

Circuiti combinatori fondamentali

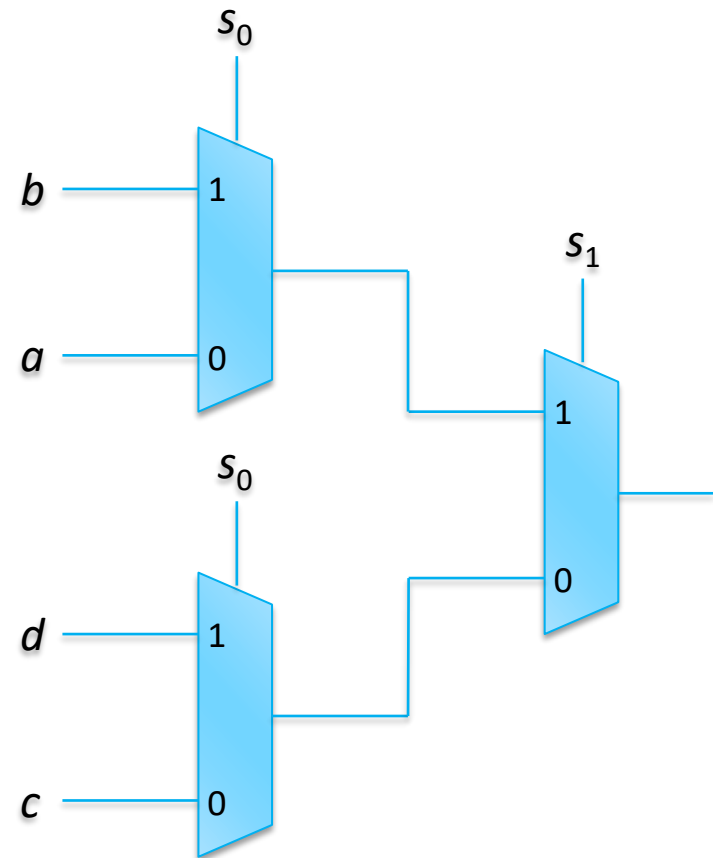
Costruiamoci molti blocchetti base utili

Metodo gerarchico

- ▶ **Il metodo visto finora è generico**
 - ▶ Basta scrivere la mappa e semplificarla
 - ▶ Gestibile fino a un numero di variabili limitato
- ▶ **Con tante variabili la semplificazione diventa complessa**
 - ▶ Sia perché ci sono troppe caselle
 - ▶ Ma soprattutto perché è difficile perfino immaginare il valore che deve avere la funzione
 - ▶ E' quindi complesso anche solo scrivere le mappe (e non solo perché sono di grandi dimensioni)
- ▶ **Conviene costruire il circuito a partire da blocchi base**
 - ▶ I blocchi li possiamo progettare con cura
 - ▶ Ne conosciamo bene il funzionamento
 - ▶ Sappiamo già che funzionano
 - ▶ In pratica aumentiamo il livello di astrazione
 - ▶ Gerarchico perché una volta fatto non lo si apre più

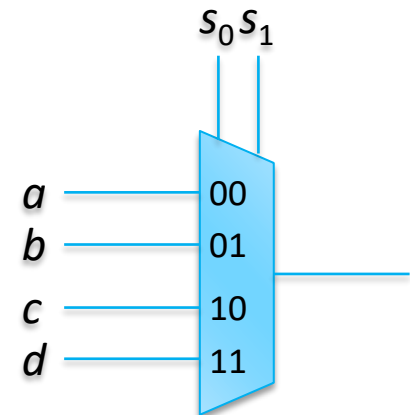
Multiplexer 4 a 1

- ▶ **Funzione a 4 ingressi di dato + 2 ingressi di controllo**
 - ▶ Ingressi a , b , c e d , più s_0 ed s_1 per la scelta
 - ▶ Totale 6 ingressi
 - ▶ Difficile da gestire come tabella o come mappa
- ▶ **Si può fare un circuito gerarchico**
 - ▶ Prima si sceglie tra a e b , e tra c e d usando s_0
 - ▶ Poi si sceglie tra i due che sono rimasti con s_1
 - ▶ Il circuito che si ottiene non è più a 2 livelli
 - ▶ Si potrebbe fare una soluzione a due livelli, forse più veloce (provare a farla a casa!)



Simbolo grafico del multiplexer

- ▶ **Si aggiungono gli ingressi di dato e quelli di selezione**
 - ▶ Si aggiornano i numerini che identificano l'ingresso attivo in corrispondenza di una combinazione degli ingressi di selezione
- ▶ **I multiplexer sono utili in molteplici applicazioni**
 - ▶ Per condividere risorse
 - ▶ Per fare delle scelte
 - ▶ Per realizzare dispositivi programmabili
 - ▶ Per selezionare celle di memoria



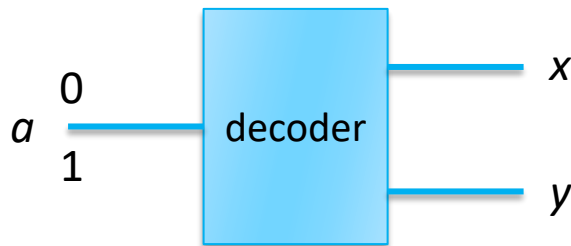
Decodifiche o decoder

- ▶ **E' utile poter passare da una codifica ad un'altra**
 - ▶ Per esempio si può passare dal codice binario al codice Gray
 - ▶ Oppure passare da fixed point a floating point
- ▶ **Decodificare significa passare da n a m cifre**
 - ▶ Con $n \leq m \leq 2^n$
 - ▶ In modo che ad ogni codice di ingresso corrisponda una uscita unica
 - ▶ In pratica si scompatta la rappresentazione
 - ▶ Con m cifre potrei rappresentare 2^m codici, ma ne uso solo 2^n

Decodifica 1 a 2

► Un ingresso e due uscite

- La prima uscita vale 1 se l'ingresso vale 0
- La seconda uscita vale 1 se l'ingresso vale 1



► Notate che le uscite non sono mai a 1 o a 0 contemporaneamente

- In teoria 2 uscite potrebbero codificare 4 diversi elementi
- Alcune combinazioni però non vengono utilizzate

Realizzazione

► Tabella della verità

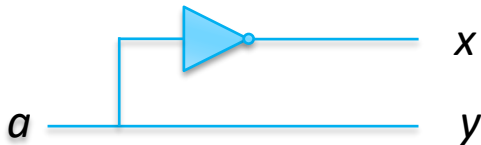
- Solo due righe e due uscite
- Non è una mappa di Karnaugh!!!

a	x	y
0	1	0
1	0	1

► Per ispezione si vede che

- L'uscita y è uguale all'ingresso a
- L'uscita x è uguale al negato dell'ingresso a

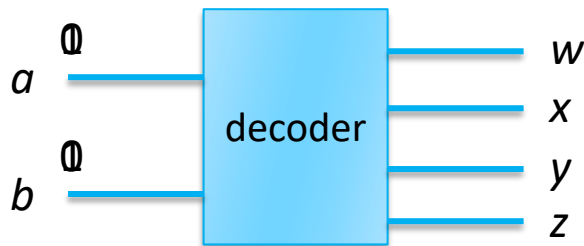
► Circuito



Decodifica 2 a 4

► Due ingressi e quattro uscite

- La prima uscita vale 1 se l'ingresso vale 0
- La seconda uscita vale 1 se l'ingresso vale 1
- La terza uscita vale 1 se l'ingresso vale 2
- La quarta uscita vale 1 se l'ingresso vale 3



Realizzazione

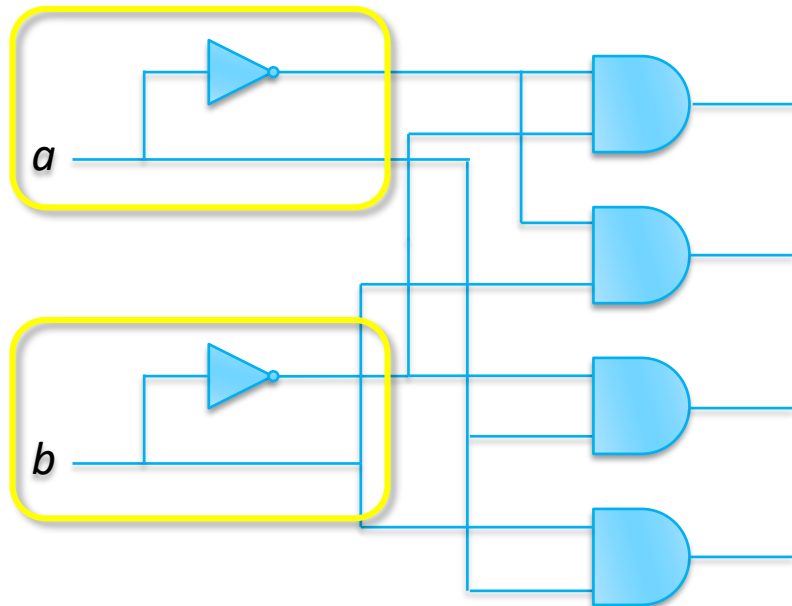
► Tabella della verità

- Ogni uscita ha un solo minterm
- Inutile farsi le mappe di Karnaugh
- Non c'è nulla da semplificare

► Circuito

- Costruito con due decodifiche 1 a 2

ab	w	x	y	z
00	1	0	0	0
01	0	1	0	0
10	0	0	1	0
11	0	0	0	1

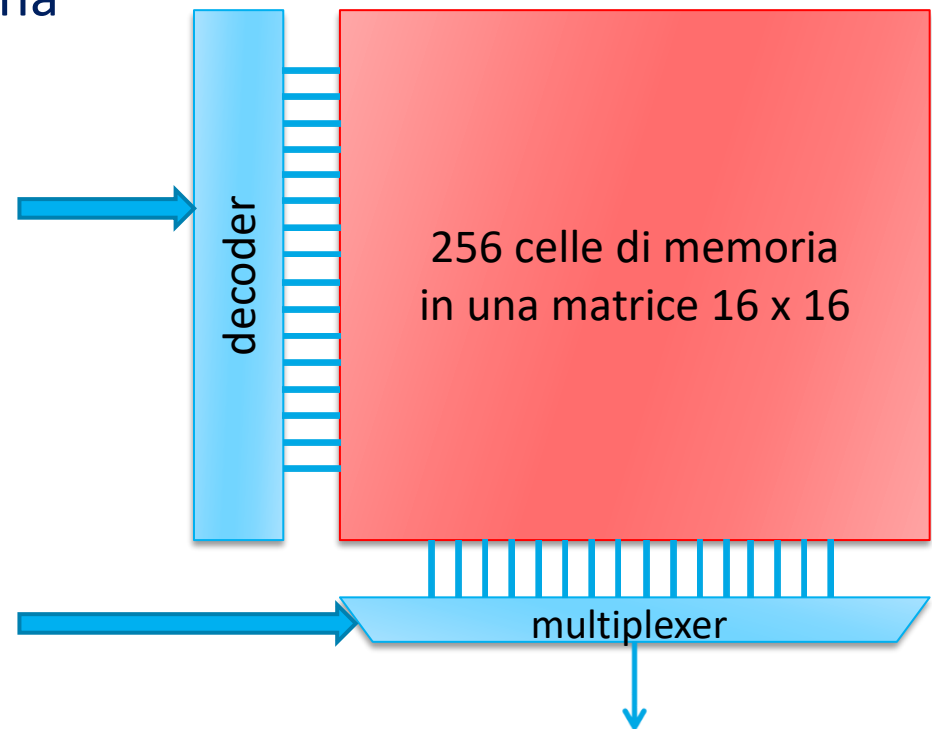


Decodifiche

► A cosa servono le decodifiche?

- Molto utili per indirizzare le memorie
- Il processore mette sul bus un indirizzo a n bit
- Metà indirizza le righe con una decodifica
- L'altra metà sceglie la colonna con un multiplexer

► Multi altri usi



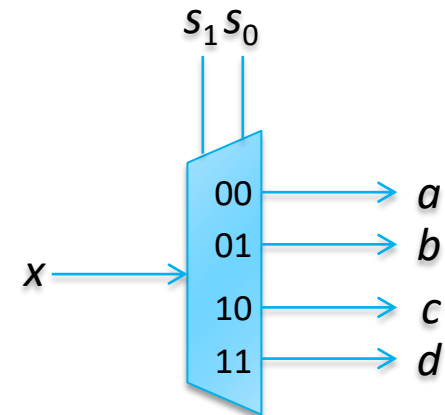
Demultiplexer

- ▶ **Fa il lavoro contrario del multiplexer**

- ▶ Prende in ingresso una variabile binaria e la presenta su una uscita a scelta tra 2^n possibili
- ▶ Le uscite non selezionate vengono tenute a valore 0
- ▶ Il demultiplexer ha
 - ▶ $n + 1$ ingressi (n per la selezione, più un ingresso di dato)
 - ▶ 2^n uscite

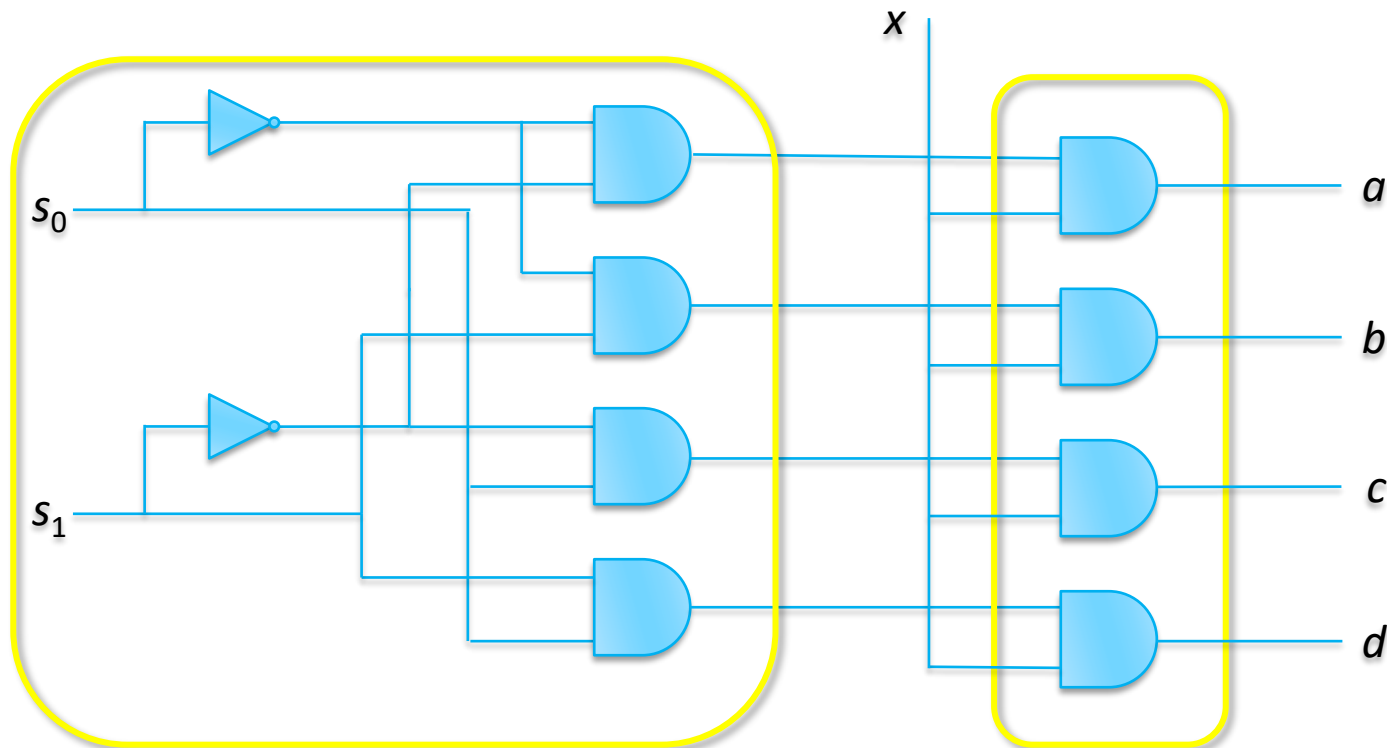
- ▶ **Esempio: demux 1 a 4**

- ▶ 2 ingressi di selezione
- ▶ 4 uscite
- ▶ Facilmente realizzabile a due livelli con quattro mappe



Realizzazione con decodifica

- ▶ Il demultiplexer si può realizzare utilizzando una decodifica
 - ▶ Delle porte AND abilitano l'ingresso a passare ad ognuna uscita
 - ▶ La decodifica sceglie quale porta AND abilitare
 - ▶ Semplicissimo fare demux a tante uscite usando decodifiche più grosse



Encoder o codifica

► E' il processo contrario della decodifica

- Per esempio si può passare da 8 fili a 3
- Molto più semplice da realizzare

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Realizzazione

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

► **Ogni uscita vale 1 per una determinata combinazione di ingressi**

► Per esempio A_2 vale 1 per D_4 , D_5 , D_6 e D_7 . Quindi

► $A_2 = D_4 + D_5 + D_6 + D_7$

► $A_1 = D_2 + D_3 + D_6 + D_7$

► $A_0 = D_1 + D_3 + D_5 + D_7$

Priority encoder

▶ Già visto in precedenza

▶ Passati da 3 ingressi a 2 uscite

- ▶ Nessuna, 1, 2, 3 (4 combinazioni)

▶ Con 4 ingressi occorrono 3 uscite

- ▶ Dobbiamo codificare anche il caso in cui nessun ingresso sia attivo
- ▶ Nessuna, 1, 2, 3, 4 (5 combinazioni)

▶ Si può codificare l'uscita in modo alternativo

- ▶ Una uscita V (valid) dice se almeno un ingresso è attivo
- ▶ Le altre 2 uscite codificano l'indice dell'ingresso attivo con priorità più alta
- ▶ Possiamo cominciare a contare da 0 invece che da 1

Tabella della verità

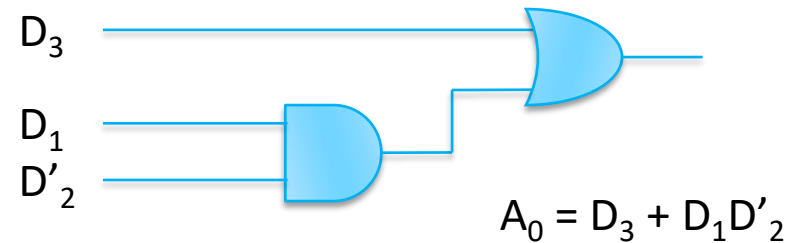
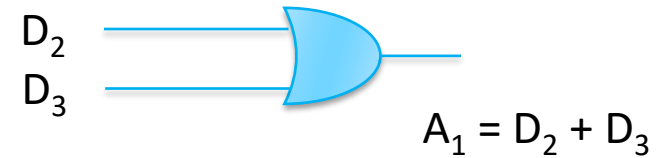
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

- ▶ **Gli ingressi a x indicano che il valore della variabile non è importante (come una wildcard)**
 - ▶ Se c'è una **x** è come in realtà indicare 2 righe contemporaneamente
 - ▶ Con due **x** si indicano quattro righe contemporaneamente
 - ▶ E' solo un modo per scrivere la tabella più velocemente, ma altrimenti non cambia assolutamente nulla
- ▶ **Per V basta mettere in OR tutti gli ingressi**
 - ▶ Per le altre uscite costruiamo le mappe

Mappe di Karnaugh

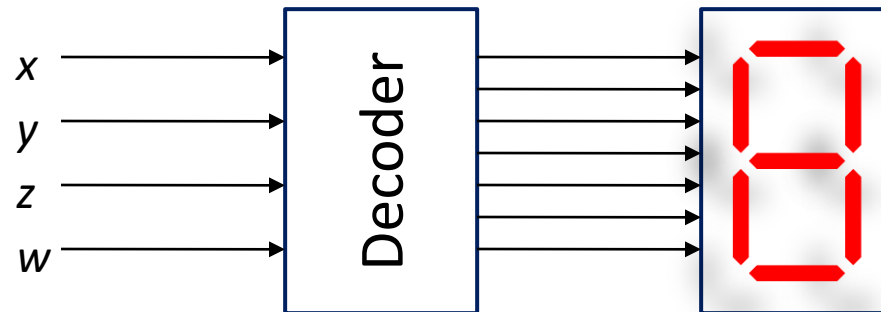
$D_3D_2 \backslash D_1D_0$	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$D_3D_2 \backslash D_1D_0$	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1



Decodifica per display a 7 segmenti

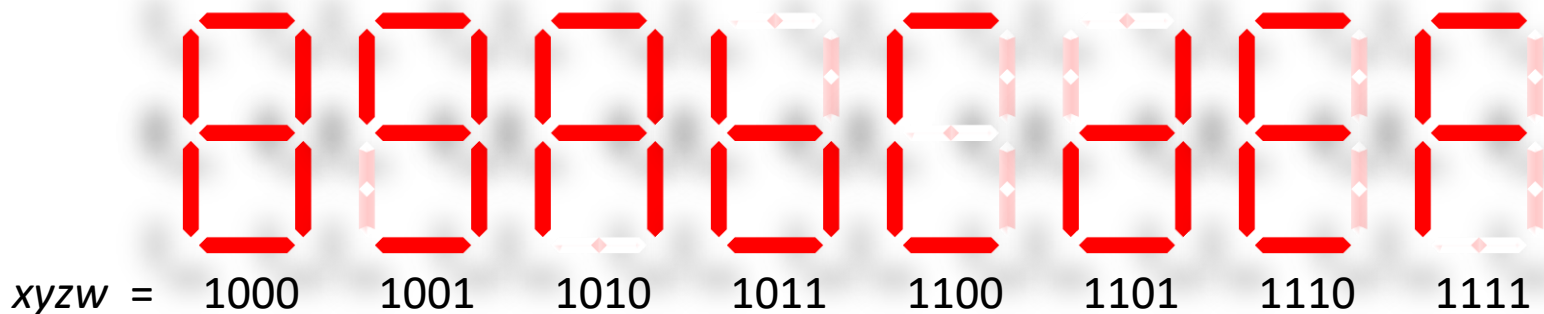
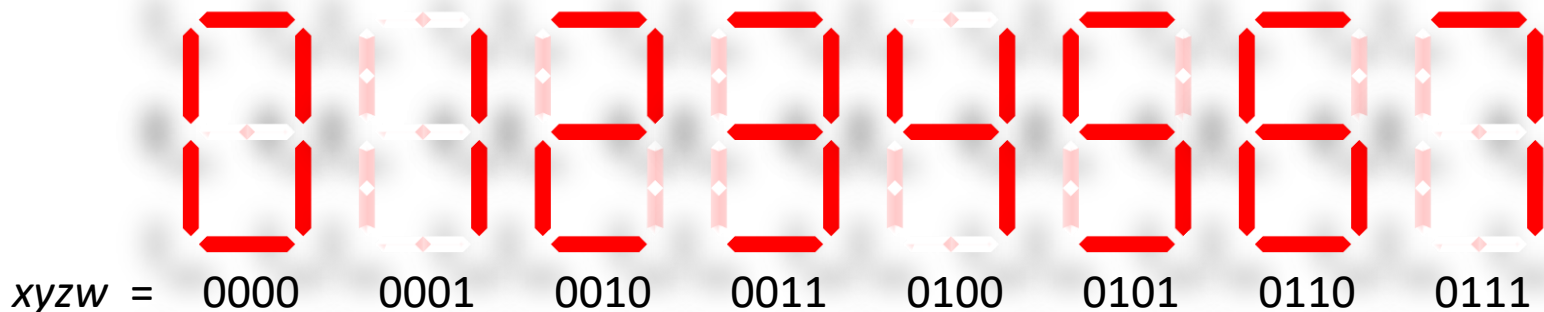
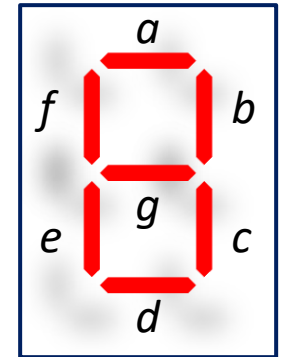
- ▶ Si vuole realizzare una decodifica in grado di pilotare un display a 7 segmenti
 - ▶ Si assume di avere un numero binario a 4 cifre in ingresso, denominate x , y , z , w
 - ▶ Si devono calcolare 7 uscite, ognuna in corrispondenza di un segmento
 - ▶ Si vuole rappresentare il dato binario in esadecimale



Display a 7 segmenti

► Identifichiamo ogni segmento con una lettera

- Per esempio *a*, *b*, *c*, *d*, *e*, *f*, *g* come mostrato a fianco
- Ogni segmento sarà acceso (uscita uguale a 1) oppure spento (uscita uguale a 0) a seconda della combinazione degli ingressi secondo lo schema seguente



Segmento a

- ▶ Isolando il segmento a si ottiene la seguente mappa

xy/zw	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	1	0	1	1
10	1	1	0	1

Implicanti primi:

zw' $x'z$ yz $y'w'$

$x'yw$ $xy'z'$ xw'

- ▶ L'unico implicante primo non essenziale è zw'
- ▶ Quelli essenziali coprono la funzione, quindi
 - ▶ $a = x'z + yz + y'w' + xw' + x'yw + xy'z'$
 - ▶ Ha un totale di 14 letterali

Segmento g

- ▶ Isolando il segmento g si ottiene la seguente mappa

xy/zw	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	1

Implicanti primi:

zw' xz xw $yz'w$
 xy' $x'yz'$ $y'z$ $x'yw'$

- ▶ Essenziali: $y'z$ e xy'
- ▶ Degli altri possiamo per esempio prendere i seguenti
 - ▶ $g = y'z + xy' + zw' + xw + x'yz'$
 - ▶ Ha un totale di 11 letterali

Minimizzazione congiunta

► Si noti che

- Il termine $xy'z'$ è un implicante primo essenziale per il segmento a
- Lo stesso termine è implicante per il segmento g , ma non è implicante primo, essendo contenuto in xy' , che è implicante primo essenziale per g

► Il termine è disponibile nella rete logica

- Può essere vantaggioso usarlo anche per g
- Gli altri 1 di xy' sono coperti nell'espressione di g da altri implicanti
- Si può allora scrivere g come segue
- $g = y'z + xy'z' + zw' + xw + x'yz'$
- Sebbene vi siano 12 letterali, un termine non contribuisce perché già presente nella rete logica, per un totale di soli 9 letterali

xy/zw	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	1	0	1	1
10	1	1	0	1

a

xy/zw	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	1

g

- ▶ **Abbiamo realizzato un gran numero di elementi base**
 - ▶ Multiplexer
 - ▶ Encoder
 - ▶ Decoder
 - ▶ Transcoder
- ▶ **Mettendoli assieme si possono fare circuiti più complessi**
 - ▶ Non necessariamente a due livelli
 - ▶ Ma facili da capire
- ▶ **Minimizzazione congiunta**
 - ▶ Può fornire risultati migliori
 - ▶ E' però molto più complicato farla a mano
 - ▶ Meglio usare programmi per calcolatore specializzati

Indifferenze o don't care

Quando non tutto serve

Le indifferenze

- ▶ **In certe occasioni il valore dell'uscita in corrispondenza di alcune combinazioni di ingressi è irrilevante**
 - ▶ Per esempio, nel caso della codifica 8 a 3, quando vi sono due o più ingressi a 1 contemporaneamente
 - ▶ Ogni volta che gli ingressi codificano un numero di combinazioni inferiore a 2^n
- ▶ **Le mappe di Karnaugh però devono includere un valore per tutte le combinazioni di ingressi**
 - ▶ Che valore dare?
 - ▶ Se l'ingresso non si presenta mai, si può dare il valore 0 o 1 indifferentemente
 - ▶ Vogliamo assegnare un valore che ci consenta di ottenere un'espressione più semplice
- ▶ **Le combinazioni di ingresso per cui non si indica un valore preciso dell'uscita si chiamano **indifferenze** o **don't care****

Dimensione degli implicanti

- ▶ **Implicanti che coprono molti 1 (cioè sono più grossi) sono anche quelli con meno letterali**
 - ▶ Infatti andiamo a prendere gli implicanti primi, che sono i più grossi per definizione, per realizzare un'espressione minima
- ▶ **Possiamo allora usare le indifferenze per ingrandire gli implicanti**
 - ▶ Cioè immaginiamo che le indifferenze siano a 1
 - ▶ Alcuni degli implicanti si possono espandere a coprire gli uni indifferenti
- ▶ **Esempio**
 - ▶ Si supponga di voler realizzare una decodifica per display a 7 segmenti che mostri solo le cifre numeriche decimali
 - ▶ Gli ingressi da 1010 (0xA) a 1111 (0xF) producono delle indifferenze, perché non ci interessa il valore delle uscite

Segmento a con indifferenze

xy/zw	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	-	-	-	-
10	1	1	-	-

Implicanti primi:

z $y'w'$

yw x

- ▶ $a = z + y'w' + yw + x$
- ▶ 6 letterali

xy/zw	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	1	0	1	1
10	1	1	0	1

Implicanti primi:

zw' $x'z$ yz $y'w'$

$x'yw$ $xy'z'$ xw'

- ▶ $a = x'z + yz + y'w' + xw' + x'yw + xy'z'$
- ▶ 14 letterali

Segmento *g* con indifferenze

xy/zw	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	-	-	-	-
10	1	1	-	-

Implicanti primi:

zw' x yz'

$y'z$ yw'

► $g = zw' + x + yz' + y'z$

► 7 letterali

xy/zw	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	1

Implicanti primi:

zw' xz xw $yz'w$

xy' $x'yz'$ $y'z$ $x'yw'$

► $g = y'z + xy' + zw' + xw + x'yz'$

► 11 letterali

Esempio

xy/zw	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	-	-
10	-	-	-	-

► Due implicant primari

- $y \cdot x$
- $f = y + x$
- $f = y$

Inutile aggiungere implicant nella copertura per coprire solamente indifferenze

Consideriamo le indifferenze come 1 quando si cercano gli implicant

Ma le consideriamo come fossero degli 0 quando occorre scegliere quali implicant utilizzare!!

Indifferenze: Take away

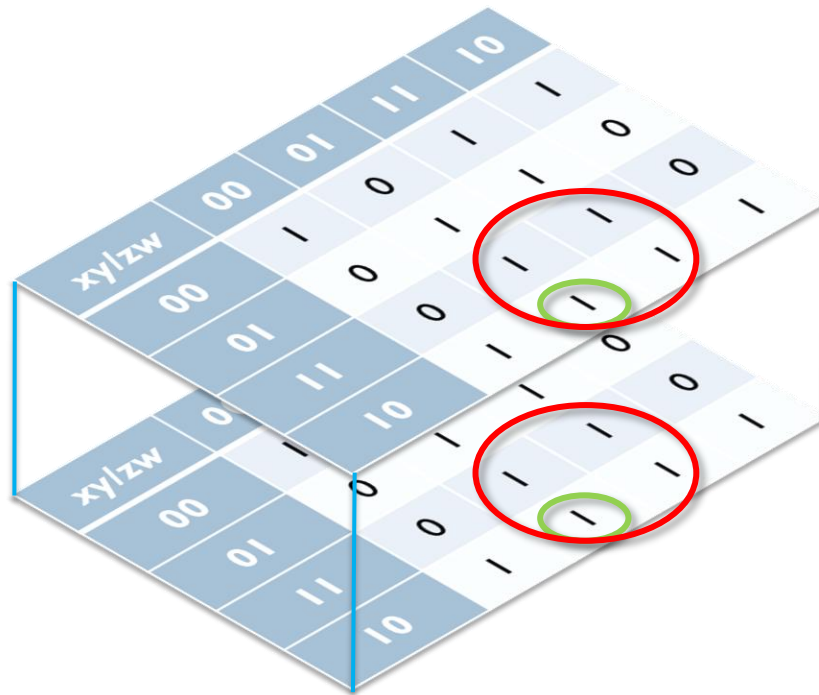


- ▶ **Le indifferenze sono combinazioni di ingresso per le quali non ci interessa il valore dell'uscita**
 - ▶ Per esempio, perché sappiamo che la combinazione di ingresso non si presenta mai
 - ▶ L'uscita può quindi essere considerata alternativamente 0 oppure 1 a seconda della convenienza
- ▶ **Sfruttiamo le indifferenze per semplificare l'espressione**
 - ▶ Assumendo che l'indifferenza valga 1, possiamo espandere gli implicant, che quindi hanno meno letterali
 - ▶ Durante la scelta degli implicant, assumiamo che l'indifferenza valga 0, così da non aggiungere implicant inutili
 - ▶ In particolare, un impicante non sarà mai essenziale a causa di una indifferenza
- ▶ **Come ottenere le indifferenze?**
 - ▶ Il caso più semplice è quando sappiamo che certi ingressi non si presentano
 - ▶ Il caso generale è molto più complesso e richiede l'analisi di una rete per identificare quando una uscita è effettivamente indifferente

Mappe a 5 variabili

- ▶ **A 5 variabili (x, y, z, w, v) si ottengono 32 caselle**
 - ▶ Anche usando il codice Gray, impossibile mantenere le vicinanze geometriche sul piano
- ▶ **Si possono usare due tabelle da 4 variabili**
 - ▶ Entrambe funzione delle variabili x, y, z , e w
 - ▶ La prima relativa al caso in cui $v = 0$
 - ▶ La seconda relativa al caso in cui $v = 1$
 - ▶ In pratica consideriamo l'espansione di Shannon sulla variabile v
- ▶ **Le vicinanze su v vanno considerate tra una tabella e l'altra**
 - ▶ Caselle che occupano la stessa posizione su entrambe le tabelle sono da considerarsi vicine, e possono formare un termine
 - ▶ Lo stesso vale per gli implicant
- ▶ **Geometricamente**
 - ▶ Immaginate che le tabelle siano poste una sopra all'altra

Mappe a 5 variabili



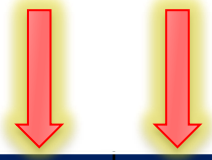
Esempio: numeri primi (o quasi)

- ▶ Ingresso: numero binario a 5 bit
- ▶ Uscita a 1 se il numero è primo o divisibile per 3, a 0 altrimenti

$a_4a_3/a_2a_1a_0$	000	001	011	010	100	101	111	110
00	0	1	3	2	4	5	7	6
01	8	9	11	10	12	13	15	14
11	24	25	27	26	28	29	31	30
10	16	17	19	18	20	21	23	22

Esempio: numeri primi (o quasi)

- ▶ Ingresso: numero binario a 5 bit
- ▶ Uscita a 1 se il numero è primo o divisibile per 3, a 0 altrimenti



Queste colonne NON sono vicine!

$a_4a_3/a_2a_1a_0$	000	001	011	010	100	101	111	110
00		1	1	1		1	1	1
01		1	1		1	1	1	
11	1		1			1	1	1
10		1	1	1		1	1	

Esempio: numeri primi (o quasi)

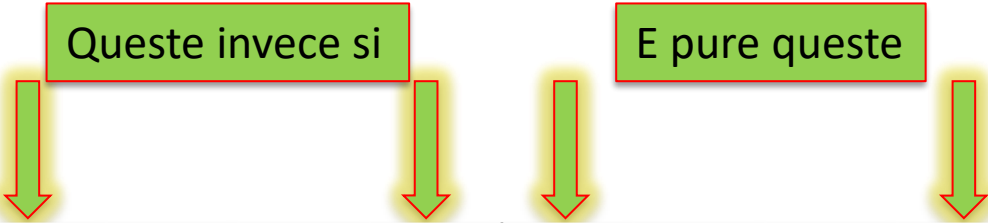
- ▶ Ingresso: numero binario a 5 bit
- ▶ Uscita a 1 se il numero è primo o divisibile per 3, a 0 altrimenti

E neanche queste!!

$a_4a_3/a_2a_1a_0$	000	001	011	010	100	101	111	110
00		1	1	1		1	1	1
01		1	1		1	1	1	
11	1		1			1	1	1
10		1	1	1		1	1	

Esempio: numeri primi (o quasi)

- ▶ Ingresso: numero binario a 5 bit
- ▶ Uscita a 1 se il numero è primo o divisibile per 3, a 0 altrimenti



$a_4a_3/a_2a_1a_0$	000	001	011	010	100	101	111	110
00		1	1	1		1	1	1
01		1	1		1	1	1	
11	1		1			1	1	1
10		1	1	1		1	1	

Esempio: numeri primi (o quasi)

- ▶ Ingresso: numero binario a 5 bit
- ▶ Uscita a 1 se il numero è primo o divisibile per 3, a 0 altrimenti

E ovviamente queste

$a_4a_3/a_2a_1a_0$	000	001	011	010	100	101	111	110
00		1	1	1		1	1	1
01		1	1		1	1	1	
11	1		1			1	1	1
10		1	1	1		1	1	

Esempio: numeri primi (o quasi)

- ▶ Ingresso: numero binario a 5 bit
- ▶ Uscita a 1 se il numero è primo o divisibile per 3, a 0 altrimenti

$a_4a_3/a_2a_1a_0$	000	001	011	010	100	101	111	110
00		1	1	1		1	1	1
01		1	1		1	1	1	
11	1		1			1	1	1
10		1	1	1		1	1	

Esempio: numeri primi (o quasi)

► Espressione minima

► Ci dobbiamo tenere tutti gli implicant

$$p = a_4'a_0 + a_3'a_0 + a_2a_0 + a_1a_0 + a_4'a_3'a_1 + a_3'a_2'a_1 + a_4a_3a_2a_1 + a_4'a_3a_2a_1' + a_4a_3a_2'a_1'a_0'$$

$a_4a_3/a_2a_1a_0$	000	001	011	010	100	101	111	110
00		1	1	1		1	1	1
01		1	1		1	1	1	
11	1		1			1	1	1
10		1	1	1		1	1	

Osservazioni

- ▶ **Espressione minima**

- ▶ E' un po' tedioso
- ▶ E' inoltre facile sbagliare con tutti gli implicant

- ▶ **Possibile un'altra numerazione delle colonne**

- ▶ Si può usare direttamente il codice Gray
- ▶ Metodo alternativo
- ▶ Attenzione però che le vicinanze cambiano
 - ▶ Per esempio, le due colonne di mezzo sarebbero vicine
- ▶ Non si può più pensare alle due mappe come sovrapposte

Mappe a 6 e più variabili

- ▶ **Si possono usare 4 mappe a 4 variabili**
 - ▶ In ogni caso non sono molto convenienti
- ▶ **Per più di 6 variabili attenzione a come si ordinano le mappe**
 - ▶ Conviene di nuovo usare il codice Gray gerarchicamente anche per le variabili che indicizzano le mappe
- ▶ **Solo più complesse geometricamente**
 - ▶ Mettono a dura prova il colpo d'occhio
 - ▶ Altrimenti non c'è nulla di nuovo
 - ▶ Spezzatele con Shannon!!

Good luck!

abc / def		0				1			
		00	01	11	10	00	01	11	10
0	00	0	1	1	0	0	-	1	-
	01	1	-	0	1	-	-	1	0
	11	0	0	-	1	1	-	0	0
	10						
1	00								
	01								
	11								
	10								

Altri modi di sintetizzare e fare il circuito

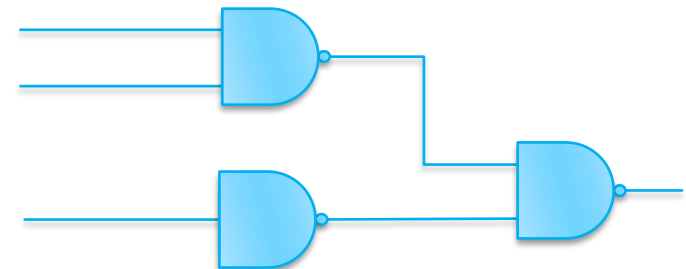
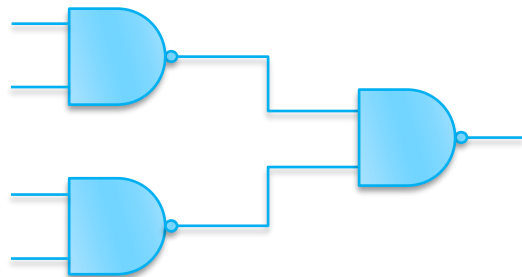
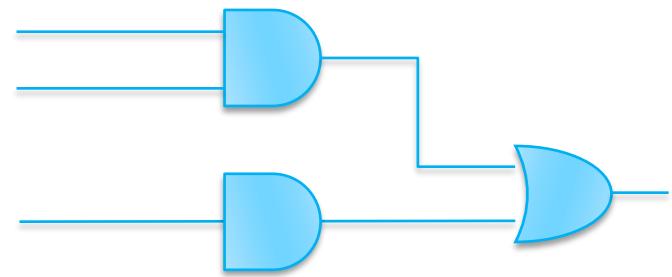
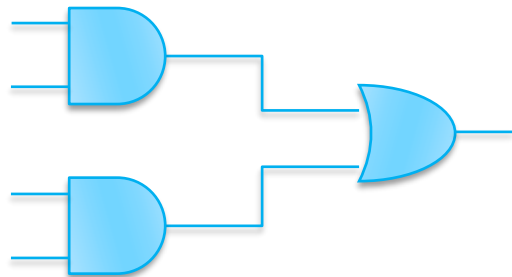
Realizzazione NAND - NAND

- ▶ **Conveniente usare sempre lo stesso tipo di porta logica**
 - ▶ Rende la realizzazione più omogenea in termini di caratteristiche elettriche
 - ▶ Le porte invertenti usano meno transistori di quelle non invertenti
- ▶ **Si ottiene la sintesi NAND – NAND partendo da quella AND – OR ed applicando la legge di De Morgan**
 - ▶ $f = z + y'w' + yw + xz'$
 - ▶ $f = (z + y'w' + yw + xz')''$
 - ▶ $f = ((z)'(y'w')')(yw)'(xz')')'$
- ▶ **Gli implicant sono quindi gli stessi della sintesi AND – OR**
 - ▶ Si individuano allo stesso modo

Nella pratica

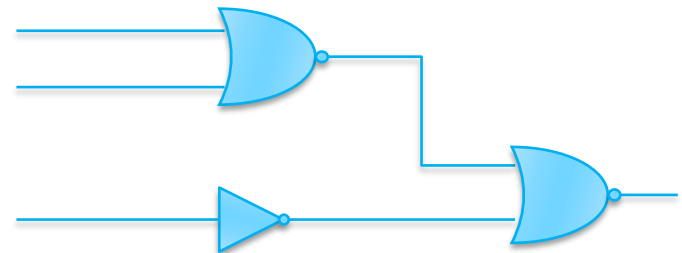
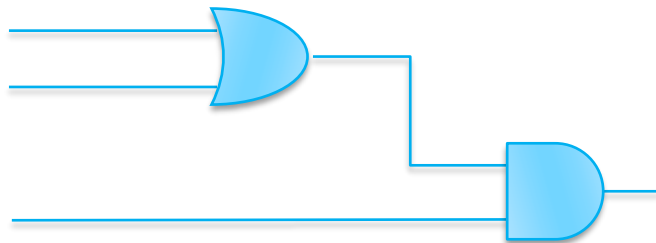
► Procedimento

- Ad ogni AND si sostituisce una NAND
- Ad ogni OR si sostituisce una NAND
- Se un ingresso va direttamente alla OR viene negato



Realizzazione NOR – NOR

- ▶ **Come prima, solo che partiamo dal prodotto di somme**
 - ▶ Gli implicant si ottengono dagli zeri della funzione
 - ▶ Applichiamo le legge di De Morgan
 - ▶ Di nuovo si sostituisce tutto con porte NOR
 - ▶ Con l'avvertenza di negare gli ingressi che vanno direttamente alla porta di uscita



- ▶ **Vi sono varie alternative realizzative**
 - ▶ Si possono differenziare per tecnologia
 - ▶ A seconda della funzione possono produrre implementazioni migliori o peggiori
 - ▶ Talvolta utile rappresentare tutto con sole porte NAND per semplificare l'analisi da parte di strumenti automatici
- ▶ **Procedimento standard**
 - ▶ In tutti i casi il procedimento non cambia
 - ▶ E' possibile passare da una rappresentazione ad un'altra applicando semplici proprietà dell'algebra Booleana
- ▶ **Complessità**
 - ▶ Possibili mappe con gran numero di variabili
 - ▶ Difficili da manipolare a mano