livelli di logica** invece di 3. Quindi, il **tempo di attraversamento** della seconda rete è minore del tempo di attraversamento della prima.

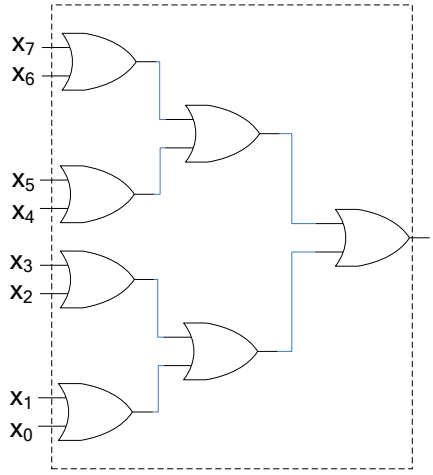
Per un AND a 8 ingressi, quindi, il modo migliore è questo:



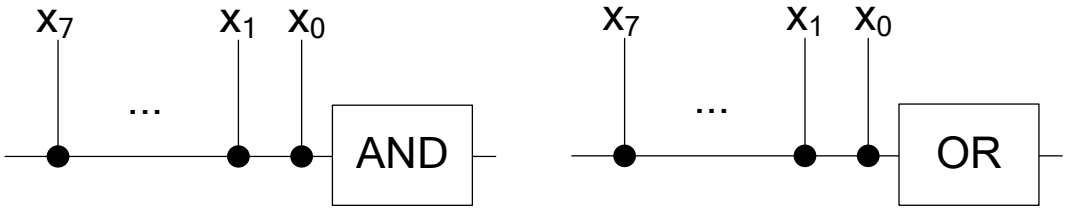
In generale, un AND ad *N* ingressi si può fare con *N*-1 AND a 2 ingressi, disposti ad **albero bilanciato**, in quanto in questa disposizione il numero di **livelli di logica** che il segnale deve attraversare **è minimo**.

2.4.1 Esempio: OR ad 8 ingressi

Il metodo è identico.



In commercio <u>esistono</u> anche AND ed OR a più di 2 ingressi. Per rappresentare graficamente porte con un numero elevato di ingressi, si usa spesso questa simbologia.



Quanto appena scritto dimostra che **si possono <u>anche</u> ottenere reti combinatorie (più complesse) interconnettendo reti combinatorie (più semplici).**

2.4.2 Esercizi (per casa)

<u>Es 1</u>

Si consideri una porta XOR (XNOR) a due ingressi, e la relativa tabella di verità. Si dispongano ad albero un certo numero di porte XOR (XNOR), e si calcoli quale sia la funzione logica calcolata da un tale montaggio, che generalizza quella di una porta XOR (XNOR) a due ingressi. Si dia una dimostrazione formale (per induzione sul numero di livelli dell'albero) della relazione trovata.

Soluzione: Mettere più porte XOR (XNOR) in cascata (eventualmente ad albero) consente di fare circuiti che **riconoscono stati di ingresso con un numero dispari (pari) di 1** (dimostrare per *induzione* sul numero di livelli dell'albero).

<u>Es 2</u>

Dimostrare (tramite un opportuno esempio) che connettendo ad albero porte NAND (oppure NOR) a due ingressi **non si ottiene** una generalizzazione della funzione logica descritta per porte a due ingressi.

3 Algebra di Boole

Introduciamo adesso un diverso formalismo per le leggi di corrispondenza delle reti combinatorie. L'algebra di Boole è un sistema algebrico basato su:

- **variabili logiche**, capaci di assumere due valori (che indichiamo, per convenzione, come 0 e 1)
- **operatori logici**, che si applicano alle variabili logiche per costruire espressioni algebriche dell'algebra di Boole.

Gli operatori logici sono **tre**: **complemento**, **prodotto logico, somma logica.** La definizione degli operatori si dà enumerando tutti i possibili casi. Date *x* ed *y* variabili logiche, posso scrivere:

Complemento: Operatore unario. Si indica con: $\overline{x}$, (anche: $!x$, $/x$). È definito come: $\overline{0} = 1, \overline{1} = 0$.

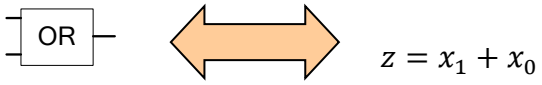
Prodotto logico: Operatore binario. Si indica con: $x \cdot y$. È definito come:

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

Somma logica: Operatore binario. Si indica con: $x + y$. È definito come:

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

Cosa abbia a che vedere tutto questo con le reti combinatorie viste finora è abbastanza ovvio. La porta OR è quella che **realizza** la funzione di somma logica tra due variabili logiche. Quindi, posso descrivere **la legge di corrispondenza *F* della porta OR a due ingressi** in termini di espressioni di algebra booleana:



$$z = x_1 + x_0$$

Lo stesso dicasi per la porta AND (quella, cioè, che realizza l'operatore prodotto logico dell'algebra di Boole) e la porta NOT (che realizza l'operatore di complemento dell'algebra di Boole).

3.1 Proprietà degli operatori booleani

Gli operatori booleani godono di alcune proprietà importanti, che si possono dimostrare per enumerazione di tutte le possibilità (cioè tramite la tabella di verità).

Le seguenti proprietà vanno capite **bene**, perché sono alla base di molte cose dette nel seguito.

Proprietà degli operatori booleani
Involutiva (del complemento): $\bar{\bar{x}} = x$
commutativa (della somma e del prodotto): $x + y = y + x, x \cdot y = y \cdot x$
associativa (della somma e del prodotto): $x + y + z = (x + y) + z = x + (y + z)$ $x \cdot y \cdot z = (x \cdot y) \cdot z = x \cdot (y \cdot z)$
distributiva della somma rispetto al prodotto **e viceversa**: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ $x + (y \cdot z) = (x + y) \cdot (x + z)$ (disegnare tabella di verità)
complementazione: $x \cdot \overline{x} = 0, x + \overline{x} = 1$
Unione e intersezione: $x + 0 = x, x + 1 = 1$ (0 el. **neutro,** 1 el. **assorbente**) $x \cdot 0 = 0, x \cdot 1 = x$ (1 el. **neutro**, 0 el. **assorbente**)
Idempotenza: $x + x = x, x \cdot x = x$
Teoremi di De Morgan $\overline{x \cdot y} = \overline{x} + \overline{y}$ $\overline{x + y} = \overline{x} \cdot \overline{y}$

3.1.1 Teoremi di De Morgan per N variabili logiche

I Teoremi di De Morgan valgono per un numero qualunque di variabili logiche.

$$1)\ \overline{x_0 \cdot x_1 \cdot \ldots \cdot x_{N-1}} = \overline{x_0} + \overline{x_1} + \ldots + \overline{x_{N-1}},$$

$$2)\ \overline{x_0 + x_1 + \ldots + x_{N-1}} = \overline{x_0} \cdot \overline{x_1} \cdot \ldots \cdot \overline{x_{N-1}}$$

Dimostrazione della prima tesi.

Si dimostra **per induzione sul numero di variabili logiche**. Dimostrare una proprietà per induzione significa:

a) dimostrare che la proprietà vale per un certo numero n_0 (passo base);
b) dimostrare che, se vale per un generico $n \geq n_0$, allora vale anche per $n + 1$ (passo induttivo)

Se ne conclude che vale per ogni $n \geq n_0$.

Passo base: *N*=2: Si fa la tabella di verità e si vede:

x_1	x_0	$x_0 \cdot x_1$	$\overline{x_0 \cdot x_1}$	$\overline{x_0} + \overline{x_1}$
0	0	0	1	1
0	1	0	1	1
1	0	0	1	1
1	1	1	0	0

Passo induttivo: Suppongo che sia vero per *N* variabili e dimostro che ciò implica necessariamente che è vero per *N*+1 variabili.

Ipotesi induttiva: $\overline{x_0 \cdot x_1 \cdot \ldots \cdot x_{N-1}} = \overline{x_0} + \overline{x_1} + \ldots + \overline{x_{N-1}}$

Tesi: $\overline{x_0 \cdot x_1 \cdot \ldots \cdot x_{N-1} \cdot x_N} = \overline{x_0} + \overline{x_1} + \ldots + \overline{x_{N-1}} + \overline{x_N}$

Pongo $x_0 \cdot x_1 \cdot \ldots \cdot x_{N-1} = \alpha$. α è una variabile logica, in quanto funzione di variabili logiche.

Quindi, dal passo base ottengo: $\overline{x_0 \cdot x_1 \cdot \ldots \cdot x_{N-1} \cdot x_N} = \overline{\alpha \cdot x_N} = \overline{\alpha} + \overline{x_N}$.

A questo punto entra in gioco l'ipotesi induttiva. $\overline{\alpha} = \overline{x_0 \cdot x_1 \cdot \ldots \cdot x_{N-1}} = \overline{x_0} + \overline{x_1} + \ldots + \overline{x_{N-1}}$, dal che ottengo la tesi.

NB: volendo, la seconda tesi del teorema di De Morgan si può dimostrare nello stesso modo. Comunque, è più semplice **complementare** entrambi i membri ed applicare la prima tesi.

$$\overline{x_0 + x_1 + \ldots + x_{N-1}} = \overline{x_0} \cdot \overline{x_1} \cdot \ldots \cdot \overline{x_{N-1}} \Leftrightarrow$$

$$\overline{\overline{x_0 + x_1 + \ldots + x_{N-1}}} = \overline{\overline{x_0} \cdot \overline{x_1} \cdot \ldots \cdot \overline{x_{N-1}}} \Leftrightarrow$$

$$x_0 + x_1 + \ldots + x_{N-1} = \overline{\overline{x_0}} + \overline{\overline{x_1}} + \ldots + \overline{\overline{x_{N-1}}} \Leftrightarrow$$

$$x_0 + x_1 + \ldots + x_{N-1} = x_0 + x_1 + \ldots + x_{N-1}$$

Dove il terzo passaggio richiede appunto la prima tesi del teorema di De Morgan.

3.1.2 Equivalenza tra espressioni dell'algebra di Boole e reti combinatorie

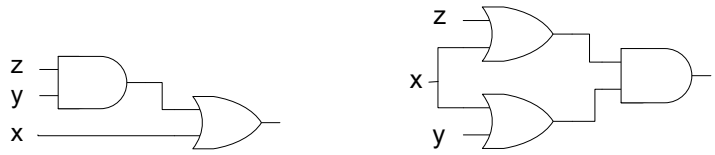
A questo punto appare chiaro che esiste una perfetta equivalenza tra operatori dell'algebra di Boole e porte logiche:

- **data una rete combinatoria** (comunque complessa), è sempre possibile trovare un'espressione booleana che ne descrive la funzione di corrispondenza (un'espressione per ogni uscita, in realtà);
- **data un'espressione booleana**, è sempre possibile sintetizzare una rete combinatoria (ad un'uscita) la cui uscita calcola quell'espressione.

Così come ci sono più espressioni logiche equivalenti – tutte quelle che si possono ottenere da una stessa espressione applicando le proprietà viste prima, esistono **reti combinatorie logicamente equivalenti tra loro**, cioè che sintetizzano la stessa tabella di verità. Tali reti non sono però necessariamente equivalenti dal punto di vista **fisico**, cioè non sono necessariamente realizzate tramite gli stessi componenti

Ad esempio:

- Data una qualunque rete combinatoria, posso aggiungere sempre 2 invertitori in cascata su un'uscita, ed ottenere una rete logicamente equivalente (proprietà involutiva del complemento).
- Le due reti sottostanti (prese dall'esempio della proprietà distributiva) hanno la stessa tabella di verità, quindi sono logicamente equivalenti.



  Ciononostante, per realizzare la seconda rete devo **comprare una porta in più**, quindi le due reti non hanno lo stesso **costo**.

Le porte logiche **costano, introducono ritardo, dissipano potenza, si rompono.** Quindi meno se ne usano, meglio è.

Parte di questo corso sarà dedicata ad individuare metodi tramite i quali, data una qualunque **descrizione** (i.e., tabella di verità), si riesce a sintetizzare la **rete combinatoria di costo minimo** che la realizza.

3.2 Esercizi (per casa)

1) Dimostrare che le seguenti identità sono vere

- $x_1 \cdot x_2 + x_2 \cdot x_3 + \overline{x_1} \cdot x_3 = x_1 \cdot x_2 + \overline{x_1} \cdot x_3$
- $(x_1 + x_2) \cdot (x_2 + x_3) \cdot (\overline{x_1} + x_3) = (x_1 + x_2) \cdot (\overline{x_1} + x_3)$

2) Utilizzando le proprietà dell'algebra, semplificare al massimo le seguenti espressioni

- $\overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot \overline{c} \cdot \overline{d} + a \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot d + b \cdot c \cdot d + a \cdot b \cdot d$
- $(a + \overline{b} + c) \cdot (\overline{a} + \overline{b} + \overline{c}) \cdot (\overline{a} + \overline{b} + c) \cdot (a + \overline{b} + \overline{c}) \cdot (\overline{a} + b)$

3) Date le proprietà elencate per gli operatori di somma e prodotto logico, controllare quali proprietà valgono per le porte XOR/XNOR

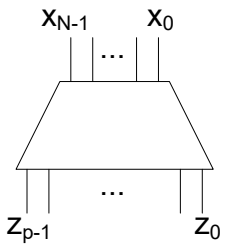
Soluzione

4 Reti combinatorie significative

Prima di parlare di come si faccia a sintetizzare reti combinatorie a costo minimo, è necessario introdurre qualche rete combinatoria significativa, di uso comune.

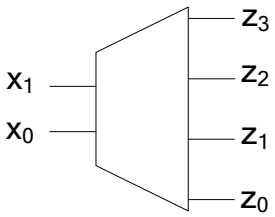
4.1 Decoder

È una rete con N ingressi e p uscite, con $p=2^N$.



Detta a parole, la sua legge di corrispondenza è la seguente: "ogni uscita riconosce uno ed un solo stato di ingresso, in particolare **l'uscita *j*-sima riconosce lo stato di ingresso i cui bit sono la codifica di *j* in base 2**, cioè se $(x_{N-1}x_{N-2}\dots x_1x_0)_{b2} \equiv j$"

4.1.1 Esempio: Decoder 2 to 4

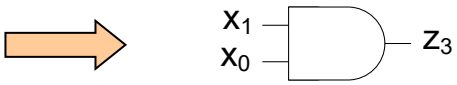


x_1	x_0	z_0	z_1	z_2	z_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Vediamo come la stessa descrizione si dà in termini di espressioni algebriche, e come questo porti immediatamente a capire come si realizzi la rete in termini circuitali.

- Prendo in esame l'uscita z_3:

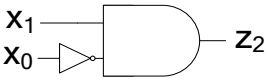
$$z_3 = \begin{cases} 1 & x_1x_0 = 11 \\ 0 & altrimenti \end{cases}$$



Tale uscita è quindi il **prodotto logico** delle variabili di ingresso. $z_3 = x_1 \cdot x_0$.

- Prendo in esame l'uscita z_2.

$$z_2 = \begin{cases} 1 & x_1x_0 = 10 \\ 0 & altrimenti \end{cases} \quad \Rightarrow \quad z_2 = \begin{cases} 1 & x_1\overline{x_0} = 11 \\ 0 & altrimenti \end{cases}$$



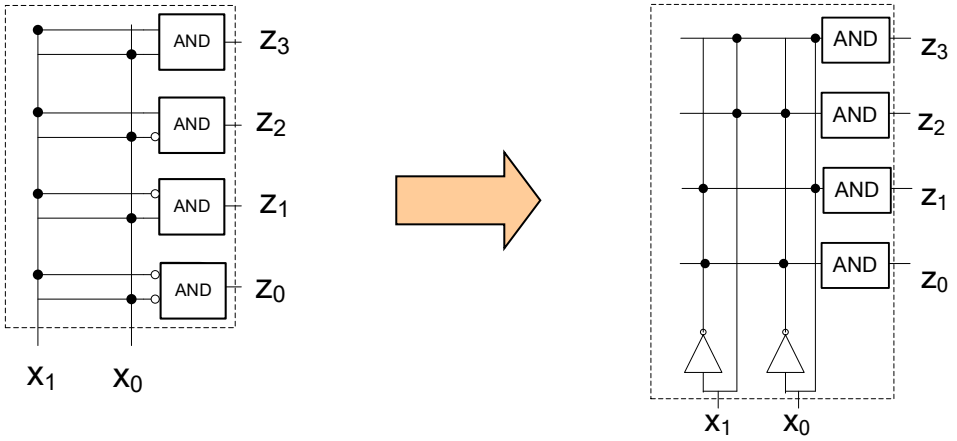
Tale uscita è quindi il **prodotto logico di x_1 per x_0 complementato**. $z_2 = x_1 \cdot \overline{x_0}$.

Lo stesso ragionamento lo posso fare per le altre uscite: $z_1 = \overline{x_1} \cdot x_0$, $z_0 = \overline{x_1} \cdot \overline{x_0}$.

Quindi, per realizzare un decoder 2 to 4 ho bisogno di:

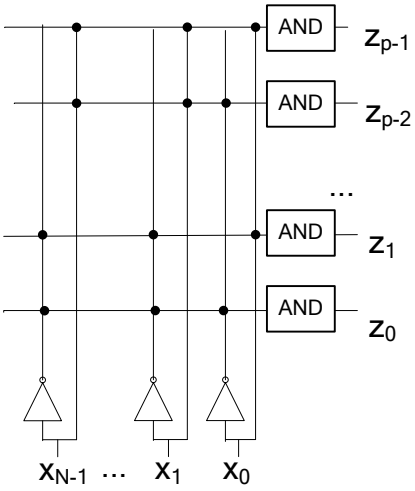
- 4 porte AND a due ingressi.
- Un certo numero di NOT. Quanti? Dipende:

Se metto gli invertitori sugli ingressi di ciascuna AND me ne servono 4 (oppure $2^{N-1}*N$). Se invece metto gli invertitori direttamente sugli ingressi, ne bastano 2, tanti quanti sono gli ingressi.



In generale, in un decoder N to 2^N ogni uscita è il **prodotto di tutti gli *N* ingressi, diretti o complementati**. Tutti i possibili stati di ingresso sono riconosciuti da una ed una sola uscita. La legge di corrispondenza, scritta per ogni uscita sotto forma di espressione booleana, è la seguente:

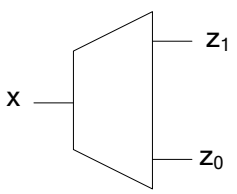
$$z_0 = \overline{x_{N-1}} \cdot \overline{x_{N-2}} \cdot \ldots \cdot \overline{x_1} \cdot \overline{x_0}$$
$$z_1 = \overline{x_{N-1}} \cdot \overline{x_{N-2}} \cdot \ldots \cdot \overline{x_1} \cdot x_0$$
$$\ldots$$
$$z_{p-2} = x_{N-1} \cdot x_{N-2} \cdot \ldots \cdot x_1 \cdot \overline{x_0}$$
$$z_{p-1} = x_{N-1} \cdot x_{N-2} \cdot \ldots \cdot x_1 \cdot x_0$$

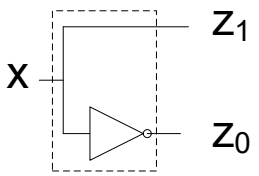


La realizzazione richiede tante porte AND (ad N ingressi) quante sono le uscite (2^N), e tanti invertitori quanti sono gli ingressi (N).

4.1.2 Esempio: Realizzazione di decoder 1 to 2

Questo è un caso degenere, che non richiede logica. Infatti:

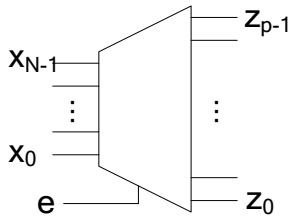




x	z_0	z_1
0	1	0
1	0	1

4.2 Decoder con enabler (espandibile)

Il decoder che abbiamo appena descritto **non è espandibile**. Non è cioè possibile costruire decoder grandi combinando decoder più piccoli. Per questo motivo, esistono decoder dotati di un ingresso aggiuntivo, detto **di abilitazione** (**enabler**).



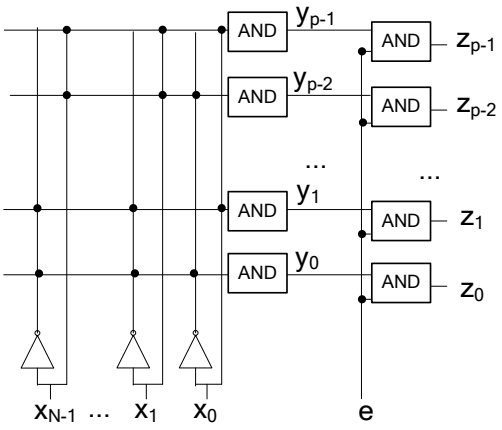
Quindi, il decoder con enabler ha N+1 ingressi e 2^N uscite. La sua descrizione funzionale è, a parole, la seguente: "**se l'ingresso di enabler è 1, la rete si comporta come un decoder *N* to 2^N. Altrimenti, tutte le uscite sono a 0**". In pratica, l'ingresso di abilitazione **"accende"** il decoder.

Come si sintetizza un decoder con enabler? Prendiamo un decoder **senza enabler**, e chiamiamo $y_0 \ldots y_{p-1}$ le sue uscite.

Ciascuna uscita z_i è 0 se e =0, altrimenti è uguale ad y_i.

$$z_i = \begin{cases} y_i & e = 1 \\ 0 & e = 0 \end{cases}$$

Ricordando che: $x \cdot 0 = 0$, $x \cdot 1 = x$, si ottiene immediatamente che $z_i = y_i \cdot e$. Il che mi dà subito un modo operativo per sintetizzare un decoder con enabler.



Ricordando come è stato ottenuto ciascun valore y_i, otteniamo la descrizione funzionale completa. Da questa si ricava immediatamente una sintesi a **costo *minore*** del decoder con enabler. Nel primo caso, infatti, è necessario mettere 2^N porte AND a 2 ingressi in più. Nel secondo caso basta aggiungere un ingresso ad ogni AND.

$$z_0 = e \cdot \overline{x_{N-1}} \cdot \overline{x_{N-2}} \cdot \ldots \cdot \overline{x_1} \cdot \overline{x_0}$$

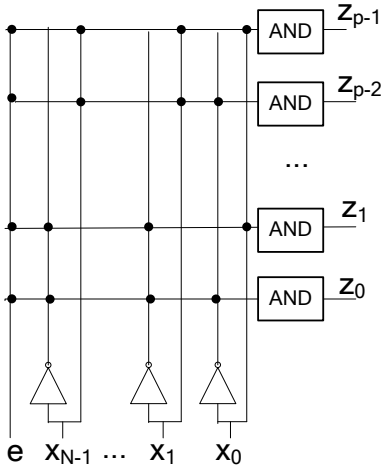
$$z_1 = e \cdot \overline{x_{N-1}} \cdot \overline{x_{N-2}} \cdot \ldots$$

…

$$z_{p-2} = e \cdot x_{N-1} \cdot x_{N-2} \ldots$$

$$z_{p-1} = e \cdot x_{N-1} \cdot x_{N-2} \ldots$$

importante – serve per confronto con demux



Osservazione: è sempre una soluzione sciocca mettere **due porte identiche in cascata**, a meno che non abbia vincoli sul numero di ingressi (di tipo elettronico o costruttivo).

4.2.1 Esempio: costruzione di un decoder 4 to 16 da decoder 2 to 4

Abbiamo detto che l'ingresso di abilitazione serve a rendere espandibile un decoder. Vediamo allora di costruire un decoder 4 to 16 mettendo insieme decoder con enabler 2 to 4. Partiamo dalla tabella di verità, che viene inusitatamente grande, ma è molto semplice da scrivere.

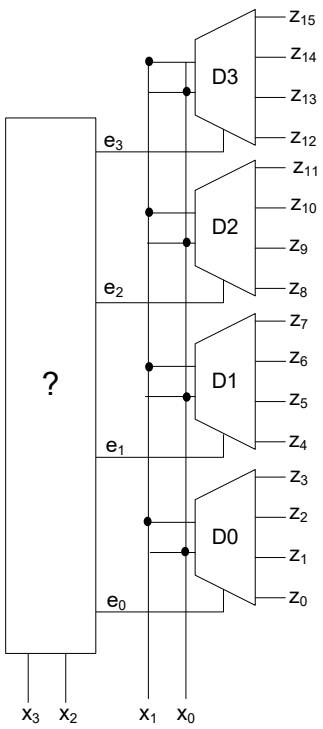
x_3	x_2	x_1	x_0	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}
0	0	0	0	1	0	0	0	0				0				0			
0	0	0	1	0	1	0	0												
0	0	1	0	0	0	1	0												
0	0	1	1	0	0	0	1												
0	1	0	0	0				1	0	0	0	0				0			
0	1	0	1					0	1	0	0								
0	1	1	0					0	0	1	0								
0	1	1	1					0	0	0	1								
1	0	0	0	0				0				1	0	0	0	0			
1	0	0	1									0	1	0	0				
1	0	1	0									0	0	1	0				
1	0	1	1									0	0	0	1				
1	1	0	0	0				0				0				1	0	0	0
1	1	0	1													0	1	0	0
1	1	1	0													0	0	1	0
1	1	1	1													0	0	0	1

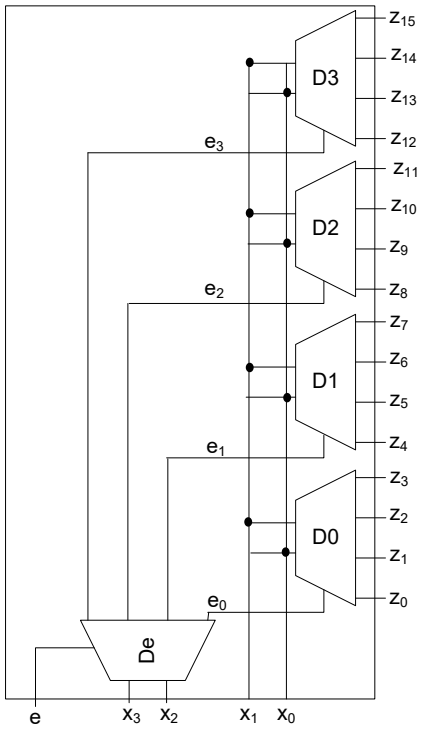
La tabella di verità di un decoder è sempre una matrice identità. I blocchi non diagonali sono nulli.
Guardiamo soltanto gli ingressi x_1x_0 e le uscite $z_0 \ldots z_3$. Posso realizzare la rete che genera queste 4 uscite con un decoder 2 to 4 che lavori soltanto quando x_3x_2 valgono 00, che cioè **è abilitato** se x_3x_2 valgono 00. Lo stesso ragionamento lo posso fare se guardo gli ingressi gli ingressi x_1x_0 e le uscite $z_4 \ldots z_7$. Quindi, posso prendere 4 decoder 2 to 4, dare loro in ingresso le variabili x_1 e x_0, e prendere tutte quante le loro uscite come $z_0 \ldots z_{15}$. Questi decoder, però, non devono essere abilitati tutti contemporaneamente:

- il primo deve lavorare soltanto quando x_3x_2 valgono 00
- il secondo deve lavorare soltanto quando x_3x_2 valgono 01
- …

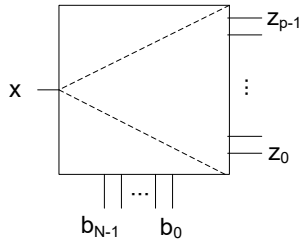
Quindi posso usare le variabili x_3x_2 per generare il segnale di abilitazione per i 4 decoder 2 to 4 D0…D3 (disegnare figura a sinistra).
Come sarà fatta la sottorete che genera i segnali di abilitazione per i 4 decoder? È una rete con 2 ingressi e 4 uscite, tale per cui a seconda delle 4 possibili configurazioni di ingresso una ed una soltanto delle uscite è attiva. Quindi, è essa stessa un decoder 2 to 4. Quindi, ho realizzato un decoder 4 to 16 usando soltanto decoder con enabler 2 to 4. Se volessi dotare quest'ultimo decoder di

enabler, **dove dovrei metterlo**? Dovrei metterlo in modo tale che, se l'enabler generale è a zero, tutte le uscite sono a zero. Quindi lo metto sul decoder che produce gli enabler per gli altri.





4.3 Demultiplexer



È una rete con N+1 ingressi e p=2^N uscite. Degli ingressi, uno (x) si chiama **variabile da commutare**, gli altri si chiamano **variabili di comando**. In pratica, un demultiplexer è un **selezionatore**. Le variabili di comando stabiliscono quale uscita è connessa all'ingresso. Tutte le altre sono nulle. La descrizione funzionale, data a parole, è la seguente: "**la *j*-sima uscita insegue la variabile da commutare se e solo se $(b_{N-1}b_{N-2}\dots b_1b_0)_2 \equiv j$, altrimenti vale 0**."

Vediamo di dare la descrizione in termini algebrici per ciascuna uscita. Per l'uscita z0 ho bisogno di una rete che dia x quando $(b_{N-1}b_{N-2}\dots b_1b_0) \equiv (00\dots00)$ e zero altrimenti.

$$z_0 = \begin{cases} x & (b_{N-1}\dots b_1b_0) = (0\dots00) \\ 0 & altrimenti \end{cases} \Longrightarrow z_0 = \begin{cases} x & \overline{b_{N-1}} \cdot \dots \cdot \overline{b_1} \cdot \overline{b_0} = 1 \\ 0 & altrimenti \end{cases}$$

Quindi, ricordando che: $\alpha \cdot 0 = 0$, $\alpha \cdot 1 = \alpha$:

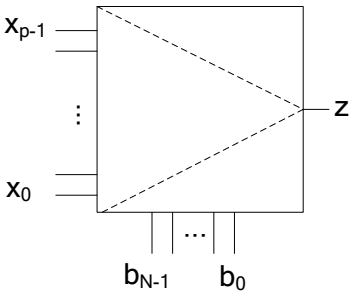
$$z_0 = x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot \overline{b_0}$$

In generale, otteniamo quanto segue:

$$\begin{aligned} z_0 &= x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot \overline{b_0} \\ z_1 &= x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot b_0 \\ &\dots \\ z_{p-2} &= x \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot \overline{b_0} \\ z_{p-1} &= x \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot b_0 \end{aligned}$$

A ben guardare, la descrizione funzionale **è identica a quella di un decoder con enabler,** salvo aver dato un nome diverso alle variabili. Infatti, "decoder con enabler" e "demultiplexer" sono due modi diversi di chiamare la stessa rete.

4.4 Multiplexer



È una rete con $N+2^N$ ingressi ed 1 uscita. Anche in questo caso le variabili b_i si chiamano **variabili di comando.** Questa rete è duale del demultiplexer, e serve a stabilire quale dei 2^N ingressi è connesso all'uscita. Tale decisione avviene, ovviamente, sulla base del valore delle variabili di comando. In altre parole:

$$z = x_i \Leftrightarrow (b_{N-1}\dots b_1 b_0)_2 \equiv i$$

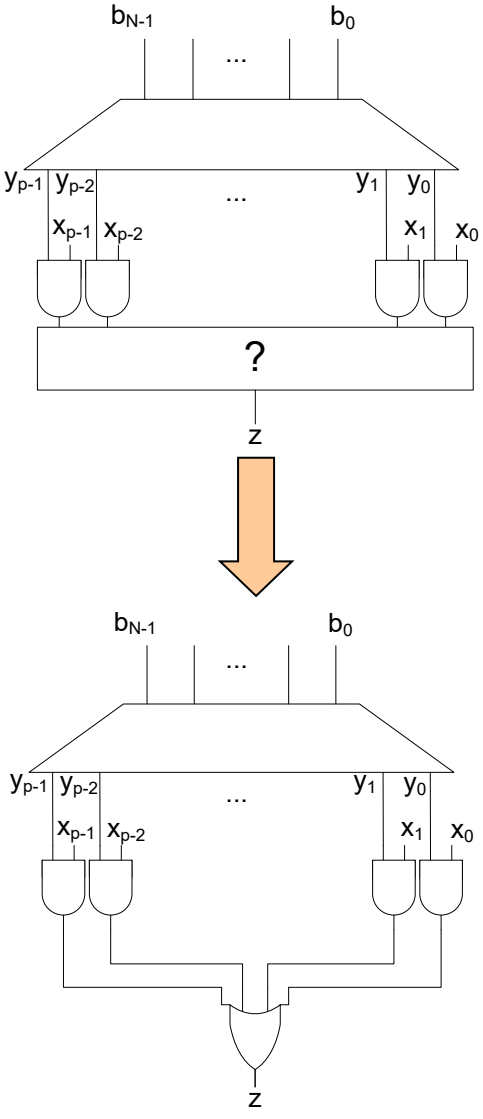
Reti di questo genere (multiplexer e demultiplexer) si trovavano (nella preistoria) nelle centrali telefoniche, e servivano a fare in modo che la stessa linea telefonica (ad esempio interurbana) potesse essere usata, in tempi diversi, da più utenti.

Invece di dare la descrizione in termini algebrici, **partiamo dalla sintesi** (è più semplice).

Prendiamo **un decoder** N to 2^N , e gli diamo come ingresso le variabili di comando. Ciascuna delle 2^N uscite si attiverà in corrispondenza di una configurazione delle variabili di comando. Quindi, possiamo mettere in AND a ciascuna uscita una delle variabili di ingresso.

In questo modo ho ottenuto 2^N variabili logiche, che hanno la seguente proprietà:

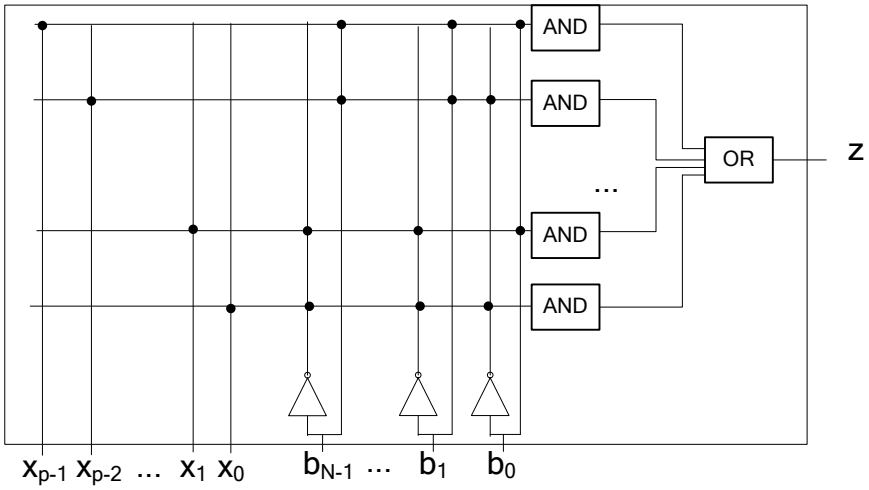
- tutte meno una sono sicuramente nulle.
- quell'unica che non sono sicuro che sia nulla (sia la j-sima) vale quanto vale x_j.



Come faccio ad ottenere l'uscita z con queste variabili?

Basta ricordare che $0 + \alpha = \alpha$. Infatti, di queste p variabili, p-1 sono sicuramente nulle. Quindi le posso sommare a quell'unica variabile che non so se è nulla senza che cambi il risultato.

Andando a guardare come è realizzato un decoder, si osserva che nella sintesi fatta sopra ci sono due porte AND in cascata. Come già osservato, si possono compattare in una AND a N+1 ingressi. Quindi, l'unica complessità in più è aver aggiunto una porta **OR a 2^N ingressi** in fondo.



Dal che ricavare una specifica funzionale è immediato:

$$
\begin{aligned}
z = {} & x_0 \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \ldots \cdot \overline{b_1} \cdot \overline{b_0} + \\
& x_1 \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \ldots \cdot \overline{b_1} \cdot b_0 + \\
& \ldots \\
& x_{p-2} \cdot b_{N-1} \cdot b_{N-2} \cdot \ldots \cdot b_1 \cdot \overline{b_0} + \\
& x_{p-1} \cdot b_{N-1} \cdot b_{N-2} \cdot \ldots \cdot b_1 \cdot b_0
\end{aligned}
$$

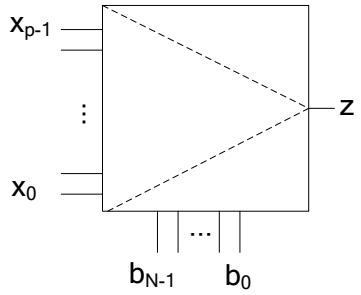
La strada più lunga da un ingresso ad un'uscita (in questo caso sono tutte uguali) transita attraverso due porte (una AND ed una OR, non contando le NOT). Un multiplexer è quindi una rete a **due livelli di logica**. Le porte AND e OR contano per un livello, le porte NOT sugli ingressi non vengono contate (il motivo sarà chiaro quando avrete fatto **i registri**).

4.4.1 Il multiplexer come rete combinatoria universale

È facile rendersi conto che un multiplexer con N variabili di comando è in grado di realizzare qualunque legge combinatoria di N ingressi ed un'uscita.

Infatti, basta connettere ai 2^N ingressi dei generatori di costante (che, in pratica, vuol dire attaccare i piedini di ingresso a tensione o a massa) in maniera opportuna. Prendo la tabella di verità della rete che voglio sintetizzare,

j	b_2	b_1	b_0	z
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0



Per ogni riga j della tabella di verità, se l'uscita vale 1 attacco l'ingresso x_j a Vcc, altrimenti lo attacco a massa.

Questo ha una conseguenza molto importante.

a) Un multiplexer si realizza con porte AND, OR, NOT, ed è una rete a due livelli di logica.
b) Un multiplexer realizza una qualunque rete combinatoria ad un'uscita.
c) una rete combinatoria a più uscite può essere scomposta in reti ad un'uscita messe "in parallelo".

Quindi:

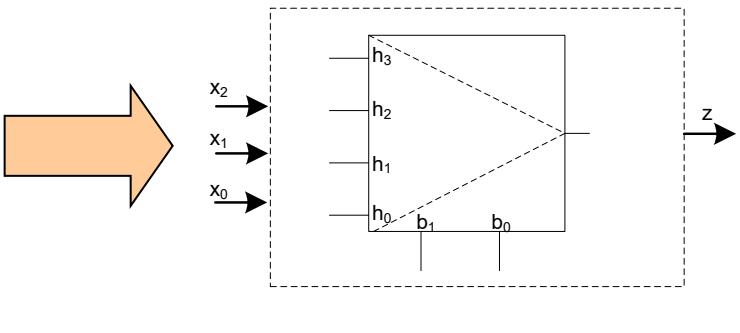
<u>Ogni</u> rete combinatoria può essere costruita combinando AND, OR, NOT in al più <u>due</u> livelli di logica

Quanto appena detto è di estrema importanza, in quanto stabilisce un limite superiore al **<u>ritardo</u>** di una qualunque rete combinatoria.

4.4.2 Esercizio: realizzazione di una RC ad N ingressi con un MUX ad $N-1$ variabili di comando

Dimostrare che la seguente tabella di verità di una RC a tre ingressi può essere implementata connettendo opportunamente lo schema della figura, aggiungendo al più una porta NOT.

x_2	x_1	x_0	z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

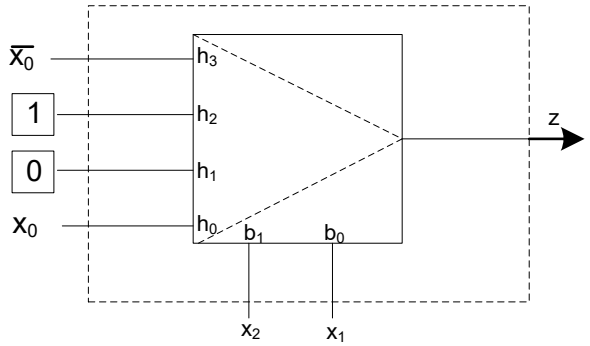


Discutere se sia sempre possibile, data una qualunque tabella di verità per rete ad N ingressi, realizzare la rete che la implementa tramite un multiplexer ad $N-1$ variabili di comando ed al più una porta NOT.

Soluzione

Si prendono gli ingressi di ordine maggiore $x_2,\ x_1$ e si collegano alle due variabili di comando. Ciò vuol dire che ciascun ingresso h_j del multiplexer sarà selezionato da una **coppia di stati di ingresso** della tabella di verità (che, in questo caso, si trovano su righe consecutive).

- Per far sì che l'uscita valga 01 quando lo stato di ingresso è 00-, connetto l'ingresso h_0 a x_0;
- Per far sì che l'uscita valga 00 quando lo stato di ingresso è 01-, connetto l'ingresso h_1 a massa;
- Per far sì che l'uscita valga 11 quando lo stato di ingresso è 10-, connetto l'ingresso h_2 a Vcc;
- Per far sì che l'uscita valga 10 quando lo stato di ingresso è 11-, connetto l'ingresso h_3 a $\overline{x_0}$.



Però posso osservare che, in corrispondenza di ciascuna coppia di stati di ingresso, la variabile di uscita può assumere soltanto 4 configurazioni diverse: 00, 01, 11, 10. Nell'esempio che ho appena risolto le ho incontrate tutte. Quindi, qualunque sia la tabella di verità, esiste un modo per realizzare una rete a 3 ingressi con un multiplexer a 2 variabili di comando.

Il procedimento avrebbe funzionato anche se avessi connesso alle variabili di comando un'altra coppia di ingressi (basta andarsi a cercare nella tabella di verità le coppie di righe interessate, che però non saranno più consecutive).

La generalizzazione ad una tabella di verità ad N ingressi appare adesso piuttosto ovvia.

Questo esercizio conferma, ancora una volta, che ci sono **diversi modi** di realizzare una stessa funzione logica. Queste diverse realizzazioni, pur equivalenti, non sono fatte con lo stesso numero di porte, e quindi non hanno lo stesso costo, lo stesso consumo di potenza, etc. Le prossime ore sono dedicate quindi a capire come si possa scegliere, tra tutte le possibili sintesi, quella di **costo minimo.**

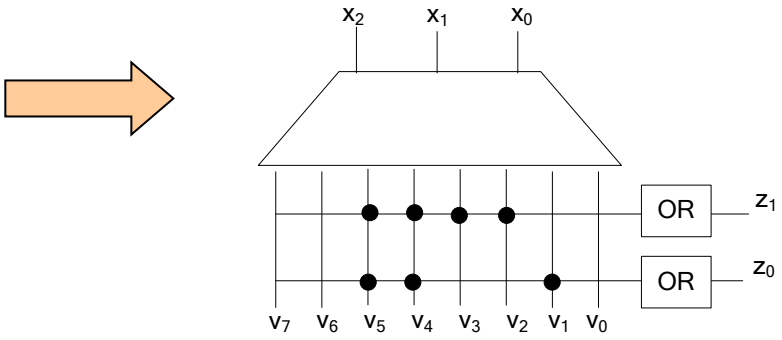
5 Modello strutturale universale per reti combinatorie

Abbiamo dato prova del fatto che **ogni rete combinatoria può essere costruita combinando AND, OR, NOT in al più due livelli di logica**

Vediamo adesso **un modo per sintetizzare** una rete logica ad *N* ingressi ed *M* uscite a partire da una tabella di verità.

Consideriamo ad esempio la seguente tabella di verità per una rete a 3 ingressi e due uscite.

	x_2	x_1	x_0	z_1	z_0
v_0	0	0	0	0	0
v_1	0	0	1	0	1
v_2	0	1	0	1	0
v_3	0	1	1	1	0
v_4	1	0	0	1	1
v_5	1	0	1	1	1
v_6	1	1	0	0	0
v_7	1	1	1	0	0

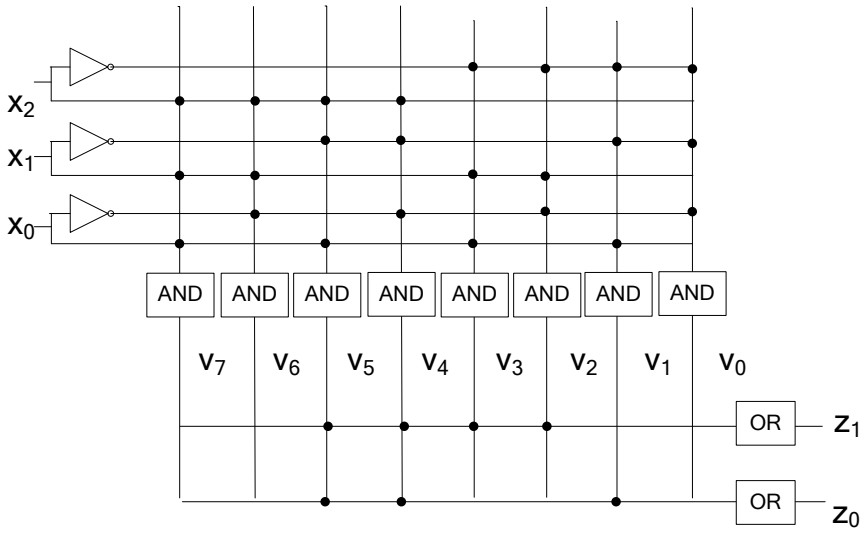


Il **modello strutturale universale** è un decoder ad *N* ingressi, seguito da una barriera di *M* OR. L'unica cosa da decidere è come collegare le uscite del decoder agli ingressi degli OR. Per farlo, vado nella tabella di verità a guardare quali sono **gli stati di ingresso riconosciuti** da ciascuna uscita. In corrispondenza di ciascuno di questi stati, una ed una sola uscita del decoder è attiva. Quell'uscita la devo collegare all'ingresso dell'OR corrispondente.

Prendo un **decoder 3 to 8**, chiamo $v_0, \ldots v_7$ le sue uscite. Prendo poi due porte OR, tante quante sono le uscite, e collego alcune uscite del decoder in ingresso alle porte OR. Quali? Quelle corrispondenti alle righe della tabella di verità in cui le uscite devono valere 1. Infatti, dalla tabella di verità ricavo che z_1 deve essere settata quando gli ingressi si trovano in una delle configurazioni che settano v_2, v_3, v_4, v_5. Quindi, se almeno una tra v_2, v_3, v_4, v_5 è settata, deve esserlo z_1. In altre parole:

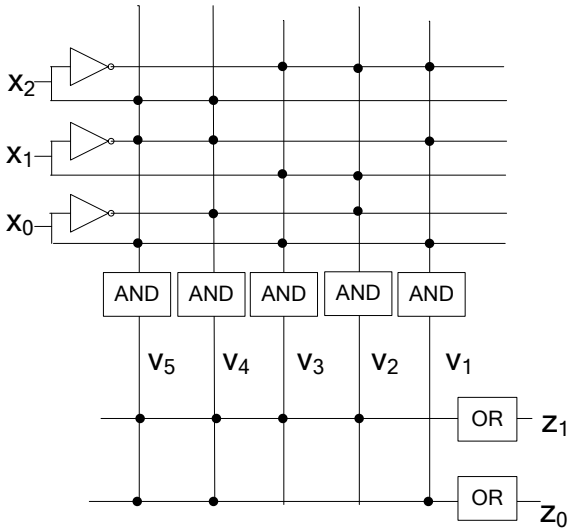
$$\begin{cases} z_1 = v_2 + v_3 + v_4 + v_5 \\ z_0 = v_1 + v_4 + v_5 \end{cases}$$

Fin qui il **modello strutturale universale**, che mi dà un modo per fare una rete. Vediamo se si può fare di meglio, e guardiamo la rete più nel dettaglio:



Ci vogliono: **8 AND a 3 ingressi, 2 OR a 4 e 3 ingressi.** Esiste una maniera di costo minore per realizzare la stessa rete?

Si vede al volo che ci sono **tre porte AND** (relative a stati di ingresso che non sono riconosciuti da nessuna uscita) che possono essere eliminate.



Possiamo scendere ancora? A occhio non si vede. Dobbiamo chiamare in aiuto l'algebra di Boole.

$$\begin{cases} z_1 = v_2 + v_3 + v_4 + v_5 \\ z_0 = v_1 + v_4 + v_5 \end{cases}$$

Espandiamo ciascuno dei termini fino a risalire alle variabili di ingresso:

$$\begin{cases} z_1 = \overline{x_2} \cdot x_1 \cdot \overline{x_0} + \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0 \\ z_0 = \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0 \end{cases}$$

L'espressione algebrica appena prodotta si chiama **forma canonica SP (somma di prodotti)** della rete combinatoria. La sintesi che le corrisponde (quella disegnata sopra) si dice **sintesi in forma canonica SP**. In una sintesi in forma canonica SP, ogni variabile di uscita è la somma del prodotto di **tutte le variabili di ingresso dirette o negate**.

Prendiamo a riferimento l'uscita z_1. È possibile trovare un'espressione equivalente più semplice? Per trovarla facciamo ricorso alle proprietà degli operatori booleani.

$$z_1 = \overline{x_2} \cdot x_1 \cdot \overline{x_0} + \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0$$
$$= \overline{x_2} \cdot x_1 \cdot \left(\overline{x_0} + x_0\right) + x_2 \cdot \overline{x_1} \cdot \left(\overline{x_0} + x_0\right)$$

Ma $\overline{x_0} + x_0 = 1$, e $\alpha \cdot 1 = \alpha$. Quindi: $z_1 = \overline{x_2} \cdot x_1 + x_2 \cdot \overline{x_1}$

Guardiamo adesso l'altra uscita z_0.

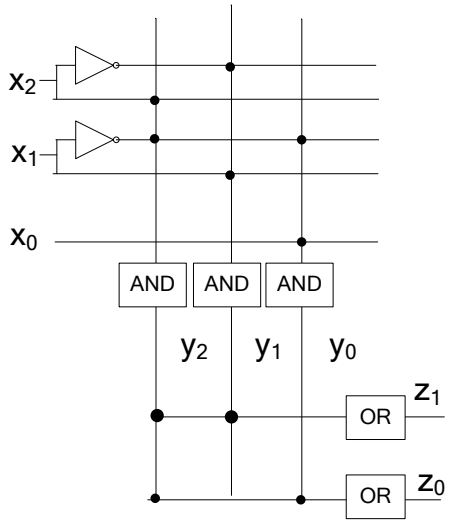
$$z_0 = \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0$$

Consideriamo che $\alpha + \alpha = \alpha$, ed aggiungiamo l'ultimo termine:

$$z_0 = \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot \overline{x_1} \cdot x_0 + x_2 \cdot \overline{x_1} \cdot x_0$$
$$= \overline{x_1} \cdot x_0 \cdot \left(\overline{x_2} + x_2\right) + x_2 \cdot \overline{x_1} \cdot \left(\overline{x_0} + x_0\right)$$

Quindi: $z_0 = \overline{x_1} \cdot x_0 + x_2 \cdot \overline{x_1}$

Una rete fatta così si sintetizza come segue:



Questa è una rete **in forma SP**, **non canonica**. Infatti, ogni uscita è la somma del prodotto di **qualche** variabile di ingresso, diretta o negata. La posso realizzare con **3 AND a 2 ingressi, 2 OR a 2 ingressi.**

Osserviamo che meno porte di così non possiamo metterne, **a meno di non uscire dal modello SP a due livelli di logica**. Infatti, si osserva facilmente che $z_1 = x_2 \oplus x_1$, $z_0 = \overline{x_1} \cdot (x_2 + x_0)$. Entrambe sono realizzazioni di costo minore, ma non sono in forma SP.

Il modo in cui, partendo da una sintesi fatta secondo il modello strutturale universale, siamo arrivati ad una sintesi di costo minore, **è artigianale**. Vediamo di introdurre un **metodo formale** per ricavare la sintesi a costo minimo di una rete.

5.1.1 Esercizio (per casa)

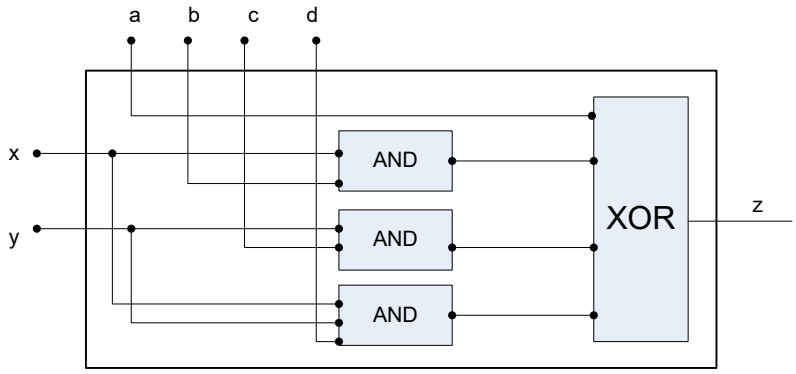
Si consideri la rete disegnata in figura, con 2 ingressi (x, y), un'uscita (z), e 4 *variabili di comando* *a, b, c, d*. Tale rete implementa una legge $f(x,y)$ diversa a seconda del valore delle variabili di comando.

x	y	z
0	0	f(0,0)
0	1	f(0,1)
1	0	f(1,0)
1	1	f(1,1)

a) Scrivere l'espressione algebrica che lega z agli ingressi e alle variabili di comando

b) Manipolando l'espressione trovata al punto precedente, calcolare *a, b, c, d* in modo da implementare una generica funzione $f(x,y)$ nota (assumendo, cioè, di conoscere $f(0,0), f(1,0), f(0,1), f(1,1)$).

c) calcolare *a, b, c, d* per i *casi particolari* a) $f(x,y) = \overline{xy}$, b) $f(x,y) = x\overline{y}$,



Soluzione

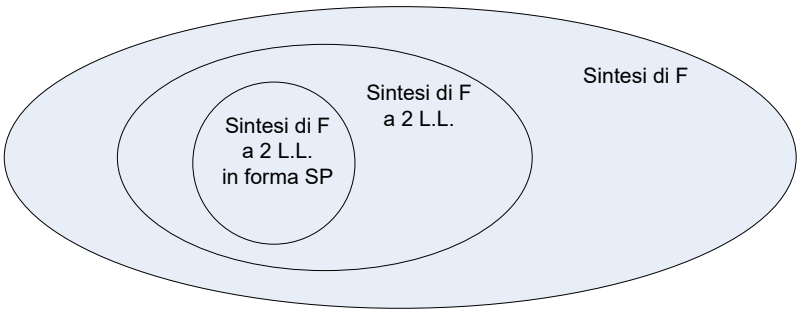
6 Sintesi di reti in forma SP a costo minimo

Decidiamo cosa voglia dire "costo". Esistono due criteri:

- criterio di costo **a porte:** ogni porta conta per un'unità di costo.
- criterio di costo **a diodi**: ogni **ingresso** conta per un'unità di costo.

Quindi, una porta AND a 2 ingressi è equivalente ad una porta AND a 10 ingressi in accordo al primo criterio, costa cinque volte meno in accordo al secondo.
Questi criteri di costo, ora che i sistemi di elaborazione si fanno a componenti integrati, lasciano un po' il tempo che trovano.

Il metodo che proponiamo si applica a reti con **una uscita**. Se ho reti a più uscite, posso comunque usarlo per affrontare l'ottimizzazione separatamente uscita per uscita. Non è detto (in generale non è vero) che applicando questo metodo a ciascuna singola uscita si trovi la rete di costo minimo in assoluto (la giustapposizione di ottimi parziali non è necessariamente un ottimo globale).
Produce **reti a 2 livelli di logica in forma SP.** Non è detto che quella che viene trovata sia la rete **di costo minimo in assoluto**. Infatti, possono esistere reti a 2 livelli di logica in forma diversa dalla forma SP che costano meno (lo vedremo), e possono anche esistere reti **a più di due livelli di logica** che costano meno. Trovare la rete di costo minimo in assoluto è però troppo complesso. Inoltre, le reti a due livelli di logica sono più veloci, e – lo vedremo più in là – presentano un miglior comportamento rispetto ai transitori. Quindi, tanto vale restare su quelle.



Punto di partenza: risultato dovuto a **Shannon**, "è sempre possibile scrivere **qualunque** legge F di una rete combinatoria come **somma di prodotti degli ingressi (diretti o negati)**". Ne abbiamo già avuto evidenza quando abbiamo parlato del multiplexer, peraltro.
Data una legge $z = f(x_{N-1}, \ldots, x_0)$, posso infatti scriverne l'**espansione di Shannon** come segue:

$$
\begin{aligned}
z = {} & f(0,...,0,0)\cdot \overline{x_{N-1}}\cdot \overline{x_{N-2}}\cdot ...\cdot \overline{x_1}\cdot \overline{x_0} \\
& + f(0,...,0,1)\cdot \overline{x_{N-1}}\cdot \overline{x_{N-2}}\cdot ...\cdot \overline{x_1}\cdot x_0 \\
& ... \\
& + f(1,...,1,0)\cdot x_{N-1}\cdot x_{N-2}\cdot ...\cdot x_1\cdot \overline{x_0} \\
& + f(1,...,1,1)\cdot x_{N-1}\cdot x_{N-2}\cdot ...\cdot x_1\cdot x_0
\end{aligned}
$$

Si osservi che la sintesi di una rete combinatoria a partire dall'espansione di Shannon si ottiene connettendo generatori di costante agli ingressi di un multiplexer, proprio come abbiamo visto.

Dall'espansione di Shannon della legge, posso ottenere la **forma canonica SP** osservando che:

- se $f(x_{N-1},\dots,x_0)=0$, dato che $0\cdot\alpha=0$, tutto il termine corrispondente sulla riga vale 0. Allora, visto che $0+\alpha=\alpha$, posso togliere l'intera riga
- se $f(x_{N-1},\dots,x_0)=1$, visto che $1\cdot\alpha=\alpha$, posso togliere uno dei fattori dal prodotto.

Un po' di nomenclatura:

- forma **SP** perché *z* è ottenuta come somma di prodotti
- forma **canonica** perché ogni prodotto ha come fattori **tutti gli ingressi** diretti o negati
- ciascuno dei termini della somma si chiama **mintermine**, e corrisponde ad uno stato di ingresso riconosciuto dalla rete.

Prendiamo come esempio la seguente tabella di verità di una rete a 4 ingressi ed 1 uscita.

x_3	x_2	x_1	x_0	z_0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Espansione di Shannon

$$
\begin{aligned}
z = {} & 0\cdot \overline{x_3}\cdot \overline{x_2}\cdot \overline{x_1}\cdot \overline{x_0} \\
& + 1\cdot \overline{x_3}\cdot \overline{x_2}\cdot \overline{x_1}\cdot x_0 \\
& ... \\
& + 0\cdot x_3\cdot x_2\cdot x_1\cdot \overline{x_0} \\
& + 0\cdot x_3\cdot x_2\cdot x_1\cdot x_0
\end{aligned}
$$

Forma canonica SP **(lista dei mintermini)**

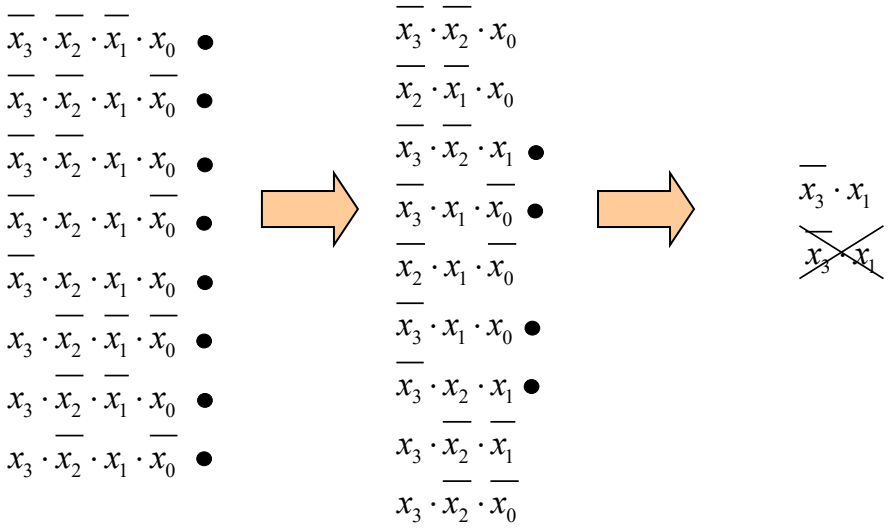
$$
\begin{aligned}
z = {} & \overline{x_3}\cdot \overline{x_2}\cdot \overline{x_1}\cdot x_0 \\
& + \overline{x_3}\cdot \overline{x_2}\cdot x_1\cdot \overline{x_0} \\
& + \overline{x_3}\cdot \overline{x_2}\cdot x_1\cdot x_0 \\
& + \overline{x_3}\cdot x_2\cdot x_1\cdot \overline{x_0} \\
& + \overline{x_3}\cdot x_2\cdot x_1\cdot x_0 \\
& + x_3\cdot \overline{x_2}\cdot \overline{x_1}\cdot \overline{x_0} \\
& + x_3\cdot \overline{x_2}\cdot \overline{x_1}\cdot x_0 \\
& + x_3\cdot \overline{x_2}\cdot x_1\cdot \overline{x_0}
\end{aligned}
$$

Cerchiamo di andare verso una soluzione di costo minore. Partendo dalla lista dei mintermini, applico **esaustivamente** le due seguenti regole:

$$
\begin{cases}
\alpha x + \alpha\overline{x} = \alpha x + \alpha\overline{x} + \alpha \\
\alpha + \alpha = \alpha
\end{cases}
$$

- La prima legge consente di **fondere i mintermini**. Dati due termini che differiscono per **una sola variabile**, che è diretta in un caso e negata nell'altro, posso produrre un termine che contiene ciò che è a comune tra i due mintermini che fondono.
- La seconda legge ci ricorda di **non inserire duplicati.**

Operativamente, si prende la lista dei mintermini (chiamiamola k_0) e si controllano tutte le coppie possibili per vedere se ce n'è qualcuna che verifica la prima regola. Nel caso, si aggiunge un nuovo termine ad un'altra lista (chiamiamola k_1), a meno che non ci sia già. Finito il procedimento, si fa lo stesso sulla lista k_1 per produrre la lista k_2, e si va avanti finché è possibile. Per un motivo che sarà chiaro tra un attimo, mi conviene **marcare** le coppie di termini che fondono ad ogni passo.



Sono partito da $z = k_0$ (cioè, detto meglio, **somma di tutti i mintermini della lista k_0)**, posso comunque scrivere $z = k_0 + k_1 + k_2$. Lo posso fare perché sto sommando termini che *comunque* sono veri quando sono veri altri termini già contenuti in k_0.

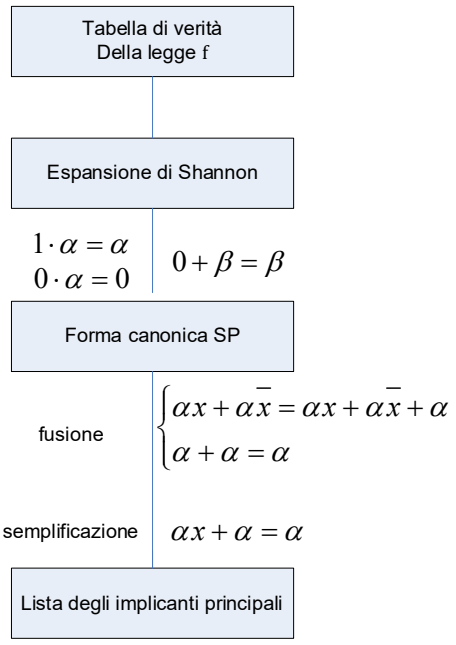
La lista che ottengo, sempre in forma SP, si chiama **lista degli implicanti**. Ciascun termine di questa lista è detto **implicante**. Un implicante è il prodotto di **alcune** variabili di ingresso dirette o negate che riconosce *alcuni* stati di ingresso. Un mintermine è un caso particolare di implicante.

La lista di implicanti deve a questo punto essere **semplificata**. Lo si fa applicando esaustivamente la legge $\boldsymbol{\alpha x + \alpha = \alpha}$**. Ciò equivale a togliere <u>tutti</u> gli implicanti che hanno prodotto qualcosa per fusione**. Per questo motivo mi è convenuto marcarli mentre producevo la lista.

Il risultato è una lista di implicanti ridotta, detta **lista degli implicanti principali**. Un implicante è detto principale se non è in grado di fondere con nessun altro implicante.

Nel nostro caso, si ottiene:

$$z = \overline{x_3}\cdot x_1 + \overline{x_3}\cdot\overline{x_2}\cdot x_0 + \overline{x_2}\cdot\overline{x_1}\cdot x_0 + \overline{x_2}\cdot x_1\cdot\overline{x_0} + x_3\cdot\overline{x_2}\cdot\overline{x_1} + x_3\cdot\overline{x_2}\cdot\overline{x_0}$$



Riepilogando: ecco il procedimento per arrivare alla lista degli implicanti principali.

Precisazione Non è vero, in generale, che **qualunque** prodotto di variabili dirette o negate è un implicante (o un mintermine, se ce le metto tutte). Qualunque **prodotto derivato dall'espansione di Shannon** di una legge di corrispondenza f è un **mintermine per quella legge**, qualunque prodotto derivato dalla fusione di mintermini è un **implicante per quella legge**.

Un mintermine vale 1 in corrispondenza di **uno ed un solo** stato di ingresso riconosciuto dalla legge f. Un **implicante** vale 1 in corrispondenza di **qualche** stato di ingresso riconosciuto dalla legge f.

Qualunque sia la legge f di partenza (cioè la tabella di verità di cui sto affrontando la sintesi), la lista degli implicanti principali produce una sintesi che costa meno ($\leq$) della forma canonica SP. Contiene infatti meno ($\leq$) termini, e normalmente con meno ($\leq$) ingressi. È lecito quindi chiedersi se la lista degli IP costituisca proprio la realizzazione SP a costo minimo che sto cercando. In generale, questo **non è vero**. Infatti, la lista degli IP potrebbe essere **ridondante**.

Lista di copertura per la funzione *f*: lista di implicanti, la cui somma è una forma SP per *f*. Ad esempio:

- La lista dei mintermini:
- La lista degli implicanti (cioè quella ottenuta prima della semplificazione):
- La lista degli implicanti principali;

sono tutte liste di copertura.

Lista di copertura **non ridondante** per *f*: tale che, se tolgo un elemento dalla lista, smette di essere una lista di copertura. Ad esempio:

$$z = \overline{x_3} \cdot x_1 + \overline{x_3} \cdot \overline{x_2} \cdot x_0 + \overline{x_2} \cdot \overline{x_1} \cdot x_0 + \overline{x_2} \cdot x_1 \cdot \overline{x_0} + x_3 \cdot \overline{x_2} \cdot \overline{x_1} + x_3 \cdot \overline{x_2} \cdot \overline{x_0}$$

È una lista ridondante (lo vedremo più avanti).

La lista dei mintermini è una lista **non ridondante**. Infatti, se ne tolgo uno, ho uno stato di ingresso riconosciuto da *f* che non è coperto (quindi sto sintetizzando una legge diversa da *f*).

È necessario dotarsi di un metodo per generare una lista di copertura non ridondante a partire dalla lista degli IP. Vediamo alcune note aggiuntive.

6.1 Alcune note aggiuntive

6.1.1 Notazione compatta

In letteratura si trova spesso, al posto della tabella di verità (che è lunga da scrivere), uno stile di descrizione equivalente, che consiste nell'elencare i mintermini come numeri in base 2. Ad esempio, per la tabella di verità usata prima, avremmo:

$$z = \Sigma m(1,2,3,6,7,8,9,10)$$

Dove "Σm" indica "somma di mintermini", ed i numeri riportati di seguito sono i valori in base 2 corrispondenti agli stati di ingresso riconosciuti.

6.1.2 Metodo di Quine-McCluskey

Il metodo di Q-McC consente di fare il lavoro del paragrafo precedente in maniera più efficiente. Si parte dalla lista degli stati di ingresso **riconosciuti**, e si raggruppano **per numero crescente di 1, separandoli in partizioni distinte**. Il numero massimo di partizioni è $N + 1$ (non è detto che ci siano tutte, dipende da come è fatta la legge f).

x_3	x_2	x_1	x_0	z_0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



	x_3	x_2	x_1	x_0
x	1	0	0	0
x	0	0	0	1
x	0	0	1	0
x	0	0	1	1
x	1	0	0	1
x	1	0	1	0
x	0	1	1	0
x	0	1	1	1



	x_3	x_2	x_1	x_0
	1	0	0	-
	1	0	-	0
	0	0	-	1
	-	0	0	1
x	0	0	1	-
	-	0	1	0
x	0	-	1	0
x	0	-	1	1
x	0	1	1	-



x_3	x_2	x_1	x_0
0	-	1	-
~~0~~	~~-~~	~~1~~	~~-~~

Il matching va fatto soltanto **tra partizioni adiacenti**, in quanto è possibile soltanto tra esse. Quando due implicanti differiscono per una sola variabile, **fondono**. Quindi, li segno in un'altra tabella. Continuo finché posso, ricordandomi di **omettere i duplicati**.

Alla fine, gli implicanti che non hanno generato fusione sono quelli **principali.**

$$z = \overline{x_3} \cdot x_1 + x_3 \cdot \overline{x_2} \cdot \overline{x_1} + x_3 \cdot \overline{x_2} \cdot \overline{x_0} + \overline{x_3} \cdot \overline{x_2} \cdot x_0 + \overline{x_2} \cdot \overline{x_1} \cdot x_0 + \overline{x_2} \cdot x_1 \cdot \overline{x_0}$$

Non si fa fatica a capire che questo metodo è identico a quello visto prima, soltanto è più veloce, perché evita confronti inutili.

Da quanto appena visto appare chiaro che un aspetto importante dell'andare verso una sintesi di costo minore consiste nell'essere in grado di cercare agevolmente stati di ingresso (o gruppi di stati di ingresso) **adiacenti** riconosciuti dalla rete. Un modo agevole per fare questo, che però è limitato a reti molto semplici, è descritto di seguito.

6.2 Mappe di Karnaugh

Le **mappe di Karnaugh** sono un altro modo di descrivere le reti combinatorie, che presenta alcuni vantaggi rispetto alle tabelle di verità. Il vantaggio principale è dato dal fatto che rendono più semplice svolgere il procedimento di sintesi di costo minimo. Una mappa di Karnaugh per una rete ad N ingressi è una matrice di 2^N celle. Ciascuna cella contiene il valore della variabile di uscita per un particolare stato di ingresso.

x_0	0	1
	1	0

$N = 1$. Mappa di Karnaugh per un invertitore.

- Ogni cella è individuata da uno stato di ingresso, che ne costituisce le **coordinate**
- Ogni cella contiene il valore dell'uscita corrispondente a quello stato di ingresso.

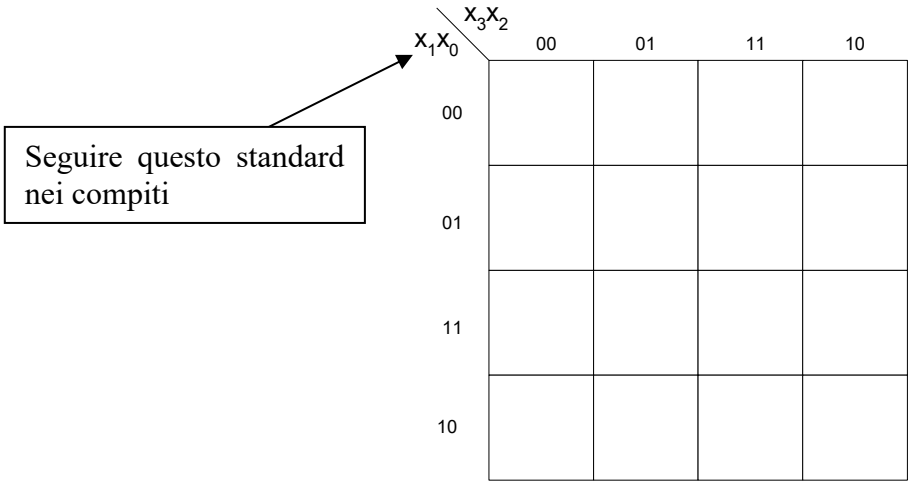
x_0 \ x_1	0	1
0	0	0
1	0	1

$N = 2$. Mappa di Karnaugh per un AND.

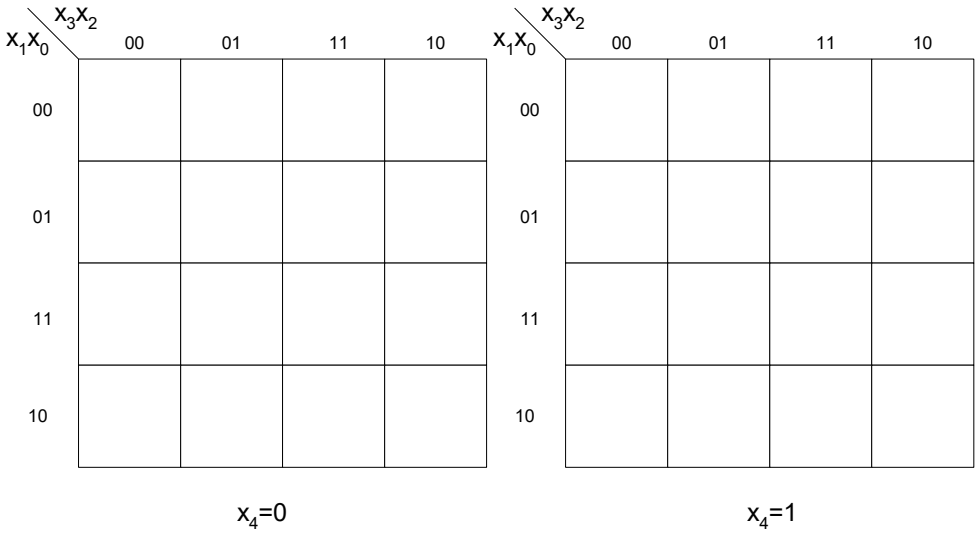
x_0 \ x_2x_1	00	01	11	10
0	0	1	1	0
1	1	1	1	1

$N = 3$. Attenzione a come scrivo le coordinate. Devo scriverle in modo che celle **contigue sulla mappa abbiano coordinate adiacenti (e viceversa)**.

Le mappe di Karnaugh vanno immaginate disegnate su superfici "**sferiche**". Ciò che si trova all'estrema destra è contiguo a ciò che si trova all'estrema sinistra. Ciò che si trova all'estremo superiore è contiguo a ciò che si trova all'estremo inferiore.



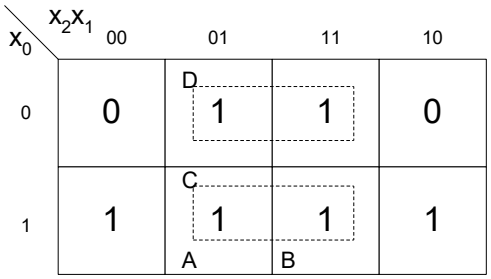
Mappa di Karnaugh di **ordine 4.** Il difetto delle mappe di Karnaugh è che non è possibile mettere **più di due variabili su un asse** mantenendo la proprietà vista sopra (cioè che **tutte** e **sole** le celle le cui coordinate differiscono per un bit siano adiacenti sulla mappa – fare la prova per credere).
Tutt'al più, è possibile usare mappe di ordine 5, usando due mappe di ordine 4 e facendo finta che siano sovrapposte:



Lavorare in queste condizioni, comunque, è estremamente difficile. Quindi, le mappe di Karnaugh si usano per reti fino a 4 ingressi. Se si devono cercare gli implicanti principali per reti con più ingressi, conviene di gran lunga usare il metodo di Quine-McCluskey.

Diamo qualche definizione:

- **sottocubo di ordine 1**: una casella che contiene un 1, corrispondente quindi ad uno stato di ingresso riconosciuto dalla rete.
- **Coordinate** di un sottocubo di ordine 1: stato di ingresso riconosciuto dalla rete corrispondente alla casella.
- **Adiacenza tra sottocubi di ordine 1**: due sottocubi di ordine 1 si dicono adiacenti se differiscono tra loro per una sola coordinata (avendo le altre *N*-1 coordinate identiche).



Ad esempio: A e B sono due sottocubi di ordine 1, le cui coordinate sono rispettivamente:

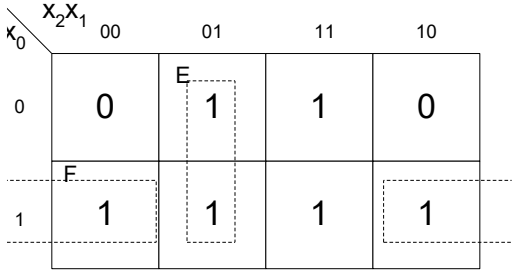
	x_2	x_1	x_0
A	0	1	1
B	1	1	1

Come tali sono adiacenti.

- Sottocubo di ordine 2: sottocubo costituito da **sottocubi adiacenti di ordine 1**. Ad esempio, C è un sottocubo di ordine 2. Le sue coordinate sono: x_2=- x_1=1 x_0=1
- Il sottocubo di ordine 2 C **copre** i sottocubi di ordine 1 A e B.
- **Adiacenza tra sottocubi di ordine 2:** due sottocubi di ordine 2 si dicono adiacenti se differiscono tra loro per una sola coordinata (avendo le altre $N-2$ coordinate identiche). Ad esempio, C e D sono sottocubi di ordine 2 adiacenti, di coordinate

	x_2	x_1	x_0
C	-	1	1
D	-	1	0

Giusto per capire meglio:

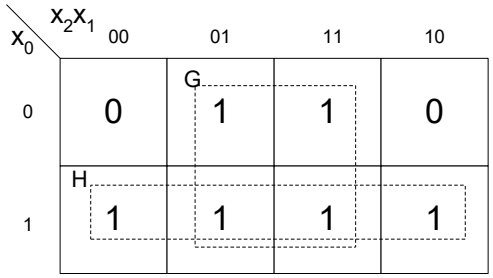


Anche E ed F sono sottocubi di ordine 2, le cui coordinate sono:

	x_2	x_1	x_0
E	0	1	-
F	-	0	1

Posso, ovviamente, iterare il ragionamento per costruire sottocubi di ordine 4 (ed eventualmente più grandi, anche se non con questo esempio).

Ad esempio, i sottocubi C ed F, entrambi di ordine 2, sono adiacenti. Quindi fondono a generare un sottocubo di ordine 4, che chiamiamo H. Anche C e D fondono per generare un sottocubo di ordine 4, il sottocubo G. I sottocubi G ed H coprono tutti gli stati riconosciuti dalla rete.

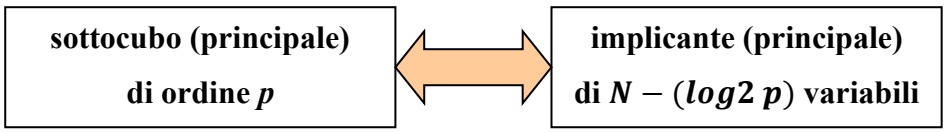


	x_2	x_1	x_0
G	-	1	-
H	-	-	1

Continuiamo con le definizioni:

- sottocubo **principale**: un sottocubo tale per cui **non esiste nessun sottocubo più grande che lo copre completamente**. Nel nostro caso, G ed H.
- **Lista di copertura:** insieme (qualunque) di sottocubi che coprono tutti i sottocubi di ordine 1, che includono, cioè, tutti le caselle della mappa che contengono un 1. G ed H sono una l.c.
- **Lista di copertura non ridondante**: tale che se tolgo un sottocubo non è più una lista di copertura.

Il bello delle mappe di Karnaugh è che esiste una **corrispondenza biunivoca tra implicanti principali della legge F e sottocubi principali nella mappa di Karnaugh.**



In particolare, da un sottocubo principale di ordine p posso ricavare un implicante principale di ordine $N - log2\ p$, cioè prodotto di $N - log2\ p$ variabili di ingresso (dirette o negate). Un sottocubo di ordine 1 è un mintermine. Per trovare l'implicante basta

- Guardare le coordinate del sottocubo
- Mettere una variabile diretta o negata a seconda che nella coordinata ci sia 1 o 0.

Cercare i sottocubi principali è facile: si fa ad occhio, con l'algoritmo che segue.

6.2.1 Algoritmo di ricerca dei sottocubi principali mediante mappe di Karnaugh

Parto dai sottocubi **di ordine più grande** che trovo sulla mappa, e li segno **tutti**. Tali sottocubi, per definizione, non sono contenuti in nessun sottocubo di ordine maggiore, e quindi sono senz'altro principali.

- domanda: l'insieme dei sottocubi che sto considerando basta a coprire tutta la mappa?
    - Se sì, ho finito.

Se no, passo ai sottocubi di ordine inferiore. Li devo **considerare tutti**, tranne quelli che sono (ovviamente) già coperti da quelli che ho già considerato al passo precedente.

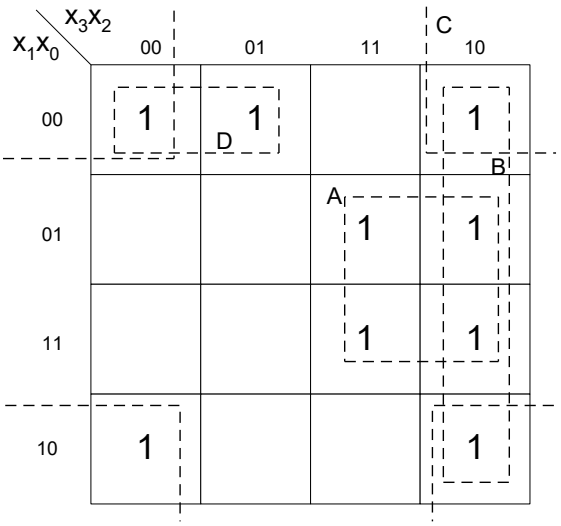
- domanda: l'insieme dei sottocubi che sto considerando basta a coprire tutta la mappa?
    - Se sì, ho finito.

Se no, passo ai sottocubi di ordine inferiore e riparto.

L'algoritmo **termina sicuramente**, perché, male che vada, avrò bisogno dei sottocubi di ordine 1 per coprire l'intera mappa.

Esempio:

Consideriamo questa mappa di Karnaugh, e cerchiamo gli implicanti principali.



sottocubi principali

	x_3	x_2	x_1	x_0
A	1	-	-	1
B	1	0	-	-
C	-	0	-	0
D	**0**	**-**	**0**	**0**

implicanti principali:

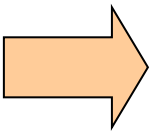
$$z = x_3 \cdot x_0 + \overline{x_3} \cdot x_2 + \overline{x_2} \cdot \overline{x_0} + \overline{x_3} \cdot \overline{x_1} \cdot \overline{x_0}$$

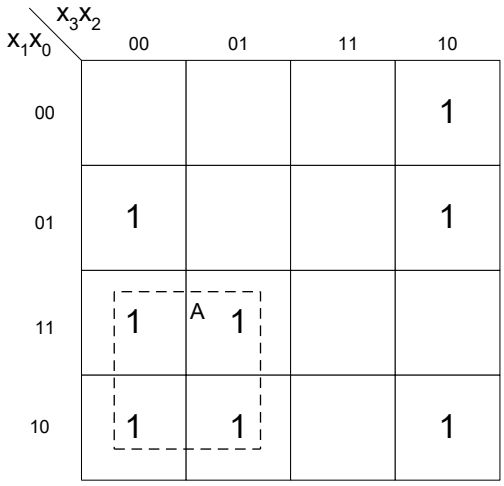
In questo caso, **non mi bastano** i tre sottocubi di ordine 4 A,B,C, perché resta scoperto uno stato riconosciuto dalla rete (0100). Devo considerare anche **tutti** i sottocubi di ordine 2, **tranne quelli già coperti** da sottocubi di ordine 4. Quindi devo aggiungere anche D. In questo modo ottengo la lista degli implicanti principali.

Esercizio:

Data la seguente tabella di verità (sempre la stessa usata per gli altri esempi, non stare a riscriverla), trovare la lista degli implicanti principali.

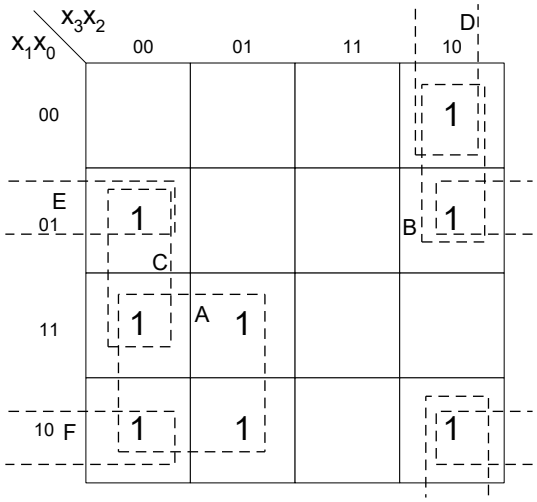
x_3	x_2	x_1	x_0	z
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0





- Passo 1: guardo tutti gli implicanti di ordine 4. Ce n'è uno solo, A, che **non** copre tutta la rete.
- Passo 2: considero **tutti** gli implicanti di ordine 2 non interamente coperti da A.

Attenzione: l'errore tipico è quello di fermarsi con il passo 2 non appena si è incluso un numero di sottocubi tale da coprire tutti gli 1 della rete. Se un passo è da compiere, **lo devo portare a termine**, prendendo **tutti** i sottocubi **che non sono già coperti da altri sottocubi <u>di ordine maggiore</u>**.



A questo punto, la lista degli implicanti principali la compilo andando a guardare le coordinate dei sottocubi che ho utilizzato:

$$z = \overline{x_3} \cdot x_1 + x_3 \cdot \overline{x_2} \cdot \overline{x_1} + \overline{x_3} \cdot \overline{x_2} \cdot x_0 + x_3 \cdot \overline{x_2} \cdot \overline{x_0} + \overline{x_2} \cdot \overline{x_1} \cdot x_0 + \overline{x_2} \cdot x_1 \cdot \overline{x_0}$$

6.2.2 Ricerca delle liste di copertura non ridondanti

Abbiamo introdotto il concetto di **lista di copertura**: lista di implicanti, la cui somma è una forma SP per la funzione *f*.

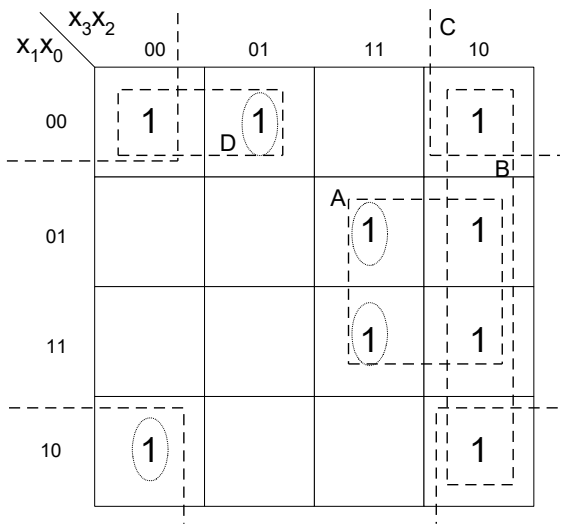
- lista dei mintermini (forma *canonica* SP)

- lista degli implicanti principali

Una lista di copertura è **non ridondante** se quando tolgo un termine non è più tale, se cioè lascia scoperti degli stati riconosciuti dalla rete.

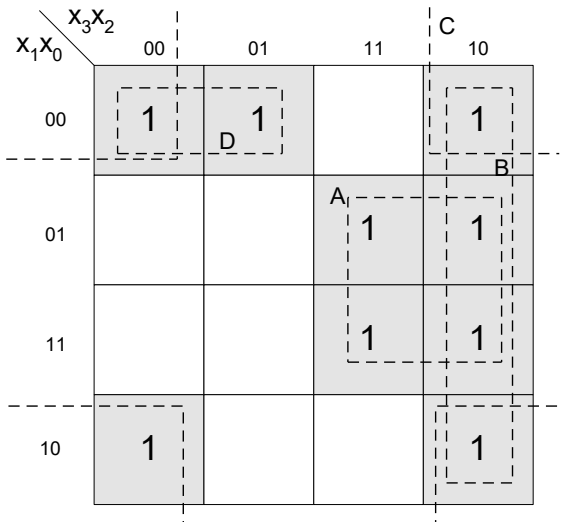
Se una lista di copertura è **ridondante**, potrei toglierne uno (o più d'uno) ed ottenere ancora una lista di copertura. Togliere un implicante significa togliere una porta AND ed un ingresso all'OR finale, quindi significa andare verso una rete **di costo minore,** quale che sia il criterio di costo adottato.

La lista degli implicanti principali può essere ridondante. Quindi, devo fornire un algoritmo per ridurla ad una forma non ridondante. L'algoritmo è il seguente: si parte dalla mappa di Karnaugh sulla quale si sono disegnati gli implicanti (i sottocubi) principali e si nota quanto segue:



a) esistono alcuni sottocubi che sono **gli unici a coprire un dato sottocubo di ordine 1**. Quei sottocubi non possono essere tolti, se vogliamo coprire tutti gli stati riconosciuti dalla rete. Tali sottocubi principali sono detti **essenziali,** e costituiscono il **cuore della mappa**.

Come si fa a trovarli sulla mappa? Basta guardare se ci sono degli 1 cerchiati una volta sola. Nel nostro esempio, i sottocubi A, C, D, sono **essenziali.**

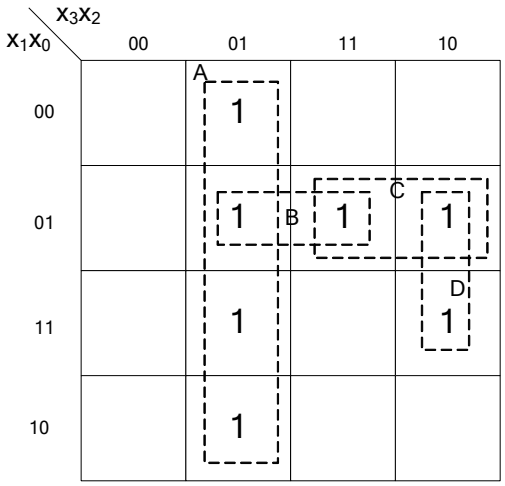


b) Si spuntano tutti i sottocubi di ordine 1 contenuti in sottocubi principali **essenziali**. Se nel fare questo si copre completamente **anche** qualche altro sottocubo (B, nel nostro caso) vuol dire che quel sottocubo è ridondante. Infatti, il suo mestiere è già svolto dai sottocubi principali essenziali. Questi sottocubi si chiamano **assolutamente eliminabili**, e **non devono comparire** in una lista di copertura irridondante.

La definizione è: un sottocubo principale è **assolutamente eliminabile** se riconosce **soltanto** stati di ingresso già riconosciuti da implicanti principali **essenziali.**

In questo esempio non restano altri sottocubi. Quindi la lista di copertura **non ridondante** per la rete di cui sopra è data dai sottocubi A, C, D. In realtà possono rimanere fuori altri sottocubi, che sono detti **semplicemente eliminabili**.

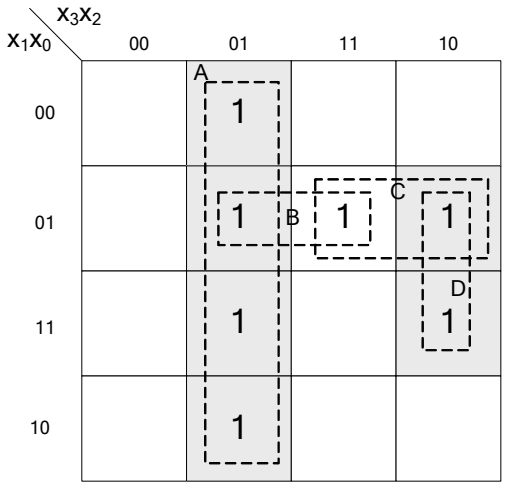
Vediamo un esempio più complesso (sul libro):



Ricerca dei **sottocubi principali:** ce n'è uno di ordine 4 (A), e poi devo passare a quelli di ordine 2 (B,C,D)

Sottocubi essenziali: A e D, che formano il c**uore della mappa.**

Qualunque lista di copertura non ridondante **deve** contenere A e D



Si spuntano tutti i sottocubi di ordine 1 contenuti in sottocubi principali **essenziali**. In questo caso, **non abbiamo coperto interamente nessun altro sottocubo principale**. Quindi, non esistono sottocubi **assolutamente eliminabili**.

I sottocubi che rimangono (B e C) si chiamano **semplicemente eliminabili**.

A questo punto, per ottenere la lista di costo minimo devo:

c) generare **tutte le possibili liste di copertura non ridondanti** che

- includono tutti i sottocubi principali essenziali
- non includono nessun sottocubo principale assolutamente eliminabile
- includono un sottoinsieme di sottocubi principali semplicemente eliminabili

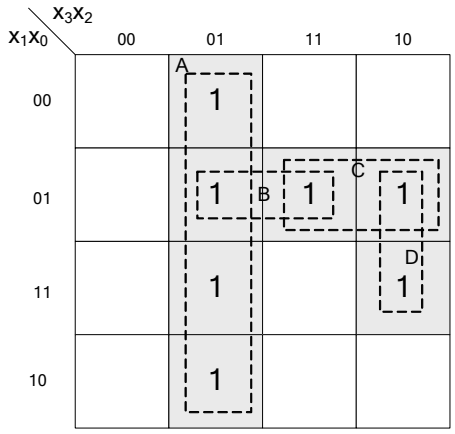
d) tra tutte queste, valuto quella di costo minimo applicando il criterio di costo voluto.

La strada per andare avanti è la seguente: prendo in considerazione un **qualunque** sottocubo semplicemente eliminabile, e formulo **due ipotesi alternative**:

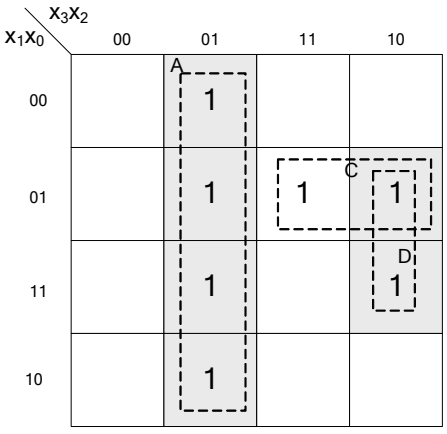
1. lo considero **come se fosse essenziale** (è chiaro che **non lo è, in realtà**), e quindi **lo aggiungo alla lista di copertura**. In conseguenza di questo, qualche **altro sottocubo** diventa assolutamente eliminabile
2. lo considero **come se fosse assolutamente eliminabile** (è chiaro che **non lo è, in realtà**), e quindi **lo tolgo da qualunque lista di copertura**. Come conseguenza di questo, qualche **altro sottocubo** diventa essenziale.

Finché ho sottocubi semplicemente eliminabili, vado avanti tenendo conto di tutti i possibili scenari. Ad ogni nuovo passo, formulo due nuove ipotesi relativamente ad un sottocubo semplicemente eliminabile. Partendo dal cuore della mappa, posso creare un **albero di decisioni binario**, alle foglie del quale ci sono **tutte** le possibili **liste di copertura non ridondanti**. Tra tutte queste, sceglierò quella di costo minimo.

Nel nostro caso, proviamo a prendere in esame il sottocubo B:

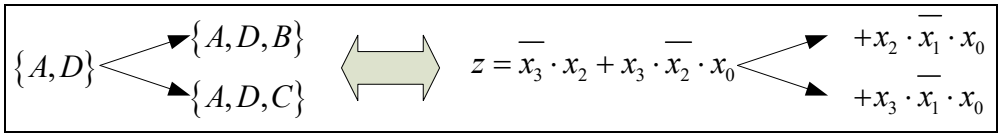


Se considero **B** come **essenziale, C diventa assolutamente eliminabile**, in quanto interamente coperto da B e D.



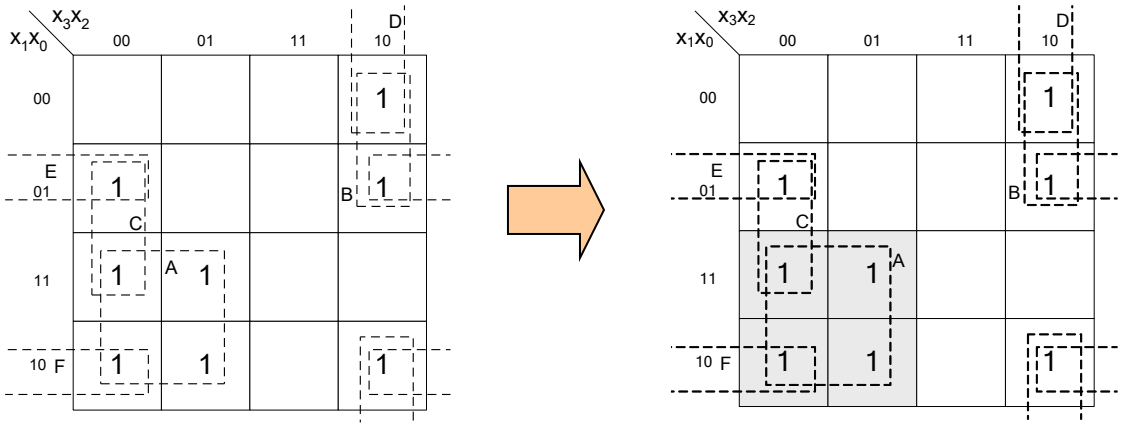
Se considero **B** come **assolutamente eliminabile, C diventa essenziale**, in quanto è l'unico sottocubo a coprire un sottocubo di ordine 1.

La scelta dà origine al seguente **albero binario**.



La sintesi di costo minimo si ottiene confrontando tutte le possibili realizzazioni ottenute in questa maniera, usando il criterio a diodi o a porte. Nel nostro caso, sono del tutto equivalenti.

Vediamo un esempio più complesso (sempre la tabella di verità scritta all'inizio):

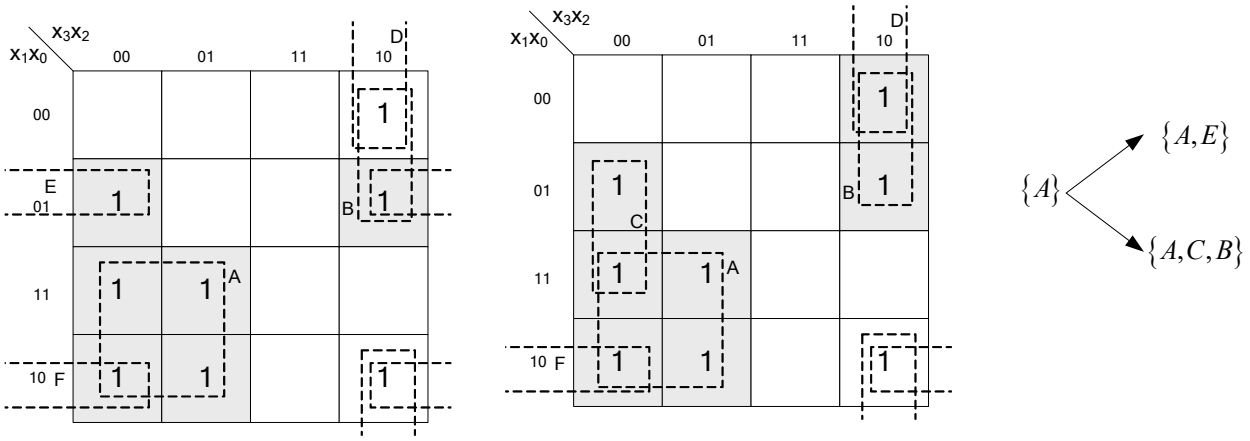


Sottocubi essenziali: soltanto **A**, che è il cuore della mappa.

Non ci sono sottocubi assolut. eliminabili. Tutti i sottocubi rimasti sono semplicemente eliminabili.

Consideriamo il sottocubo semplicemente eliminabile **E**, ed applichiamo l'algoritmo.

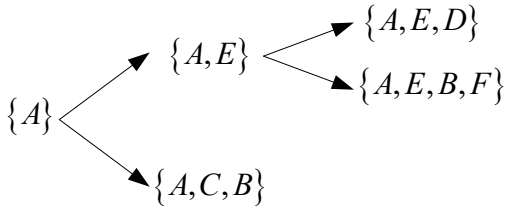
1. Se considero **E come se** fosse essenziale, **C** diventa assolutamente eliminabile
2. Se considero **E come se** fosse assolutamente eliminabile, **C** e **B** diventano essenziali.



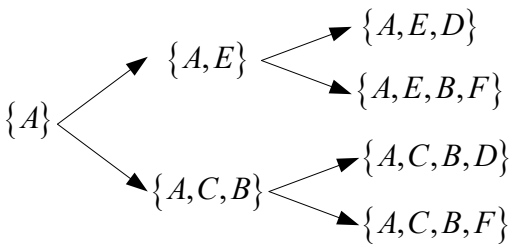
Andiamo avanti secondo la prima ipotesi, cioè che E sia **essenziale** (figura di sinistra). Tolto C, non ho più implicanti assolutamente eliminabili. Devo quindi fare un'altra ipotesi. La faccio su D.

- se D è essenziale, B ed F sono assolutamente eliminabili
- se D è assolutamente eliminabile, B ed F sono essenziali

In entrambi i casi, ho trovato una lista di copertura.

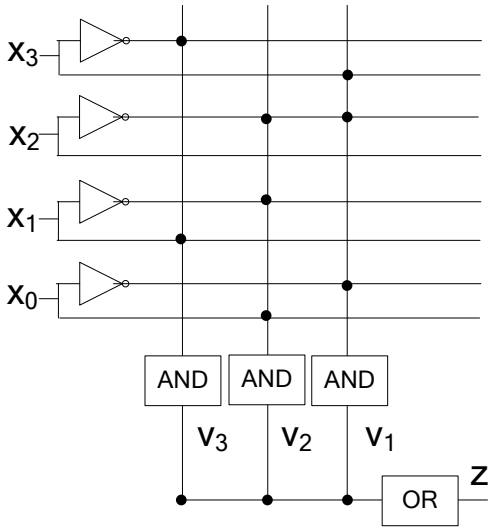


Mettiamoci nella seconda ipotesi, cioè che il cuore della mappa sia **A, C, B**. A questo punto, se D è essenziale, F è assolutamente eliminabile e viceversa.



A questo punto, ho **tutte le liste di copertura non ridondanti** per la mappa (le foglie dell'albero binario). Tra queste, c'è la lista di costo minimo. Per trovare quale sia, devo applicare il criterio di costo. In questo caso particolare, ho B, C, D, E, F che hanno tutti lo stesso costo (sono infatti sotto-cubi dello stesso ordine), quindi si vede subito che la lista di copertura di costo minimo è $\{A, E, D\}$. Quindi:

$$z = \overline{x_3} \cdot x_1 + \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_3 \cdot \overline{x_2} \cdot \overline{x_0}$$



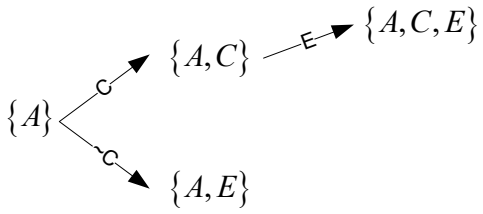
Quella scritta sopra è la **realizzazione di costo minimo** per la rete la cui tabella di verità abbiamo scritto all'inizio.

<u>Osservazione</u>: Dove è che intervengono i criteri di costo (a porte o a diodi)? **Nello scegliere quale, tra tutte le liste di copertura non ridondanti trovate**, sia quella di costo minore. Quindi, il procedimento per trovare le liste di copertura non ridondanti è identico, quale che sia il criterio di costo.

Data una rete con N ingressi, e scelta una realizzazione con N_s sottocubi, ciascuno di dimensione D_j, $1 \leq j \leq N_s$, le formule per i costi sono:

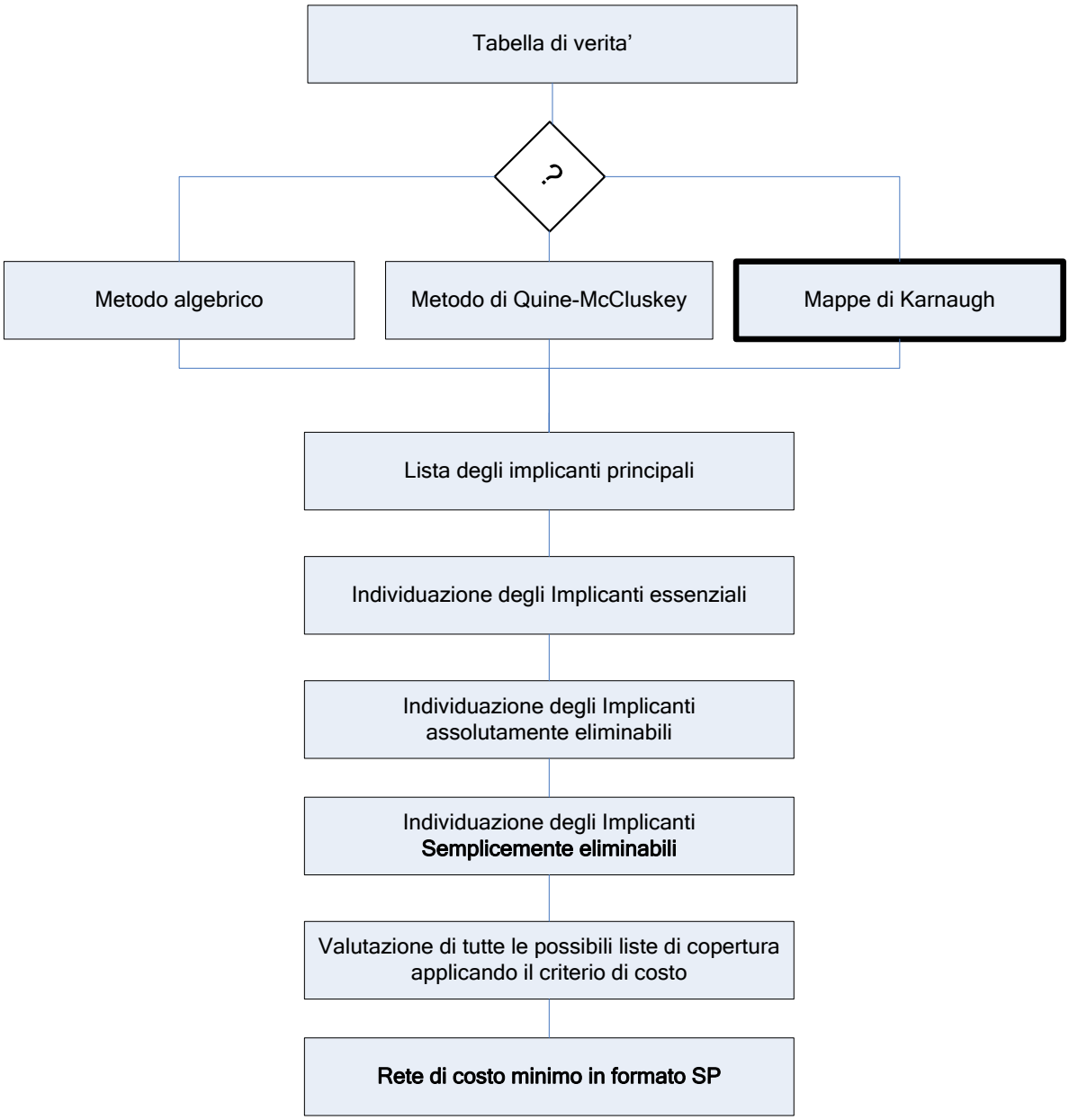
- costo a porte: $CP = N_s + 1$
- costo a diodi: $CD = N_s + \sum_{j=1}^{N_s} (N - log_2 D_j)$

Osservazione: Il risultato finale **non dipende** da quale sottocubo si considera volta per volta ad ogni passo dell'algoritmo. In tutti i casi, verranno prodotte tutte le possibili liste di copertura non ridondanti. Si deve, però, stare attenti, a far sì che **ad ogni passo** non si generino liste non ridondanti. Ad esempio:



In questo caso **non posso** inserire E dopo aver inserito C, altrimenti ottengo una lista ridondante.

6.2.3 Riepilogo – procedura per la sintesi a costo minimo SP



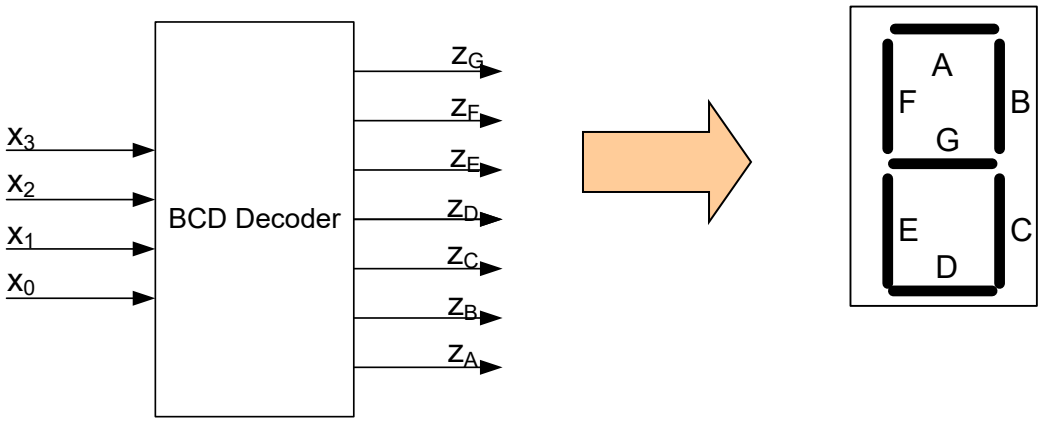
6.2.4 Riepilogo – definizione e classificazione di implicanti e sottocubi

Tipica domanda d'esame, alla quale bisogna saper rispondere.

Algebra	Definizione
Forma Canonica SP	Data una legge $z = f(x_{N-1}, \dots, x_0)$, la FC SP è quella che si ottiene dall'espansione di Shannon della legge, cioè: $$z = f(0,...,0,0) \cdot \overline{x_{N-1}} \cdot \overline{x_{N-2}} \cdot ... \cdot \overline{x_1} \cdot \overline{x_0}$$ $$+ f(0,...,0,1) \cdot \overline{x_{N-1}} \cdot \overline{x_{N-2}} \cdot ... \cdot \overline{x_1} \cdot x_0$$ $$...$$ $$+ f(1,...,1,0) \cdot x_{N-1} \cdot x_{N-2} \cdot ... \cdot x_1 \cdot \overline{x_0}$$ $$+ f(1,...,1,1) \cdot x_{N-1} \cdot x_{N-2} \cdot ... \cdot x_1 \cdot x_0$$ applicando le identità $0 + \alpha = \alpha$, $1 \cdot \alpha = \alpha$, $0 \cdot \alpha = 0$
Mintermine	Prodotto di **tutte** le variabili di ingresso dirette o negate, che compare in una **forma canonica SP** e riconosce **uno stato di ingresso**.
Implicante	Prodotto di **alcune** variabili di ingresso dirette o negate, che compare in una **forma SP** di una legge di corrispondenza. Si ottiene a partire dalla forma canonica SP applicando esaustivamente le seguenti regole: $$\begin{cases} \alpha x + \alpha \overline{x} = \alpha x + \alpha \overline{x} + \alpha \\ \alpha + \alpha = \alpha \end{cases}$$ Riconosce **alcuni stati di ingresso**.
Implicante Principale	Implicante che si ottiene dalla lista degli implicanti applicando esaustivamente la legge $\boldsymbol{\alpha x + \alpha = \alpha}$
I.P. Essenziale	Implicante che è **l'unico**, tra quelli principali, ad implicare un dato mintermine (a riconoscere uno stato di ingresso).
I.P. Assolut. Eliminabile	Implicante che riconosce solo stati di ingresso già riconosciuti da I.P. essenziali.
I.P. Semplic. Eliminabile	Implicante che riconosce solo stati di ingresso riconosciuti da altri I.P., almeno uno dei quali non riconosciuto da I.P. essenziali.

6.2.5 Esercizio – sintesi di leggi non completamente specificate - decodificatore BCD a 7 segmenti

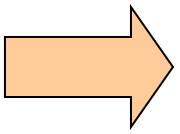
Un **decodificatore BCD a 7 segmenti** è una rete che ha 4 variabili di ingresso, che vanno interpretate come la **codifica in base 2** di una cifra decimale, e produce **7 uscite**, che vanno ad accendere i segmenti di un display a cristalli liquidi che illumina la cifra data in ingresso.



BCD sta per **Binary Coded Decimal**, ed è un modo per rappresentare i numeri naturali (ne vedremo altri quando parleremo di aritmetica), tipico delle calcolatrici tascabili con display LCD. Con tale tipo di rappresentazione, ciascuna cifra decimale viene rappresentata su 4 bit.

Abbiamo visto che la sintesi a costo minimo per una rete con *M* uscite si può affrontare **un'uscita alla volta**. Prendiamo a riferimento l'uscita z_E, e sintetizziamo la rete di costo minimo che la realizza.

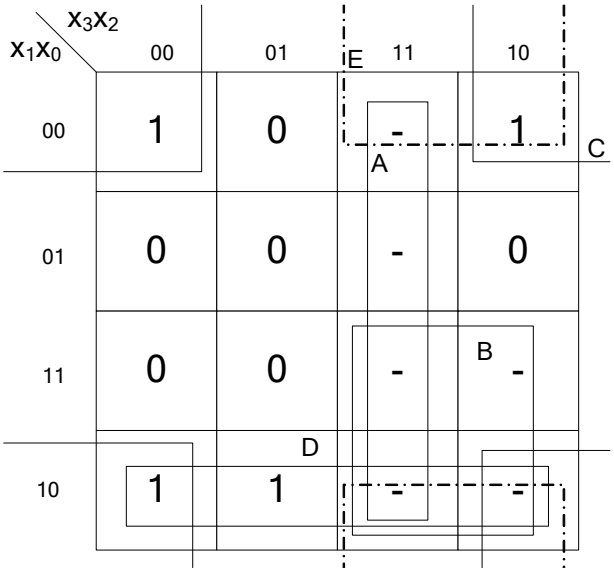
J	x_3	x_2	x_1	x_0	z_E
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
?	1	0	1	0	-
?	1	0	1	1	-
?	1	1	0	0	-
?	1	1	0	1	-
?	1	1	1	0	-
?	1	1	1	1	-



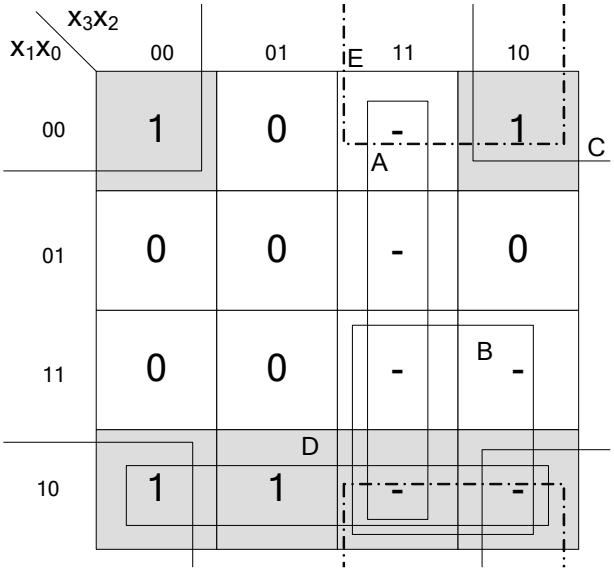
x_1x_0 \ x_3x_2	00	01	11	10
00	1	0	-	1
01	0	0	-	0
11	0	0	-	-
10	1	1	-	-

Si vede subito che **non mi interessa** cosa succeda quando in ingresso si presenta uno stato che che non è la codifica di una cifra in base 10, in quanto nelle ipotesi di funzionamento gli unici stati di ingresso possibili sono quelli relativi a cifre da 0 a 9. Quindi, **non ha senso chiedersi** quanto valga l'uscita, o meglio, **non ha senso volerlo decidere adesso**. Mi riservo di assegnare all'uscita **il valore che più mi converrà**, nell'ottica di ottenere la rete di **costo minimo**. Per dire questo, metto nella tabella di verità, e nella mappa di Karnaugh che le corrisponde, un **non specificato.**
Vediamo come si affronta la sintesi in questo caso.

Ricerca degli implicanti principali: considero i valori non specificati **come degli 1**, in quanto mi fa comodo ottenere implicanti più grandi.
Ce la faccio con 5 implicanti di ordine 4, proprio grazie al fatto di aver considerato i non specificati come 1.

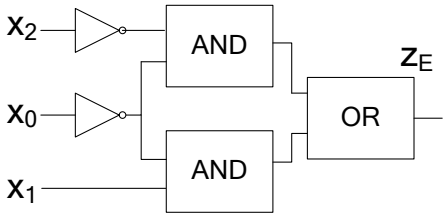


Ricerca degli implicanti essenziali: mi preoccupo di riconoscere soltanto gli stati effettivamente corrispondenti ad un 1 (che è come dire: tratto i non specificati come se fossero 0). Pertanto, **solo C e D** sono essenziali. A, B ed E sono assolutamente eliminabili.



La rete di costo minimo in forma SP che realizza l'uscita desiderata è, quindi:

$$z_E = \overline{x_2} \cdot \overline{x_0} + x_1 \cdot \overline{x_0}$$



6.2.6 Esercizio (per casa)

Data la seguente rete a 3 ingessi e due uscite:

a	b	c	z_1	z_0
0	0	0	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	0

a) Effettuare la sintesi a costo minimo delle due uscite, secondo il procedimento noto.

b) Discutere se la sintesi così ottenuta sia o meno la sintesi a costo minimo in assoluto, tra quelle in forma SP a due livelli di logica (eventualmente mostrare un controesempio). Rispondere, in particolare, alla seguente domanda: in una rete a più uscite, continua ad essere vero che la sintesi a costo minimo in assoluto contiene soltanto implicanti *principali* delle funzioni che calcolano le *singole* uscite?

Osservazione

Il metodo di sintesi ottima per reti a più uscite generalizza quello per reti ad una sola uscita. È discretamente più complicato da applicare a mano, e per questo non viene svolto a lezione. Diventa *molto* complesso nel caso in cui la legge per alcune uscite non sia completamente specificata.

Soluzione

6.2.7 Esercizio (per casa)

1) Dimostrare che, data una legge di corrispondenza F, una rete a due livelli di logica così costruita:

- ogni stato di ingresso riconosciuto da F è coperto da un numero *dispari* di prodotti
- ogni stato di ingresso *non* riconosciuto da F è coperto da un numero *pari* di prodotti
- l'uscita di tutti i prodotti va in ingresso ad uno XOR

È una sintesi di F (nota in letteratura con il nome di sintesi *ESOP, Exclusive Sum Of Products*).

x_0 \ x_2x_1	00	01	11	10
0	0	1	1	0
1	1	0	0	1

2) Data la mappa di Karnaugh di figura,

- calcolare la sintesi SP di costo minimo
- assumendo che il costo di una porta XOR sia identico a quello di una porta AND/OR (sia secondo il criterio a diodi che a porte), trovare una sintesi ESOP a costo minore

Soluzione

6.2.8 Esercizio (per casa)

Data la seguente tabella di verità che descrive una rete combinatoria a 4 ingressi ed 1 uscita,

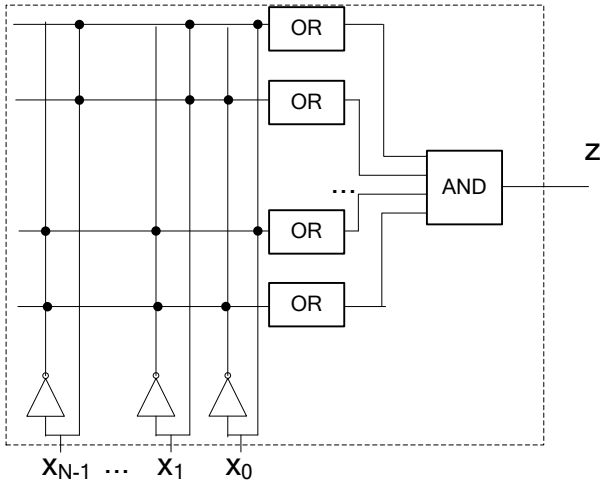
1) Scrivere la forma canonica SP per la rete
2) Individuare la lista degli implicanti principali usando il metodo di Quine-McCluskey
3) Disegnare la mappa di Karnaugh corrispondente alla tabella di verità
4) Individuare e classificare gli implicanti principali sulla mappa
5) Trovare tutte le liste di copertura irridondanti, e scegliere quella (o quelle) di costo minimo sia con il criterio di costo a porte che a diodi

x_3	x_2	x_1	x_0	z_0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Soluzione

7 Sintesi di reti in formato PS

La forma SP non è l'unica possibile per una rete combinatoria. Esiste anche – **sempre –** la possibilità di sintetizzare una rete in formato **PS,** cioè prodotto di somme. La sua realizzazione è data da uno schema del tutto identico a quelli già visti, ottenuto sostituendo le porte OR a quelle AND e viceversa.



Dal punto di vista algebrico, si dice che una legge è in **forma PS** se può essere scritta come:

$$z = S_1 \cdot S_2 \cdot \ldots \cdot S_k$$

I vari S_i sono **somme di variabili di ingresso dirette o negate**, e prendono il nome di **implicati**.

È possibile sviluppare tutta una teoria **duale** rispetto a quella delle reti in forma SP:

- forma canonica PS: quella in cui si hanno **somme di tutte le variabili dirette o negate**
- lista degli implicati principali, etc.

Dal punto di vista operativo, la sintesi di reti di tipo PS di una data legge *F* si affronta così:

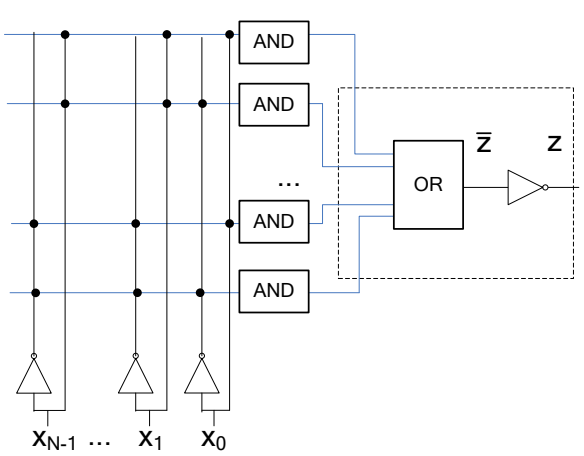
Punto 1: data la legge *F*, ricavo la legge $\overline{F}$, cioè la legge che fa corrispondere ad ogni stato di ingresso **il complemento** di quello che farebbe *F*. In pratica, scrivo la tabella di verità con 1 al posto dello 0.

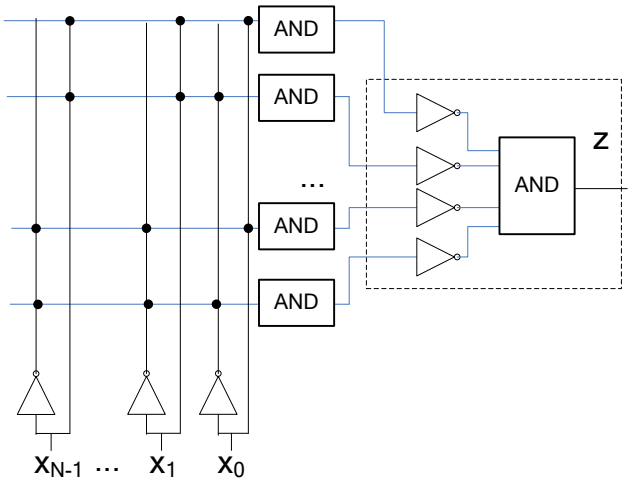
Punto 2: realizzo una **sintesi SP** della legge $\overline{F}$.

Punto 3: ottengo una sintesi della legge *F* inserendo un **invertitore** in uscita alla rete ottenuta al punto precedente, quella cioè che calcolava $\overline{F}$.

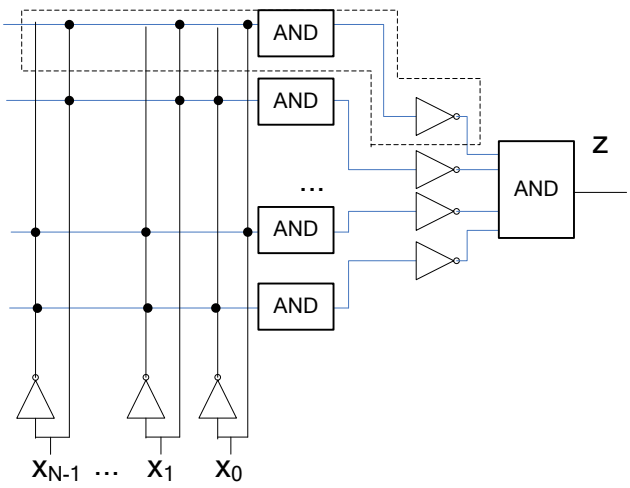
Punto 4: applico i **Teoremi di De Morgan** all'indietro, a partire dall'ultimo livello di logica, ed ottengo:

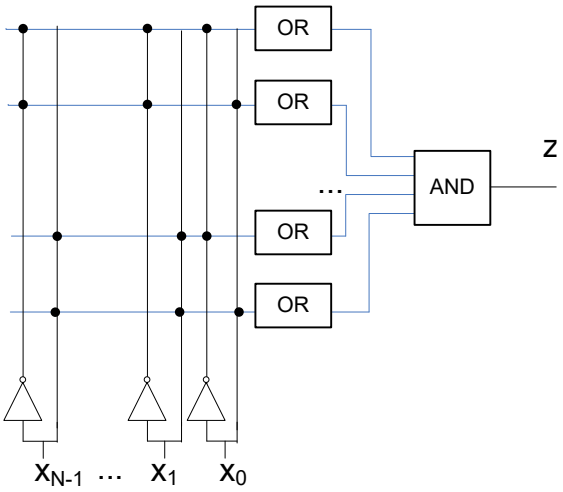
- Al posto della somma finale negata, il **prodotto dei suoi ingressi negati.**





- Nel primo livello di logica, al posto di ciascun prodotto negato, **somme dei suoi ingressi negati.**





Dal punto di vista algebrico, le cose funzionano in questo modo:

- scrivo la legge $\overline{F}$ in forma SP: $\overline{z} = P_1 + P_2 + \ldots + P_k$, in cui i P_i sono prodotti di variabili.
- applico De Morgan, ottenendo: $z = \overline{z} = \overline{P_1 + P_2 + \ldots + P_k} = \overline{P_1} \cdot \overline{P_2} \cdot \ldots \cdot \overline{P_k}$.
- Applico ancora De Morgan per ottenere, da $\overline{P_i} = \overline{\prod x_j} = \sum \overline{x_j}$

La rete così ottenuta è in forma PS, e se:

- $\overline{F}$ è in forma **canonica SP**, allora *F* è in **forma canonica PS;**
- **la sintesi SP di** $\overline{F}$ è di costo minimo (tra le possibili sintesi SP), lo è anche la sintesi PS di F (tra le possibili sintesi PS).

Quest'ultima affermazione si dimostra per assurdo. Infatti, data una realizzazione SP di una legge $\overline{F}$, la realizzazione PS di *F* che si ottiene mettendo un invertitore in fondo ed applicando De Morgan da destra verso sinistra ha lo stesso costo, sia a porte che a diodi. Quindi, se esistesse una realizza-

zione PS di *F* di costo minore, esisterebbe anche una realizzazione SP di $\overline{F}$ di costo minore, che è contro l'ipotesi, in quanto abbiamo utilizzato il procedimento di sintesi SP a costo minimo.

Riepilogando:

- data una legge *F*, siamo in grado di effettuarne la sintesi a costo minimo in forma SP e la sintesi a costo minimo in forma PS. Quest'ultima si ricava a partire dalla sintesi a costo minimo in forma SP di $\overline{F}$.
- Data una sintesi qualunque (SP, PS) per F, siamo in grado di ricavare la sintesi **"duale"** di $\overline{F}$ applicando un invertitore in coda e De Morgan. Tale sintesi:
    - È in **forma duale** rispetto a quella di partenza (PS se quella di partenza era SP e viceversa)
    - Ha **lo stesso costo** di quella di partenza, sia a porte che a diodi.

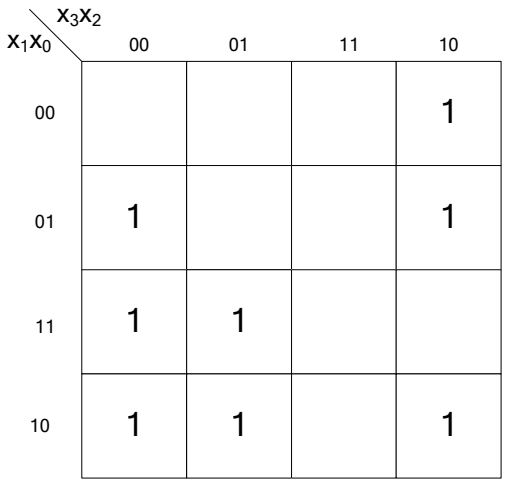
Per via di questa similitudine, fa comodo **dare un nome** alle due sintesi. Presa una sintesi di una legge *F*, definiremo **sintesi duale** la sintesi di $\overline{F}$ ottenuta inserendo un invertitore in coda ed applicando due volte De Morgan.

<u>Domanda</u>: data una legge *F*, **costa meno la sua sintesi a costo minimo in forma SP o la sua sintesi a costo minimo in forma PS?**

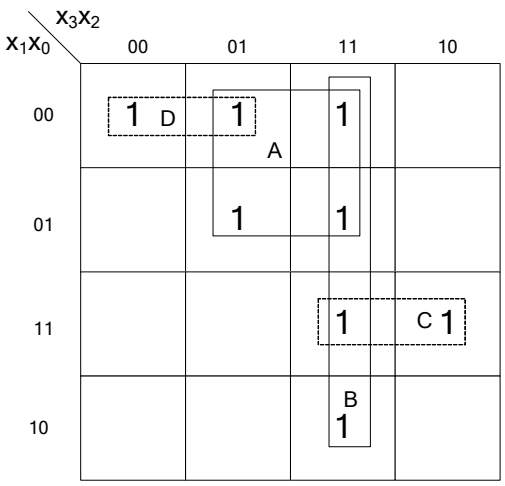
Non è possibile dare una risposta valida in generale a questa domanda. Dipende dalla legge *F* che si vuole realizzare.

<u>Esercizio:</u> sintesi in forma PS di costo minimo – confronto con sintesi SP già fatta. Effettuare la sintesi a costo minimo in forma PS della legge *F* usata negli esempi

Mappa di Karnaugh per F



Mappa di Karnaugh per $\overline{F}$



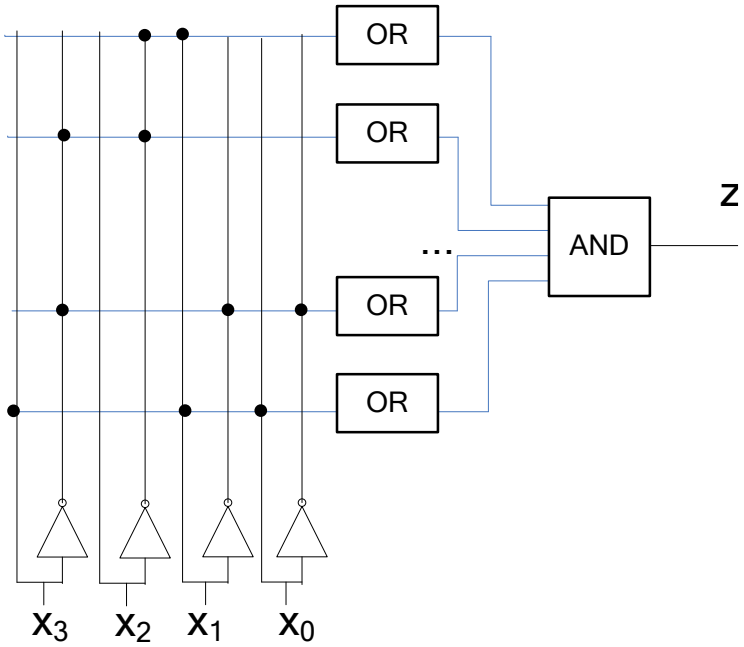
In questo caso gli implicanti principali sono 4, e si vede subito che sono **tutti essenziali.** Quindi, la sintesi SP di costo minimo di $\overline{F}$ è:

$$\overline{z} = \overline{x_1} \cdot x_2 + x_3 \cdot x_2 + x_3 \cdot x_1 \cdot x_0 + \overline{x_3} \cdot \overline{x_1} \cdot \overline{x_0}$$

Da cui, applicando De Morgan:

$$\begin{aligned} z = \overline{\overline{z}} &= \overline{\overline{x_1} \cdot x_2 + x_3 \cdot x_2 + x_3 \cdot x_1 \cdot x_0 + \overline{x_3} \cdot \overline{x_1} \cdot \overline{x_0}} \\ &= \left(\overline{\overline{x_1} \cdot x_2}\right) \cdot \left(\overline{x_3 \cdot x_2}\right) \cdot \left(\overline{x_3 \cdot x_1 \cdot x_0}\right) \cdot \left(\overline{\overline{x_3} \cdot \overline{x_1} \cdot \overline{x_0}}\right) \\ &= \left(x_1 + \overline{x_2}\right) \cdot \left(\overline{x_3} + \overline{x_2}\right) \cdot \left(\overline{x_3} + \overline{x_1} + \overline{x_0}\right) \cdot \left(x_3 + x_1 + x_0\right) \end{aligned}$$

La rete che si ottiene è la seguente, che è di costo **maggiore** rispetto a quella SP (quale che sia il criterio con il quale viene giudicata).



Importante: in sede d'esame servono **le espressioni algebriche, non i disegni.**

7.1 Approfondimento: Sintesi PS per via algebrica

Esiste una teoria completamente duale a quella SP per arrivare alla forma PS. In particolare, data una legge $z = f(x_{N-1}, \ldots, x_0)$, posso infatti scriverne l'**espansione di Shannon** come segue:

$$\begin{aligned} z = & \left[f(0,...,0,0) + x_{N-1} + x_{N-2} + ... + x_1 + x_0 \right] \\ & \cdot \left[f(0,...,0,1) + x_{N-1} + x_{N-2} + ... + x_1 + \overline{x_0} \right] \\ & ... \\ & \cdot \left[f(1,...,1,0) + \overline{x_{N-1}} + \overline{x_{N-2}} + ... + \overline{x_1} + x_0 \right] \\ & \cdot \left[f(1,...,1,1) + \overline{x_{N-1}} + \overline{x_{N-2}} + ... + \overline{x_1} + \overline{x_0} \right] \end{aligned}$$

Notare che le variabili a sommare sono complementate rispetto allo stato su cui è calcolata la funzione.

Dall'espansione di Shannon della legge, posso ottenere la **forma canonica PS** osservando che:

- se $f(x_{N-1}, \ldots, x_0) = 1$, tutto il termine corrispondente sulla riga vale 1 (in quanto $1 + \alpha = 1$). Allora, visto che $1 \cdot \alpha = \alpha$, posso togliere l'intera riga
- se $f(x_{N-1}, \ldots, x_0) = 0$, visto che $0 + \alpha = \alpha$, posso togliere uno degli addendi della somma.

Un po' di nomenclatura:

- forma **PS** perché *z* è ottenuta come prodotto di somme
- forma **canonica** perché ogni prodotto contiene **tutti gli ingressi** diretti o negati
- ciascuno dei termini del prodotto si chiama **maxtermine**, e corrisponde ad uno stato di ingresso non riconosciuto dalla rete.

Si osserva che ogni maxtermine a prodotto vale **sempre** 1, tranne quando lo stato di ingresso ha la configurazione opposta di variabili. Quindi, il prodotto vale zero se e soltanto se uno dei maxtermini è nullo, altrimenti vale uno.

Prendiamo come esempio la tabella di verità di una rete a 4 ingressi ed 1 uscita, già usata nella sintesi SP.

x_3	x_2	x_1	x_0	z_0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Espansione di Shannon

$$z = \left[0 + x_3 + x_2 + x_1 + x_0\right] \cdot \left[1 + x_3 + x_2 + x_1 + \overline{x_0}\right] \cdot \ldots \cdot \left[0 + \overline{x_3} + \overline{x_2} + \overline{x_1} + x_0\right] \cdot \left[0 + \overline{x_3} + \overline{x_2} + \overline{x_1} + \overline{x_0}\right]$$

Forma canonica PS

(lista dei maxtermini)

$$z = \left[x_3 + x_2 + x_1 + x_0\right] \cdot \left[x_3 + \overline{x_2} + x_1 + x_0\right] \cdot \left[x_3 + \overline{x_2} + x_1 + \overline{x_0}\right] \cdot \left[\overline{x_3} + x_2 + \overline{x_1} + \overline{x_0}\right] \cdot \left[\overline{x_3} + \overline{x_2} + x_1 + x_0\right] \cdot \left[\overline{x_3} + \overline{x_2} + x_1 + \overline{x_0}\right] \cdot \left[\overline{x_3} + \overline{x_2} + \overline{x_1} + x_0\right] \cdot \left[\overline{x_3} + \overline{x_2} + \overline{x_1} + \overline{x_0}\right]$$

Partendo dalla lista dei maxtermini, applico **esaustivamente** le due seguenti regole:

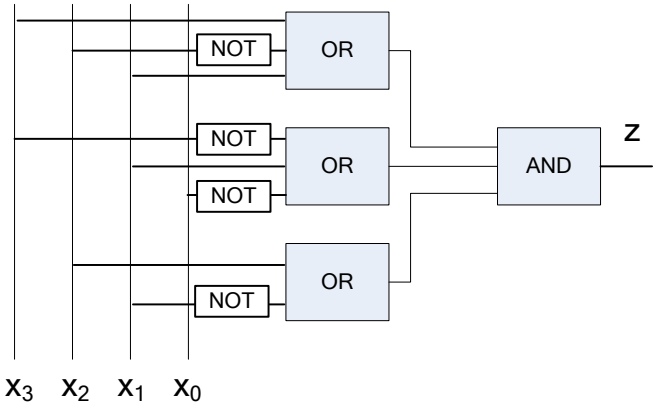
$$\begin{cases} (\alpha + x) \cdot (\alpha + \overline{x}) = (\alpha + x) \cdot (\alpha + \overline{x}) \cdot \alpha \\ \alpha \cdot \alpha = \alpha \end{cases}$$

- La prima legge consente di **fondere i maxtermini**. Dati due termini che differiscono per **una sola variabile**, che è diretta in un caso e negata nell'altro, posso produrre un termine che contiene la sola parte comune
- La seconda legge ci ricorda di **non inserire duplicati.**

In questo modo, dalla lista dei maxtermini costruisco la lista degli **implicati.**

7.2 Esercizio (per casa)

Data la rete combinatoria di figura:



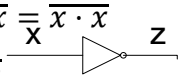
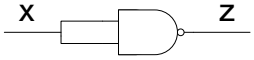
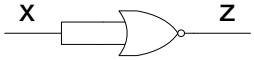
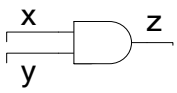
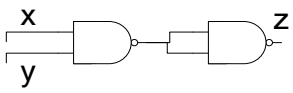
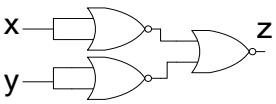
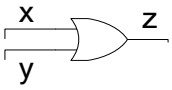
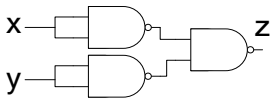
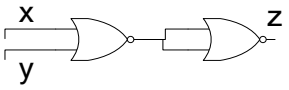
1) disegnare la mappa di Karnaugh per la legge *z*, sapendo che non è possibile che si presentino stati di ingresso in cui tutte le variabili hanno lo stesso valore.

2) Individuare e classificare gli implicanti principali, e trovare *tutte* le liste di copertura irridondanti. Sintetizzare la rete in forma SP, scegliendo la realizzazione di costo minimo secondo il criterio a porte.

Soluzione

8 Porte logiche universali

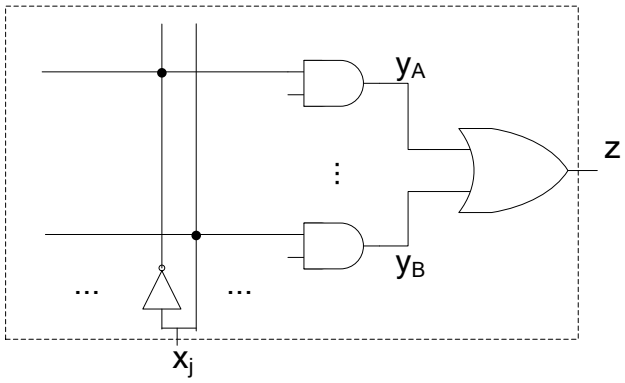
Le porte NAND e NOR sono dette **porte logiche universali**. Infatti, ogni legge combinatoria può essere sintetizzata usando esclusivamente porte NAND (o usando esclusivamente porte NOR). Vediamo come:

	Porta	realizz. a NAND	realizz. a NOR
NOT $x = x \cdot x \Rightarrow \overline{x} = \overline{x \cdot x}$ $x = x + x \Rightarrow \overline{x} = \overline{x + x}$			
AND $x \cdot y = \overline{(\overline{x \cdot y})}$ $x \cdot y = \overline{\overline{x} + \overline{y}}$			
OR $x + y = \overline{\overline{x} \cdot \overline{y}}$ $x + y = \overline{(\overline{x + y})}$			

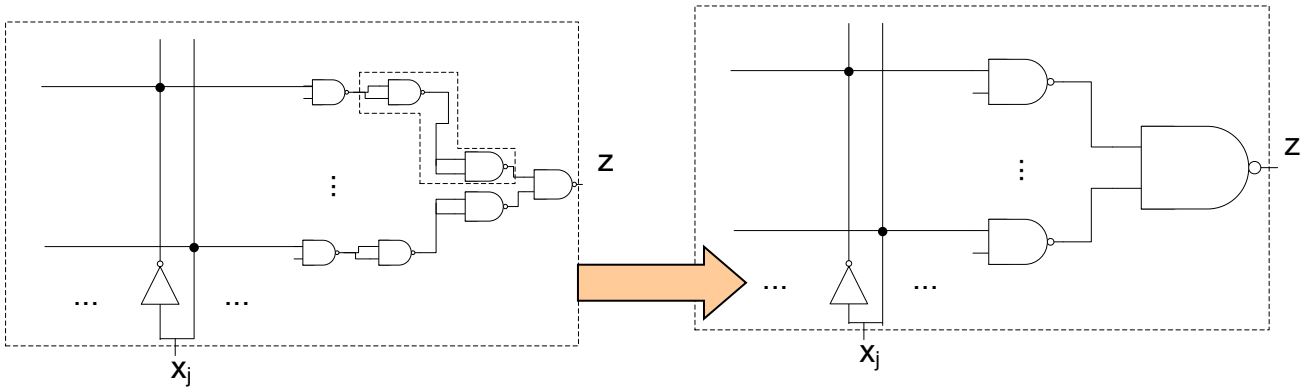
E visto che qualunque legge combinatoria può essere realizzata con sole porte AND, OR, NOT, posso sintetizzare qualunque legge combinatoria **soltanto** con porte NAND, oppure **soltanto** con porte NOR.

8.1 Sintesi a porte NAND

Si parte da un circuito **in forma SP**:



- Sostituisco la porta OR con il suo equivalente a NAND
- Sostituisco ciascuna AND con il suo equivalente a NAND
- I NOT sugli ingressi li posso lasciar stare, tanto **non fanno parte della sintesi** (sono gratis)
- **elimino ogni coppia di** NAND in cascata



Data una sintesi SP di un circuito, la sintesi a porte NAND che gli corrisponde ha **lo stesso costo**, sia a diodi che a porte (nel costo non si contano gli invertitori sugli ingressi).

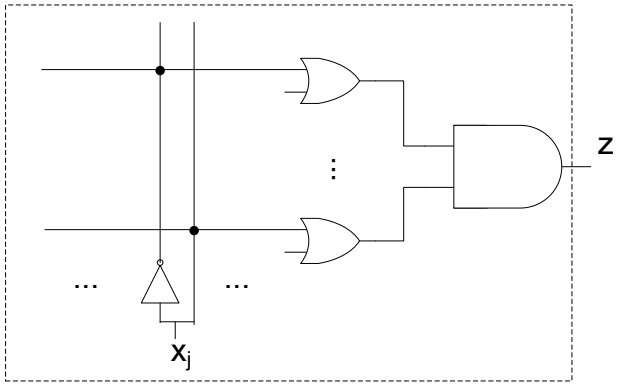
Dal punto di vista **algebrico**, un'espressione a porte NAND si ottiene a partire da quella SP, **complementando due volte ed applicando De Morgan una volta:**

$$\begin{aligned} z &= P_1 + P_2 + ... + P_K \\ &= \overline{\overline{P_1 + P_2 + ... + P_K}} \\ &= \overline{\overline{P_1} \cdot \overline{P_2} \cdot ... \cdot \overline{P_K}} \end{aligned}$$

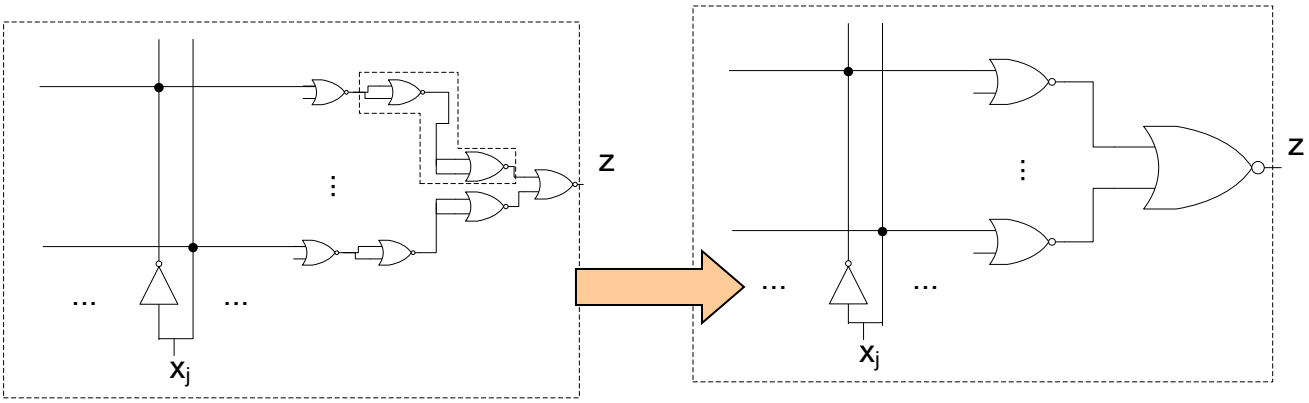
Ciascuno dei termini P_i è un prodotto, e quindi questa è una sintesi fatta a NAND.

8.2 Sintesi a porte NOR

Si parte da un circuito **in forma PS**:



- Sostituisco la porta AND con il suo equivalente a NOR
- Sostituisco ciascuna porta OR con il suo equivalente a NOR
- non sto a toccare i NOT sugli ingressi, per lo stesso motivo visto prima
- **elimino ogni coppia di** NOR in cascata



Data una sintesi PS di un circuito, la sintesi a porte NOR che gli corrisponde ha **lo stesso costo**, sia a diodi che a porte (nel costo non si contano gli invertitori sugli ingressi). Dal punto di vista **algebrico**, un'espressione a porte NOR si ottiene a partire da un'espressione in forma PS, complementando due volte ed applicando De Morgan una volta, come nel caso SP-NAND.

8.3 Esercizio d'esame

Data la rete combinatoria descritta da questa mappa di Karnaugh:

- sintetizzarla a costo minimo, sia a porte NAND che a porte NOR
- utilizzando il **criterio di costo a diodi**, stabilire quale dei due modelli ha costo minore, e disegnare il circuito corrispondente

x_1x_0 \ x_3x_2	00	01	11	10
00	1	0	1	-
01	0	0	1	1
11	1	-	0	-
10	-	0	0	1

Dobbiamo fare una sintesi di tipo **SP, e trasformarla a porte NAND**. Dobbiamo fare la sintesi di tipo **PS, e trasformarla a porte NOR.** Visto che la rete è parzialmente specificata, considererò i non specificati come 1 per trovare gli implicanti principali, e come 0 per trovare la lista di copertura di costo minimo.

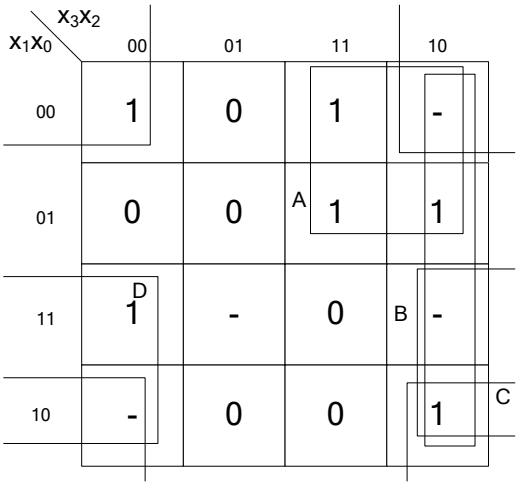
8.3.1 Sintesi a porte NAND

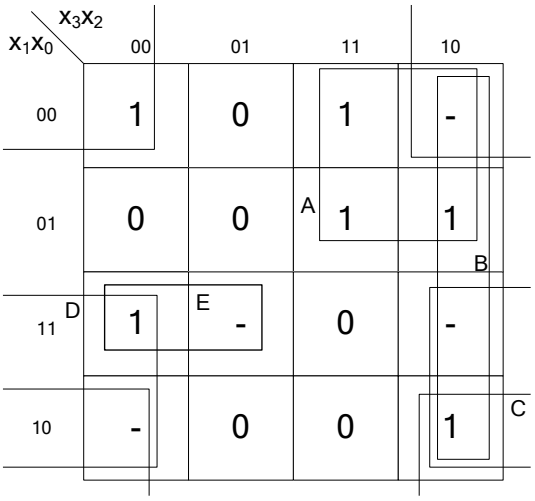
Per prima cosa cerco gli implicanti principali:

parto da quelli di ordine 4, e trovo:

	x_3	x_2	x_1	x_0
A	1	-	0	-
B	1	0	-	-
C	-	0	-	0
D	-	0	1	-

Non bastano a coprire tutta la rete. Devo passare a quelli di ordine 2.

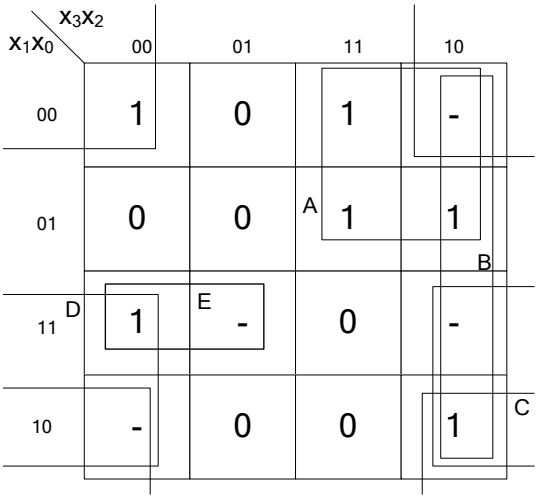




Non bastano a coprire tutta la rete. Devo passare a quelli di ordine 2.

	x_3	x_2	x_1	x_0
A	1	-	0	-
B	1	0	-	-
C	-	0	-	0
D	-	0	1	-
E	0	-	1	1

Adesso ho bisogno di classificare gli implicanti essenziali, assolutamente eliminabili, semplicemente eliminabili.



- A e C sono essenziali, e costituiscono il cuore della mappa.
- B è assolutamente eliminabile, in quanto riconosce soltanto stati già riconosciuti da implicanti essenziali.
- D ed E sono semplicemente eliminabili.

Quindi, le due possibili liste di copertura sono: $\{A, C, D\}, \{A, C, E\}$. Tali liste richiedono lo **stesso numero di porte** (3 AND ed una OR), ma un **diverso numero di ingressi**. Infatti, il sottocubo D è

di ordine 4 (2 ingressi), mentre il sottocubo E è di ordine 2 (3 ingressi). Quindi, la lista di copertura di costo minimo è $\{A, C, D\}$. Il suo costo è di 2+2+2+3=9 ingressi.

La sintesi SP a costo minimo del circuito è:

$$z = x_3 \cdot \overline{x_1} + \overline{x_2} \cdot \overline{x_0} + \overline{x_2} \cdot x_1$$

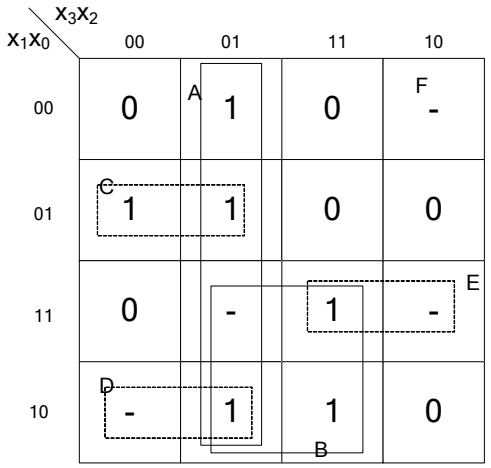
Da questa, complementando due volte ed applicando DeMorgan, ottengo l'espressione in termini di NAND:

$$\boxed{\begin{aligned} z &= \overline{\overline{\left(x_3 \cdot \overline{x_1}\right) + \left(\overline{x_2} \cdot \overline{x_0}\right) + \left(\overline{x_2} \cdot x_1\right)}} \\ &= \overline{\overline{\left(x_3 \cdot \overline{x_1}\right)} \cdot \overline{\left(\overline{x_2} \cdot \overline{x_0}\right)} \cdot \overline{\left(\overline{x_2} \cdot x_1\right)}} \end{aligned}}$$

8.3.2 Sintesi a porte NOR

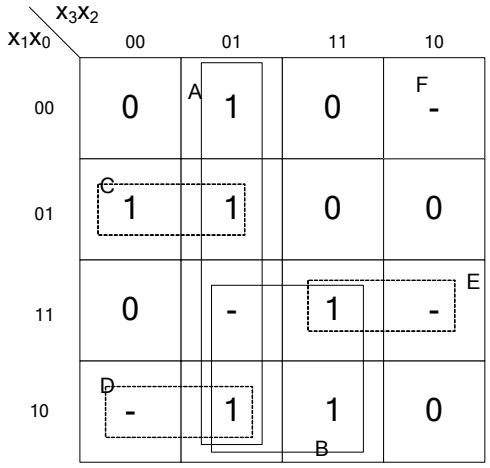
La sintesi a porte NOR si ottiene da quella di tipo PS. Peraltro, quest'ultima si ricava a partire dalla sintesi SP della mappa ottenuta complementando gli zeri e gli uni. I valori di non specificato **restano tali**. (vuol dire che "non voglio sapere" cosa fa la rete in quel caso).

	x_3	x_2	x_1	x_0
A	0	1	-	-
B	-	1	1	-
C	0	-	0	1
D	0	-	1	0
E	1	-	1	1
F	1	0	0	0



Mi serve anche un implicante di ordine 1.

Adesso ho bisogno di classificare gli implicanti essenziali, assolutamente eliminabili, semplicemente eliminabili.



- A, B, C sono essenziali, e costituiscono il cuore della mappa.
- D, E ed F sono assolutamente eliminabili.

Quindi, ho un'unica lista di copertura: $\{A, B, C\}$, che è a costo minimo. Il suo costo è di 2+2+3+3=10 ingressi.

La sintesi SP a costo minimo della rete con gli 1 al posto dello zero è:

$$\overline{z} = \overline{x_3} \cdot x_2 + x_2 \cdot x_1 + \overline{x_3} \cdot \overline{x_1} \cdot x_0$$

Da cui ottengo la sintesi PS:

$$z = (x_3 + \overline{x_2}) \cdot (\overline{x_2} + \overline{x_1}) \cdot (x_3 + x_1 + \overline{x_0})$$

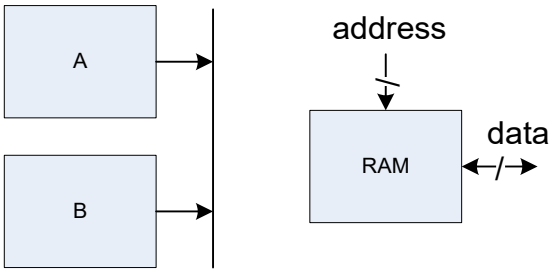
E da questa, complementando due volte ed applicando DeMorgan, l'espressione della sintesi a porte NOR

$$\boxed{\begin{aligned} z &= \overline{\overline{\left(x_3 + \overline{x_2}\right) \cdot \left(\overline{x_2} + \overline{x_1}\right) \cdot \left(x_3 + x_1 + \overline{x_0}\right)}} \\ &= \overline{\left(x_3 + \overline{x_2}\right) + \left(\overline{x_2} + \overline{x_1}\right) + \left(x_3 + x_1 + \overline{x_0}\right)} \end{aligned}}$$

Tra le due, quella che costa meno secondo il criterio di costo a diodi è quella SP.

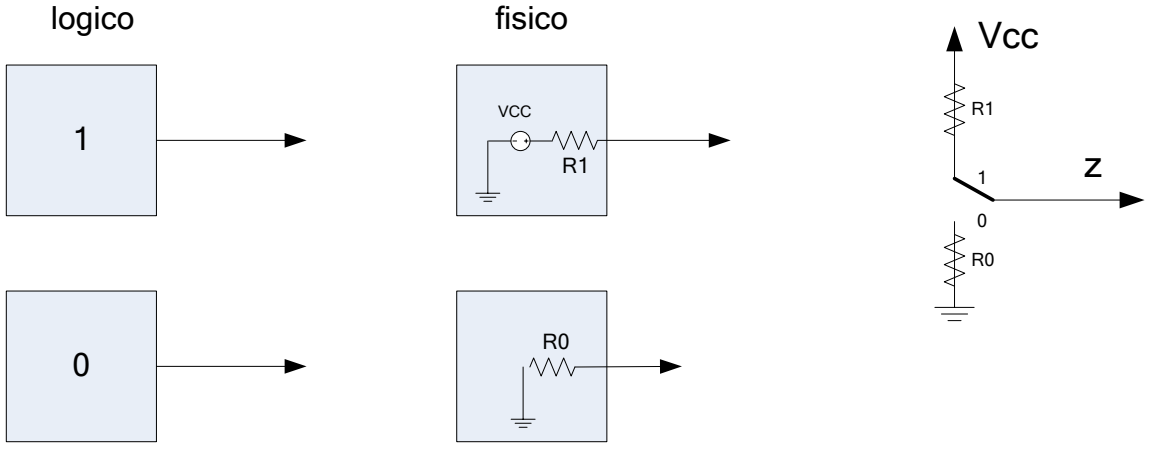
9 Porte tri-state

In molti casi, fa comodo poter connettere insieme le **uscite** delle reti. Ad esempio, se ho un **bus condiviso**, ho una o più linee che più reti diverse possono impostare ad un dato valore. Ovviamente, non lo possono fare **contemporaneamente**, ma dovranno fare a turno. Posso inoltre avere delle linee che servono, in tempi diversi, come **ingressi ed uscite** ad una data rete (ad esempio, le linee dati rispetto alla memoria RAM).



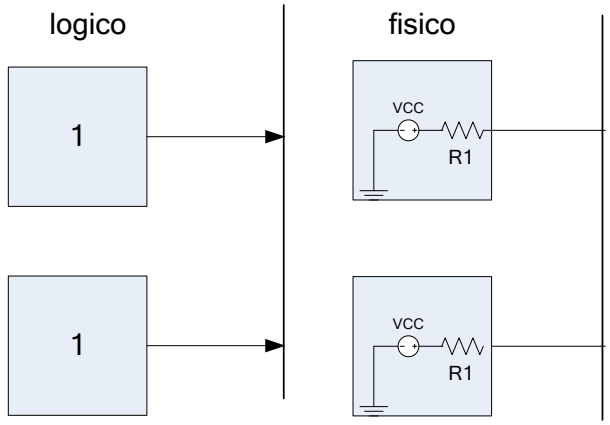
La notazione con la freccia tagliata da uno slash significa "un certo numero di fili". Se voglio specificare quanti sono, lo scrivo accanto allo slash.

Questa cosa **comporta dei problemi a livello elettrico**. Abbiamo visto che le nostre reti logiche sono **modelli** di circuiti elettronici. Tali circuiti sono **alimentati** da tensioni continue (ad esempio di **5 volts**). Un modello elettrico **semplificato**, ma ragionevole, di cosa succede su un'uscita è quello della figura sottostante: un'uscita è un **generatore di tensione** in serie ad una resistenza.



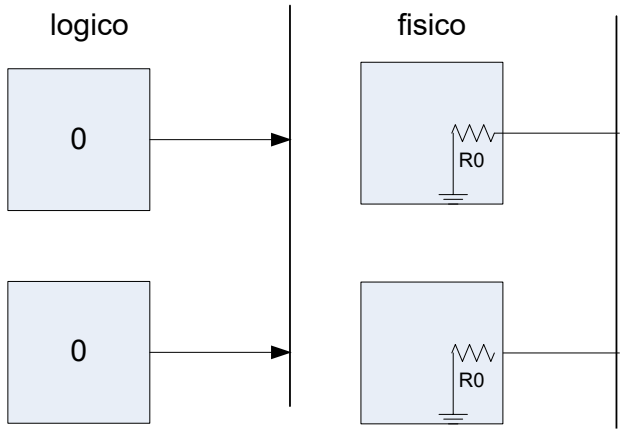
Quando l'uscita vale 1, sull'uscita c'è la stessa tensione di alimentazione. Quando l'uscita vale 0, sull'uscita c'è la tensione di riferimento (*massa*). Che succede se connetto **insieme due uscite?**

Caso 1: entrambe le uscite valgono 1:



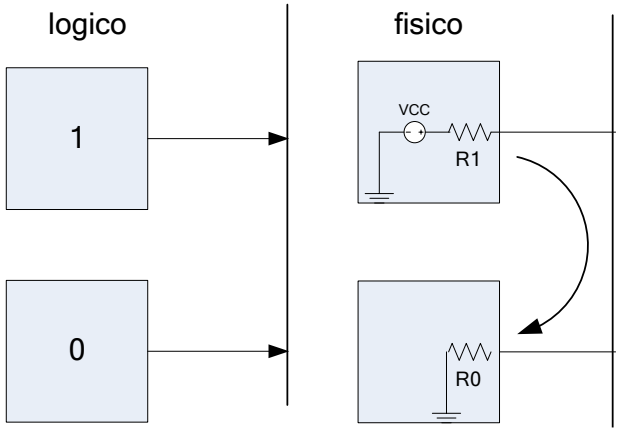
In questo caso ho due generatori di tensione messi in parallelo, quindi la linea condivisa ha la stessa tensione dei due generatori. Vale a dire che **la linea condivisa vale 1**.
Nel circuito non scorre corrente.

Caso 2: entrambe le uscite valgono 0



In questo caso la linea condivisa ha lo stesso potenziale di riferimento. Cioè, **la linea contiene il valore logico 0.**
Nel circuito non scorre corrente.

Caso 3: le uscite sono discordi:

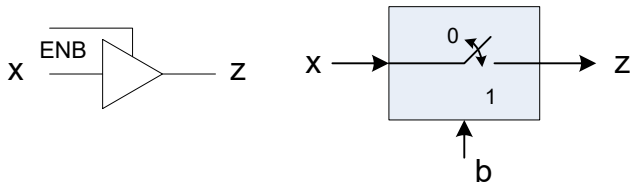


In questo caso ci sono **due grossi problemi**. Il primo è di tipo **elettrico: c'è una maglia chiusa dove scorre corrente**. La corrente è limitata solo dalle resistenze R0 ed R1, che peraltro possono essere **piccole**, quindi ne scorre **tanta**. I circuiti si **bruciano** facilmente in queste condizioni. Il secondo problema è di tipo **logico**. La tensione che c'è sulla linea condivisa **dipende dal valore di R0 ed R1,** e potrebbe anche trovarsi nella fascia di indeterminazione. Quale che sia il valore di questa tensione, è chiaro che questa è una situazione che **non deve accadere**.

Fin qui abbiamo visto cosa succede quando ci sono **due** uscite connesse sulla stessa linea. In realtà, potrei pensare di averne più di due. In quel caso, il problema peggiora: che succede se N-1 valgono 0 ed 1 vale 1? Qual è il livello elettrico in uscita? Potrò mai garantire che i livelli elettrici siano sempre corretti, quale che sia la configurazione delle uscite?

Lo stesso problema ce l'ho quando ho delle linee di **ingresso-uscita**. Se sono uscite, si comportano come quelle scritte sopra. Se sono ingressi, sono uscite per qualche altra rete, e quindi si verificano gli stessi problemi.

Per poter gestire configurazioni del genere, è necessario disporre di apparati **capaci di disabilitare le uscite**, cioè di **disconnettere** fisicamente un'uscita da una linea condivisa. Tali apparecchi prendono il nome di **porte tri-state.**

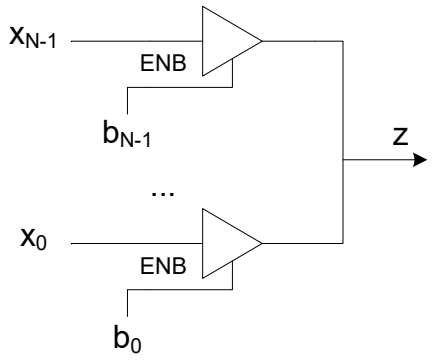


b	x	z
1	0	0
1	1	1
0	-	HiZ

Quando **b vale 1** la porta tri-state si comporta come un **elemento neutro**. Quando **b vale 0**, l'uscita è (come se fosse) disconnessa dal resto della rete. Si dice in tal caso che l'uscita si trova **in alta impedenza**. Quando la porta si trova in alta impedenza, qualunque tensione venga applicata sull'uscita, non scorrerà comunque corrente (o ne scorrerà una quantità trascurabile).

Quando ci sono **N uscite collegate insieme**, su **ogni uscita** deve essere presente una porta **tri-state**. Ad ogni istante, **almeno N-1 porte** devono trovarsi in alta impedenza.

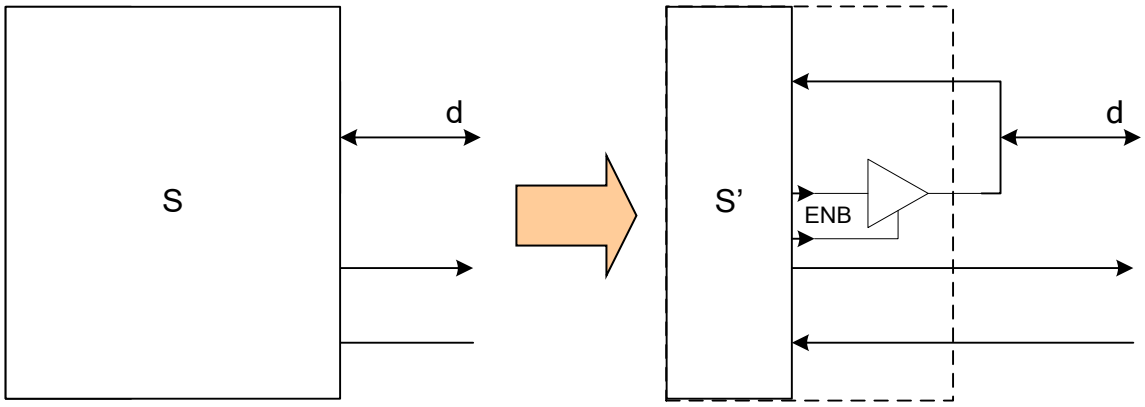
<u>Esempio</u>: multiplexer decodificato



- Se tutte le variabili di comando valgono 0, l'uscita è in **alta impedenza**.

- Altrimenti, **una** sola variabile di comando deve essere ad 1, nel qual caso l'uscita insegue l'ingresso corrispondente.

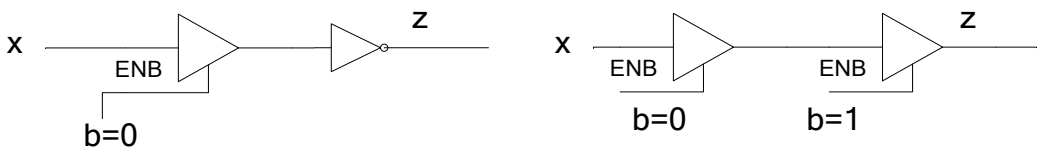
Esempio: rete con linea di ingresso/uscita.



A stretto rigor di logica, le porte tri-state **non ricadono** nell'ambito del modello definito all'inizio del corso. Visto che ci sono, e sono indispensabili, è necessario prenderne atto.

L'alta impedenza non è un valore logico. Come tale non si propaga.

Esempio



Quanto vale *z* nelle due figure? **Vale zero o vale 1**. Una porta tri-state è una **porta logica**, che come tale **rigenera i livelli logici**, cioè produce in uscita un valore che è **molto vicino** al fondo scala (che sia uno o zero), **quale che sia** il suo ingresso.

10 Circuiti di ritardo e formatori di impulsi

Abbiamo visto che si possono pensare circuiti ad un ingresso ed un'uscita detti **buffer** o elementi di ritardo. In genere questi circuiti si sintetizzano collegando in cascata **un numero pari di porte NOT**. Il numero di porte viene dimensionato in base al rapporto tra il ritardo desiderato ed il tempo di attraversamento della porta NOT.

Tali circuiti hanno un ritardo **simmetrico**, cioè identico sulla transizione 0-1 e su quella 1-0.

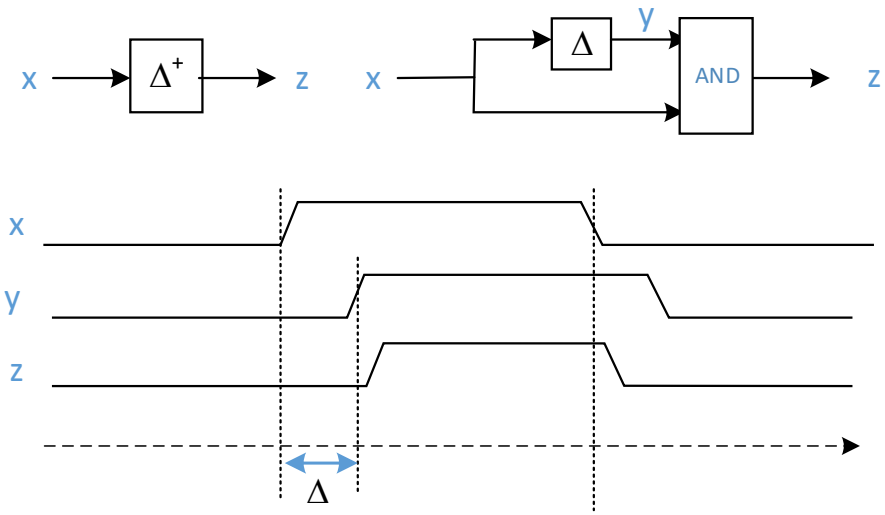
Può far comodo disporre di circuiti con **ritardo asimmetrico**:

- Grande (e dimensionabile) per la transizione 0-1, piccolo per quella 1-0;
- Piccolo per la transizione 0-1, grande (e dimensionabile) per quella 1-0.

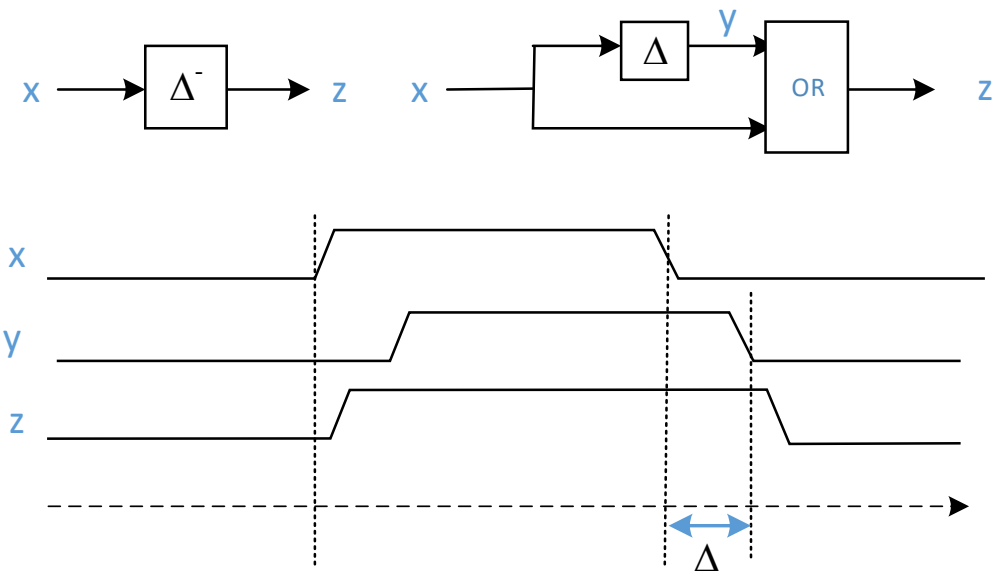
Circuiti del genere si realizzano prendendo una porta **AND o OR a due ingressi, più un buffer** di ritardo noto.

Il circuito con ritardo grande sulla transizione 0-1 si indica con Δ^+, e si realizza con una AND. Detto y l'ingresso ritardato, il diagramma temporale basta a convincersi che

- Nella transizione 1-0, il primo ingresso che va a 0 (x) porta a zero l'uscita. Il ritardo è quindi pari a quello (piccolo) della porta AND.
- Nella transizione 0-1, il **secondo** ingresso che va a 1 (y) porta a 1 l'uscita. Il ritardo è quindi pari a quello (grande) del buffer, più quello (piccolo) della porta AND.



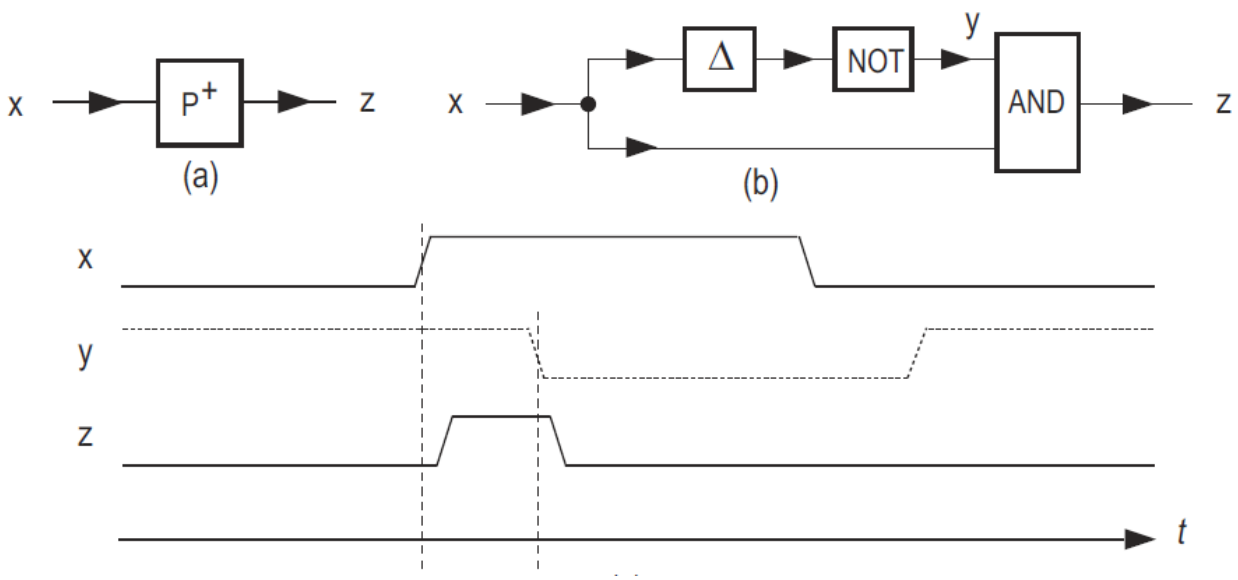
Il circuito con ritardo grande sulla transizione 1-0 si indica con Δ^-, e si realizza con una OR, dove la situazione è simmetrica.



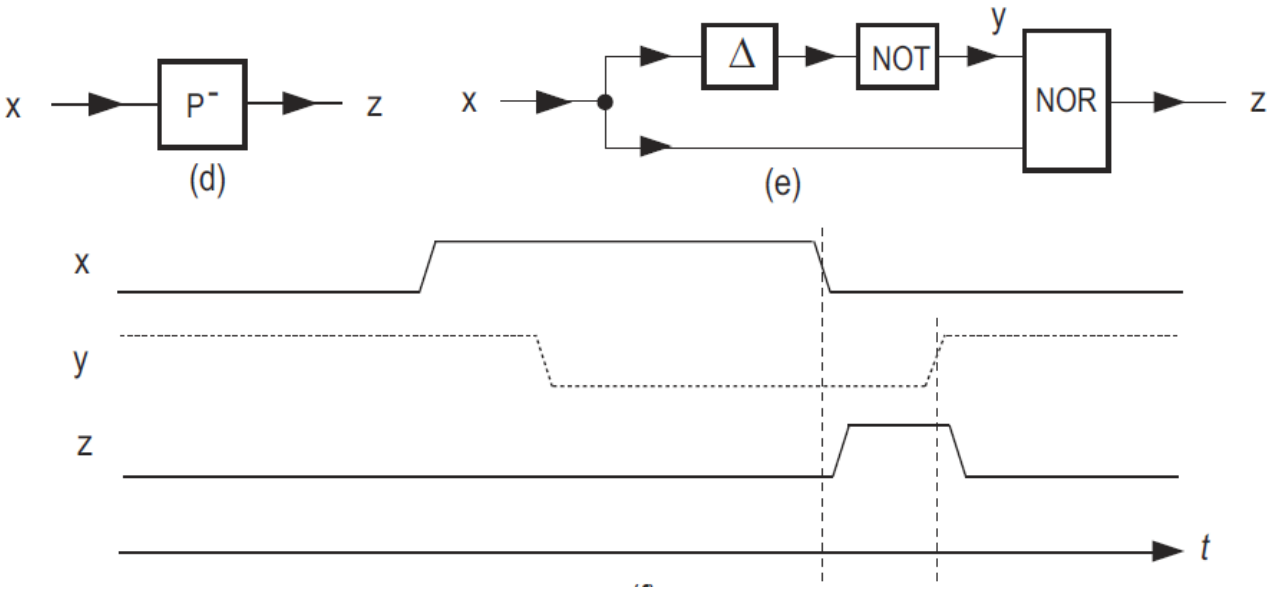
In maniera simile, possiamo costruire dei **formatori di impulsi**, cioè dei circuiti combinatori che generano **un impulso di durata nota sull'uscita** quando l'ingresso ha:

- Un fronte di salita (circuito P^+);
- Un fronte di discesa (circuito P^-).

Tali circuiti si realizzano nello stesso modo di quelli appena visti, **negando** l'ingresso che subisce ritardo.



Per realizzare il circuito P^- è però necessario usare una **porta NOR**, in quanto ci si aspetta che l'uscita sia sempre a zero, tranne quando l'ingresso ha una transizione 1-0. Se usassimo una porta OR, avremo un circuito con l'uscita invertita.



11 Soluzioni degli esercizi per casa

11.1 Esercizio 3.2

1) Dimostrare che le seguenti identità sono vere

- $x_1 \cdot x_2 + x_2 \cdot x_3 + \overline{x_1} \cdot x_3 = x_1 \cdot x_2 + \overline{x_1} \cdot x_3$

$$x_1 \cdot x_2 + x_2 \cdot x_3 + \overline{x_1} \cdot x_3 =$$
$$x_1 \cdot x_2 + (x_1 + \overline{x_1}) \cdot x_2 \cdot x_3 + \overline{x_1} \cdot x_3 =$$
$$x_1 \cdot x_2 + x_1 \cdot x_2 \cdot x_3 + \overline{x_1} \cdot x_2 \cdot x_3 + \overline{x_1} \cdot x_3 =$$
$$x_1 \cdot x_2 \cdot (1 + x_3) + \overline{x_1} \cdot x_3 \cdot (1 + x_2) =$$
$$x_1 \cdot x_2 + \overline{x_1} \cdot x_3$$

- $(x_1 + x_2) \cdot (x_2 + x_3) \cdot (\overline{x_1} + x_3) = (x_1 + x_2) \cdot (\overline{x_1} + x_3)$

  Si dimostra allo stesso modo della precedente, *mutatis mutandis*.

2) Utilizzando le proprietà dell'algebra, semplificare al massimo le seguenti espressioni

- $\overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot \overline{c} \cdot \overline{d} + a \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot d + b \cdot c \cdot d + a \cdot b \cdot d$

$$\overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot \overline{c} \cdot \overline{d} + a \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot d + b \cdot c \cdot d + a \cdot b \cdot d =$$
$$\overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot \overline{c} \cdot \overline{d} + \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot d + b \cdot c \cdot d + a \cdot b \cdot d =$$
$$\overline{a} \cdot \overline{c} \cdot \overline{d} + \overline{b} \cdot \overline{c} \cdot \overline{d} + \overline{a} \cdot b \cdot d + b \cdot c \cdot d + a \cdot b \cdot d =$$
$$\overline{a} \cdot \overline{c} \cdot \overline{d} + \overline{b} \cdot \overline{c} \cdot \overline{d} + b \cdot d + b \cdot c \cdot d =$$
$$\overline{a} \cdot \overline{c} \cdot \overline{d} + \overline{b} \cdot \overline{c} \cdot \overline{d} + b \cdot d$$

- $\left(a+\overline{b}+c\right)\cdot\left(\overline{a}+\overline{b}+\overline{c}\right)\cdot\left(\overline{a}+\overline{b}+c\right)\cdot\left(a+\overline{b}+\overline{c}\right)\cdot\left(\overline{a}+b\right)\left(a + \overline{b} + c\right) \cdot \left(\overline{a} + \overline{b} + \overline{c}\right) \cdot$

$\left(\overline{a} + \overline{b} + c\right) \cdot \left(a + \overline{b} + \overline{c}\right) \cdot (\overline{a} + b) =$

$$\left(\overline{a} + \overline{b}\right) \cdot \left(a + \overline{b} + c\right) \cdot \left(a + \overline{b} + \overline{c}\right) \cdot (\overline{a} + b) =$$
$$\overline{a} \cdot \left(a + \overline{b} + c\right) \cdot \left(a + \overline{b} + \overline{c}\right) =$$
$$\overline{a} \cdot \left(a + \overline{b}\right) =$$

$\overline{a} \cdot \overline{b}$

3) Date le proprietà elencate per gli operatori di somma e prodotto logico, controllare quali proprietà valgono per le porte XOR/XNOR

Proprietà degli operatori XOR, XNOR
commutativa: $x \oplus y = y \oplus x, x \overline{\oplus} y = y \overline{\oplus} x$
associativa: $x \oplus y \oplus z = (x \oplus y) \oplus z = x \oplus (y \oplus z)$ $x \overline{\oplus} y \overline{\oplus} z = (x \overline{\oplus} y) \overline{\oplus} z = x \overline{\oplus} (y \overline{\oplus} z)$
distributiva del prodotto rispetto allo XOR (ma **non viceversa**): $x \cdot (y \oplus z) = (x \cdot y) \oplus (x \cdot z)$ $x \oplus (y \cdot z) \neq (x \oplus y) \cdot (x \oplus z)$ (disegnare tabella di verità)
complementazione: $x \oplus \overline{x} = 1, x \overline{\oplus} \overline{x} = 0$
elemento neutro: $x \oplus 0 = x, x \overline{\oplus} 1 = x$ (0 el. **neutro** per XOR, 1 el. **neutro** per XNOR)
??: $x \oplus 1 = \overline{x}$ $x \overline{\oplus} 0 = \overline{x}$
??: $x \oplus x = 0, x \overline{\oplus} x = 1$
autoduale: $x \oplus y = \overline{x} \oplus \overline{y}$ $x \overline{\oplus} y = \overline{x} \overline{\oplus} \overline{y}$

11.2 Esercizio 5.1.1

a) $z = a \oplus bx \oplus cy \oplus dxy$.

b) Ricordando che $x \cdot 1 = x$, $x \cdot 0 = 0$, $x \oplus 0 = x$, $x \oplus 1 = \overline{x}$, dall'espressione trovata al punto 1 si ottiene:

$f(0,0) = a, f(1,0) = a \oplus b, f(0,1) = a \oplus c, f(1,1) = a \oplus b \oplus c \oplus d.$

Da cui si ricava:

- $a = f(0,0)$
- $f(1,0) = f(0,0) \oplus b$, che equivale a $b = f(1,0) \oplus f(0,0)$, come si può facilmente controllare scrivendo la tabella di verità.
- Seguendo lo stesso procedimento, si ottiene $c = f(0,1) \oplus f(0,0)$
- Infine $f(1,1) = a \oplus b \oplus c \oplus d$, da cui sviluppando si ottiene

$$f(1,1) = f(0,0) \oplus \big(f(1,0) \oplus f(0,0)\big) \oplus \big(f(0,1) \oplus f(0,0)\big) \oplus d$$
$$= f(0,0) \oplus f(1,0) \oplus f(0,1) \oplus d$$

(L'ultimo passaggio dovuto alla nota identità $x \oplus x = 0$). E quindi:

$d = f(0,0) \oplus f(1,0) \oplus f(0,1) \oplus f(1,1)$

c) Sostituendo nelle espressioni trovate al punto 1) si ottiene

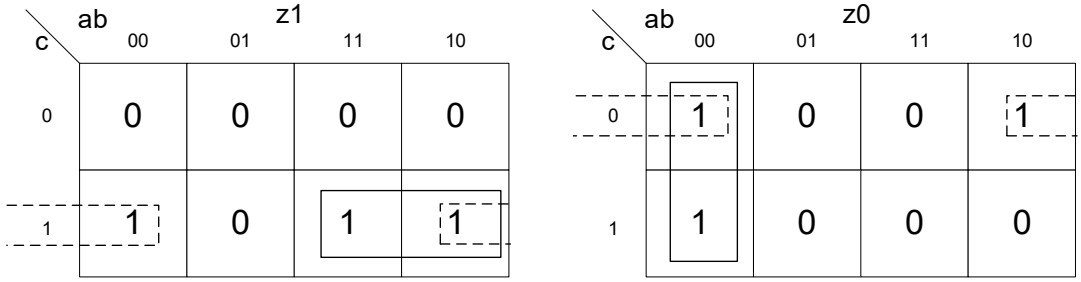
1) $a = 1, b = 0, c = 0, d = 1.$

2) $a = 0, b = 1, c = 0, d = 1.$

Il caso 1) si può inoltre ricavare immediatamente osservando che $\overline{x \cdot y} = (x \cdot y) \oplus 1 = (x \cdot y \cdot 1) \oplus 1$

11.3 Esercizio 6.2.6

1) La sintesi a costo minimo (tra quelle a due livelli di logica in forma SP) delle singole uscite è, come mostrato in figura, $z_1 = a \cdot c + \overline{b} \cdot c$, $z_0 = \overline{b} \cdot \overline{c} + \overline{a} \cdot \overline{b}$.



2) La sintesi di cui al punto 1 costa

- (secondo il criterio a porte) 3+3=6
- (secondo il criterio a diodi) 6+6=12

In realtà, è immediato rendersi conto che il mintermine $\overline{a} \cdot \overline{b} \cdot c$ è comune ad entrambe le leggi per z_1 e z_0, e che, usando quel mintermine, posso realizzare la rete come $z_1 = a \cdot c + \overline{a} \cdot \overline{b} \cdot c$, $z_0 = \overline{b} \cdot \overline{c} + \overline{a} \cdot \overline{b} \cdot c$. Tale realizzazione ha un costo:

- (secondo il criterio a porte) 3+3-1=5
- (secondo il criterio a diodi) 7+7-3=11

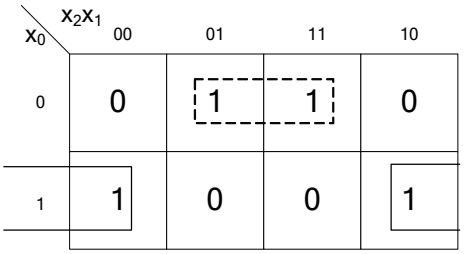
In entrambi i casi, il fattore a sottrarre è dovuto al fatto che la porta $\overline{a} \cdot \overline{b} \cdot c$ va realizzata una volta sola

3) L'esempio mostrato al punto precedente si basa, appunto, sull'utilizzo di un implicante *non principale*, quale il mintermine $\overline{a} \cdot \overline{b} \cdot c$, per ottenere una sintesi a costo minimo di una rete a più uscite. In generale, qualora si voglia affrontare il problema della sintesi *globalmente* di costo minimo per una rete a più uscite, è possibile che nella sintesi entrino anche implicanti che non sono principali per nessuna delle singole uscite.

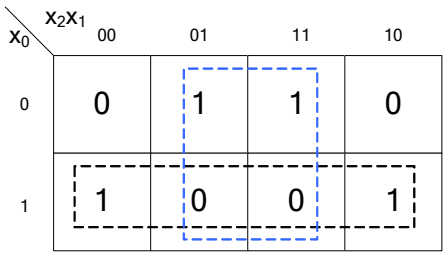
11.4 Esercizio 6.2.7

1) È noto che una porta XOR a *n* ingressi riconosce gli stati di ingresso in cui il numero di 1 presenti è *dispari* (la cosa può essere facilmente osservata disponendo ad albero un numero arbitrario di porte XOR a due ingressi). Pertanto, uno stato di ingresso riconosciuto da F sarà riconosciuto da una rete così fatta se e solo se, in corrispondenza di quello stato, un numero *dispari* di porte AND ha l'uscita pari ad 1.

2) La sintesi SP di costo minimo è $\overline{x_0} \cdot x_1 + x_0 \cdot \overline{x_1}$, il cui costo a porte è pari a 3, ed il cui costo a diodi è pari a 6.



Una sintesi ESOP a costo minore è: $x_0 \oplus x_1$. Il suo costo è pari ad 1 (a porte) e 2 (a diodi), ed è senz'altro minore che nella sintesi SP.



11.5 Esercizio 6.2.8

1) Per ispezione diretta della tabella di verità si ottiene:

$$z = \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1} \cdot x_0 + \overline{x_3} \cdot \overline{x_2} \cdot x_1 \cdot x_0 + \overline{x_3} \cdot x_2 \cdot x_1 \cdot x_0 + x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_3 \cdot x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_3 \cdot x_2 \cdot \overline{x_1} \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0$$

2) Si applica il metodo di Quine-McCluskey (il marcatore "▸" è aggiunto per indicare implicanti che fondono):

#1		x_3	x_2	x_1	x_0
1	▸	0	0	0	1
	▸	1	0	0	0
2	▸	0	0	1	1
	▸	1	0	0	1
	▸	1	1	0	0
3	▸	0	1	1	1
	▸	1	1	0	1
4	▸	1	1	1	1

#1		x_3	x_2	x_1	x_0
1		0	0		1
			0	0	1
	▸	1	0	0	
	▸	1		0	0
2		0		1	1
	▸	1		0	1
	▸	1	1	0	
3			1	1	1
		1	1		1

#1	x_3	x_2	x_1	x_0
1	1		0	
	~~1~~		~~0~~	

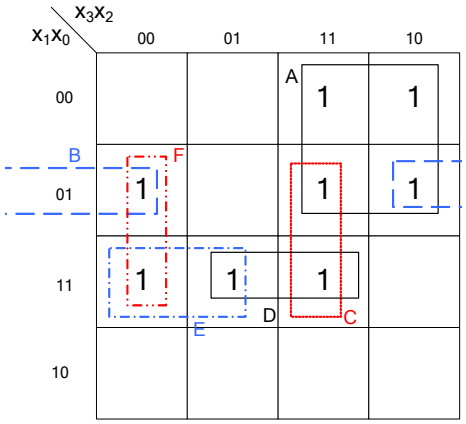
La lista degli implicanti principali può essere ricavata guardando quelli che non hanno fuso:

$$z = \overline{x_3} \cdot \overline{x_2} \cdot x_0 + \overline{x_2} \cdot \overline{x_1} \cdot x_0 + \overline{x_3} \cdot x_1 \cdot x_0 + x_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_0 + x_3 \cdot \overline{x_1}$$

3) La mappa di Karnaugh è la seguente:

x_1x_0 \ x_3x_2	00	01	11	10
00			1	1
01	1		1	1
11	1	1	1	
10				

4) Gli implicanti principali si possono individuare direttamente sulla mappa come segue:



sottocubi principali

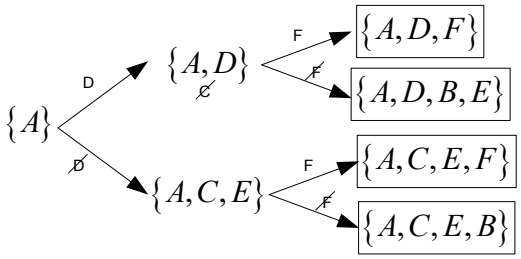
	x_3	x_2	x_1	x_0
A	1		0	
B		0	0	1
C	1	1		1
D		1	1	1
E	0		1	1
F	0	0		1

implicanti principali:

$$z = x_3 \cdot \overline{x_1} + \overline{x_2} \cdot \overline{x_1} \cdot x_0 + x_3 \cdot x_2 \cdot x_0 + x_2 \cdot x_1 \cdot x_0 + \overline{x_3} \cdot x_1 \cdot x_0 + \overline{x_3} \cdot \overline{x_2} \cdot x_0$$

Ovviamente, il risultato è lo stesso ottenuto con il metodo di Quine-McCluskey. Degli implicanti trovati, A è essenziale, gli altri sono tutti *semplicemente eliminabili*.

5) Le liste di copertura si trovano a partire dal cuore della mappa, inserendo o non inserendo implicanti semplicemente eliminabili



Le quattro liste di copertura non ridondanti sono quelle cerchiate nella figura superiore. Di queste, quella di costo minimo è A,D,F, quale che sia il criterio utilizzato. Pertanto, la realizzazione di costo minimo (tra quelle in forma SP a 2 livelli di logica) per la funzione combinatoria data come esercizio è la seguente:

$$z = x_3 \cdot \overline{x_1} + x_2 \cdot x_1 \cdot x_0 + \overline{x_3} \cdot \overline{x_2} \cdot x_0$$

11.6 Esercizio 7.2

1) Dalla mappa si ricava immediatamente: $z = (x_3 + \overline{x_2} + x_1) \cdot (\overline{x_3} + x_1 + \overline{x_0}) \cdot (x_2 + \overline{x_1})$, da cui: $\overline{z} = (\overline{x_3} \cdot x_2 \cdot \overline{x_1}) + (x_3 \cdot \overline{x_1} \cdot x_0) + (\overline{x_2} \cdot x_1)$. Da quest'ultima, si ricava la mappa di Karnaugh per $\overline{z}$, e, tenendo conto di quanto scritto al punto 1), anche quella di z, riportate di seguito.

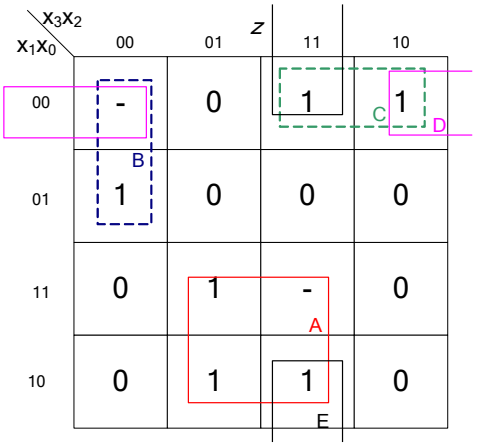
$\overline{z}$

x_1x_0 \ x_3x_2	00	01	11	10
00	-	1	0	0
01	0	1	1	1
11	1	0	-	1
10	1	0	0	1

z

x_1x_0 \ x_3x_2	00	01	11	10
00	-	0	1	1
01	1	0	0	0
11	0	1	-	0
10	0	1	1	0

2) La sintesi in forma SP può essere fatta come segue:



- Implicanti essenziali: A=$x_2 \cdot x_1$, B=$\overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1}$
- Implicanti assolutamente eliminabili: *nessuno*
- Implicanti semplicemente eliminabili: C=$x_3 \cdot \overline{x_1} \cdot \overline{x_0}$, D=$\overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0}$, E=$x_3 \cdot x_2 \cdot \overline{x_0}$

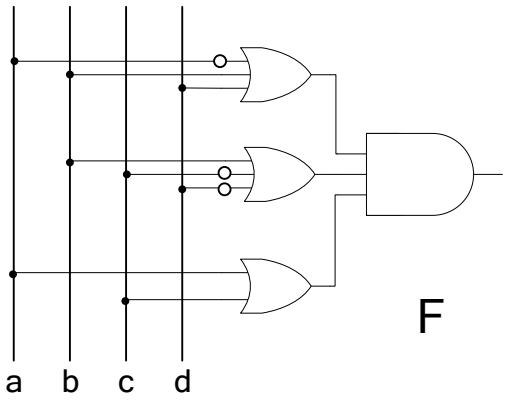
- Liste di copertura irridondanti: {A, B, C}, {A, B, D, E}
- Lista di copertura di costo minimo: {A, B, C}

Quindi, la sintesi di costo minimo in forma SP è: $z = (x_2 \cdot x_1) + (\overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1}) + (x_3 \cdot \overline{x_1} \cdot \overline{x_0})$

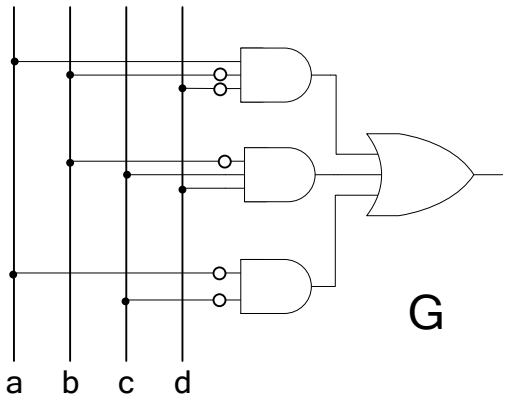
12 Esercizi di riepilogo

12.1 Esercizio (Sintesi PS-SP a costo minimo)

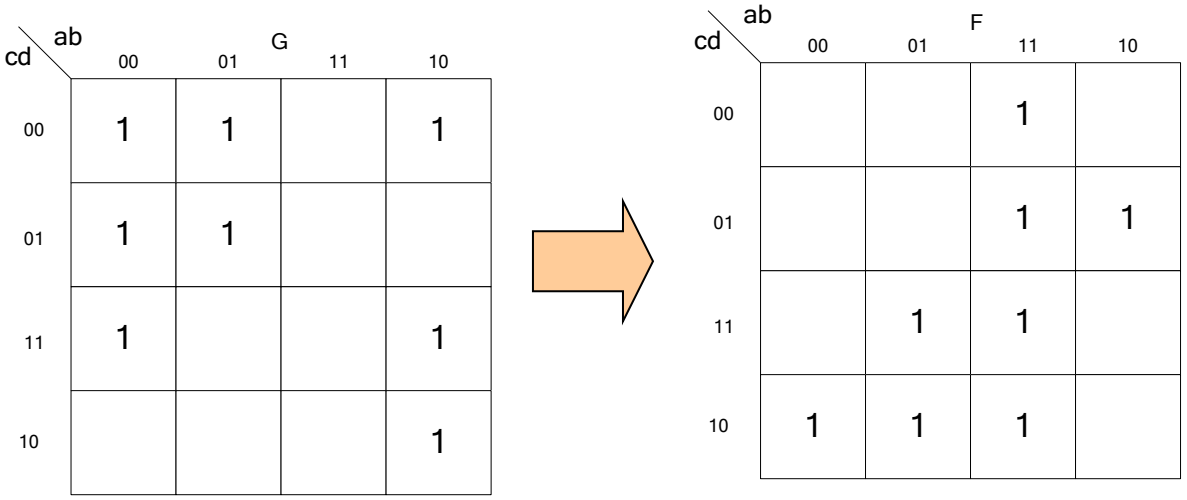
Dato il seguente circuito in forma PS, che sintetizza la legge F, trovare la sintesi di F in forma SP a costo minimo.



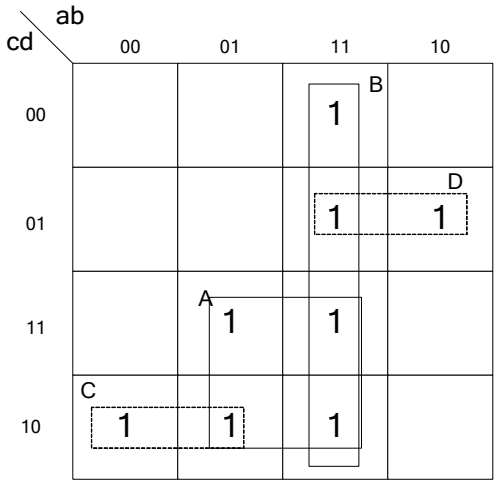
Il circuito dato è in forma PS. Per trovarne la tabella di verità mi conviene passare attraverso il circuito che sintetizza la legge "complementata" rispetto a questa. Tale circuito si ottiene scambiando OR ed AND e complementando gli ingressi.



Per **questo** circuito so trovare immediatamente un'espressione algebrica in formato SP: $z = a \cdot \overline{b} \cdot \overline{d} + \overline{b} \cdot c \cdot d + \overline{a} \cdot \overline{c}$, dalla quale trovo la mappa di Karnaugh. Da questa, ricavo la mappa di Karnaugh per il **circuito originale**:

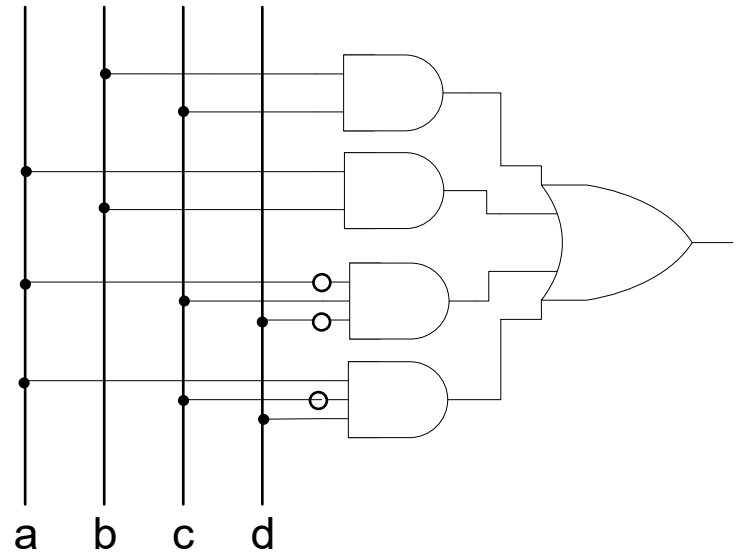


Posso quindi sintetizzare il circuito in forma SP a costo minimo, usando tecniche note.



Ci sono 4 implicanti principali, tutti essenziali.

Quindi, la sintesi a costo minimo in forma SP è la seguente:



Questo circuito costa già di più della sintesi PS dalla quale siamo partiti (e non sappiamo se quella fosse a costo minimo). Quindi, sicuramente la sintesi PS a costo minimo costa meno.

12.2 Esercizio (sintesi a costo minimo)

Data la mappa in figura:

1. indicare e classificare tutti gli implicanti principali;
2. trovare tutte le possibili liste di copertura cui corrispondono forme di tipo SP di costo minimo secondo il criterio di costo a porte;

Specificare le espressioni utilizzando esclusivamente le variabili e l'ordinamento in figura.

x_1x_0 \ x_3x_2	00	01	11	10
00	1	0	0	-
01	-	1	0	-
11	-	1	-	0
10	-	1	0	1

z

Soluzione

Gli implicanti principali sono:

$A = x_2 \cdot x_1 \cdot x_0$, $B = \bar{x}_3 \cdot \bar{x}_2$, $C = \bar{x}_2 \cdot \bar{x}_1$, $D = \bar{x}_2 \cdot \bar{x}_0$, $E = \bar{x}_3 \cdot x_0$, $F = \bar{x}_3 \cdot x_1$.

Gli implicanti *D*, *E*, e *F* sono essenziali, gli altri risultano assolutamente eliminabili. L'unica lista di copertura di costo minimo, indipendentemente dal criterio, è quindi {*D*, *E*, *F*}, cui corrisponde la forma SP di costo minimo: $z = \bar{x}_2 \cdot \bar{x}_0 + \bar{x}_3 \cdot x_0 + \bar{x}_3 \cdot x_1$.

13 Appendice

13.1 Domande a risposta multipla

Domande di questo tipo capitano nella prima parte della prova scritta. Le risposte sono alla fine.

1) Quante reti combinatorie diverse con 3 ingressi ed un'uscita si possono costruire?

a) 8
b) 256
c) 1024
d) Nessuna delle precedenti

	00	01	11	10
00	1	0	0	1
01	0	0	1	-
11	0	-	0	0
10	-	1	0	-

2) Nella mappa di Karnaugh sopra disegnata gli implicanti principali essenziali sono

a) 1
b) 2
c) 3
d) Nessuna delle precedenti

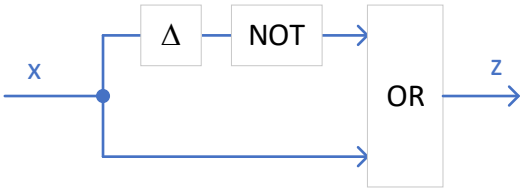
3) $a \cdot b \cdot c \cdot d + \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} =$

a) 0
b) 1
c) Non specificato
d) Nessuna delle precedenti

	00	01	11	10
00	1	1	0	1
01	0	0	1	-
11	0	-	1	0
10	-	1	0	-

4) Nella mappa di Karnaugh sopra disegnata gli implicanti principali essenziali sono

a) 1
b) 2
c) 3
d) Nessuna delle precedenti

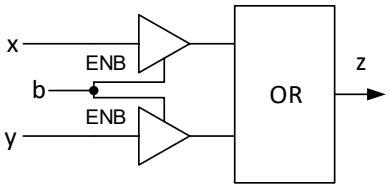


5) Nel circuito riportato di sopra, l'uscita z

a) va ad 1 (per circa Δ) dopo il fronte di salita di x
b) va ad 1 (per circa Δ) dopo il fronte di discesa di x
c) va a 0 (per circa Δ) dopo il fronte di salita di x
d) va a 0 (per circa Δ) dopo il fronte di discesa di x

6) è possibile realizzare una porta XOR sia con porte NAND, sia con porte NOR

a) Vero
b) Falso: solo a porte NAND
c) Falso: solo a porte NOR
d) Falso: in nessuno dei due modi



7) L'uscita z della rete di figura, quando $b = 0$:

a) È in alta impedenza
b) È una tensione nella fascia di indeterminazione
c) Oscilla
d) Nessuna delle precedenti

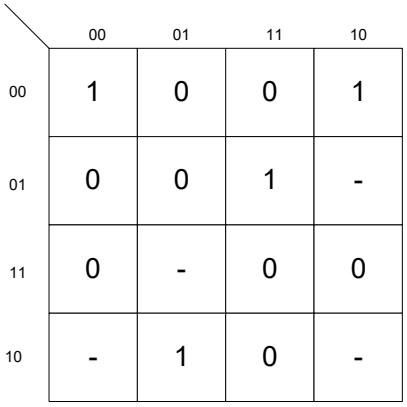
8) Le sintesi in forma canonica PS e a NAND di una rete combinatoria:

a) Hanno sempre lo stesso costo
b) Hanno in genere costi differenti
c) Nessuna delle precedenti

13.2 Risposte

1) Quante reti combinatorie diverse con 3 ingressi ed un'uscita si possono costruire?

a) 8
b) 256
c) 1024
d) Nessuna delle precedenti

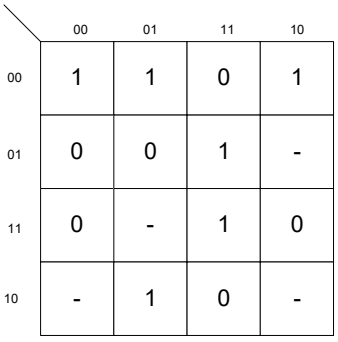


2) Nella mappa di Karnaugh sopra disegnata gli implicanti principali essenziali sono

a) 1
b) 2
c) 3
d) Nessuna delle precedenti

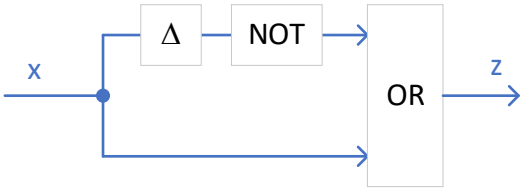
3) $a \cdot b \cdot c \cdot d + \overline{a} \cdot \overline{b} \cdot \overline{c} \cdot \overline{d} =$

a) 0
b) 1
c) Non specificato
d) Nessuna delle precedenti



4) Nella mappa di Karnaugh sopra disegnata gli implicanti principali essenziali sono

a) 1
b) 2
c) 3
d) Nessuna delle precedenti

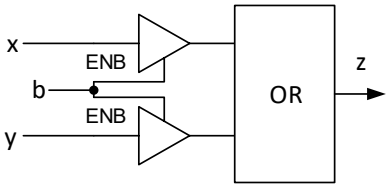


5) Nel circuito riportato di sopra, l'uscita z

a) va ad 1 (per circa Δ) dopo il fronte di salita di x
b) va ad 1 (per circa Δ) dopo il fronte di discesa di x
c) va a 0 (per circa Δ) dopo il fronte di salita di x
d) va a 0 (per circa Δ) dopo il fronte di discesa di x

6) è possibile realizzare una porta XOR sia con porte NAND, sia con porte NOR

a) Vero
b) Falso: solo a porte NAND
c) Falso: solo a porte NOR
d) Falso: in nessuno dei due modi



7) L'uscita z della rete di figura, quando $b = 0$:

a) È in alta impedenza
b) È una tensione nella fascia di indeterminazione
c) Oscilla
d) Nessuna delle precedenti

8) Le sintesi in forma canonica PS e a NAND di una rete combinatoria:

a) Hanno sempre lo stesso costo
b) Hanno in genere costi differenti
c) Nessuna delle precedenti