CAP2. LE RETI LOGICHE COMBINATORIE NEL CALCOLATORE

Prof. Rita Cucchiara

3.1 LE RETI LOGICHE

Il livello delle **reti logiche o della logica digitale** è il livello di base del modello del calcolatore che implementa la microarchitettura, al di sopra del livello della elettronica. È anche il livello fondamentale per ogni sistema digitale sia esso sistema di elaborazione, di comunicazione o semplicemente sistema elettronico. Conoscere il livello delle reti logiche è necessario per l'analisi del comportamento di un calcolatore elettronico e per la sintesi, ossia il progetto. In questo capitolo si studiano i fondamenti delle reti logiche combinatorie come elementi di base del progetto dei computer, ad es., come si realizza una rete logica combinatoria complessa come il sommatore della ALU della CPU.

Il livello delle reti logiche è il livello di astrazione che studia i sistemi digitali a livello di componenti LOGICI elementari, chiamati gate, indipendentemente dalla loro realizzazione fisica.

Cosa e' una rete logica ? La rete logica è il MODELLO MATEMATICO con cui si formalizza un sistema digitale e ne costituisce la sua rappresentazione astratta.

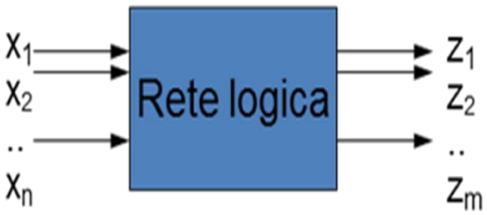


Fig. 1: Rete logica.

Def. Dal punto di vista funzionale, *la **rete logica è la astrazione di un sistema digitale** avente n segnali binari di ingresso X ed m segnali binari di uscita Z ed una funzione logica f che pone in relazione logica i segnali di uscita con quelli di ingresso.*

E' una funzione che dipende dal tempo perché i segnali sono grandezze funzioni del tempo:

$$X = \{x_1(t), \dots, x_n(t)\} \qquad Z = \{z_1(t), \dots, z_m(t)\} \qquad z_i(t) = f\big(x_1(t), \dots, x_n(t)\big)$$

La funzione f(x1,..xn) e' **una funzione logica** che segue le leggi dell'algebra binaria anche detta algebra di Boole (in onore di George Boole) e alla base della logica informatica.

Lo strumento formale si chiama "**Algebra di Boole**", introdotta nel 1874 da George Boole per fornire una rappresentazione algebrica della logica; per questo motivo i circuiti elettronici che

lavorano su valori binari assumono il nome di circuiti “logici” o porte “logiche”. Fu poi applicata nel 1936 da Claude Shannon allo studio delle reti di commutazione telefonica.

La funzione logica o booleana e’ una applicazione su variabili booleane (gli ingressi della rete logica) che si definisce come

$f(x1, ..xn): \{0,1\}^n \rightarrow \{0,1\}$

Indicare i segnali di ingresso a 0 o 1 e’ come indicare i valori logici FALSO e VERO.

I segnali di ingresso e di uscita delle reti logiche possono essere singoli segnali binari (es. RESET), o segnali digitali composti in *parole* codificate come un insieme di segnali binari.

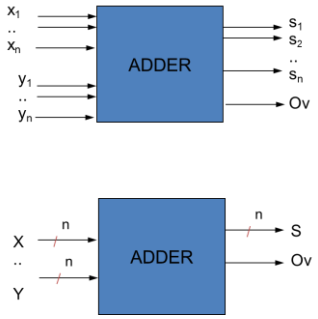


Fig. 2: Rete logica con ingressi singoli e con parole di ingresso di n bit.

Lo schema della figura 2 mostra una rete logica ADDER che può essere descritta a livello comportamentale:

- sia a livello di rete logica – l’adder o sommatore e’ una rete che ha 2n ingressi e somma n bit di ingresso x1..xn con n bit di ingresso y1..yn per creare n bit di uscita s1..sn ed un segnale unico OV di overflow-
- sia a livello RTL (register transfer level) – l’adder e’ una rete con due ingressi X, Y, come parole a n bit e con due uscite di cui S a n bit e OV ad 1 bit.

Le reti logiche godono di alcune proprietà:

- **Proprietà di interconnessione:** l’interconnessione di più reti logiche, aventi per ingresso segnali esterni o uscite di altre reti logiche e per uscite segnali di uscita esterni o ingressi di altre reti logiche, è ancora una rete logica.
- **Proprietà di decomposizione**: una rete logica complessa può essere decomposta in reti logiche più semplici (fino all’impiego di soli blocchi o gate elementari).
- **Proprietà di decomposizione in parallelo**: una rete logica a m uscite può essere decomposta in m reti logiche ad 1 uscita, aventi ingressi condivisi.

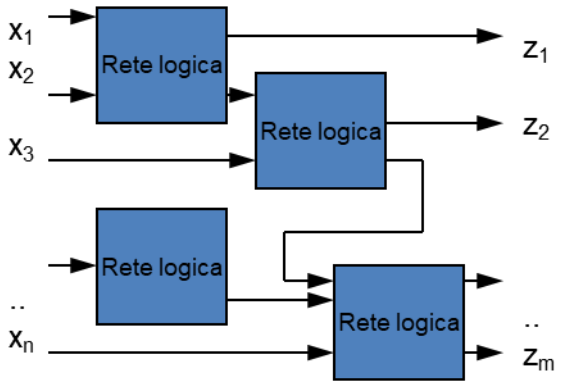


Fig. 3: Esempi di proprietà di interconnessione.

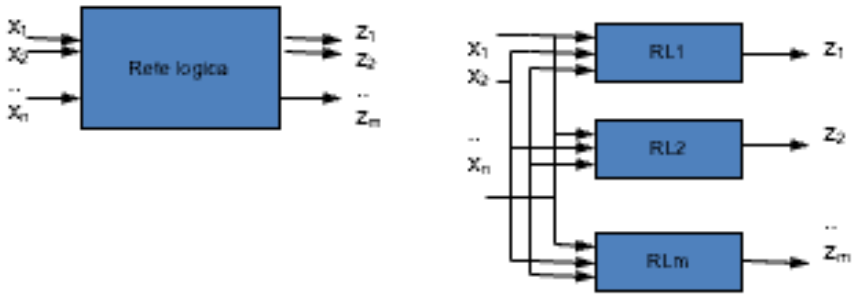


Fig. 4: Esempi di proprietà di decomposizione in parallelo.

Grazie a queste proprietà è stato possibile negli anni realizzare reti logiche sempre più complesse per realizzare sia reti elettroniche digitali sia i blocchi che compongono la micro-architettura di CPU, memorie e periferiche di input-output anche di grandi dimensioni.

Le reti logiche si dividono in reti combinatorie e sequenziali.

Una rete è combinatoria se ogni segnale di uscita dipende solo dai valori degli ingressi in quell'istante:

$$z_i(t) = f\big(x_1(t), \dots, x_n(t)\big).$$

Una rete è sequenziale se ogni segnale di uscita dipende dai valori degli ingressi in quell'istante e dai valori che gli ingressi hanno assunto negli istanti precedenti: $z_i(t) = f(x_1(t), \dots, x_n(t), t)$.

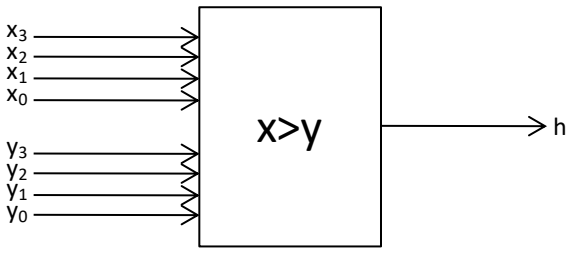
Una rete combinatoria è anche detta rete senza memoria (l'uscita cambia "istantaneamente" dopo che l'ingresso è cambiato), mentre una rete sequenziale è una rete con memoria: è una rete in cui l'uscita cambia in funzione del cambiamento dell'ingresso e della specifica configurazione interna in quell'istante (STATO).

Esercizio 1: Si vuole progettare un comparatore di due cifre decimali BCD X ed Y rappresentate con 4 bit la cui uscita sia vera se X>Y e 0 altrimenti. Un comparatore è quindi la rete logica che confronta due valori binari *x* e *y* a 4 bit e produce in uscita un bit *h* (higher) che vale 1 se *x* è maggiore di *y*, 0 se sono uguali o se *x* è minore di *y*.

Quale e' la descrizione funzionale con uno schema ?

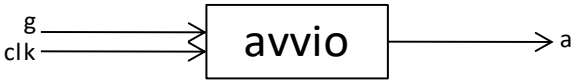
```
While (forever)
        {If (X>Y) h=1;else h=0; }
```

La codifica chiamata **BCD binary-coded digit** consiste semplicemente nel rappresentare le 10 cifre digitali 0…9 con 4 valori binari da 0000 a 1001; viene usata nei led, nelle sveglie.. ovunque si voglia rappresentare delle cifre decimali.



L'uscita Z={h} dipende in ogni istante dalla configurazione degli ingressi {x_3, x_2, x_1, x_0, y_3, y_2, y_1, y_0}. È una rete combinatoria dato che un numero è maggiore di un altro indipendentemente da quali numeri sono stati confrontati prima.

Esercizio 2: Si vuole progettare la logica di AVVIO dei giochi a gettoni con due gettoni. La rete logica collegata tra la gettoniera e il timer di avvio del gioco riceve un segnale in ingresso *g* che vale 1 ogni volta che un gettone viene correttamente inserito e produce in uscita un segnale di avvio *a* che vale 1 dopo la ricezione di due 1 su *g*. Il sistema viene temporizzato con un segnale *clk* (clock). Come e' la descrizione funzionale con uno schema logico?



A parità di ingresso (*g*=1) l'uscita *a* assume a volte il valore 0 e a volte il valore 1. Quindi questa rete non è combinatoria. Per calcolare l'uscita è necessario infatti ricordare che il gettone è stato inserito la prima volta. Quindi e' una rete dotata di memoria.

2.2 DESCRIZIONE COMPORTAMENTALE CON TABELEL DIVERITA'

Come descrivere la funzione *f(x1..xn)* attraverso gli strumenti della logica?

La più immediata descrizione funzionale o comportamentale e' la **"Descrizione a parole"**: il metodo più semplice è la descrizione a parole (in linguaggio naturale) del comportamento della rete logica, ma è poco formale e preciso e difficilmente porta direttamente ad una sintesi. Un es. di descrizione: la rete logica, dati due ingressi, fornisce uscita "vero" solo se i due ingressi sono uguali. Non specifica se gli ingressi sono ad 1 o piu' bit ad es..

La forma esaustiva per rappresentare il comportamento di una rete logica è la **Tabella di verità. (True table)** La **Tabella di verità** *è la descrizione di tutte le configurazioni di uscita per ogni possibile configurazione di ingresso ed è rappresentata da una tabella che associa tutte le possibili combinazioni degli ingressi alle corrispondenti configurazione delle uscite, e indica esaustivamente il comportamento della rete logica.*

E' il metodo basilare *per la descrizione comportamentale di una rete logica.* Se la rete combinatoria ha *n* ingressi e *m* uscite, allora la tabella di verità ha *(n+m)* colonne e 2^n righe. Per la proprietà di decomposizione una tabella con m uscite può essere definita anche come tante tabelle della verità quante sono le uscite.

Es. Come e' fatta la tabella della verità che descrive l'esercizio 1?

La tabella di verità si dice **COMPLETAMENTE SPECIFICATA** se ogni valore della tabella assume il valore logico di vero o falso (1, 0).

La tabella si dice **NON COMPLETAMENTE SPECIFICATA** se sono presenti condizioni di indifferenza (di solito rappresentati con "-" o con "*") per indicare che l'uscita non è specificata nella descrizione. Essa si impiega in diversi casi: 1) *se alcune configurazioni di ingressi sono vietate oppure 2) se le uscite sono indifferenti per alcune configurazioni di ingresso.*

Eserzicio 3: Descrivere un comparatore a 2 bit per ogni ingresso: è la rete logica con due ingressi da 2 bit X ed Y ossia x1,x0 e y1,y0 rispettivamente che confronta i due ingressi e fornisce l'uscita ad 1 se il primo e' maggiore del secondo.

Si può descrivere anche con uno pseudo linguaggio (o con un HDL) come

```
Define X[0,1];Y[0,1];

        If (X>Y) h=1; else h=0;
```

E' una rete completamente specificata che si rappresenta con la tabella di verità come segue

Esercizio *4:* tabella non completamente specificata (per il caso 1): una cifra decimale (0-9) viene codificata con un segnale binario a quattro bit X(codice BCD binary coded digit).La rete logica ha ingresso X e ha uscita mincinque attiva a 1 se X ha un valore inferiore a 5.

Tabella della verità Es 3

X_1	X_0	y_1	y_0	h
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
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	1
1	1	1	1	0

Tabella della verità Es 4

x (decimale)	x_3	x_2	x_1	x_0	**mincinque**
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10 (non valido)	1	0	1	0	-
11 (non valido)	1	0	1	1	-
12 (non valido)	1	1	0	0	-
13 (non valido)	1	1	0	1	-
14 (non valido)	1	1	1	0	-
15 (non valido)	1	1	1	1	-

È chiaro che non serve specificare il valore dell'uscita per valori binari superiori a 9, dato che non si verificheranno mai in ingresso. Le configurazioni di uscita sono non specificate perchè le configurazioni di ingresso sono vietate.

Esercizio 5: tabella non completamente specificata (per il caso 2): un segnale binario a due bit X viene inviato su un canale di trasmissione. Per proteggere la trasmissione da errori, si decide di inviare anche una sua copia invertita Y. Si progetti una rete logica che fornisca due uscite: E e Z. E indica errore se si è verificato o no un errore; Z indica se l'ingresso vale 0 ma solo in caso di configurazione giusta.

x_1	x_0	y_1	y_0	*e*	*Z*
0	0	0	0	1	-
0	0	0	1	1	-
0	0	1	0	1	-
0	0	1	1	0	1
0	1	0	0	1	-
0	1	0	1	1	-
0	1	1	0	0	0
0	1	1	1	1	-
1	0	0	0	1	-
1	0	0	1	0	0
1	0	1	0	1	-
1	0	1	1	1	-
1	1	0	0	0	0

1	1	0	1	1	-
1	1	1	0	1	-
1	1	1	1	1	-

Se quindi x=00, y=11 o se x=10, y=01 e cosi' via . In ricezione una rete logica verifica se x=00 e mette l'uscita z (zero) a 1, altrimenti la lascia a 0. È presente inoltre un segnale e (errore) che vale 1 se y non è corretto.

Quando e=1, è inutile specificare il valore di z, dato che i dati ricevuti sono sbagliati. Perciò in questo caso l'uscita è detta "indifferente".

Dalla descrizione funzionale della tabella di verità esistono metodi manuali ed algoritmi che permettono di arrivare ad una possibile sintesi ossia una rappresentazione strutturale con gate elementari.

Esercizio 6 : Sia data una funzione logica con tre ingressi, A, B e C e tre uscite, D, E e F. La funzione è definita come segue: D è vero se è presente almeno un ingresso vero, E è vero se esattamente due ingressi sono veri e F è vero solo se tutti e tre gli ingressi sono vere. Quale è la tabella della verità per questa funzione?

Si veda la appendice C del testo di Patterson Hennessey.

2.3 ALGEBRA DI BOOLE O DI COMMUTAZIONE

Le funzioni logiche possono essere descritte, oltre che in modo esaustivo tramite tabella, anche in forma algebrica mediante l'**Algebra di Commutazione** o detta anche **di Boole** da **George Boole .**

Mediante l'algebra di commutazione, è possibile effettuare una progettazione logica con una descrizione strutturale (che di conseguenza e' anche funzionale), indipendentemente dalla descrizione fisica.

L'algebra di Boole è diventato uno strumento fondamentale per la **sintesi dell'hardware**. Ricordiamoci però che nasce come strumento formale alla base della **logica matematica**, poi diventata fondamentale nella logica informatica (logica del primo ordine e successive) fondamento **dell'Intelligenza artificiale.** Gli studi che ora sono alla base dei metodi di ragionamento deduttivi dei calcolatori (sistemi esperti) nascono dai lavori di **George Boole** sull'analisi algebrica della sillogistica tradizionale (1847) e sulla costruzione da parte di **Gottlob Frege** di un linguaggio formale in grado di riprodurre e analizzare la struttura logica del linguaggio in uso nella pratica matematica (1879). Il primo ha prodotto una indagine completa della struttura matematica sottostante ai metodi propri del ragionamento formale, mentre il secondo ha definito una fondazione per la matematica stessa che attraverso l'analisi logica ne ha giustificato le definizioni e le procedure deduttive. Ora studiamo però come la logica di Boole ha permesso la sintesi con una diretta corrispondenza tra logica e gate elementari e circuiti.

Come detto si lavora ad un livello di astrazione superiore del livello fisico ed indipendente dal fatto che l'informazione venga trasmessa da una grandezza fisica elettronica, meccanica, biologica o quant'altro. *La logica di commutazione si basa sul fatto che esistono segnali trasmessi ed elaborati di tipo elettronico, ma trasformati in natura digitale come valori 1 o 0 (vero o falso).* La descrizione

tramite l'algebra di Boole ne sottende una struttura logica ; e' anche una descrizione funzionale che e' univoca nel momento in cui si ritorna alla tabella della verità. Attraverso i teoremi dell'algebra si possono poi realizzare sintesi diverse.

In questa corrispondenza impieghiamo la **logica positiva**, cioè utilizziamo i simboli 0 e 1 per rappresentare — rispettivamente — il valore inferiore e il valore superiore dei segnali in corrispondenza dei valori logici di falso e vero. Il valore 0 (falso) corrisponde all'assenza di segnale o un segnale sotto la soglia minima (di solito di tensione); il valore 1 corrisponde alla presenza di un segnale (sopra ad una soglia massima). Se il segnale si trova in uno stato intermedio non e' utile per la logica booleana. E per ora, non si considera questa possibilità.

Esiste anche la possibilità di lavorare in **logica negativa** con le convenzioni opposte (1, gate attivo basso rappresenta l'assenza di segnale o il segnale inferiore alla soglia e 0 il contrario).

Come si definisce l'algebra di Boole?

Def. L'algebra di Boole è un sistema matematico che descrive funzioni di variabili binarie: è composto da:

- un insieme di simboli: $\boldsymbol{B} = \{\mathbf{0}, \mathbf{1}\}$
- un insieme di operazioni: $\boldsymbol{O} = \{+, \cdot, '\}$
  - $+$: somma logica (OR, oppure V)
  - $\cdot$: prodotto logico (AND oppure ^)
  - $'$: complementazione (NOT oppure $^-$ oppure ¬)
- un insieme di postulati (assiomi) $\boldsymbol{P}$:

P1) 0+0=0	P5) 0 •0= 0	P9) 0' =1
P2) 0+1=1	P6) 0 •1= 0	P10) 1' =0
P3) 1+0=1	P7) 1 •0= 0	
P4) 1+1=1	P8) 1 •1= 1	

Nell'algebra esistono **le COSTANTI** dell'algebra (0 ed 1) e le **VARIABILI**, ossia un qualsiasi simbolo che può essere sostituito da una delle due costanti.

Def. Un'**espressione** secondo l'algebra di Boole è una stringa di elementi di B che soddisfa una delle seguenti regole:

a) una costante è un'espressione.
b) una variabile è un'espressione.
c) Se X è un'espressione allora il complemento di X è un'espressione.
d) Se X ed Y sono espressioni allora la somma logica di X e Y è un'espressione.
e) se X ed Y sono espressioni allora il prodotto logico di X ed Y è un'espressione.

Percio' se e' vero che una rete logica e' rappresentata con una espressone Booleana questa definizione spiega le proprietà di interconnessione e di decomposizione.

3.4 FUNZIONI BINARIE E GATE ELEMENTARI

Quante sono le possibili funzioni binarie di n variabili indipendenti**?** Una tabella di verità di n variabili di ingresso ha 2^n righe. Ma quante diverse tabelle di verità ad una uscita si possono costruire? Tutte le combinazioni delle uscite per ogni configurazione di ingresso, ossia 2 elevato al numero delle possibili configurazioni di ingresso, quindi $2^{(2\text{^}n)}$ (attenzione che questo non è 4n!).
Dati n segnali di ingresso sono possibili $2^{(2\text{^}n)}$ reti logiche diverse, ossia $2^{(2\text{^}n)}$ funzioni booleane diverse che corrispondono ad altrettante tabelle di verità.
Ad es. con 4 ingressi la tabella ha $2^{(2\text{^}n)} = 2^4$=16 righe, ed esistono 2^{16} =64K possibili reti logiche diverse con quei 4 segnali di ingresso.

FUNZIONI DI UNA VARIABILE INDIPENDENTE

Sia data una tabella di verità con tutte le possibili reti logiche o funzioni di una sola variabile indipendente. Quante sono le reti logiche che si possono progettare con 1 variabile booleana di ingresso? L'ingresso può assumere due valori (0 o 1) e quindi saranno possibili 4 funzioni booleane: z_0, z_1, z_2, z_3.

Fig. 5: Funzioni di una variabile.

Queste 4 funzioni a una variabile $z_i = f(x)$ hanno un significato fisico immediato. z_1 e z_3 (*massa* e *alimentazione*) sono legati ai concetti del livello più basso dei circuiti elettrici: una funzione che ritorna sempre 0 corrisponde a un collegamento a massa, mentre una funzione che ritorna sempre 1 è come un circuito collegato all'alimentazione V_{CC}. La funzione z_1 che non modifica il segnale si chiama ***buffer*** (anche se in elettronica questo termine implica anche una funzione di amplificazione), mentre z_2 che cambia sempre l'ingresso si chiama **NOT o *invertitore***.

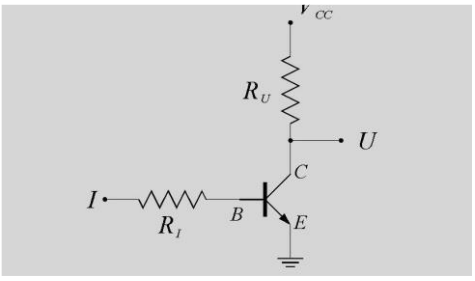


Fig.6 Transistor bipolare che realizza il Not Gate (NOT)

Attraverso perciò un gate elementare NOT si può sempre realizzare la operazione logica di complementazione NOT ed i teoremi conseguenti. Ad es. due NOT in cascata non cambiano il valore dell'ingresso. Perche' (X')'=X.

FUNZIONI DI DUE VARIABILI INDIPENDENTI

Con due ingressi, gli ingressi possono assumere 4 configurazioni, quindi si possono costruire 16 funzioni. Delle 16 possibili funzioni logiche di 2 variabili indipendenti, solo 6 meritano di essere nominate. 3 hanno un corrispondente diretto nelle espressioni logiche:

- AND (prodotto logico): l'uscita è vera solo se entrambi i termini sono veri.
- OR (somma logica): l'uscita è vera se almeno uno dei termini è vero.
- EXOR (disuguaglianza): l'uscita è vera se uno solo dei termini è vero, ma non entrambi.

Le altre 3 funzioni sono il complemento (negato) di queste tre: NAND, NOR e EXNOR.

AND, OR, NOT assieme a EXOR, NOR, NAND e EXNOR sono detti "**gate elementari**" delle reti logiche e corrispondono direttamente all'algebra booleana.

Questi hanno una sola *descrizione funzionale* (la funzione di ingresso ed uscita), ma possono essere *sintetizzati* a livello di circuiti in modi diversi utilizzando un certo numero di *transistor*. La sintesi delle reti logiche si studia nel corso di Elettronica.

Fig. 7: Funzioni di due variabili indipendenti.

Questi gate elementari (indipendentemente dalla loro realizzazione tecnologica) possono poi essere combinati per formare reti logiche più complesse e a più ingressi. La combinazione viene descritta grazie a diagrammi detti *schemi logici* in cui i segnali di ingresso e uscita vengono rappresentati da linee che provengono dall'esterno ed entrano in alcune porte elementari. Le uscite delle porte elementari possono poi essere trasferite ad altre porte elementari o in uscita. L'indipendenza tecnologica è reale, dato che si possono realizzare funzioni logiche tramite sistemi elettrici (relè ed interruttori), tramite sistemi idraulici e pneumatici (i segnali sono linee che possono essere in pressione o meno), con metodi meccanici (i primi progetti di sistemi di calcolo erano appunto meccanici) ed infine elettronici, tecnologia ad oggi notoriamente vincente e dominante.

Tra tutte le funzioni logiche spesso si impiegano solo un loro sottoinsieme:

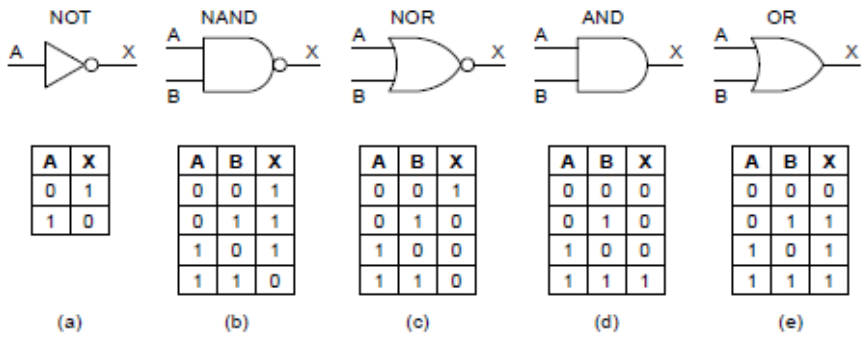
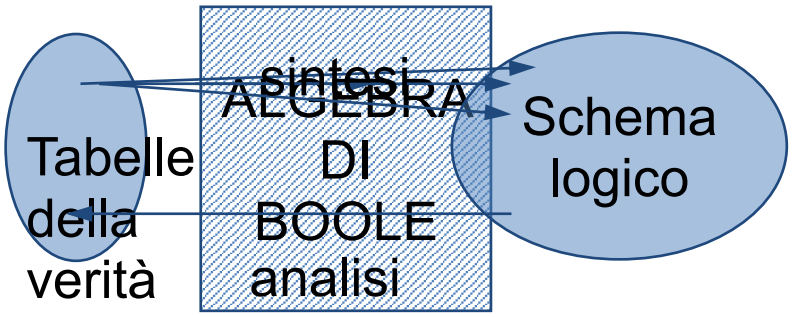


Fig. 8: Gate elementari utilizzati più di frequente.

Si può dimostrare che tutte le reti logiche possono essere realizzate impiegando solo AND , OR e NOT, in corrispondenza con l'algebra di Boole.

Esiste una **diretta corrispondenza tra algebra di Boole e alcuni blocchi di base detti gate elementari**. I gate elementari che sono gli elementi di base delle reti logiche e sono stati progettati e realizzati per corrispondere alle più comuni funzioni binarie. E se è vero che si possono realizzare attraverso circuiti (ora elettronici) gli elementi fondamentali dell'algebra di Boole allora **TUTTE LE FUNZIONI LOGICHE rappresentabili dall'algebra di Boole si possono rappresentare con RETI LOGICHE**.



Legame tra algebra di Boole e Reti Logiche

C'e una corrispondenza quindi tra espressione booleana e schema logico (della rete logica). Avendo a disposizione uno schema logico si può ottenerne l'analisi attraverso una descrizione comportamentale mediante espressioni dell'algebra di Boole o con una rappresentazione esaustiva tramite la tabella della verità. Dallo schema logico, tramite le espressioni è possibile ricavare il comportamento di una rete logica, ossia si può passare dallo schema alla descrizione comportamentale ottenendo la funzione booleana corrispondente.

3.5 I TEOREMI DELL'ALGEBRA DI BOOLE

Esistono diversi teoremi dell'algebra di Boole che sono stati dimostrati e che sono alla base delle reti logiche e necessari per la sintesi di reti logiche:

Proprietà di chiusura: $\forall a, b \in B \Rightarrow a + b \in B, a \cdot b \in B$.

Ci indica che ogni operazione logica con ingessi dati da due segnali binari rappresentati da variabili binarie, porta una uscita che è ancora una variabile binaria. Percio' se una RL rappresenta una operazione logica la sua uscita può essere l'ingresso di un'altra RL.

Principio di Dualità:

- ogni espressione algebrica presenta una forma duale ottenuta scambiando l'operatore OR con AND e viceversa, la costante 0 con la costante 1 e viceversa e mantenendo i letterali invariati.
- ogni proprietà vera per un'espressione è vera anche per la sua duale.

Il principio di dualità è molto utile per trattare segnali attivi alti e segnali attivi bassi, ossia lavorare con logica positiva o negativa e realizzare sintesi differenti in base a vincoli (ad es. se possiamo impiegare solo AND e non OR etc).

TEOREMA DI IDENTITÀ

(T1) X + 0 = X

(T1') X · 1 = X

Le costanti 0 ed 1 sono gli elementi neutri (che portano all'identità) per la somma e il prodotto logico, rispettivamente.

TEOREMA DEGLI ELEMENTI NULLI

(T2) X + 1 = 1

(T2') X · 0 = 0

Le costanti 1 e 0 sono gli elementi nulli (che portano alla costante) per la somma e il prodotto logico, rispettivamente.

TEOREMA DELL'IDEMPOTENZA

(T3) X + X = X

(T3') X · X = X

Le operazioni di somma e prodotto logico sono idempotenti e portano al collasso funzionale verso la variabile stessa.

TEOREMA DELL'INVOLUZIONE

(T4) (X ')' = X

La operazioni di negazione logica è involutiva (due negazioni affermano) .

TEOREMA DELLA COMPLEMENTARIETÀ (O DELL'INVERSIONE)

(T5) X + X' = 1

(T5') $X \cdot X' = 0$

Ogni variabile sommata (o moltiplicata) logicamente a se stessa porta alla costante della somma ,1 . (o del prodotto 0) come per la proprietà degli elementi nulli.

PROPRIETA COMMUTATIVA

(T6) $X + Y = Y + X$

(T6') $X \cdot Y = Y \cdot X$

PROPRIETA ASSOCIATIVA

(T7) $(X + Y) + Z = X + (Y + Z) = X + Y + Z$

(T7') $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$

Somma e prodotto logico godono della proprietà commutativa ed associativa .

TEOREMA DELL'ASSORBIMENTO

(T8) $X + X \cdot Y = X$

(T8') $X \cdot (X + Y) = X$

Questo teorema è molto importante e permette di ridurre il numero di porte logiche. Si dimostra dai teoremi precedenti anche per enumerazione.

PROPRIETÀ DISTRIBUTIVA

(T9) $X \cdot Y + X \cdot Z = X \cdot (Y + Z)$

(T9') $(X + Y) \cdot (X + Z) = X + Y \cdot Z$

PROPRIETÀ DELLA COMBINAZIONE

(T10) $(X + Y) \cdot (X' + Y) = Y$

(T10') $X \cdot Y + X' \cdot Y = Y$

Si può vedere come una applicazione della proprietà distributiva. Infatti:

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

per il teorema della complementarietà $X + X' = 1$

$(X + X') \cdot Y = 1 \cdot Y = Y$ per il teorema di identità.

TEOREMA DEL CONSENSO

(T11) $(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$

(T11') $X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$

Questo teorema è indispensabile per semplificare le espressioni logiche. Quando si arriva alla fine di un Esempio , controllare sempre se per caso si può applicare questo teorema!

Da questo si può ricavare un altra forma molto comune:

$X + X' \cdot Z = X + Z$

Il primo termine è uguale al secondo termine di T11' ponendo Y=1, quindi:

$X + X' Z = X \cdot 1 + X' Z = X \cdot 1 + X' \cdot Z + 1 \cdot Z = X + X' \cdot Z + Z =$

applicando il teorema dell'assorbimento $X' \cdot Z + Z = Z$, quindi $= X + Z$

TEOREMA DI DE MORGAN

(T12) $(X + Y)' = X' \cdot Y'$

(T12') $(X \cdot Y)' = X' + Y'$

Quindi, esprimendo a parole il teorema possiamo dire che: il valore negato della somma è il prodotto dei valori negati e il valore negato del prodotto è la somma dei valori negati. Questo teorema è facilmente generalizzabile per più variabili.

Identità	(T1) $X + 0 = X$ (T1') $X \cdot 1 = X$	Associativa	(T7) $(X + Y) + Z = X + (Y + Z) = X + Y + Z$ (T7') $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z$
Elementi nulli	(T2) $X + 1 = 1$ (T2') $X \cdot 0 = 0$	Assorbimento	(T8) $X + X \cdot Y = X$ (T8') $X \cdot (X + Y) = X$
Idempotenza	(T3) $X + X = X$ (T3') $X \cdot X = X$	Distribuzione	(T9) $X \cdot Y + X \cdot Z = X \cdot (Y + Z)$ (T9') $(X + Y) \cdot (X + Z) = X + Y \cdot Z$
Involuzione	(T4) $(X ')' = X$	Combinazione	(T10) $(X + Y) \cdot (X' + Y) = Y$ (T10') $X \cdot Y + X' \cdot Y = Y$
Complementarietà	(T5) $X + X' = 1$ (T5') $X \cdot X' = 0$	Consenso	(T11) $(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$ (T11') $X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$
Commutativa	(T6) $X + Y = Y + X$ (T6') $X \cdot Y = Y \cdot X$	De Morgan	(T12) $(X + Y)' = X' \cdot Y'$ (T12') $(X \cdot Y)' = X' + Y'$

Fig. 9: Teoremi dell'algebra di Boole

3.6 DESCRIZIONE DELLE RETI LOGICHE

Esistono diversi metodi per descrivere le reti logiche, e sono fondamentali per tre motivi

a) per l'analisi, per dare una descrizione comportamentale,
b) per la documentazione, per rappresentare in modo standard e codificato cosa fa un sistema
c) per la sintesi, per generare le reti e l'hardware in modo automatico al calcolatore.

Questi metodi di descrizione sono

1) **Descrizione a parole**: descrizione a parole in linguaggio naturale del comportamento della rete logica.

2) **Tabelle di verità:** descrizione esaustiva di tutte le configurazioni di uscita per ogni possibile configurazione di ingresso. È la descrizione comportamentale più diretta ed è il punto di partenza più semplice per ottenere la sintesi automatica.

3) **Mappe (di Karnaugh):** altra rappresentazione delle tabelle della verità, in forma più compatta utilizzabile manualmente solo per funzioni molto semplici con pochi ingressi.

4) **Espressioni dell'algebra Booleana:** metodo formale impiegato come ingresso da sistemi di sintesi automatica.

5) **Schema logico:** descrizione strutturale, utile per la documentazione.

6) **Forme d'onda:** descrizione comportamentale in funzione del tempo, tipico delle reti sequenziali.

7) **Linguaggi di descrizione dell'hardware:** impiego di linguaggi procedurali e paralleli come il VHDL, sia per reti combinatorie che sequenziali.

Le mappe sono una rappresentazione matriciale o *geometrica* della tabella di verità, che ci permettono di rappresentarla in modo più compatto, in cui le righe indicano tutte le possibili configurazioni di un sottoinsieme delle variabili di ingresso e le colonne tutte le configurazioni delle variabili rimanenti, il valore nelle celle indica il valore dell'uscita nella configurazione corrispondente.

Tra le diverse mappe si ricordano le storiche **Mappe di Karnaugh**: mappe in cui le configurazioni successive in ogni lato sono **ADIACENTI**.

Def: *Due configurazioni sono adiacenti se differiscono di un solo bit (due celle sono adiacenti se corrispondono a configurazioni adiacenti) .*

La differenza bit a bit tra due espressioni si chiama distanza di Hamming. Due configurazioni adiacenti hanno distanza di Hamming=1

X_2	X_1	X_0	Z
0	0	0	0

0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	1

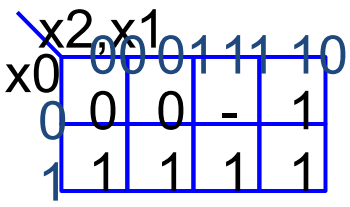


Fig. 10: Tabella e Mappa di Karnaugh di tre variabili.

Rappresentare le **tabelle come mappe è** molto utile per ottenere sintesi minime di reti logiche in modo manuale, ma senza l'ausilio del calcolatore il procedimento è applicabile solo per un numero limitato di ingressi. Esistono delle regole manuali, e poi algoritmi e programmi per passare da una descrizione con la mappa a una sintesi ottimale secondo alcuni parametri (risparmio di numero di componenti, di tempo..).

ESPRESSIONI DELL'ALGERA BOOLEANA

Il metodo più impiegato per descrivere le funzioni combinatorie è di lavorare con i gate elementari che sono la rappresentazione logica dell'algebra di Boole e sono direttamente realizzabili con due o più transistor in logica digitale. Le reti logiche combinatorie sintetizzano funzioni combinatorie e si possono descrivere facilmente con l'algebra di Boole usando gli operatori algebrici AND, OR e NOT. Ad es. osservando la tabella della Fig. 10 o la mappa corrispondente si può osservare l'uscita Z (Vero=1 Falso=0). Innanzitutto esiste una uscita non completamente specificata che per la sintesi deve essere specificata indifferentemente o con uscita 0 o con uscita 1.

Ci sono quindi due soluzioni come due sintesi possibili, **espresse come somme (OR) di prodotti (AND)**

Z= (x2'*x1'*x0) + (x2'*x1*x0) + (x2*x1'*x0') + (x2*x1'*x0) +(x2*x1*x0)

se – viene considerata 0

Z= (x2'*x1'*x0) + (x2'*x1*x0) + (x2*x1'*x0') + (x2*x1'*x0) (x2*x1*x0')+(x2*x1*x0)

se – viene considerata 1.

Perchè sono vere entrambe? Perche' la tabella non è completamente specificata e quindi sono indifferenti. La sintesi non è univoca. In ogni caso anche se fosse completamente specificat,a la sintesi che porta ad una descrizione strutturale non è univoca perché possono essere impiegati i teoremi dell'algebra di Boole per ottenere altre sintesi possibili. Considerando la seconda ad esempio si ottiene una grande semplificazione.

Esercizio 7. Si semplifichi la espressione che segue, usando i teoremi dell'algebra di Boole

Z= (x2'*x1'*x0) + (x2'*x1*x0) + (x2*x1'*x0') + (x2*x1'*x0)+ (x2*x1*x0')+(x2*x1*x0)

Per proprietà commutativa essa è equivalente a

Z= (x2' *x0)*x1' + (x2'*x0)*x1 + (x2*x1')*x0' + (x2*x1')*x0' +(x2*x1)*x0' + (x2*x1)*x0

Per combinazione

Z= (x2' *x0) + (x2*x1') +(x2*x1)

Per la proprietà di combinazione

Z= (x2' *x0) + x2

Per assorbimento

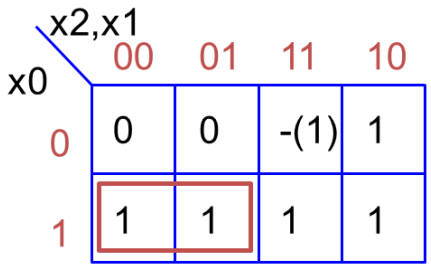
Z= (x2' *x0) + x2 + x2*x0

Per commutazione ed assorbimento

Z= x0+x2.

Sono tutte espressioni equivalenti, di cui l'ultima e' detta in **forma minima** perche' non puo' essere ulteriormente semplificata.

La forma Minima puo' essere trovata anche lavorando direttamente sulla mappa di Karnaough (k-map) in forma grafica usando le stesse regole e modelli visuali chiamati **Raggruppamenti rettangolari (RR).**



Trovare i RR significa verificare dove esistendo configurazioni adiacenti si può applicare la regola della combinazione . L'adiacenza vale anche in forma ciclica tra la prima e l'ultima configurazione.

(x2'*x1'*x0) + (x2'*x1*x0) → (x2'*x1)

SCHEMA LOGICO E FORME D'ONDA

Lo schema logico è una descrizione strutturale che rappresenta come è fatta la rete logica in termini di gate elementari o di aggregati e moduli di gate elementari in RTL. Come si sa la sintesi non è univoca e quindi esistono diversi schemi logici equivalenti che descrivono la stessa identica funzione. Essi sono equivalenti dal punto di vista logico ma essendo poi realizzati con gate elementari reali sono differenti in termini di area, consumi, tempi di risposta. I tempi di risposta in reti combinatoria sono in teoria ZERO ma di fatto essendo realizzati con circuiti elettronici sono tempi che dipendono dalla tecnologia impiegata e dalla sintesi stessa.

Le forme d'onda rappresentano il legame consequenziale e temporale tra ingressi d uscite, poco importante nelle reti combinatorie ma molto più importante nelle reti sequenziali.

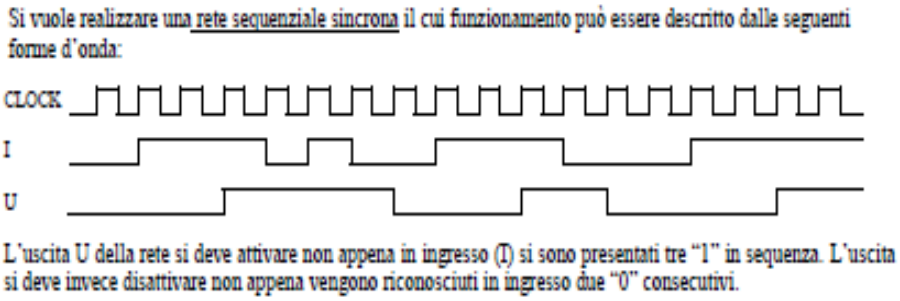


Fig. 11: Forma d'onda.

3.7 ANALISI DI UNA RETE LOGICA

Fare l'analisi di una rete logica significa ottenerne la descrizione comportamentale ossia ottenere la descrizione esaustiva mediante tabella della verità od una espressione di Boole equivalente.

Dato che l'analisi e' univoca esiste **un algoritmo** semplice che permette di passare dallo schema logico alla tabella. Data una rete logica combinatoria

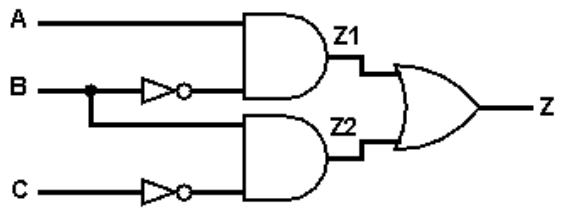
1) si nominano tutte le uscite dei gate logici;

2) si sostituiscono i gate elementari a partire dalle uscite con le corrispondenti espressioni della logica;

3) si semplifica, se necessario con i teoremi dell'algebra di Boole;

4) si fa la valutazione dell'espressione per ottenere in modo esaustivo al tabella della verità.

Esercizio 8: analizzare la rete logica



$$\boldsymbol{Z = Z_1 + Z_2}$$

$\boldsymbol{Z_1 = A\overline{B}}$ e $\boldsymbol{Z_2 = B\overline{C}}$

$$\boldsymbol{Z = A\overline{B} + B\overline{C}}$$

Poi bisogna farne la valutazione.

Def. Valutazione di una espressione: *ogni espressione di n variabili descrive una funzione completamente specificata che può essere* ***valutata*** *attribuendo ad ogni variabile un valore assegnato ed applicando i postulati e teoremi dell'algebra.*

Esempio: $f(a,b,c) = (A' + B) \cdot C$. *Con* $(A,B,C) = (0,0,1)$, *la funzione* $f(A,B,C) = 1$.

Esercizio 9 Valutare la rete logica descritta dalla espressione

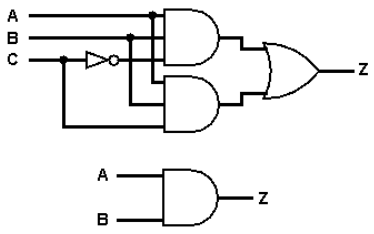
$$\boldsymbol{Z = A\overline{B} + B\overline{C}}$$

A	*B*	*C*	*Z*
0	*0*	*0*	*0*
0	*0*	*1*	*0*
0	*1*	*0*	*1*
0	*1*	*1*	*0*
1	*0*	*0*	*1*
1	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*0*

La valurazione è la stessa sia che si parta da una espressione semplificata o piu' complesse, che sono esempi equivalenti con la stessa tabella della verità.

I teoremi dell'algebra booleana sono impiegati (non solo manualmente, ma anche nei software di sintesi logica) per semplificare le espressioni e usare, quando necessario, meno gate elementari.

Esercizio 10*: le due reti logiche sono equivalenti?*



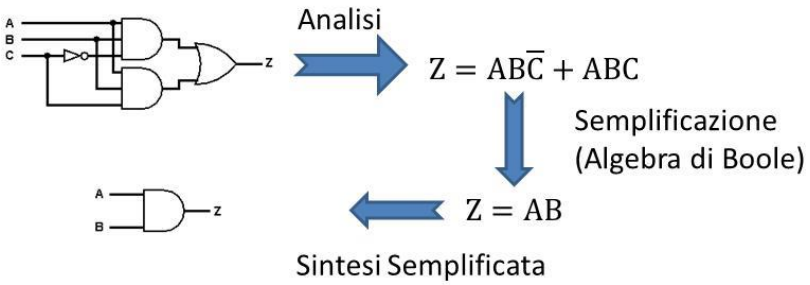
$Z = AB\overline{C} + ABC$

$Z = (AB)(\overline{C}C)$ *proprietà distributiva*

$Z = (AB) \cdot 1$ *teorema complementarietà*

$Z = AB$ *teorema identità*

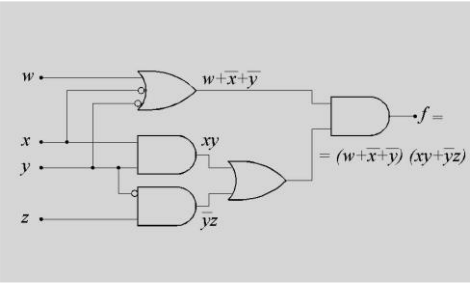
Dall'Esercizio si vede che la rete logica di sopra può essere semplificata con una espressione logica più semplice da cui ottenerne una sintesi più semplice.



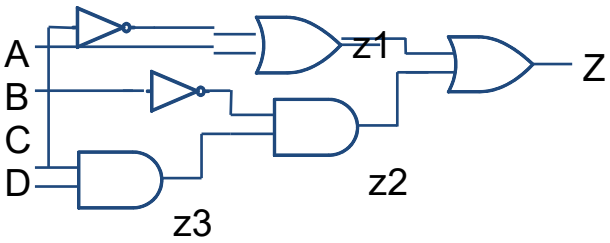
A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Fig. 12: Semplificazione di una rete logica

Esercizio 11*: valutare la seguente espressione ottenendone la tabella della verità e semplificarla con algebra di Boole*



Esercizio 12: *semplificare la rete logica con l'algebra di Boole*



3.8 SINTESI DI UNA RETE LOGICA (COMBINATORIA)

Progettare una rete logica significa farne la sintesi a partire da una descrizione comportamentale (ad es. con la tabella di verità) fino al suo schema logico che ne rappresenta una descrizione strutturale.

Una espressione logica è una descrizione funzionale ma se messa in corrispondenza ai gate elementari ne rappresenta anche una possibile sintesi. Esistono molti modi di fare la sintesi. In passato si cercava di ottenere (spesso in modo manuale) sempre **la sintesi *minima*** che potesse impiegare il numero minimo di gate elementari, anche a scapito della leggibilità del progetto e del numero di connessioni tra gate. Nelle sintesi attuali, dove si ottengono reti con milioni di gate è conveniente invece ottenere una sintesi facilmente leggibile e facilmente comprensibile anche con strumenti formali di logica. Questa si ottiene attraverso la **sintesi canonica.** Sia dato un circuito combinatorio con n variabili di ingresso, X: {X1..Xn}.

Si chiama **letterale** ogni occorrenza di una singola variabile, sia in forma semplice Xi che complementata Xi ' .

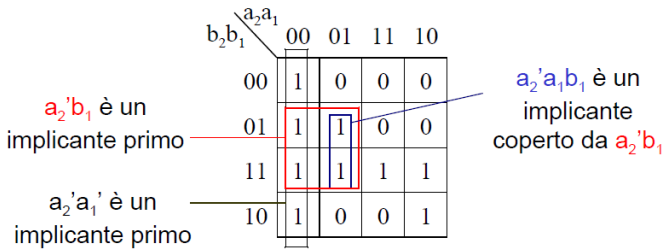


Fig. 13: Esempio di implicanti

Si dice **Implicante di una funzione** f(X1..Xn) *un prodotto P(X') X'⊂X di letterali tale che se P=1 ⇒ f=1.*

Date due funzioni f(x1,x2 ,..,xn) e g(x1,x2,..,xn) si dice che **f copre g** (oppure **g implica f**) e si scrive f ⊃g se f(x1,x2,..,xn) =1 quando g(x1,x2,..,xn) =1; Se P è il prodotto di letterali e f copre P, si dice che P è un implicante di f.

Esempio f= abc+ ab'+ac e P= ab' allora P è un implicante di f.

Si chiama **implicante primo** di una funzione *f un implicante di f che non è coperto da un altro implicante di f con meno letterali.* Gli implicanti e gli implicanti primi si possono vedere nelle

mappe di Karnaugh dove i Raggruppamenti rettangolari che coprono le uscite pari a 1, indicano gli implicanti e gli implicanti primi se non esistono altri raggruppamenti rettangolari che li coprono.

Si dice ***Mintermine*** *un implicante in cui appaiono tutti i letterali ossia tutte le variabili di ingresso,* cioè X' o X. La funzione prende il nome di **prodotto fondamentale (o mintermine**) dal momento che è costituita dal prodotto logico dei letterali delle variabili di ingresso, presi negati se nella riga considerata la corrispondente variabile di ingresso vale 0, non negati se tale variabile vale 1.

Si definisce mintermine di una funzione f(X1..Xn) un punto Bx nello spazio booleano Bn tale per cui f(Bx)=1

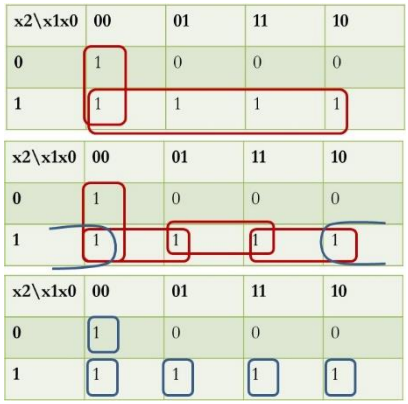


Fig. 14: Esempio di mintermini, implicanti, implicanti primi

Un implicante si dice ***implicante primo essenziale*** se comprende almeno un mintermine che non è coperto da nessun altro implicante primo.

Si definisce **maxtermine** di una funzione f(X1..Xn) un punto Bx nello spazio booleano Bn tale per cui f(Bx)=0. La (seconda) funzione prende il nome di **somma fondamentale (o maxtermine)** poiché è costituita dalla somma logica dei letterali delle variabili di ingresso, presi non negati se nella riga considerata la corrispondente variabile di ingresso vale 0, negati se tale variabile vale 1. Si definisce **implicato** una somma di letterali che impongono 0 alla funzione, e nello stesso modo si parla di implicato primo od essenziale.

Nelle mappe di Karnaough (e nelle tabelle) si possono vedere gli implicanti, ossia quelle configurazione di ingressi che implicano che la funzione valga 1 e naturalmente anche gli implicati). Sia data la funzione:

Z= x2+x1'x0'

Entrambi x2 e (x1'x0') sono implicanti primi ed essenziali. Ma e' anche vero che se considerassimo una forma non minima

Z= x2+x2x1'+x1'x0' + x2x1'x0'

x2x1' e' un implicante ma non e' essenziale perche' esiste l'implicante x2 che lo copre. Invece ad es. x2x1'x0' e' un mintermine.

Esercizio 13: Si vuole progettare una rete logica che ha tre ingressi A,B e C. Sono attivi quando assumono il valore 1 (attivi alti). L'uscita e' vera quando i tre ingressi sono uguali oppure quando a e' diverso dagli altri due oppure quando a e c non sono attivi contemporaneamente. Quali sono i mintermini? quali gli implicanti ? quali gli implicanti primi? quali i primi essenziali?

SINTESI CANONICA

La sintesi canonica e' la sintesi realizzata attraverso i mintermini o i maxtermini. Si ottiene osservando ogni riga della tabella della verità. Ad ogni riga di una tabella delle verità si possono associare due funzioni logiche particolari: una che assume valore 1 solo quando gli ingressi presentano la configurazione associata a tale riga, una che assume valore 0 solo quando gli ingressi presentano la configurazione associata a tale riga.

Esistono due tipi di **sintesi canonica** equivalenti e duali:

- **SP**: Somma di prodotti;
- **PS**: Prodotto di somme.

La **prima forma canonica SP** e' definita come la **somma logica dei mintermini associati alle righe della tabella nelle quali l'uscita assume valore 1**.
La **seconda forma canonica PS** e' definita dal **prodotto logico dei maxtermini associati alle righe della tabella nelle quali l'uscita assume valore 0**.

Esercizio 14: Sia data la tabella che segue. Quale e' la sintesi canonica SP e PS?

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

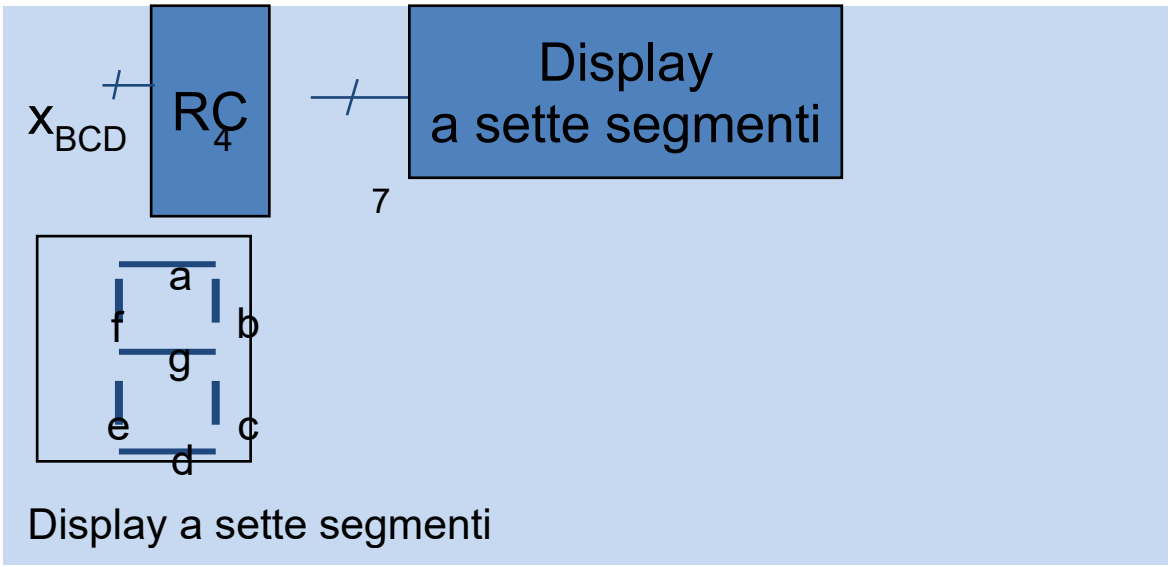
La sintesi canonica SP si ottiene come
Z= (ABC')+(ABC) È la somma di due prodotti a tre ingressi che sono mintermini.
La sintesi canonica PS si ottiene come
Z= (A+B+C)* (A+B+C')* (A+B'+C)*(A+B'+C')*(A'+B+C)*(A'+B+C')
È il prodotto di 6 somme logiche a tre ingressi.

Si dimostra con l'algebra di Boole (e con de Morgan) che le due reti sono assolutamente equivalenti. È conveniente impiegare la sintesi SP o PS? Dipende da quante sono le uscite che assumono il valore 1, o in genere dai componenti che si hanno a disposizione.
Nelle reti logiche complesse e' il tool di sintesi logica che ottimizza la sintesi scegliendo quella migliore.

La sintesi canonica non è minima ma può essere minimizzata con diversi algoritmi tra i quali quelli che impiegano la semplificazione algebrica.
La sintesi canonica ha pero' il vantaggio di garantire la massima velocità dato che impiega due livelli di gate S e P o P ed S, ed inoltre permette il riuso degli elementi logici se si devono realizzare piu' reti logiche con gli stessi ingressi.
Non e' necessario quindi cercare per forza la sintesi minima.

Si progetti con sintesi canonica la rete combinatoria di *Conversione di valori BCD su display a sette segmenti*.

Descrizione comportamentale (a parole): progettare una rete logica che permetta la visualizzazione su un display a sette segmenti di un valore in codice BCD, avendo in ingresso i 4 bit del codice BCD ed in uscita i 7 segmenti a,b,…g che valgono 1 on 0 off e ordinati in sequenza oraria a partire dall'alto. In questo caso l'uscita {a,b,…,g} dipende in ogni istante dalla configurazione degli ingressi $\{x_1,x_2,x_3,x_4\}$ per ogni cifra BCD, e per questo è una rete combinatoria.



Nella codifica a 7 segmenti, la tabella non è completamente specificata perché per la codifica BCD le configurazione 1010, 1011, 1100, 1101, 1110 e 1111 sono vietate.

x3	*x2*	*x1*	*x0*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	-	-	-	-	-	-	-
1	0	1	1	-	-	-	-	-	-	-
1	1	0	0	-	-	-	-	-	-	-
1	1	0	1	-	-	-	-	-	-	-
1	1	1	0	-	-	-	-	-	-	-
1	1	1	1	-	-	-	-	-	-	-

Tab. 4: La tabella di verità del codice a sette segmenti.

Esercizio 15: Come e' la sintesi canonica di tipo SP ? e PS? Del codice a 7 segmenti? Quante sono le sintesi canoniche per ogni codice a sette segmenti?

Perche' e' utile usare questa forma?
Provare a disegnarne lo schema.

3.9 VALUTAZIONE DELLA COMPLESSITA' DI UNA RETE LOGICA

Si può valutare con una notazione convenzionale la complessità realizzativa della funzione indicata. Tale notazione riporta (senza contare i livelli NOT)

- L) il **numero di Livelli di porte logiche** che il segnale associato a un qualsiasi letterale di ingresso deve attraversare per raggiungere l'uscita (quindi un'indicazione della rapidità della rete),
- G) il **numero di Gate** (cioè di porte logiche) necessari alla sintesi della rete (quindi un'indicazione della dimensione della rete),
- I) il **numero di Ingressi totali** alle porte logiche usate (quindi un'indicazione della dimensione media delle porte stesse).

Ossia le tre sintesi sono equivalenti

Z1= (ABC')+(ABC)

Z2= (A+B+C)* (A+B+C')* (A+B'+C)*(A+B'+C')*(A'+B+C)*(A'+B+C')

Z3=AB
Dal punto di vista logico-comportamentale ma non dal punto di vista strutturale; con la notazione precedente si può indicare
Z1→ 2L3G8I (sono 2 livelli con 3 gate, 2 AND e 1 OR, con 8 ingressi in tutto)
Z2→2L7G24I
Z3→1L1G2I

L**e sintesi canoniche sono sempre 2L**. Per ottenere la sintesi canonica di una rete logica, quindi

a) si descrive il suo funzionamento a partire dalla tabella della verità
b) si decide se operare la sintesi SP facendo al somma logica dei mintermini delle righe in cui l'uscita vale 1 o al sintesi PS con il prodotto logico dei maxtermini delle righe in cui l'uscita vale 0.

Se la tabella non è completamente specificata nella sintesi la si deve specificare.

Esercizio 16: la rete logica combinatoria di Maggioranza: $M = f(A, B, C)$ vale 0 se la maggioranza degli ingressi è 0, e 1 altrimenti. Farne una sintesi canonica

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

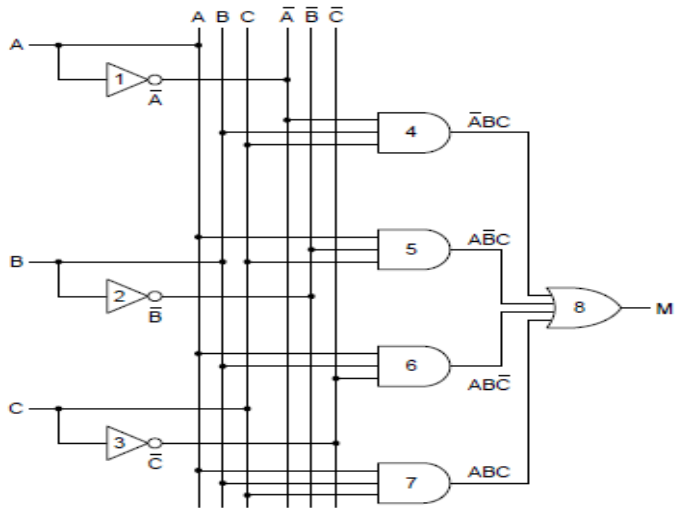


Fig. 15: Funzione di maggioranza.

Molto spesso nella logica, si usano solo i simboli AND, OR e NOT, ma essi potrebbero essere ottenuti impiegando solo NOR o solo NAND come nella figura che segue.

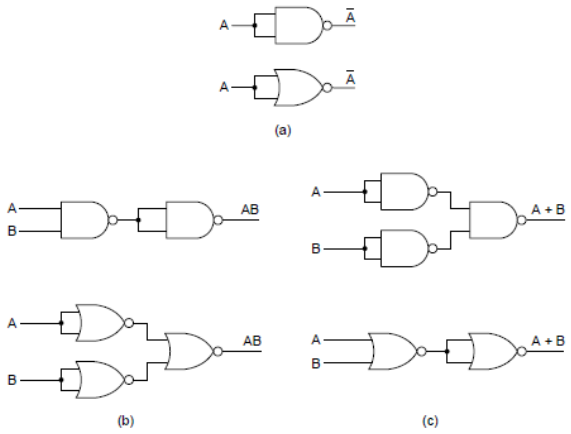


Fig. 16: Gate elementari con solo NAND e solo NOR.

Esercizio 17: sintesi con l'algebra di Boole e con la sintesi canonica

Viene fornita una assicurazione: se uomo e o ha meno di 30 anni o ha più di 30 anni ed ha figli; se ha più di 30 anni ma non ha figli e ,o è uomo o è sposato ;se ha più di 30 anni ma non ha figli e non è sposato. Fare la sintesi Ps della rete. Farne una Valutazione: Riceve una assicurazione una donna con figlio non sposata e con meno di 30 anni?

Esistono molti modi per realizzare in modo manuale e automatico la sintesi logica

1) usare le forme canoniche e minimizzare con i teoremi dell'algebra di Boole
2) usare le mappe di Karnough e determinare gli implicanti primi ed essenziali. E' un metodo manuale che funziona solo per reti assai piccole
3) usare algoritmi di sintesi logica: il piu' famoso e' l'algoritmo di Quine Mc Cluskey.

3.10 ALGORITMO DI QUINE MC CLUSKEY

L'algoritmo di Quine Mc Cluskey e' il metodo piu' impiegato per una sintesi minima di una rete canonica di un qualsiasi numero di ingressi (cosa non possibile con la sintesi manuale con tabelle). E' un metodo di minimizzazione tabellare (facilmente automatizzato in strumenti CAD), facile da tradurre in un algoritmo con il numero di variabili trattate teoricamente illimitato[1]. Consideriamo l'algoritmo per SP anche se esiste anche per PS.

Si impiega la **Distanza di Hamming definita come il numero di bit diversi tra due variabili binarie:** *Con due variabili binarie a e a' di n bit la distanza di Hamming vale d (numero intero tra 0 e n) se d è il numero di cifre binarie corrispondenti diverse tra le due variabili.*

Ad es.: *a = 00110010 e a' = 01110100* hanno distanza di Hamming pari a 2.

[1] Provare il codice http://arxiv.org/ftp/arxiv/papers/1404/1404.3349.pd

Se due configurazioni (mintermini o implicanti) hanno dH=1 allora avle la proprietà di combinazione e possono essere semplificati.

https://www.youtube.com/watch?v=G9_oICLaLBU

Si usa per ottenere la sintesi minima da una sintesi canonica anche con indifferenze

Es

Σ (2,3,6), Σ d(1,0)

L'algoritmo di **Quine McCluskey** parte dalla tabella della verita', e si compone di due passi

1. calcola tutti gli **implicanti primi, a** partire dai mintermini di una funzione
2. cerca la **copertura minima** ossia ottima perche' minimizza i costi della rete che abbia la descrizione funzionale richiesta (sia equivalente come equazione booleana combinatoria) ma contenga il numero minimo di implicanti primi.

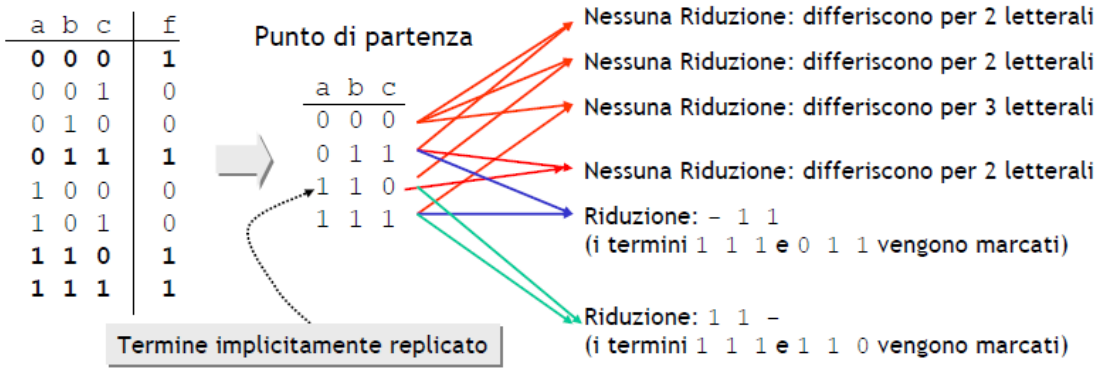
Passo 1: ricerca degli implicanti primi

a) Vengono tabellati tutti i mintermini della funzione in forma binaria, in ordine crescente come per la tabella della verità;

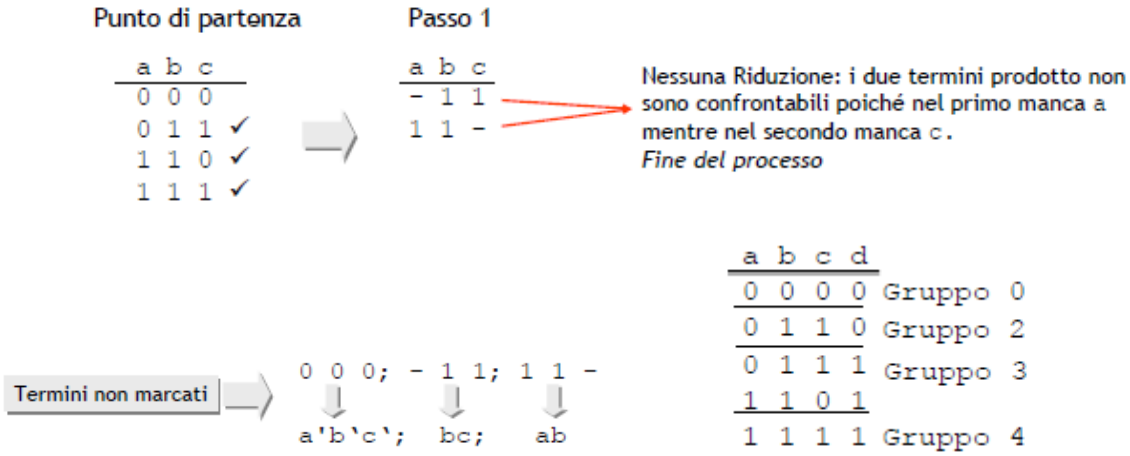
b) Si applica in forma iterativa la semplificazione *con la proprietà di combinazione* *xiP+xi'P= (xi+ xi')P= P* provando tutti i letterali xi.
In teoria si devono confrontare tra loro tutti i termini esaustivamente: per rendere la forma piu' agevole o iterativa in caso di codice si semplificano in modo iterativo i termini che differiscono per un solo bit (**distanza di Hamming unitaria**) e si marcano, in quanto essi hanno contribuito alla creazione di un implicante; quelli non marcati non si possono ridurre;

c) Si crea una nuova tabella con tutti i termini prodotto che derivano dalla semplificazione della prima tabella e si ripete il passo b);

d) il processo termina quando non si possono più fare riduzioni.



Nell'Esempio si considerano i 4 mintermini e si valutano a 2 a due in modo esaustivo; si marcano i minterimi che si riducono e si ricomincia.



I termini non marcati, che non si possono piu' ridurre sono quindi la sintesi con solo implicanti primi è Z= a'b'c'+ bc+ab.

Il numero di combinazioni a 2 a 2 di m mintermini cresce con il coefficiente binomiale e puo' essere troppo elevato. Per ridurre il numero di combinazioni si puo' evitare di confrontare i termini che hanno distanza di Hamming>1. Perciò si partiziona l'ON set in gruppi con lo stesso numeri di 1 e si confrontano solo gli elementi tra gruppi adiacenti. Cosi' nell'Esempio il gruppo 0 non viene confrontato con nulla, il gruppo 2 solo col gruppo 3 e il gruppo 3 col gruppo 4.

Si definisce il **gruppo come insieme Sij dei termini prodotto all'iterazione j con un numero di 1 pari a i;** ad ogni gruppo e' collegata una etichetta (label) che indica l'insieme dei mintermini che esso copre.

Algoritmo per Passo 1: implicanti primi;

j=0;

costruzione di S_i^o (la partizione in cui ogni S_i^o contenga tutti i mintermini con un numero di "1" pari a i);

repeat

for (k= min(i); k<=(max(i)-1) ; k++)

{si confronta S_k^j S_{k+1}^j;

se due configurazioni hanno dH=1 si genera un nuovo implicante (propr. combinazione)

si elimina la variabile con valore differente;

si marcano le configurazioni considerate e si pone il risultato in S_k^{j+1};}

j=j+1;

until (nessuna riduzione possibile);

Si consideri la k-map

X2/ x1x0	00	01	11	10
0	-	-	1	1
1	0	0	0	1

S0
000 (0)
S1
001 (1)
010 (2)
S2
011 (3)
110 (6)

Diventa con le combinazioni di configuirazioni adiacenti

00-	0,1
0-0	0,2
0-1	1,3
01-	2.3
-10	2,6

Continuando
S0
00- (0,1)
0-0 (0,2)
S1
0-1 (1,3)
01- (2,3)
-10 (2,6)

Semplificando ancora

0--	0,1, 2,3
0--	0,2, 1,3
-10	2,6

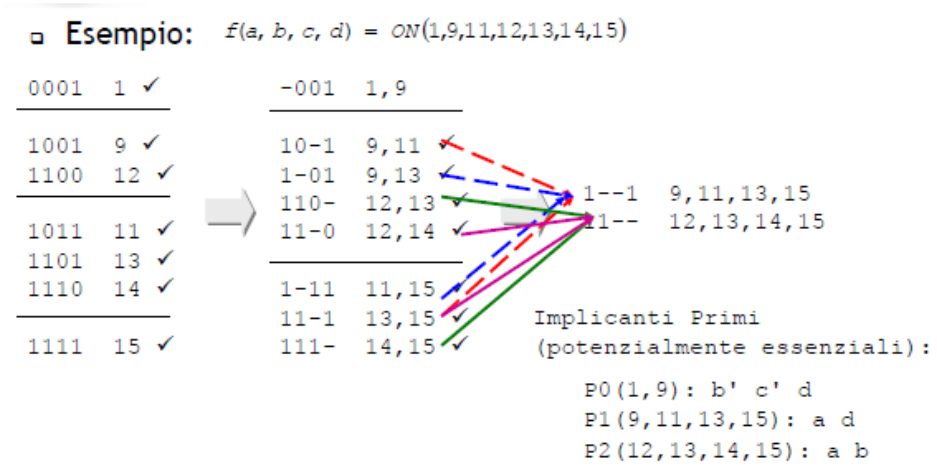
Sono implicanti primi. Anche essenziali? In questo caso si' e l'algoritmo e' finito.

Tutte le configurazioni non marcate sono implicanti primi e potenzialmente essenziali: un implicante si ricorda che si dice ***implicante primo essenziale*** se comprende almeno un mintermine che non è coperto da nessun altro implicante primo.

Esercizio 18: sia data una tabella di verita' con l' ON set dato dalle seguenti configurazioni f(a,b,c,d,)=ON(1,9,11,12,13,14,15). Farne la sintesi con QmC

Forma canonica: Z= a'b'c'd+ab'c'd+ab'cd+abc'd'+abc'd+abcd'+abcd

Gruppi: alla iterazione 0 S1 contiene la configurazione 1, S2 le configurazioni 9 e 12, S3 11,13,e 14 ed S4 con 15 . S0 e' vuota. Si eseguono le semplificazioni per combinazioni e sono tutti marcati e si passa alla iterazione 1 in cui si ha S0 con (1,9), S1 vuoto, S2 con label 9,11 e 9,13 e 12, 13 e 12, 14, S3 con label 11,15 e 15,15 e 14,15; S0 non e' confrontato con nessuno e non si puo' marcare. S2 e' confrontato con S3.Si passa alla iterazione 2 dove nulla si puo' semplificare. Gli implicanti primi sono quindi
P(1,9)= b'c'd'
P(9,11,13,15)= ad
P(12,13,14,15)=ab



Z= b'c'd+ad+ab questa e' una forma minima con implicanti primi. Ma sono anche essenziali?

Passo 2: copertura

Per vedere se sono essenziali si devono trovare tra gli implicanti primi, l'insieme minimo di quelli che ottengono la copertura di tutto l'ON_SET. Si usa la tabella degli implicanti (o **tabella di copertura**) *che e' una matrice binaria i cui indici di riga sono gli implicanti primi e la colonna sono i mintermini dell'ON-set e l'elemento aij della tabella viene marcato se l'implicante i copre il mintermine j.* Nell'Esempio di sopra:

	1	9	11	12	13	14	15
P0	x	x					
P1		x	x		x		x
P2				x	x	x	x

La ricerca della copertura ottima dipende dalla funzione di costo.

Se gli implicanti hanno tutti lo stesso costo si minimizza solo la cardinalità della copertura. Algoritmi piu' sofisticati mettono in costo per ogni implicante. Il problema della copertura e' NP-completo e la complessità cresce esponenzialmente con le dimensioni. Si usano tre criteri e metodi di ricerca operativa come il branch&bound per trovare l'ottimo.

a)Criterio di essenzialità: si verifica se esistono implicanti primi ed essenziali ; *se in una colonna esiste una sola marcatura con x la riga e' essenziale.* Si eliminano le righe e le colonne essenziali.

	1	9	11	12	13	14	15
P0	x	x					
P1		x	x		x		x
P2				x	x	x	x

P0 e P2 sono essenziali per 1 e 12; si ricomincia. E si considera anche P1. In questo Esempio semplice non ci sono riduzioni.

Esercizio piu' complesso; P3 e' implicante primo ed essenziale per J e copre anche A,C,D,F,G,H Rimangono B,E,I, K da coprire

	A	B	C	D	E	F	G	H	I	J	K
P0	x	x									x
P1		x	x				x		x		
P2				x	x	x	x				x
P3	x		x	x		x	x	x		x	
P4	x	x			x	x		x	x		

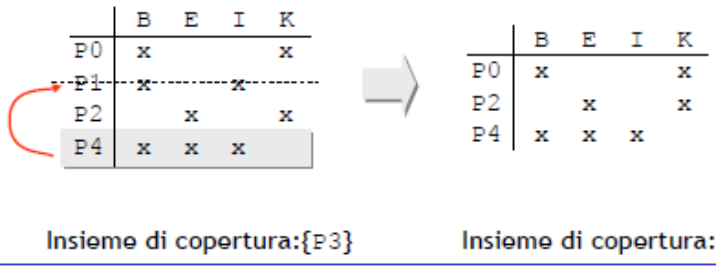
Insieme di copertura:∅

	B	E	I	K
P0	x			x
P1	x		x	
P2		x		x
P4	x	x	x	

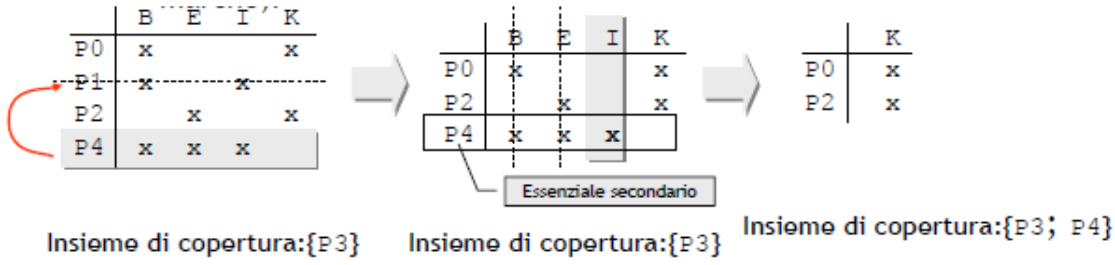
Insieme di copertura:{P3}

b) Criterio di dominanza di riga: *un implicante Pi domina un implicante Pj se Pi copre almeno tutti i mintermini di Pj*. Nel qual caso Pj puo' essere eliminato dalla tabella

Nell'Esempio P4 domina P1 e P1 non serve e si puo' eliminare.



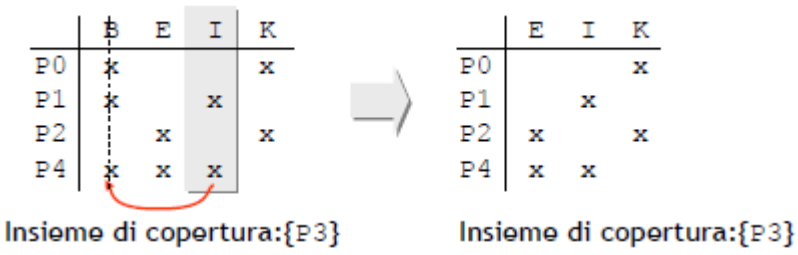
Questa eliminazione crea implicanti pseudoessenziali o essenziali secondari che possono essere considerati per la copertura.



P4 diventa essenziale secondario per I e copre anche B ed E.

c)Criterio di Dominanza tra colonne : un mintermine mi domina un mintermine mj se ogni implicante che copre mj copre anche mi. Per questo si tiene solo mj che assicura tutte le coperture di mi e mi viene eliminato dalla tabella.

Ad es. la copertura del mintermine I induce anche la copertura di B. ossia se si copre I sicuramente anche B sara' coperto e B si puo' eliminare.



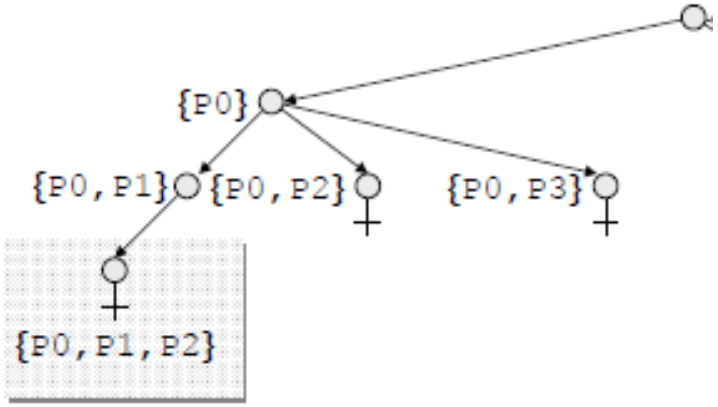
Al termine delle rimozioni delle righe essenziali e delle righe e colonne dominate si ottiene la

Tabella ciclica degli implicanti primi. Se la tabella e' vuota allora l'insieme di copertura e' stato trovato, altrimenti si deve trovare una tra le soluzioni ottime e si usa un algoritmo di ottimizzazione: il piu' noto è l'algoritmo di **branch&bound** o altri algoritmi di ottimizzazione per trovare la miglior copertura: ogni scelta (branch) crea un ramo dell'albero delle scelte; se esso corrisponde all'implicante Po si eliminano le righe e le colonne e si cerca di ridurre la tabella altrimenti si itera fino a mantenere la soluzione di costo minore (bound) .

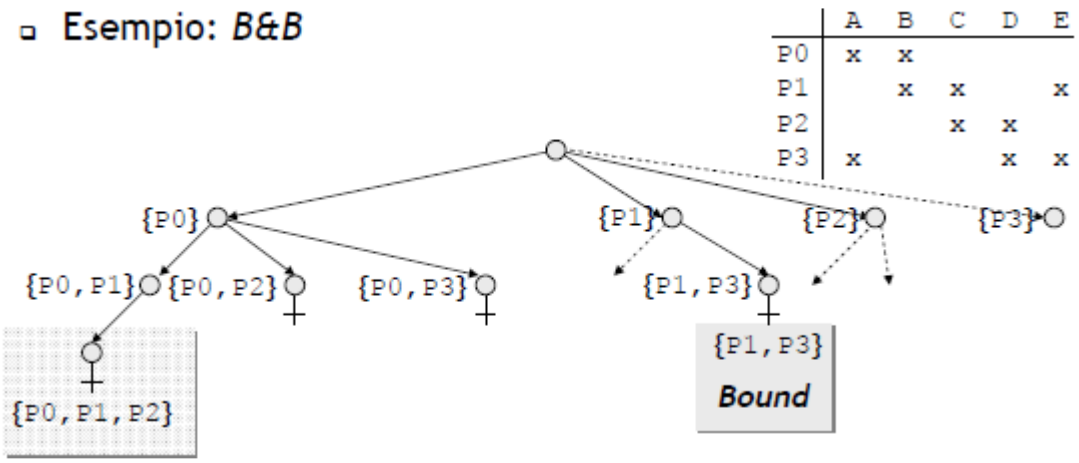
Si consideri al tabella di implicanti primi

	A	B	C	D	E
P0	x	x			
P1		x	x		x
P2			x	x	
P3	x			x	x

L'albero di branch:



Con P0 si coprono A e B; aggiungendo P1 si coprono C ed E; si deve aggiungere anche P2 e si copre al tabella. P0,P1 e P3 ha lo stesso costo della precedente e non viene esaminata; si risale a P0, ma P0, P2 oppure P0 e P3 devono continuare come il precedente e non serve esaminarle: Si risale alla radice e si trova la forma minima che e' P1,P3.



Algoritmo Per passo 2 di copertura:

1 si identificano implicanti primi essenziali primari;
2 si esaminano dominanze di righe e colonne;
3 se esistono implicanti primi essenziali secondari si torna al passo 2
4.si applica il B&B.

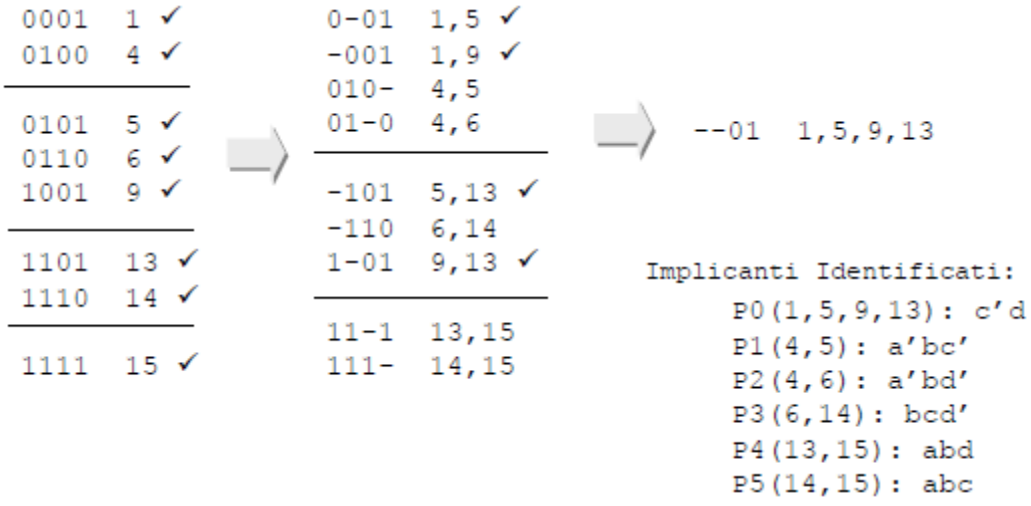
L'algoritmo e' leggermente piu' complesso per tabelle non completamente specificate e per funzioni combinatorie con piu' uscite.

Esercizio 19: sia data una rete combinatoria a 4 variabili la cui funzione booleana sia al seguente: F(a,b,c,d)= a'b'c'd+a'bc'd'+a'bc'd+a'bcd'+ab'c'd+abc'd+abcd'+abcd

trovare la forma minima.

L'ON-set e' dato da f(a,b,c,d,)= ON(1,4,5,6,9,13,14,15)

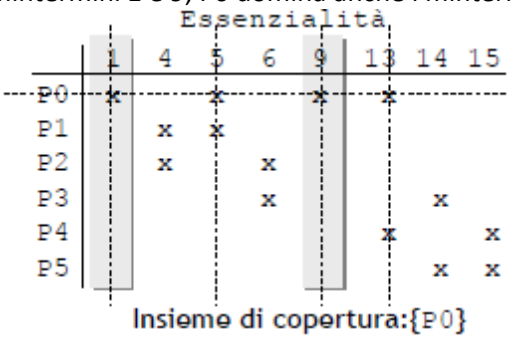
1) ricerca degli implicanti primi; dividendoli in gruppi



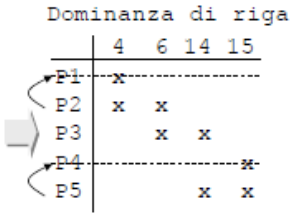
Prima riduzione considerando gli implicanti primi

F(a,b,c,d)= c'd+ a'bc'+a'bd'+bcd'+abd+abc

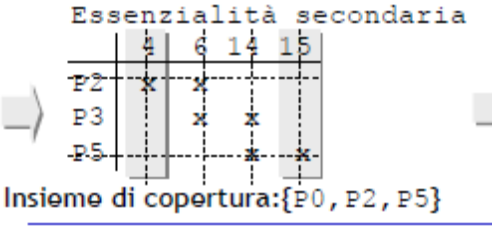
2) tabella delle coperture e ricerca dei primi essenziali:P0 e' primo ed essenziale per i mintermini 1 e 9; P0 domina anche i minterimini 5 e 13.



3) ricerca delle dominanze di righe e colonne



E essenzialità secondarie



Al termine bastano P0, P2 e P5. Perciò F(a,b,c,d,)= c'd+ a'b'c'+abc.

LA stessa soluzione si poteva trovare con al semplificazione delle equazioni booleane o in modo molto semplice con le mappe di Karnaough.

Si puo' sviluppare il codice in un qualsiasi linguaggio e lo si trova in rete.
Questo algoritmo e' NP-hard perchè cresce esponenzialmente con il numero di ingressi . ad es. se n = 32 ci possono essere più di 6.5 * 1015 implicanti primi. Pertanto, le funzioni con un grande numero di variabili booleane devono essere minimizzate con metodi euristici, come ad es. il minimizzatore logico **Espresso (Espresso heuristic logic minimizer**)..

Esercizio 20: Si semplifichi con Quine McCluskey la funzione combinatoria a 4 ingressi il cui onset e' ONset=(0,1,4,5,13,14)

Esercizio 21: Si semplifichi con Quine McCluskey la funzione combinatoria a 4 ingressi il cui onset e' ONset=(0,2,4,5,6,7,8,9,13,15)

Esercizio 22: si vuole costruire una rete logica che ha due ingressi X ed Y a 2 bit. L'uscita vale 1 se si verifica almeno una delle seguenti condizioni (x=3)&(y=0) oppure (y=1)&(x≠1) oppure(x≥2)&(y=3) oppure (y≥2)&(x=3). Calcolare la sintesi canonica SP e la sintesi minima con l'algoritmo di Quine McCluskey.

Attenzione se la rete ha indifferenze(don't care): Le indifferenze devono essere usate nella ricerca degli implicanti primi (passo 1) ma non nelle tabelle delle coperture (passo 2). Nel primo passo devono essere incluse per utilizzare al meglio le semplificazioni di Boole. Nel secondo passo non servono perche' non devono essere coperte

Esercizio 23: Si semplifichi con Quine McCluskey la funzione combinatoria a 4 ingressi il cui onset e' ONset=(0,2,4,5,6,7,13,15) e DON'TCARESet=(11,12). Indicarne anche la mappa e la tabella e le forme canoniche e vedere come si riduce la complessità con la sintesi di QmC.

3.11 RETI LOGICHE COMBINATORIE SEMPLICI

Sommatore (half adder)

Un Esempio di rete logica elementare è **il sommatore (half adder):** Per eseguire somme in una qualsiasi base è necessario saper ottenere il risultato della somma tra due cifre. Lo stesso vale per il calcolo binario ed in particolare la somma di due bit viene eseguita da una rete logica combinatoria detta **semi sommatore o half adder**. La somma binaria è molto semplice, in quanto ogni cifra può assumere solo i valori 0 e 1, quindi è facile elencare le possibilità:

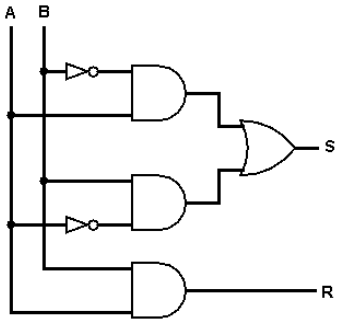
0+0 = 0 (nessun riporto)
0+1 = 1 (nessun riporto)
1+0 = 1 (nessun riporto)
1+1 = 0 (riporto di 1)

Rappresentando con una tabella di verità questo comportamento è il seguente, chiamando a e b i due bit in ingresso, r il riporto e s la somma:

a	b	R	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

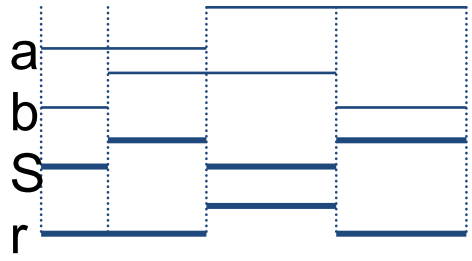
r ed s sono funzioni di due variabili indipendenti come dalle funzioni precedenti. È possibile fare la sintesi canonica utilizzando solo gate AND, OR e NOT. Secondo l'alg

oritmo precedente. **La sintesi canonica SP che ne deriva è**



$S = \bar{a}b + a\,\bar{b}$ ed $r = ab$

Si può ottenere anche la sintesi PS. Impiegando al logica però è possibile notare che il riporto è l'AND di a e b, mentre la somma è data dall'EXOR dei due. Lo schema logico risultante è quindi anche rappresentabile come:



La forma sopra indicata a è quella più nota per l'half adder.

Tutto il calcolatore sia all'interno della CPU, delle memorie e dell'i/o sia all'interno del bus controller è composto da reti logiche di cui alcune combinatorie e altre sequenziali.

Nei calcolatori si usa il simbolo # per indicare che il segnale ha valore logico invertito e ha un significato logico vero quanto vale 0 e falso quando vale 1 (logiga negativa). Ad es. RD# significa che esegue la lettura Read se RD# e' attivo ossia se RD#=0.

***Esercizio 24**: progettare una rete logica che gestisca l'accesso ad un banco di memoria tramite i segnali M/IO# (quando il segnale ha valore logico alto indica accesso alla memoria, quando basso indica accesso ai dispositivi di I/O) RD#, WR# (attivi bassi); l'accesso alle memorie è possibile se M/IO# è attivo e se sono attivi o il segnale di lettura o quello di scrittura (mai contemporaneamente). L'uscita della rete è il segnale di abilitatazione del dispositivo (chiamato normalmente chip select CS_M) spesso disponibile come input delle memorie solo in forma attiva basa CS_M# .*

La tabella della verità è data da:

M/IO#	*RD#*	*WR#*	*CS_M*	*CS_M#*
0	*0*	*0*	-	-
0	*0*	*1*	*0*	*1*
0	*1*	*0*	*0*	*1*
0	*1*	*1*	*0*	*1*
1	*0*	*0*	-	-
1	*0*	*1*	*1*	*0*
1	*1*	*0*	*1*	*0*
1	*1*	*1*	*0*	*1*

Sintesi canonica SP, considerando entrambe le indifferenze come 0:

$$CS\_M = (M/IO\# \cdot \overline{RD\#} \cdot WR\#) + (M/IO\# \cdot RD\# \cdot \overline{WR\#})$$

Determinazione della sintesi minima SP

Si considera ad 1 l'uscita corrispondente alla combinazione degli ingressi (1,0,0).

$$CS\_M = (M/IO\# \cdot\ RD\# \cdot WR\#) + (M/IO\# \cdot\ \overline{RD\#} \cdot WR\#) + (M/IO\# \cdot RD\# \cdot \overline{WR\#}\ )$$

Per il teorema di idempotenza si può aggiungere una "copia" del primo mintermine.

$$CS\_M = (M/IO\# \cdot\ RD\# \cdot WR\#) + (M/IO\# \cdot\ \overline{RD\#} \cdot WR\#) + (M/IO\# \cdot\ RD\# \cdot WR\#) + ((M/IO\# \cdot RD\# \cdot \overline{WR\#}\ )$$

Per il teorema della combinazione applicato tra il primo e il secondo, e tra il terzo e il quarto mintermine si ottiene la sintesi minima SP:

$$CS\_M = \left(M/IO\# \cdot\ \overline{WR\#}\right) + (M/IO\# \cdot \overline{RD\#})$$

Per ricavare l'espressione dell'uscita attiva bassa $CS\_M\#$, si nega l'espressione appena ricavata e si applica il teorema di de Morgan.

$$\begin{aligned} CS\_M\# &= \overline{\left(M/IO\# \cdot\ \overline{WR\#}\right) + \left(M/IO\# \cdot \overline{RD\#}\right)} \\ &= \overline{\left(M/IO\# \cdot\ \overline{WR\#}\right)} \cdot \overline{\left(M/IO\# \cdot \overline{RD\#}\right)} \\ &= \left(\overline{M/IO\#} + WR\#\right) \cdot (\overline{M/IO\#} + RD\#) \end{aligned}$$

La rete sopraindicata e' una delle tante reti combinatorie che si trovano nei calcolatori all'interno del bus controller, per gestire i segnali di controllo che dalle CPU vanno verso le memorie.

3.12 RETI LOGICHE A LIVELLO RTL

La sintesi di reti logiche a medie dimensioni che si ottiene di solito attraverso librerie hardware di dispositivi fisici (o di routine software di VHDL o SystemC) permette la realizzazione di reti a livello di parola e non di singolo bit. Si parla quindi di Sintesi a livello di registro o RTL (**Register Transfer Level**). È la sintesi di blocchi logici elementari (ma più complessi dei singoli gate) con cui realizzare per composizione ed interconnessione reti logiche e sistemi complessi.

I blocchi più comuni che si trovano nelle librerie e che si impiegano nei calcolatori spesso con numero di input e di output parametrici sono molti, come ad es.:

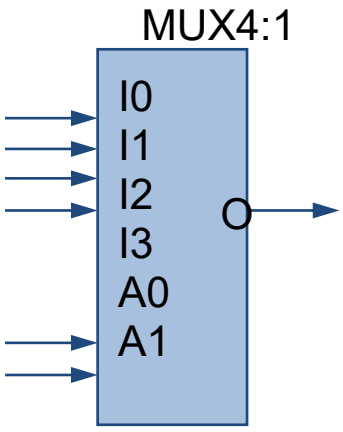
- Multiplexer
- Decoder
- Adder, ALU
- …

MULTIPLEXER

Il **Multiplexer** è quel blocco logico che permette di derivare verso una unica uscita un segnale proveniente da uno tra n possibili ingressi.

Formalmente: *il MULTIPLEXER è una rete logica avente 2^n ingressi di tipo dati e n ingressi di tipo segnali di controllo (o indirizzo) ed 1 uscita: in ogni istante il dato di input corrispondente alla configurazione dei segnali di controllo viene posto in uscita.*

I muliplexer o selettori permettono di selezionare gli ingressi tra più possibili sorgenti.



A_1	A_0	I_3	I_2	I_1	I_0	O
0	0	×	×	×	0	0
0	0	×	×	×	1	1
0	1	×	×	0	×	0
0	1	×	×	1	×	1
1	0	×	0	×	×	0
1	0	×	1	×	×	1
1	1	0	×	×	×	0
1	1	1	×	×	×	1

Fig. 17: Multiplexer (attenzione non e' una tabella completamente specificata ma solo un modo compatto per rappresentare una tabella che dovrebbe avere 64 configurazioni di ingresso,e che è comunque completamente specificata dato che non ci sono indifferenze sulle uscite)

Si può creare la sintesi attraverso la tabella della verità o direttamente dalle eq. Booleane.

Ad es.: con un MUX 4:1 che ha 4 ingressi da selezionare, due di selezione ed 1 uscita, la funzione di uscita vale 1 quando la selezione vale 00 e l'ingresso 0 vale 1 o quando la selezione vale 01 e l'ingresso 1 vale 1 o quando la selezione vale 10 e l'ingresso 2 vale 1° quando la selezione vale 11 e l'ingresso 3 vale 1.

O=I3A1A0 +I2A1A0' +I1A1'A0+I0A1'A0'

Esegue la somma di tanti prodotti quanti gli ingressi (di dato I0..In) , in cui ogni prodotto è un implicante che contiene la configurazione delle selezioni e l'ingresso corrispondente.

La realizzazione logica che deriva e' immediata. Ad es per un Mux 8:1

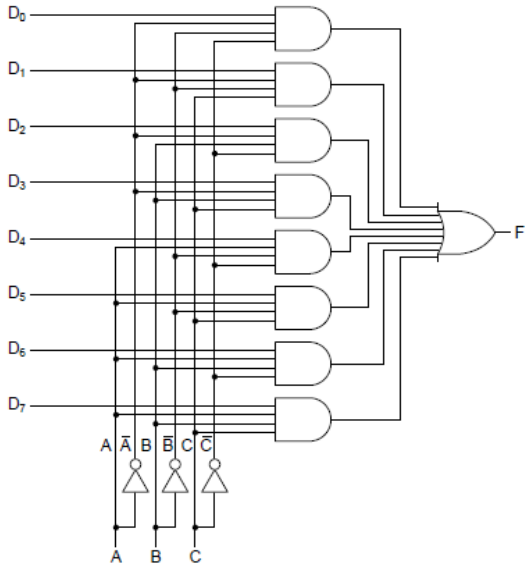


Fig. 18: Rete logica canonica per un multiplexer 8:1.

Nei multiplexer la forma pseudo-canonica (relativamente ai soli segnali di controllo) è anche la forma minima (non è semplificabile).

I multiplexer sono impiegati nei calcolatori sia per instradare i segnali utili a seconda di un selettore, sia per realizzare funzioni combinatorie.

Per la prima modalità si pensi ad es. l'ingresso di un bus che può provenire dalla sorgente A o B a seconda del segnale di selezione. Impiegando un Mux si evita che entrambi i segnali confluiscano sul bus e si possa verificare un conflitto se un segnale elettrico è posto a zero (collegato a massa) ed un segnale e posto a 1 (collegato alla alimentazione).

I multiplexer sono usatissimi all'interno dei calcolatori per gestire i bus e all'esterno nei canali di comunicazioni per portare su una sola linea nel tempo, segnali provenienti da piu' sorgenti, mandando in sequenza i segnali di controllo. Un Esempio e' nello schema che segue

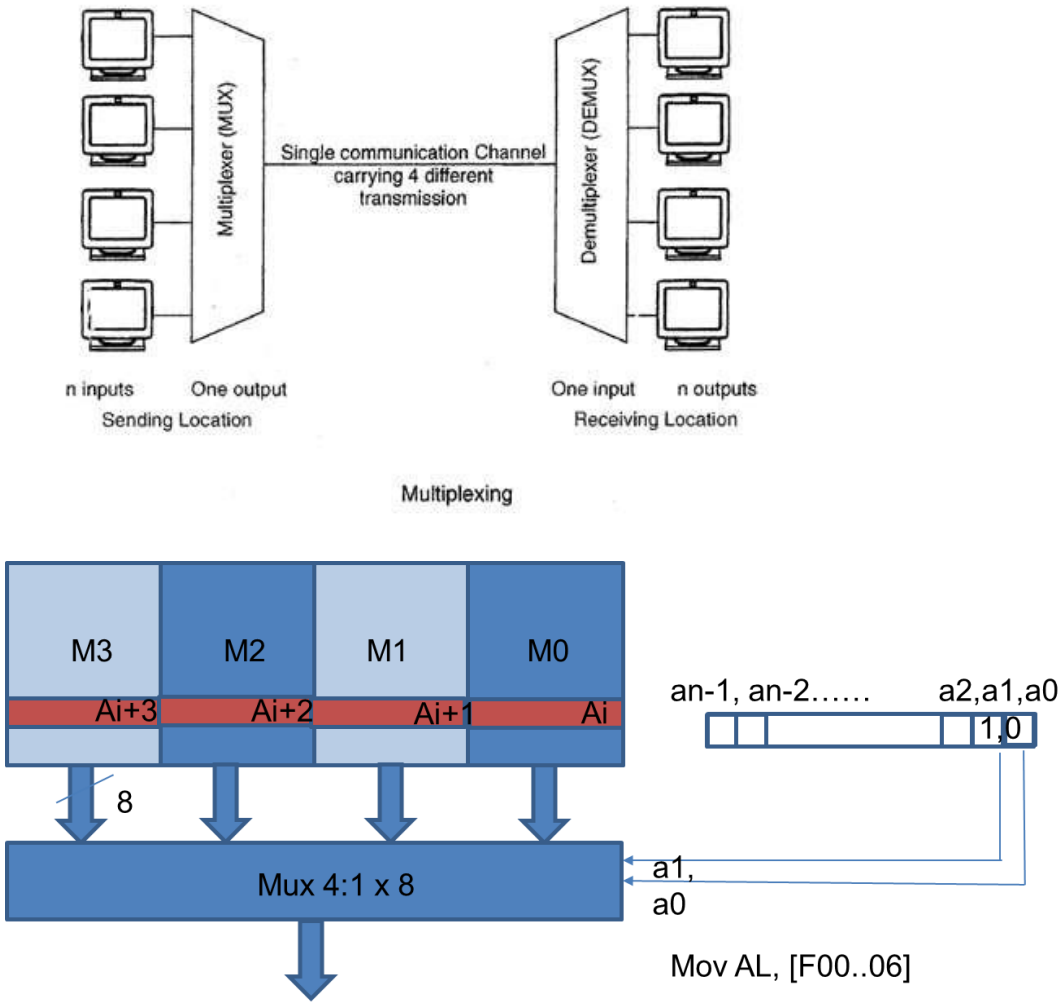


Fig. 19: interfaccia con la memoria

Si pensi ad es. ad una memoria con 4 banchi ”in parallelo” ognuno da 1 byte per iottenere una parola da 32 bit. Supponiamo che la CPU abbia un modello di indirizzamento non allineato e voglia leggere solo 1 byte a partire da un indirizzo che termina con 06h (o con 02 o con A o con E). In questi 4 casi il byte si trova nel terzo banco M2 (cosi’ come nel banco M0 troviamo gli indirizzi che terminano per 0 per 4 per 8 o per C etc..).

Cosi’ il bus degli indirizzi viene diviso in due parti, la parte alta o piu’ significativa per selezionare l’indirizzo che potenziamente va a tutti i banchi e la parte bassa per selezionare quale banco ad un mux. Verrà impiegato nella parte alta [an-1…a2] per selezionare la memoria mentre la parte bassa in questo caso a1 ed a0 per selezionare quale banco della memoria leggere.

Il mux disegnato e’ in realta’ composto da 8 mux 4:1, uni per ognuno dei bit del byte della memoria, che hanno tutti gli stessi bit di selezione

Il mux che si vede una volta si trovava nel bus controller quando la CPU aveva un bus dei dati "piccolo". Ora invece il mux e' integrato nel MDR (MMU) interno alla CPU per selezionare se si vogliono leggere solo parti delle parole. Il vero mux integrato nel MDR ora e' un po' piu' complesso perche' permette di selezionare anche piu' byte, ma usa lo stesso concetto.

Il mux puo' essere usato come dispositivo per realizzare una qualsiasi funzione canonica di tipo SP; nella figura che segue si vede un mux usato per realizzare la funzione combinatoria di maggioranza tra i due ingressi di selezione. Si può ossia **impiegare per rappresentare direttamente la tabella della verità basta collegare a '1' i mintermini della funzione canonica SP ed a '0' i prodotti di tutti i letterali che danno uscita 0.**

Naturalmente e' una sintesi semplice in termine di velocità ma non minima perche' ha alcuni AND che non servono dato che hanno un ingresso fissato a '0'.

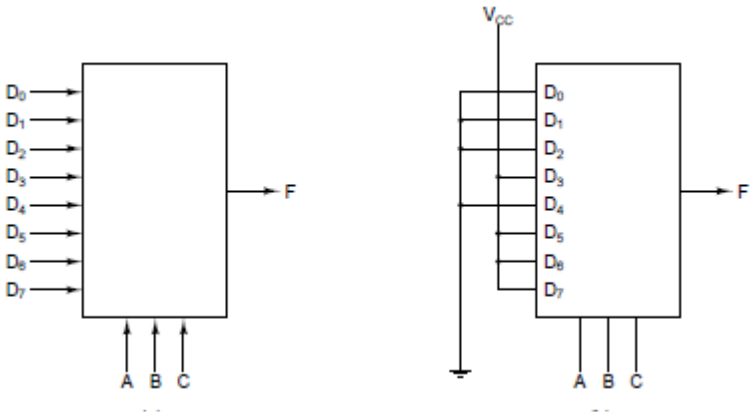


Fig. 20: Reti logiche con multiplexer.

Come vale l'uscita F della Figura? Considerando Vcc, la alimentazione ossia sorgente del valore logico 1 e la massa come sorgente del valore logico 0 e considerando ABC i segnali di controllo in sequenza (con A piu' significativo)

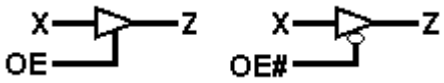
F= A'BC+AB'C+ABC'+ABC

AMPLIFICATORE TRI-STATE

L'amplificatore tri-state è una rete logica speciale che va al di là dell'algebra di Boole. È un generatore di segnale in terzo stato (Z = alta impedenza); se non e' abilitato il segnale di controllo OE output Enable emula la presenza di un circuito aperto o, se OE è abilitato, lascia passare il segnale di ingresso.

In	OE	Out

-	0	Z
1	1	1
0	1	0



Spesso con OE# attivo basso: l'uscita è uguale all'ingresso quando l'output enable è asserito (basso se OE#). Si usa molto spesso per realizzare multiplexer distribuiti nei bus; per evitare cortocircuiti bisogna che in ogni istante solo un tri-state sia abilitato.

I registri all'interno del calcolatore sono delle reti logiche sequenziali composte da tanti elementi di memoria (che vedremo essere i D-Latch) quanti sono i bit da memorizzare con un segnale di scrittura che corrisponde all'enable del D-Latch ed un segnale di lettura che corrisponde all'OE#

Il multiplexer si può usare anche per molti altri scopi, ad es. per eseguire un **convertitore parallelo-serie**. Il dato viene posto nell'ingresso ad otto bit e ai selettori si collega un contatore che fa presentare nel tempo i valori da 000 a 111 e i dati escono in serie. Il contrario del multiplexer è il **demultiplexer** che accetta un segnale in serie e lo trasferisce in una delle possibili n uscite a seconda dei $\log_2 n$ ingressi di selezione.

DECODER

Il decoder è la rete logica che asserisce un valore 1 a una sola delle possibili 2^n uscite in base agli n ingressi. È come il de-multiplexer con ingresso fisso a 1.

Viene usato quando un sistema digitale realizza in parallelo un certo numero di funzioni ma il sistema si comporta in base ad una sola di esse alla volta selezionata da una opportuna configurazione dei segnali di controllo

*Formalmente il **decoder/de-multiplexer** è una rete logica con 1 ingresso di dato (fissato ad 1 nel decoder) , n segnali di controllo e 2^n uscite: l'uscita contrassegnata dall'indice pari alla configurazione dei segnali di controllo riceve l'ingresso mentre le altre non sono abilitate e forniscono il valore 0.*

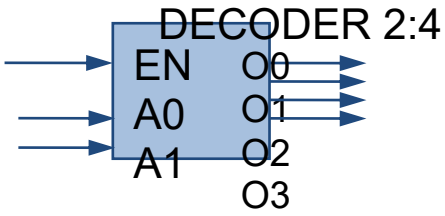


Fig. 21: EN è il segnale di abilitazione (posto a 1).

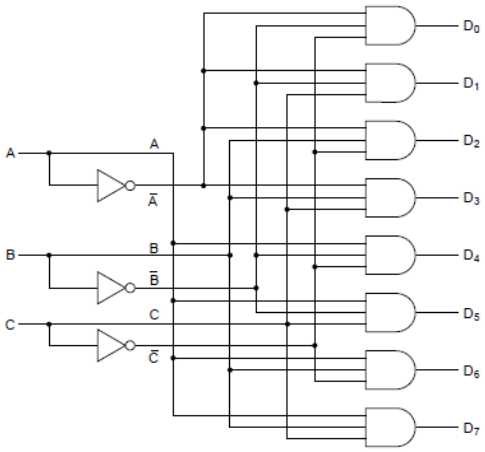


Fig. 22: Decoder 3:8, senza segnale di abilitazione.

Si dice decoder in quanto viene usato per decodificare un segnale binario in codice 1/n. Il decoder è molto usato nei bus controller del calcolatore per decodificare i bit di indirizzo verso i segnali di selezione (chiamati chip select) delle memorie e dei dispositivi di input/output. Se ci sono n dispositivi di memorie sono necessari n segnali di selezione e si controllano con log2(n) segnali di indirizzo. Ad es. per 8 dispositivi servono 3 segnali di indirizzo.

I componenti RTL e quelli ad integrazione più alta, così come i componenti delle motherboard dei computer sono descritti tramite schemi logici e funzioni VHDL nelle librerie di sintesi logica. Se sono disponibili separatamente sono descritti nei **data sheet**. Il data sheet sono i manuali con la documentazione degli IC (integrated circuit) che si trovano in commercio con cui realizzare reti logiche. Contengono :

- *descrizione a parole (*es: il comparatore contiene 4 NOR)
- *tabella della verità*
- *schema logico del gate e espressione booleana*
- *packaging Pin out (lista dei segnali)*
- *absolute maximum ratings* (temperatura, max supply voltage, max input voltage 5.5V)
- *raccomended operating conditions* (es: 74LS: supply voltage 4.75,5, 5.25 V input voltage Vih >2V, Vil <0.8, output currents ioh, iol : -0.4, 8 mA)

– *electrical characterics* (voltage , current, ...)

Negli schemi sono indicati i blocchi interni a diversi livelli di astrazione in logica positiva e logica negativa: logica negativa è indicata dalla presenza di una *bubble* nel simbolo (e non indica la necessità di un NOT esterno). Ad es. il decoder che ha codice 47 (o 7447per applicazioni non militari) usato per convertire una cifra BCD in codice a sette segmenti, ha il data sheet che si può trovare qui: http://focus.ti.com/lit/ds/sdls111/sdls111.pdf.

COMPARATORE

Il comparatore è il blocco logico che ha uscita uguale ad uno se le due parole di ingresso sono uguali.

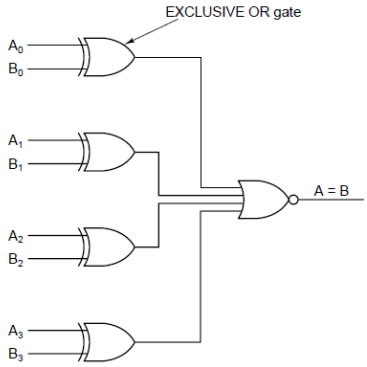


Fig. 23: Comparatore a 4 bit.

3.13 PROGETTO DI UNA ALU

Una delle reti logiche combinatorie più importati nelle CPU è la ALU **arithmetic logic unit,** ossia l'unità che in base a segnali di controllo dell'istruzione esegue un'operazione aritmetia o logica tra gli operandi.

Usando una sintassi assembly a due ingressi ed una uscita con sintassi: <op, sorg1, sorg2, dest>, le tipiche operazioni di ALU sono:

- somma add r1,r2,r3
- sottrazione sub r1,r2,r3
- prodotto logico and r1,r2,r3
- somma logica or r1,r2,r3
- negazione not r1,r2

- shift shr/shl r1,r2,r3
- confronto

Il tipico schema della ALU è il seguente:

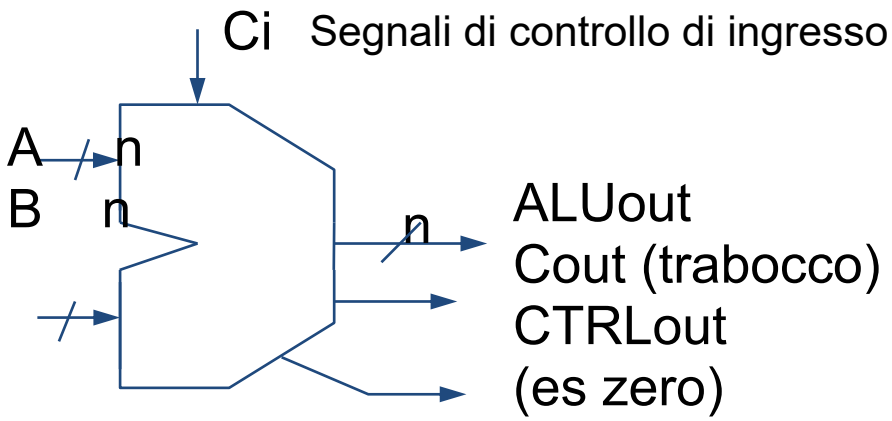
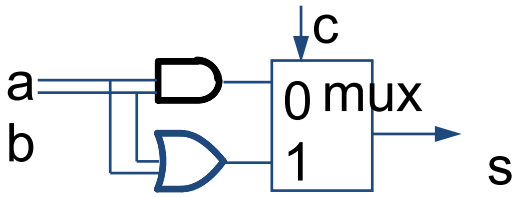


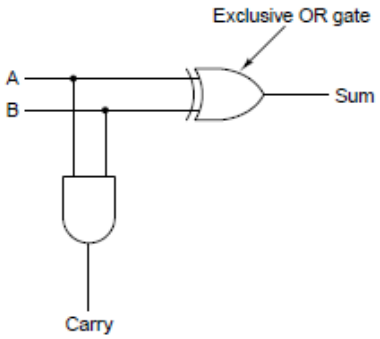
Fig. 24: ALU.

All'interno la rete logica è decomposta in reti logiche semplici per ogni funzione (poi spesso tutta l'implementazione è ottimizzata per avere il minor numero di gate in cascata). Ad es. per ottenere le operazioni di AND e OR logico per ogni bit della parola si ha uno schema come segue:



Per la somma la rete logica si compone del sommatore (half adder) che crea la somma e il riporto:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



L'half adder serve per sommare due bit, i meno significativi di una parola, ma non gli altri perché deve gestire anche il riporto di ingresso nello schema chiamato **full adder:**

A	B	Carry In	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

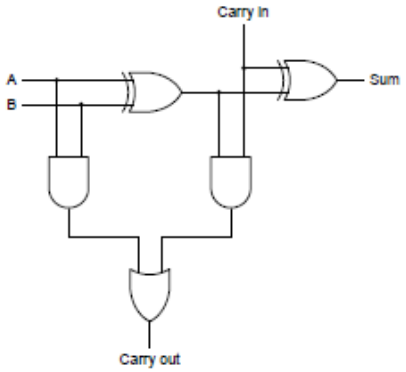


Fig. 25: Full adder.

Il full adder può essere messo in cascata con altri nel sommatore a propagazione di riporto. Esso è il più semplice (ma inefficiente) sommatore, dove il tempo di attesa è proporzionale al tempo di propagazione del riporto lungo tutta la rete.

Inputs			Outputs		
a	b	CarryIn	CarryOut	Sum	Comments
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

(da Patterson Hennesey)

Naturalmente puo' essere fatta la sintesi canonica. Esistono molte altre sintesi di sommatori più efficienti e veloci anche se spesso richiedono più gate elementari. Questi di solito sono realizzate nelle nuove CPU dove il numero di gate elementari è elevato. Le diverse funzioni della ALU possono essere create o da reti logiche separate o da uniche reti logiche integrate.

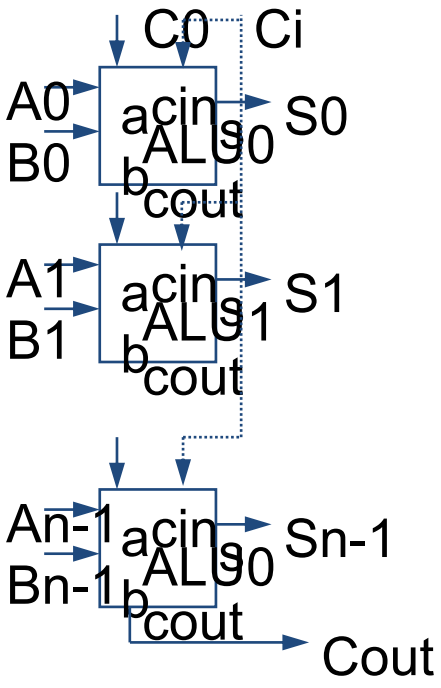
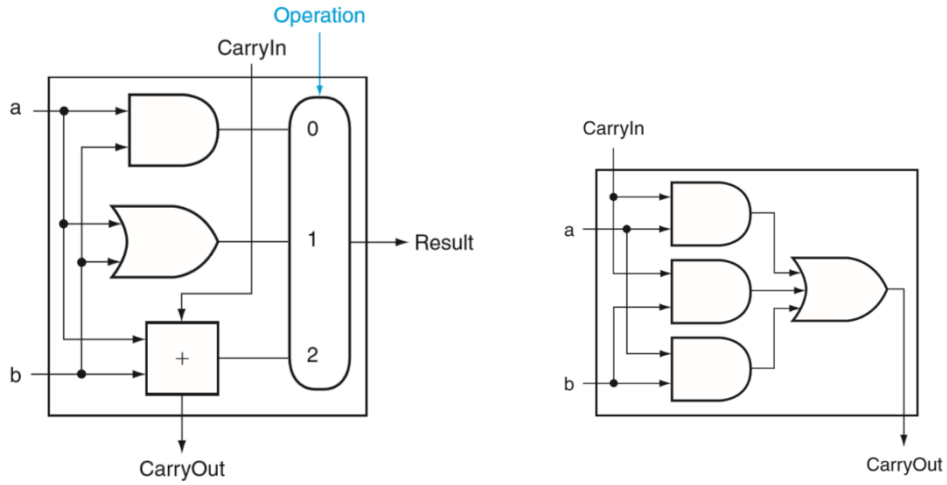


Fig. 26 Sommatore a propagazione di riporto.

Nella figura che segue si vede una ALU ad 1 bit dove può essere realizzato un insieme di funzioni pari a:

- AND A,B
- OR A, B
- NOT B
- ADD A,B

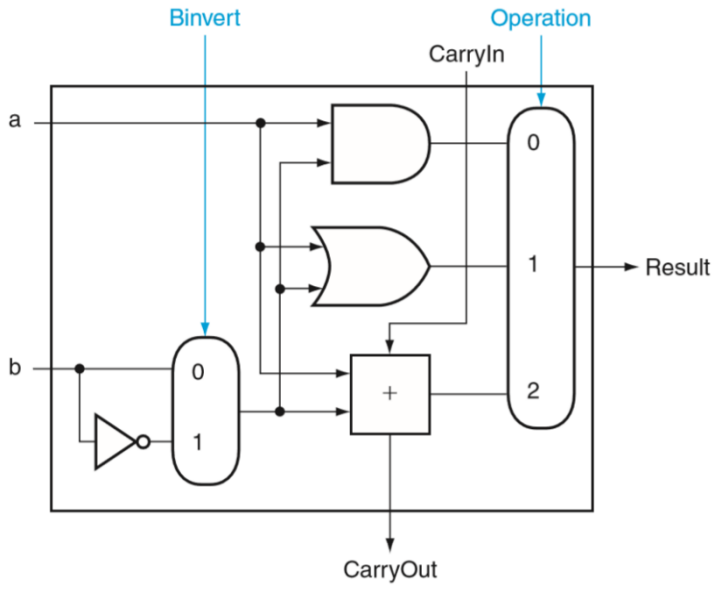
Mettendo assieme la parte logica ed aritmetica puo' quindi essere snitetizzata una prima ALU AD 1 bit che compie due operazioni LOGICHE e 1 ARTIMETICA , la somma



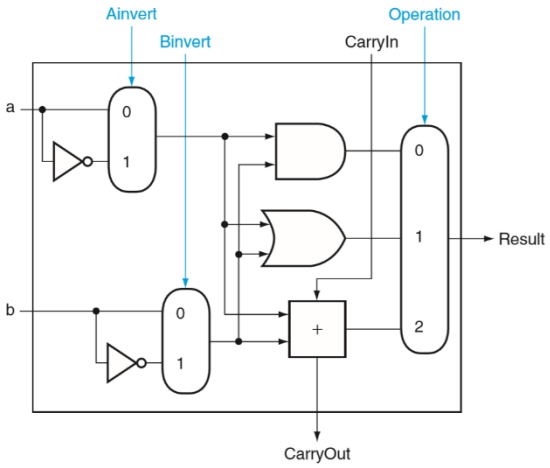
Gli ingressi a e b vanno a tre blocchi, ad un OR, ad un AND o a un FA che crea la somma ed il riporto. Le tre uscite and, or e somma vanno ad 1 multiplexer che ha 4 ingressi (di cui 1 non usato) ed 1 uscita.

Se si volesse realizzare anche la sottrazione bisognerebbe invertire il secondo operando e sommare 1 (complemento a 2)

Come fare ? basta avere =1 e usare un secondo multiplexer di controllo Binvert



Inoltre se I ha la possibilità di invertire anche A si ottengono anche altre funzioni logiche come il NAND e il NOR (si veda appendice Patterson Hennessey)



Infine considerando che esiste un quarto ingresso non usato del mux finale si puo' pensare di mettere un quarto ingresso Less. Questo puo' servire per una istruzione "slt" (set on less) che vale 1 se a<b (istruzione del processore RISC V).

Attenzione che per ottenere questo risultato in una catena ad esempio di 32 bit bisogna vedere la differenza: se questa e' negativa allora a<b. Ossia bisogna vedere il bit piu' significativo della somma (con a+ (-b)) come ingresso di Less che se 1 significa che (a-b) negativo ossia a minore di b , e 0 significa positivo ossia il contrario. Ad esempio a 4 bit se a vale 1010 e b vale 1011 (a is less) la sottrazione a-b vale in complemento a 2: (1010) + (0100)+1= 1111 con carry 0

La rete finale (courtesy Patterson Hennessey)

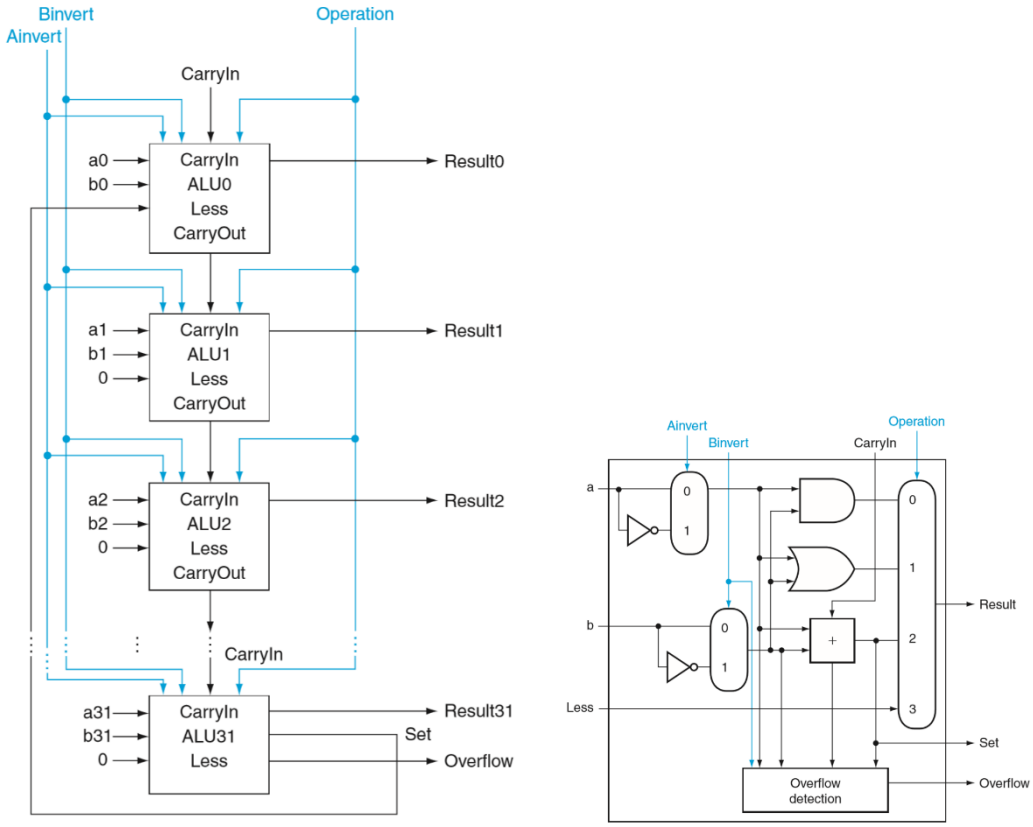


Fig. 27: ALU a 32 bit del RISC V.

Questo schema mostra che l'ingresso Less vale sempre 0 tranne il bit meno significativo che ha come ingresso il valore Set che vine dal bit piu' significativo. In questo modo il risultato vale 000000…01 se a e' minore di b o 0000000..00 altrimenti.

Si puo' anche aggiungere il Test Zero (importante per eseguire la istruzione di btanch ife qual) ; lo zero non e' altro che ' l'OR negato di tutti i valori del risultato.

Di fatto questa ALU e' controllata da 4 segnali di controllo che nella tabella sono nell'ordine Anegate, Bnegate, e i 2 bit di Operation del Mux finale.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR