Prof. Mariacarla Staffa
a.a. 2022/2023

Laboratorio di Architettura Degli Elaboratori

Logica Combinatoria: blocchi logici funzionali: Adder, Multiplexer e Decoders

Logica Combinatoria

- Una rete combinatoria è un circuito logico avente n ingressi (x_1,x_2,…,x_n) ed m uscite (y_1,y_2, …,y_m), ciascuno dei quali assume valori binari (0/1), tale che a ciascuna combinazione degli ingressi corrisponde un'unica combinazione delle uscite.

- Da un punto di vista logico, ogni uscita può essere definita come una funzione booleana degli ingressi $y_i = y_i(x_1,x_2,\ldots,x_n)$.

- Ad ogni istante, il valore delle uscite dipende unicamente dal valore assunto dagli ingressi nello stesso istante.

Logica Combinatoria

- La procedura per progettare una rete logica combinatoria passa attraverso i seguenti stadi:

  - definizione completa e univoca del problema da risolvere;
  - analisi del problema, con individuazione delle variabili d'ingresso e delle funzioni di uscita;
  - scrittura della tabella della verità di ogni funzione;
  - sintesi delle funzioni e loro semplificazione con le mappe di Karnaugh;
  - disegno della schema logico della rete.

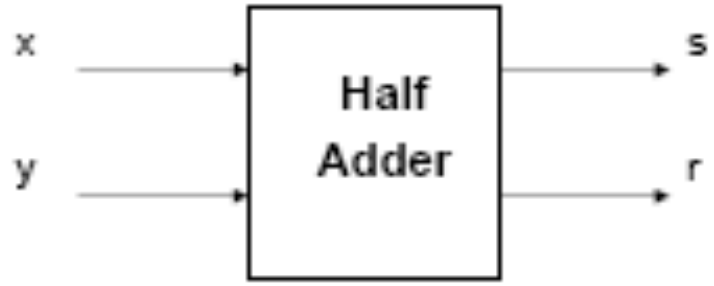
Blocchi logici funzionali

- Diamo qualche esempio di blocco logico funzionale, elemento costitutivo di schemi logici complessi.

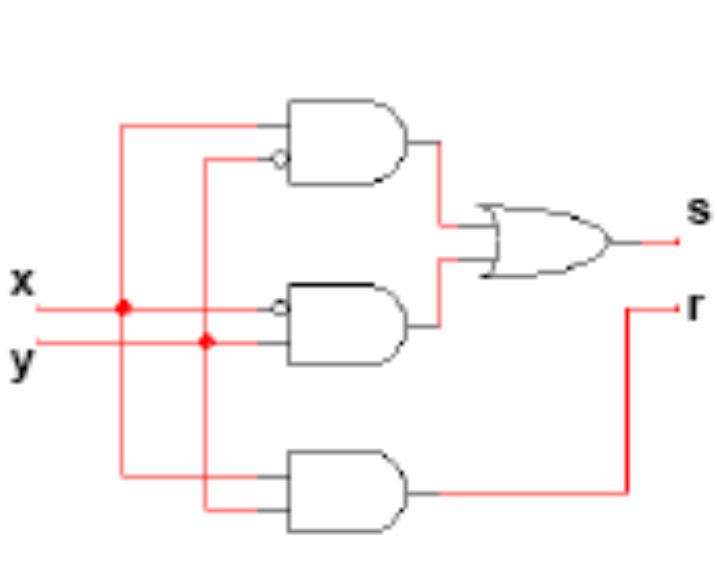
Sommatore

- Esegue l'addizione di cifre binarie fornendo in uscita la cifra somma e la cifra riporto. Sono possibili due schemi:
  - semiaddizionatore (half adder)
    - **2 cifre in ingresso**
    - **somma e riporto in uscita**
  - addizionatore completo (full adder)
    - **2 cifre in ingresso + carry in ingresso**
    - **somma e riporto in uscita**

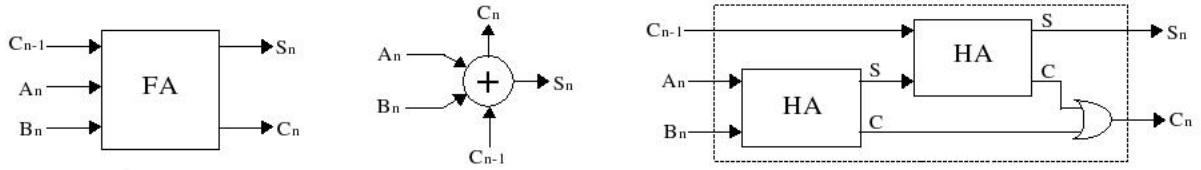
Half adder



x	y	s	r
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full Adder

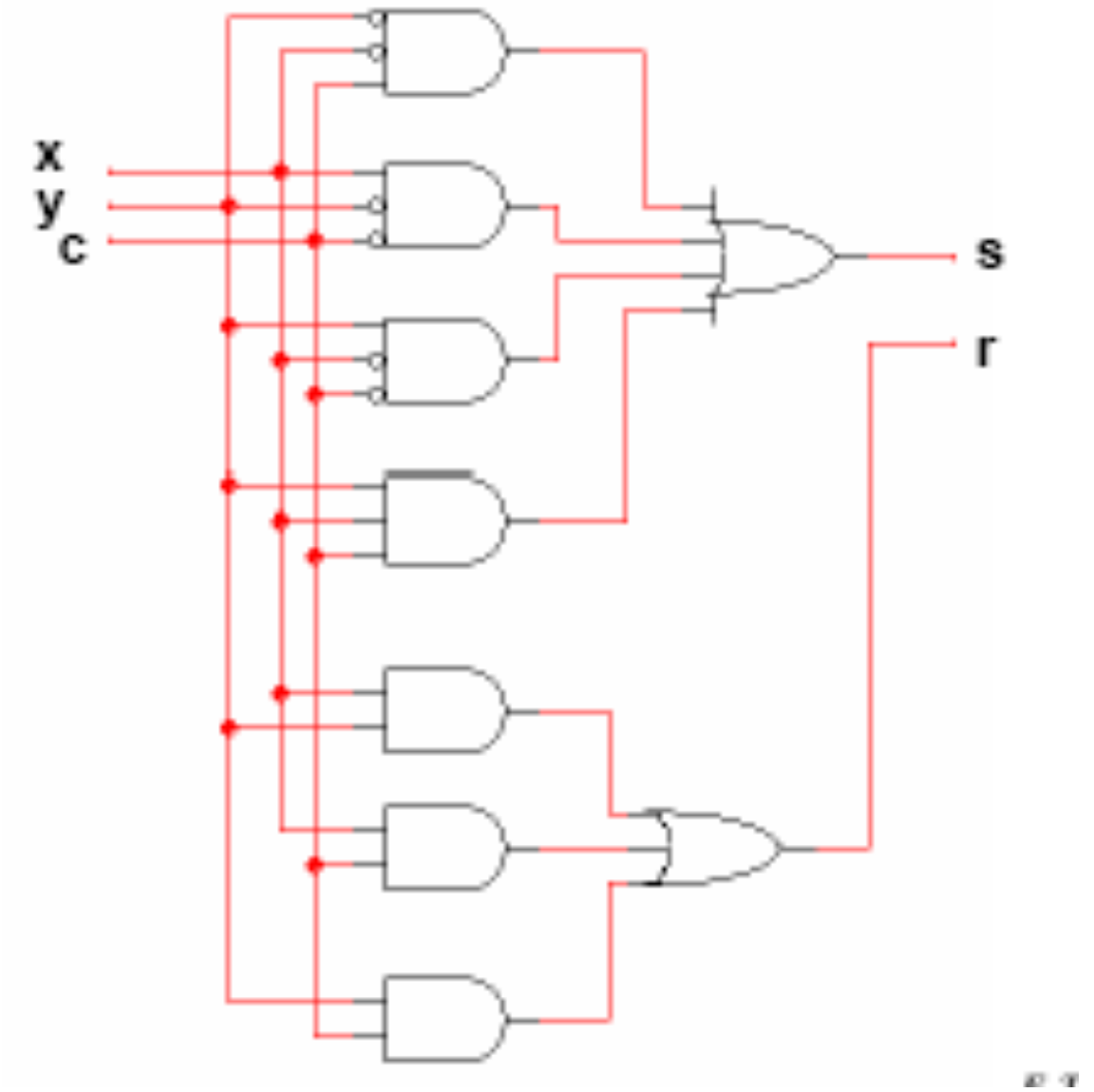


Simboli logici e realizzazione di un FA

- Il **full-adder** o **sommatore completo**[1] è un circuito logico caratterizzato da tre ingressi e due uscite.
- La sua funzionalità è quella di eseguire una somma tra due numeri espressi in formato binario con lunghezza di parola a un bit.
- È un componente fondamentale dell'elettronica digitale perché, connesso opportunamente con altri full-adder e porte logiche può dare luogo alle unità di elaborazione ALU (*Arithmetic Logic Unit*) dei processori.
- I full-adder sono le fondamenta su cui è basata la costruzione di semplici calcolatrici. Il full-adder è costituito dall'insieme di due half-adder e una porta logica OR, opportunamente collegati.

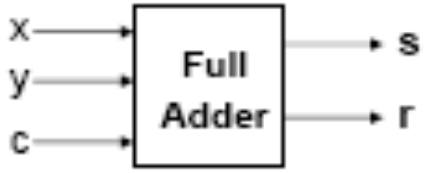
Full Adder – sintesi diretta

x	y	c	s	r
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

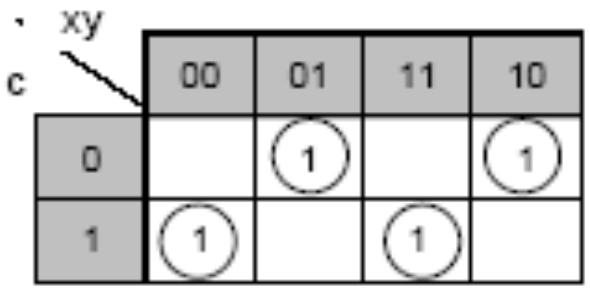


Full Adder

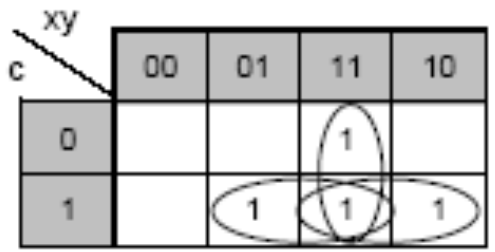
In logica binaria esegue questa semplice operazione: A + B + Ci = S + Co
dove A e B sono gli operandi, Ci il riporto (C → *carry)* in ingresso della precedente somma e S e Co sono la somma e il riporto di uscita. Ogni variabile è un bit (0 oppure 1)



x	y	c	s	r
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



s = !x!yc+!xy!c+xyc+x!y!c



r = xy+yc+xc

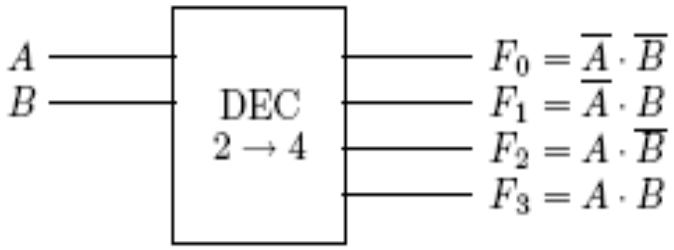
Il decoder binario

Il **decoder** o **decodificatore**, è un tipo di componente usato nell'elettronica digitale.

La sua funzione è opposta a quella dell'encoder: in base alla combinazione dei bit presenti ai suoi ingressi, attiva una corrispondente combinazione di bit sulle linee di uscita.

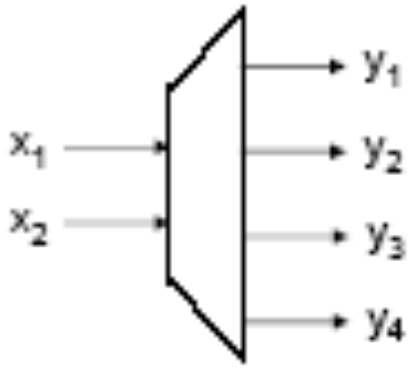
Un decoder binario ha tante uscite quante sono le combinazione delle variabili d'ingresso.

E' fatto in modo che sia attiva la sola uscita che corrisponde alla combinazione presente in ingresso.



Il decoder binario

- Rete combinatoria ad n ingressi ed a 2^n uscite. Per ogni combinazione degli ingressi, solo una uscita assume valore 1 mentre le altre sono uguali a 0.



decodificatore 1/4

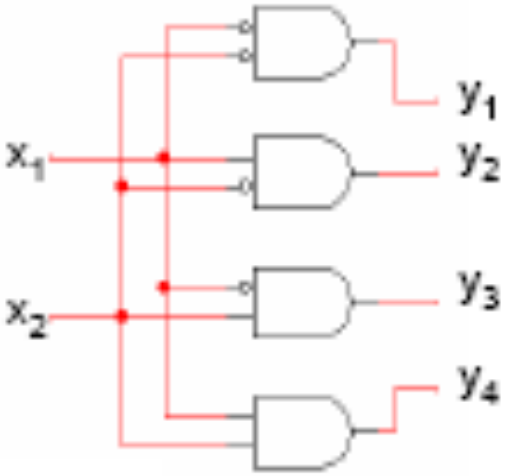
x_1	x_2	y_1	y_2	y_3	y_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$y_1 = !x_1!x_2$

$y_2 = !x_1x_2$

$y_3 = x_1!x_2$

$y_4 = x_1x_2$



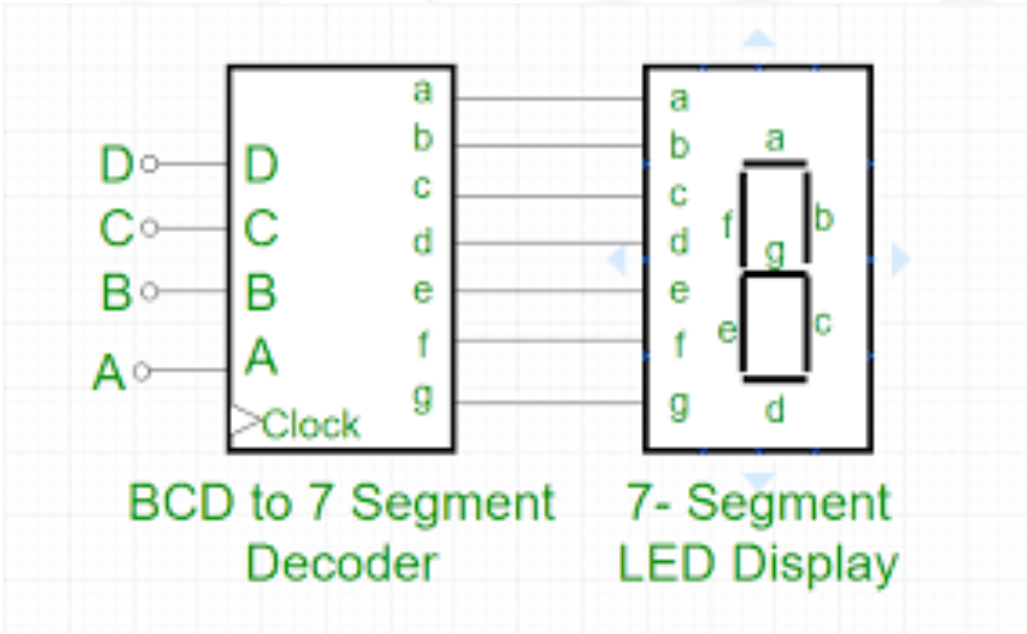
Il decoder BCD

- Il codice BCD (Binary Code Decimal) **viene usato per la rappresentazione di numeri che possano essere convertiti rapidamente in cifre decimali**. In ogni byte vengono rappresentate due cifre decimali; ogni cifra del numero decimale occupa 4 bit, che assumono il valore della corrispondente rappresentazione binaria.
- La codifica binary-coded decimal è un modo comunemente utilizzato in informatica ed elettronica per rappresentare le cifre decimali in codice binario, che sfrutta in parte la convertibilità da base 2 a base 16

DECIMALE	ESADECIMALE	BINARIO	BCD
0	0	0000	0000 0000
1	1	0001	0000 0001
2	2	0010	0000 0010
3	3	0011	0000 0011
4	4	0100	0000 0100
5	5	0101	0000 0101
6	6	0110	0000 0110
7	7	0111	0000 0111
8	8	1000	0000 1000
9	9	1001	0000 1001
10	A	1010	0001 0000
11	B	1011	0001 0001
12	C	1100	0001 0010
13	D	1101	0001 0011
14	E	1110	0001 0100
15	F	1111	0001 0101

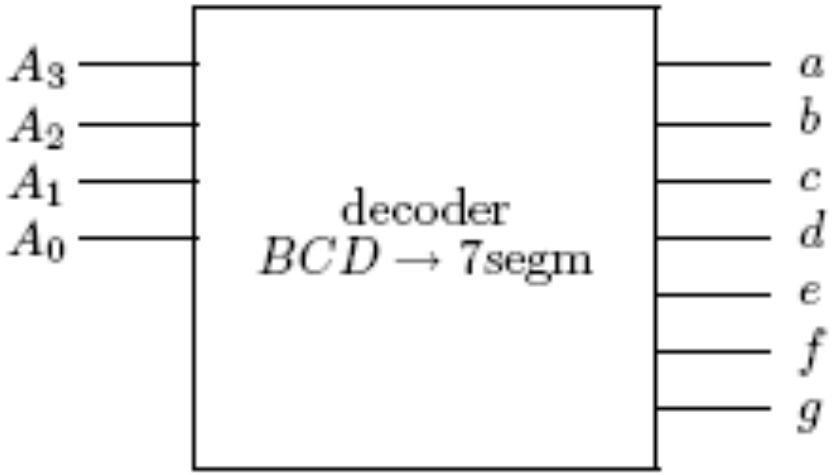
Il decoder BCD - 7 segmenti

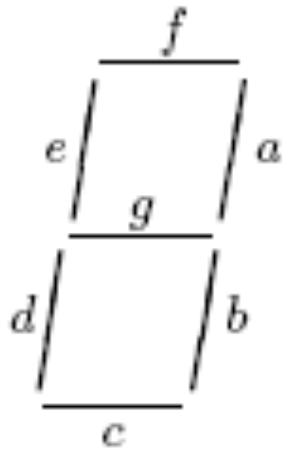
- Esercizio:
  - Realizzare un decoder BCD con un display a 7 segmenti.
  - Eseguire la minimizzazione con le mappe di Karnaugh.
  - Eseguire la rappresentazione grafica del circuito con un simulatore online.
  - Confrontare i risultati



Il decoder BCD - 7 segmenti

- Rete combinatoria a 4 ingressi e a 7 uscite. Serve per pilotare display numerici a 7 segmenti.

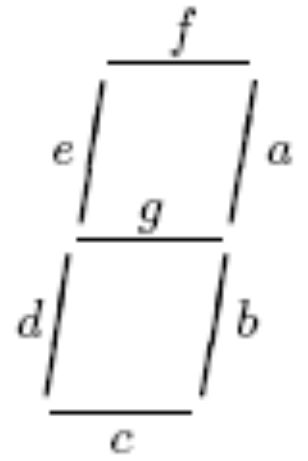




Il decoder BCD - 7 segmenti

- Tabella della verità

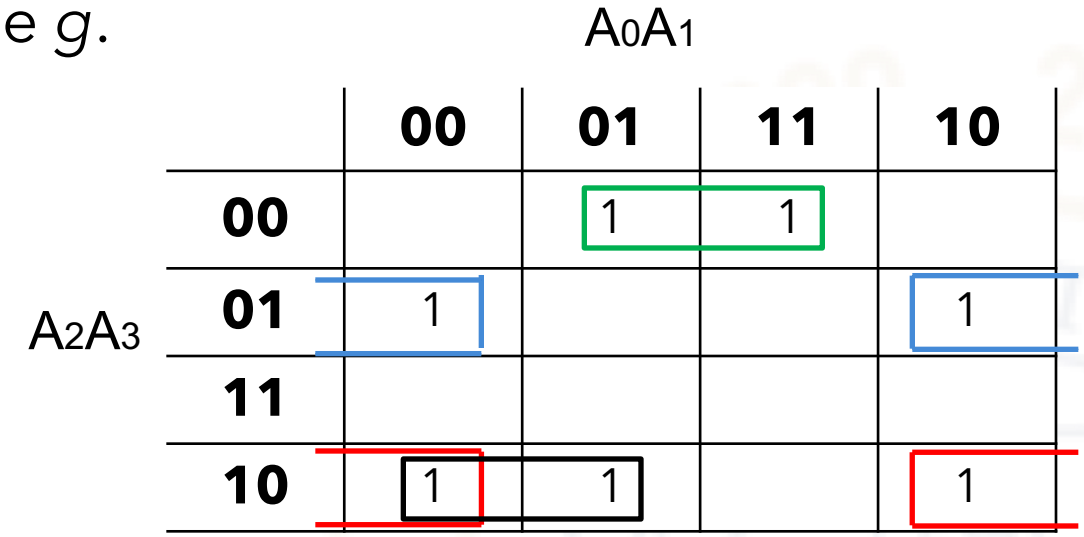
	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	1	1	0	0	0	0	0
2	0	0	1	0	1	0	1	1	0	1	1
3	0	0	1	1	1	1	1	0	0	1	1
4	0	1	0	0	1	1	0	0	1	0	1
5	0	1	0	1	0	1	1	0	1	1	1
6	0	1	1	0	0	1	1	1	1	1	1
7	0	1	1	1	1	1	0	0	0	1	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	0	0	1	1	1



Il decoder BCD - 7 segmenti

- Semplificazione della funzione *g*.

	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	1	1	0	0	0	0	0
2	0	0	1	0	1	0	1	1	0	1	1
3	0	0	1	1	1	1	1	0	0	1	1
4	0	1	0	0	1	1	0	0	1	0	1
5	0	1	0	1	0	1	1	0	1	1	1
6	0	1	1	0	0	1	1	1	1	1	1
7	0	1	1	1	1	1	0	0	0	1	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	0	0	1	1	1



$$g = (!A_2!A_3!A_0A_1 + !A_2!A_3A_0A_1) + (!A_2A_3!A_0!A_1 + !A_2A_3A_0!A_1) +$$

$$+ (A_2!A_3!A_0!A_1 + A_2!A_3A_0!A_1) + (A_2!A_3!A_0!A_1 + A_2!A_3!A_0A_1) =$$

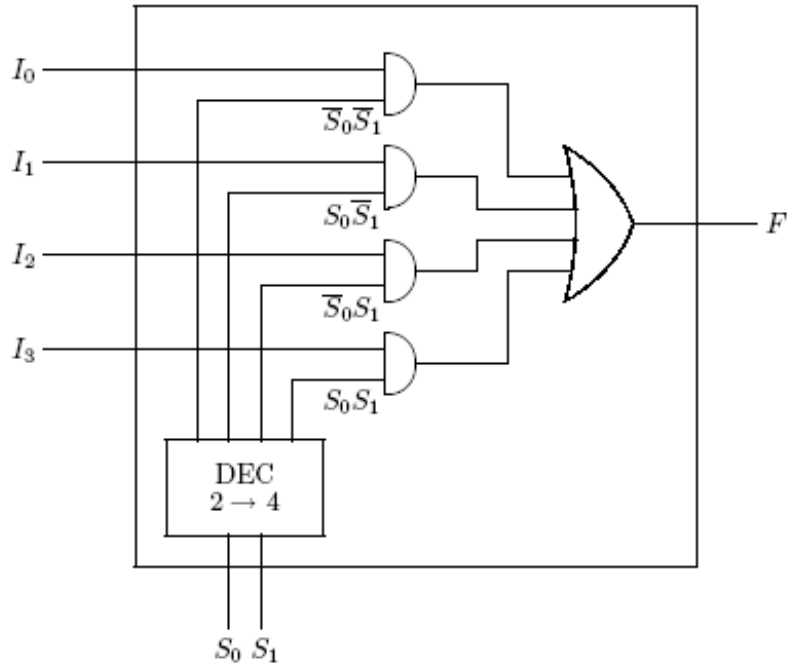
$$= !A_2!A_3A_1 + !A_2A_3!A_1 + A_2!A_3!A_1 + A_2!A_3!A_0$$

Combinational Building Blocks

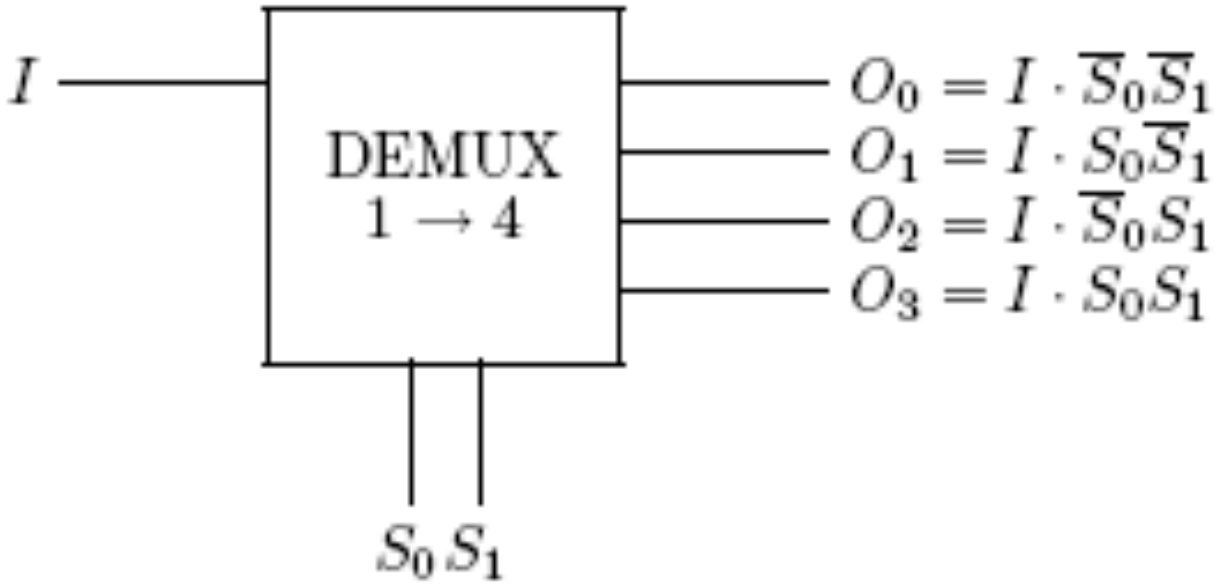
- Multiplexers
- Decoders

Il multiplexer binario

- Il multiplexer è la realizzazione interamente elettronica di un commutatore meccanico.
- Esso permette di selezionare un ingresso fra tanti e di inviare in uscita il suo stato.
- Il multiplexer si avvale di un decoder per effettuare la selezione degli ingressi.
- A fianco è rappresentato, in forma di blocco funzionale, un multiplexer a 4 ingressi (4 a 1).



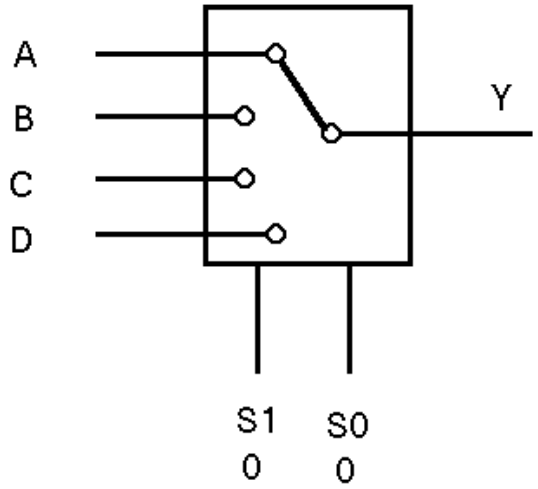
Il demultiplexer



- Il demultiplexer svolge la funzione opposta rispetto al multiplexer: invia il segnale in ingresso su una delle possibili uscite selezionate da un decoder. A fianco è dato il blocco logico funzionale del demultiplexer 1 a 4 e sono scritte le espressioni logiche delle uscite.

Multiplexer

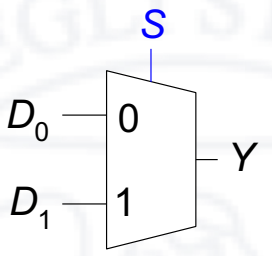
- Un multiplexer è sostanzialmente un selettore di linea



- In generale è costituito da N ingressi (dove N è una potenza di 2), 1 uscita, e $\log_2 N$ linee di selezione che indicano a quale ingresso deve corrispondere l'output

Multiplexer (Mux)

- Seleziona tra uno degli N ingressi da collegare all'uscita
- $\log_2$N-bit input di selezione – input di controllo
- **Example:** **2:1 Mux**

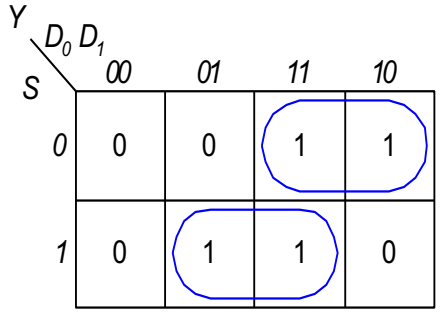


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

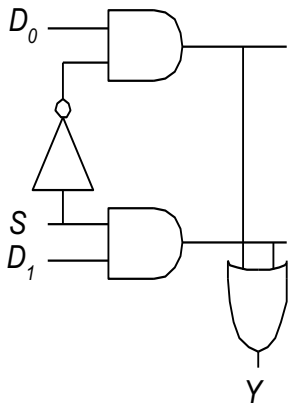
S	Y
0	D_0
1	D_1

Multiplexer Implementations

- **Logic gates**
  - Forma SOP (somma di Prodotti)

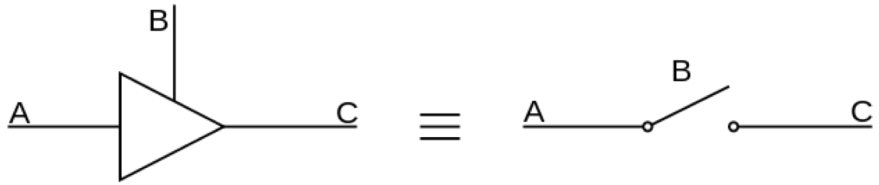


$$Y = D_0\overline{S} + D_1S$$



Multiplexer Implementations: **Tristates**

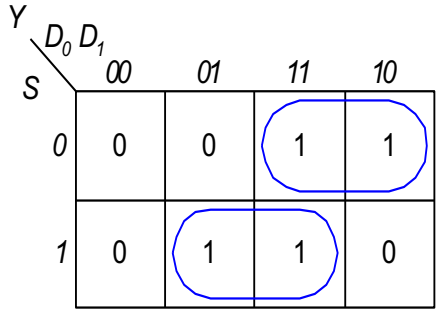
- Per un MUX a N-input mux, usa N tristates
- Attivarne esattamente uno per selezionare l'ingresso appropriato
- In elettronica digitale, una porta logica si dice three state, tri-state o 3-state quando la sua uscita può trovarsi in un terzo stato di alta impedenza, spesso indicato con il simbolo Z, oltre ai due livelli logici già presenti nella logica binaria.
- Uno dei principali dispositivi a tre stati è il **buffer a tre stati** (o **buffer tri-state**), che è possibile pilotare in modo che si comporti come un interruttore aperto nello stato di alta impedenza, e come un buffer altrimenti. Questa condizione di lavoro viene attuata tramite il condizionamento logico di un ingresso, detto *enable*, preposto allo scopo.
  La sua tabella di verità è dunque la seguente:



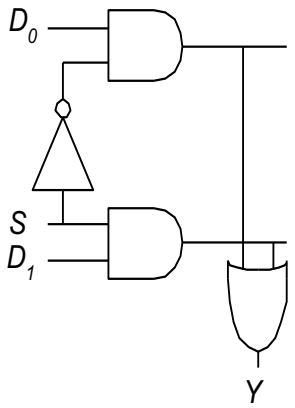
In	En	Out
0	0	Z
1	0	Z
0	1	0
1	1	1

Multiplexer Implementations

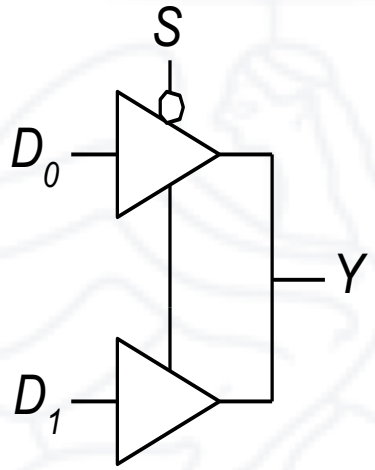
- **Logic gates**
  - Forma SOP (somma di Prodotti)

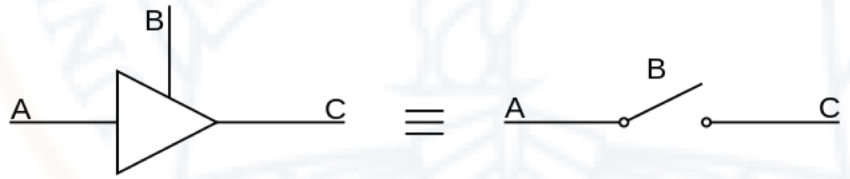


$$Y = D_0\overline{S} + D_1S$$



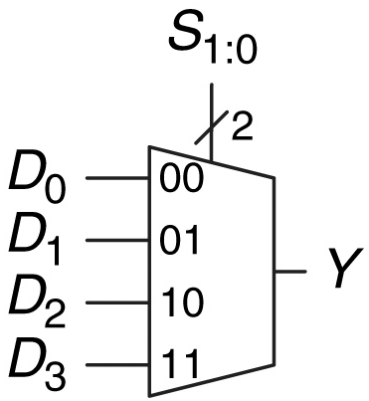
- **Tristates**
  - Per un MUX a N-input mux, usa N tristates
  - Attivarne esattamente uno per selezionare l'ingresso appropriato





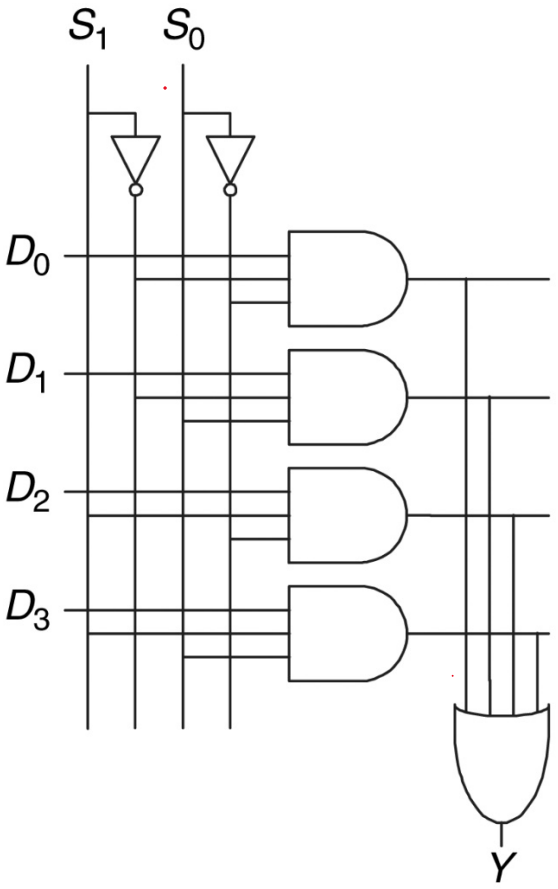
Multiplexer 4:1

4:1 MUX



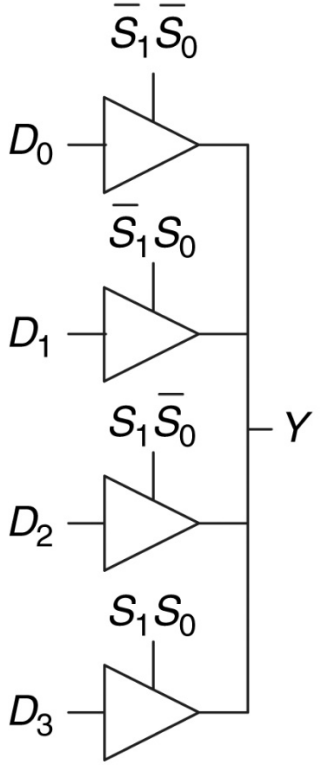
=

SOP Logic



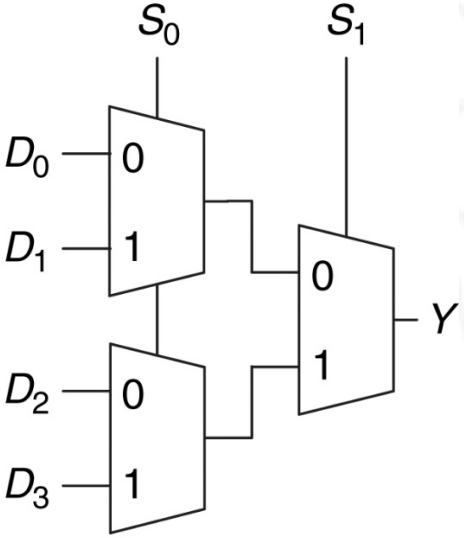
(a)

TRISTATES



(b)

Multiple 2:1 MUX



(c)

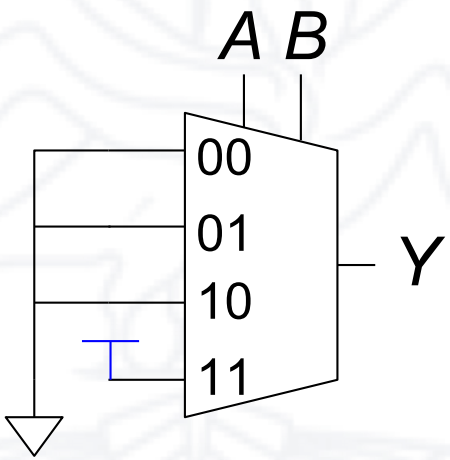
Logic using Multiplexers

I Multiplezer possono essere utilizzati come *Lookup Table* di funzioni logiche:

Per esempio un multiplexer 4:1 può essere usato per implementare una porta logica AND a due ingressi

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$Y = AB$

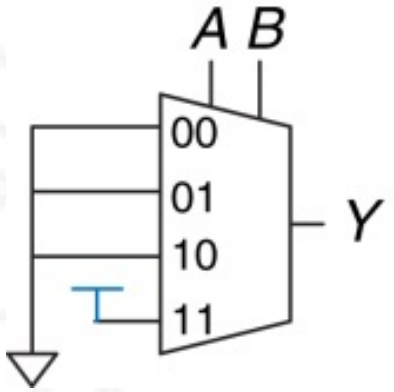


Sintetizzare funzioni booleane con mux

- I multiplexer possono essere usati anche per sintetizzare delle funzioni booleane
- Sintetizzare una funzione di m variabili con un mux a 2^m linee è molto semplice: le variabili saranno linee di selezione. Data una certa configurazione delle variabili, la linea di ingresso corrispondente sarà posta al valore della funzione in quella configurazione
- Di fatto le linee di ingresso riproducono la tabella di verità della funzione

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$Y = AB$



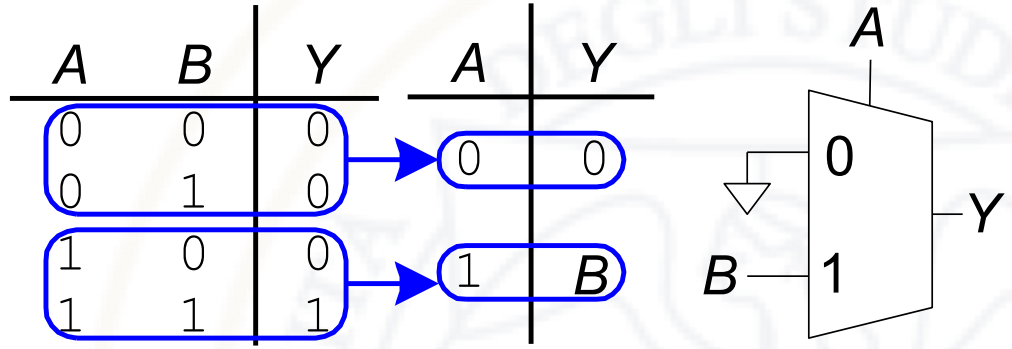
Sintetizzare funzioni booleane con mux

- E' possibile utilizzare un mux con 2^{m-1} ingressi per sintetizzare un funzione ad m variabili
- Le prime m-1 variabili saranno linee di selezione, mentre le linee di ingresso possono essere poste a 0,1, oppure all'ultima variabile (positiva o negata)

Logic using Multiplexers

Reducing the size of the mux

$Y = AB$

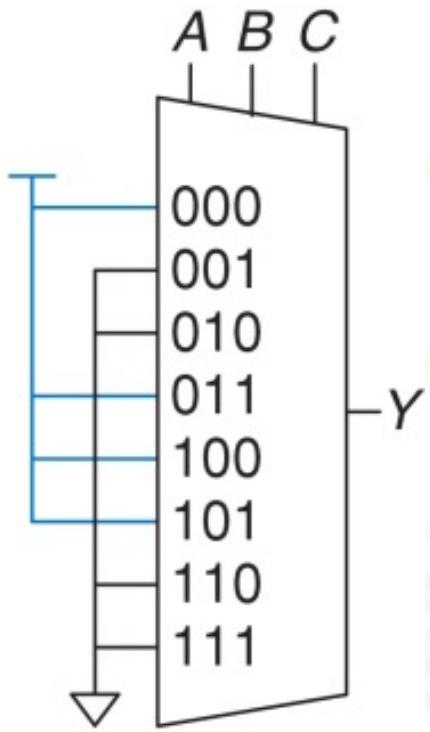


Sintetizzare funzioni booleane con mux

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$Y = A\overline{B} + \overline{B}\overline{C} + \overline{A}BC$

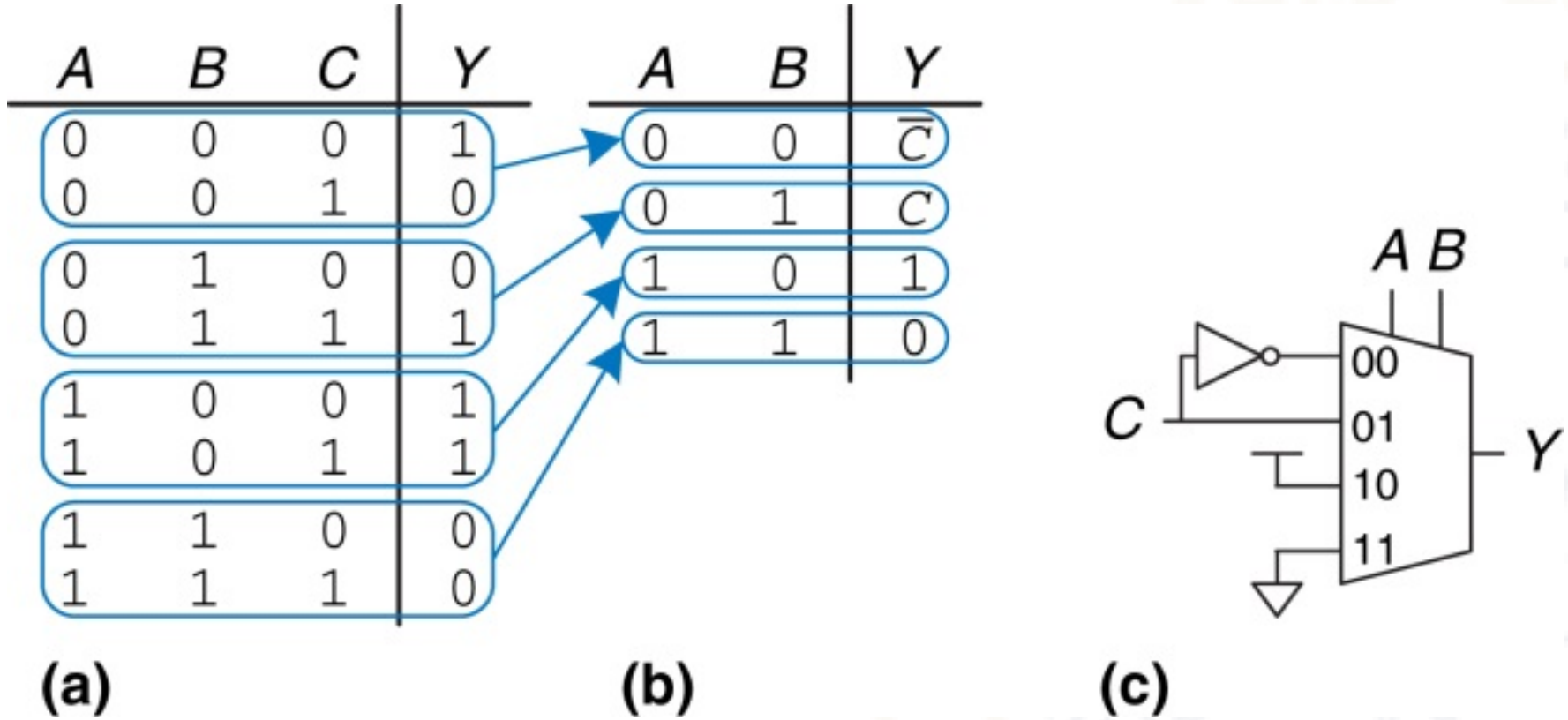
(a)



(b)

$Y = \Sigma(0,3,4,5)$

Sintetizzare funzioni booleane con mux



A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

(a)

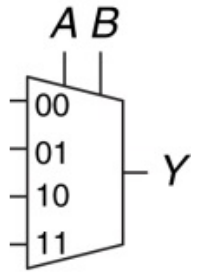
A	B	Y
0	0	$\overline{C}$
0	1	C
1	0	1
1	1	0

(b)

(c)

Sintetizzare funzioni booleane con mux

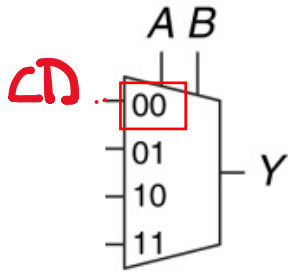
Supponiamo di avere mux 4:1 e di voler sintetizzare una funzione di 4 variabili



A B C D	Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	1
0 1 0 1	1
0 1 1 0	0
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

Sintetizzare funzioni booleane con mux

Supponiamo di avere mux 4:1 e di voler sintetizzare una funzione di 4 variabili

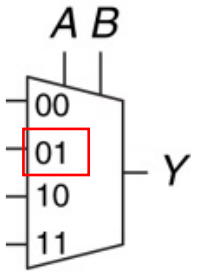


A B C D	Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	1
0 1 0 1	1
0 1 1 0	0
0 1 1 1	0
1 0 0 0	1
1 0 0 1	1
1 0 1 0	0
1 0 1 1	0
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

CD

Sintetizzare funzioni booleane con mux

Supponiamo di avere mux 4:1 e di voler sintetizzare una funzione di 4 variabili

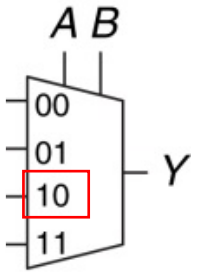


A B C D	Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	1
0 1 0 1	1
0 1 1 0	0
0 1 1 1	0
1 0 0 0	1
1 0 0 1	1
1 0 1 0	0
1 0 1 1	0
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

$\bar{C}$

Sintetizzare funzioni booleane con mux

Supponiamo di avere mux 4:1 e di voler sintetizzare una funzione di 4 variabili

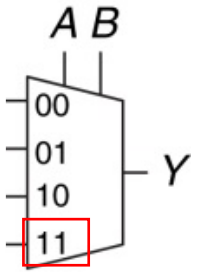


A B C D	Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	1
0 1 0 1	1
0 1 1 0	0
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

$C \oplus D$

Sintetizzare funzioni booleane con mux

Supponiamo di avere mux 4:1 e di voler sintetizzare una funzione di 4 variabili

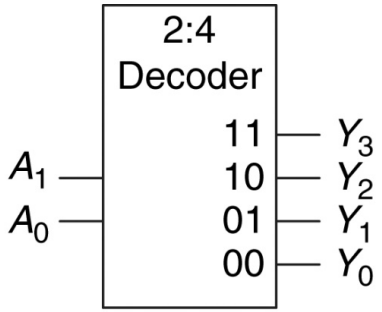


A B C D	Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	1
0 1 0 1	1
0 1 1 0	0
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

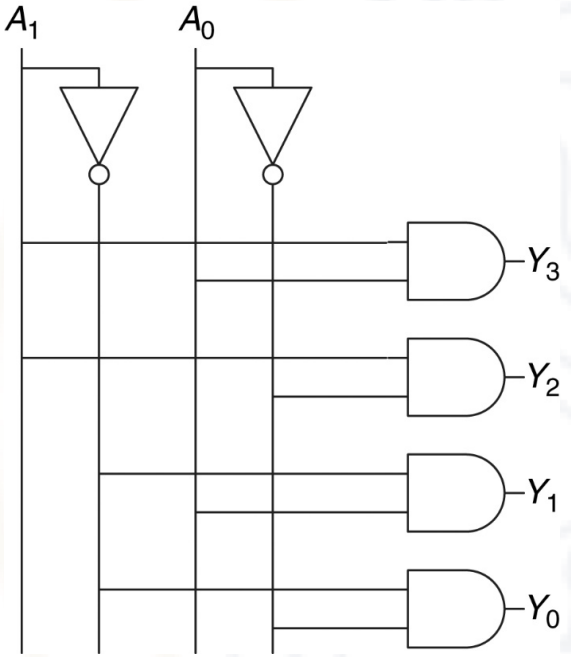
1

Decoder

- Un decoder ha N linee di ingresso e 2^N linee di uscita
- se m è numero rappresentato dagli input allora solo l'm-esima linea di uscita è pari a 1 mentre tutte le altre sono a 0

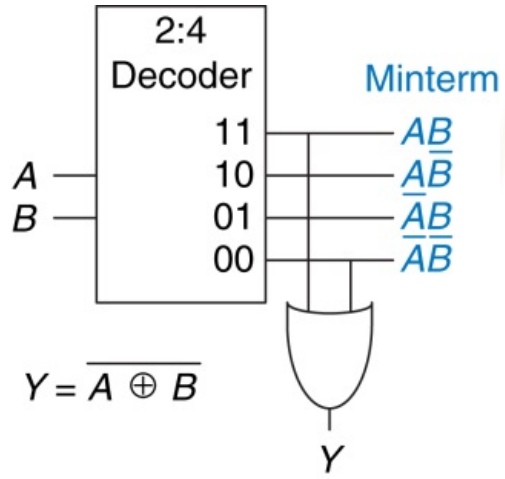


A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Sintetizzare funzioni booleane con decoder

- Anche i decoder possono essere usati per sintetizzare funzioni booleane
- Basta mettere in OR tutte e solo le linee di uscita che occorrono nella sigma-espressione della funzione da sintetizzare

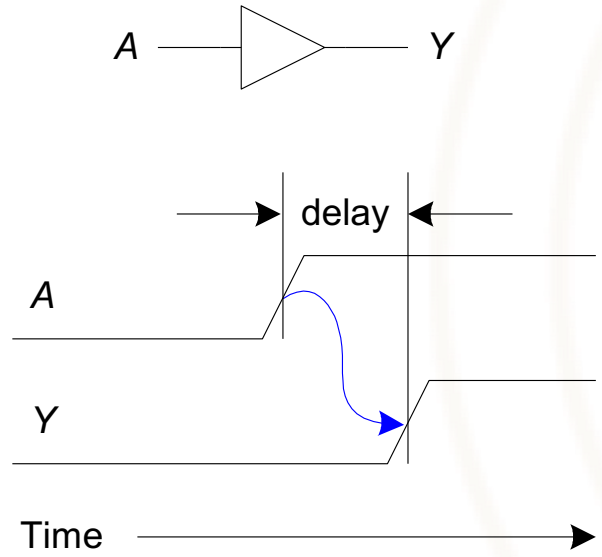


Y= Σ(0, 3)

Timing

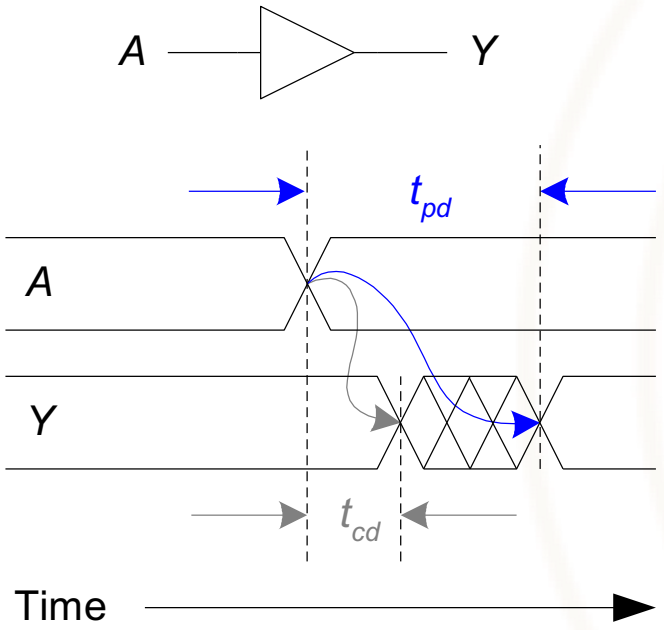
Timing

- **Delay** (Ritardo): tempo tra la modifica dell'ingresso e la modifica dell'uscita
- Come costruire circuiti veloci?



Propagation & Contamination Delay

- **Propagation delay:** t_{pd} = Massimo ritardo dall'input all'output
- **Contamination delay:** t_{cd} = minimo ritardo dall'input all'output



Propagation & Contamination Delay

Il ritardo è causato da

- Capacità e resistenza in un circuito
- Limitazione della velocità della luce

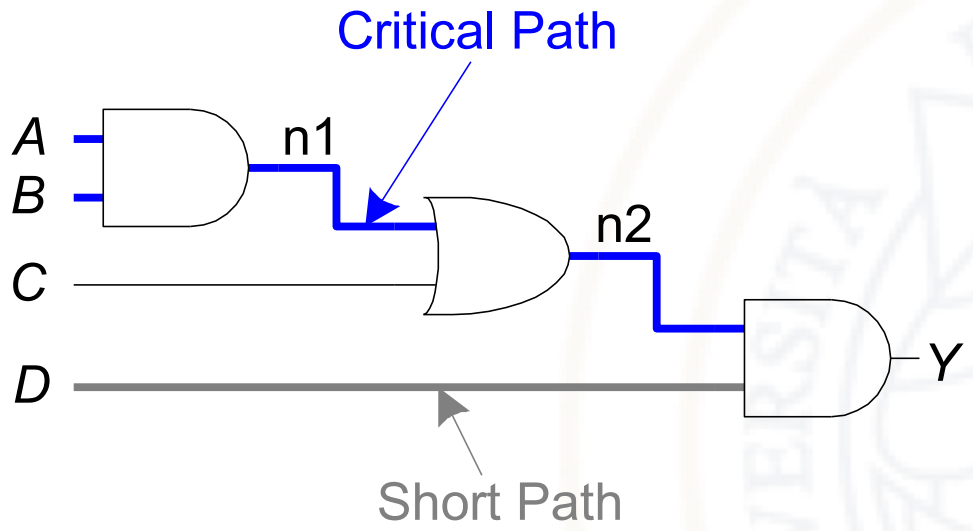
Motivi per cui t_{pd} e t_{cd} possono essere diversi:

- Diversi ritardi di salita e discesa
- Ingressi e uscite multipli, alcuni dei quali sono più veloci di altri
- I circuiti rallentano quando sono caldi e accelerano quando sono freddi

Critical (Long) & Short Paths

I ritardi possono anche essere determinati dal percorso (*path*) che il segnale percorre da un input ad un output.

Es.: circuito logico a 4-nput

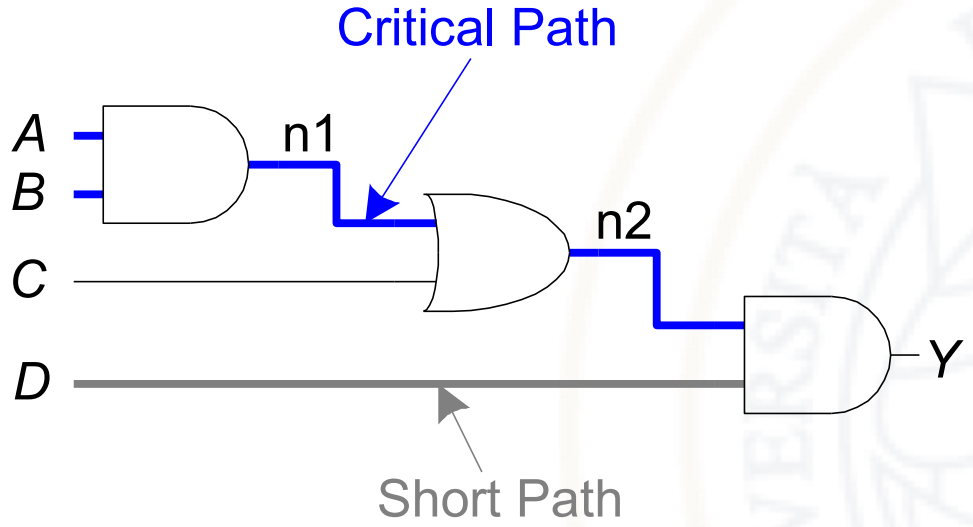


Critical (Long) Path

Short Path

Critical (Long) & Short Paths

- Il ritardo di propagazione di un circuito combinatorio è la somma dei ritardi di propagazione attraverso ogni element del Critical path
- Il ritardo di contaminazione è la somma dei ritardi di contaminazione attraverso ogni element del percorso piu' breve



Critical (Long) Path: $t_{pd} = 2t_{pd\_AND} + t_{pd\_OR}$

Short Path: $t_{cd} = t_{cd\_AND}$ → the *faster*

Exercise:

- Book Digital Design and Computer Architecture ARM Edition
- Pag.90 Ex.2.15

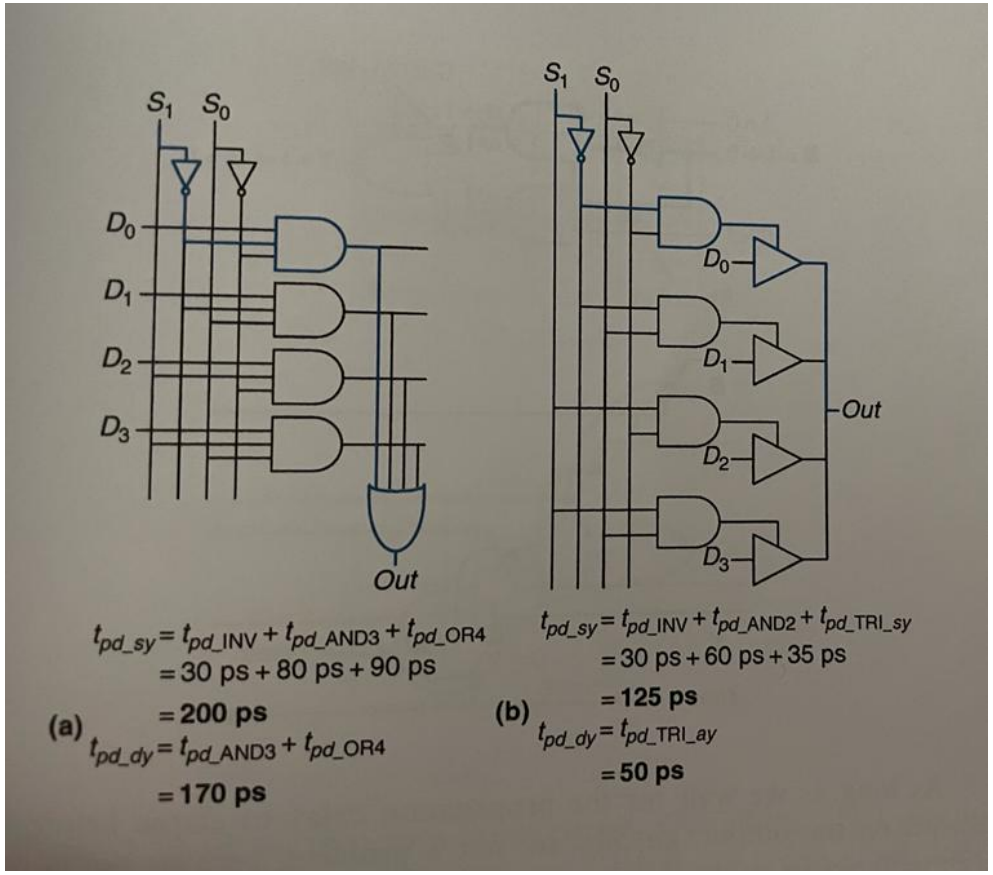


Table 2.7 Timing specifications for multiplexer circuit elements

Gate	t_{pd} (ps)
NOT	30
2-input AND	60
3-input AND	80
4-input OR	90
tristate (A to Y)	50
tristate (enable to Y)	35

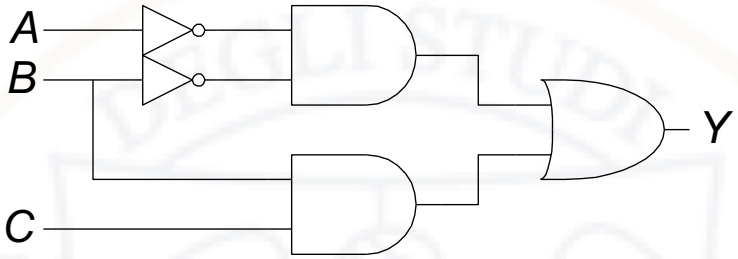
Glitches

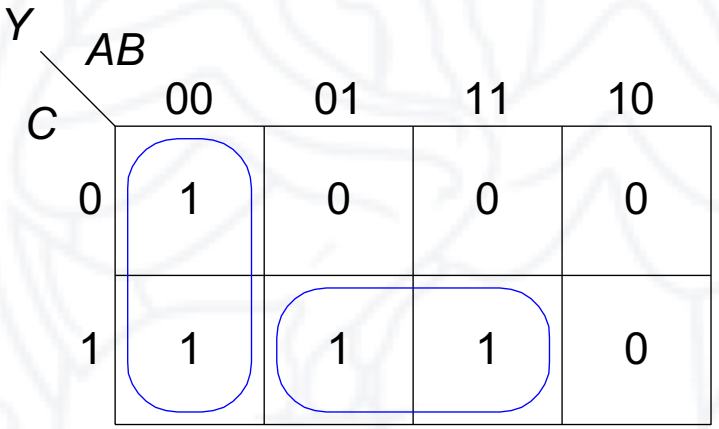
- Fino ad ora abbiamo discusso il caso in cui la transizione di un singolo input causa una transizione di stato di un singolo output
- Quando una modifica di un singolo input provoca la modifica di output multipli si parla di **Glitch**

Anche se i glitch non causano problem, è opportune considerare le influenze che hanno sul **timing**

Glitch: Example

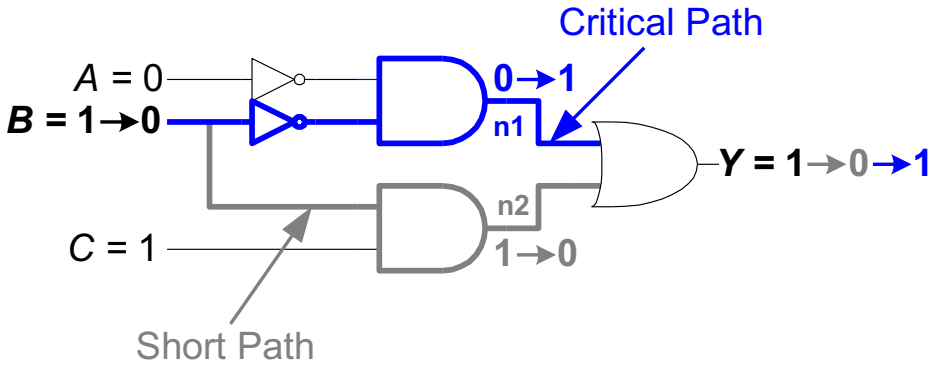
- Cosa Succcede quando A = 0, C = 1e B falls (cioè passa da 1 a 0?
- L'equazione è correttamente minimizzata:
- Y = !A!B+BC

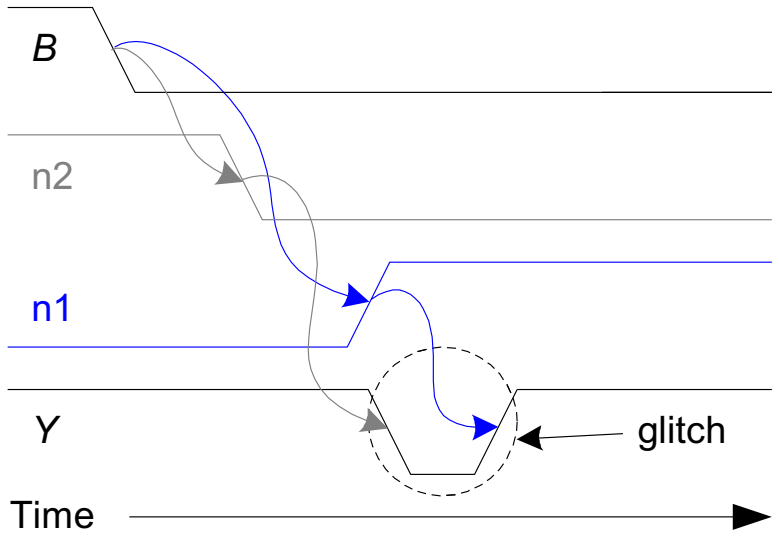




$Y = \bar{A}\bar{B} + BC$

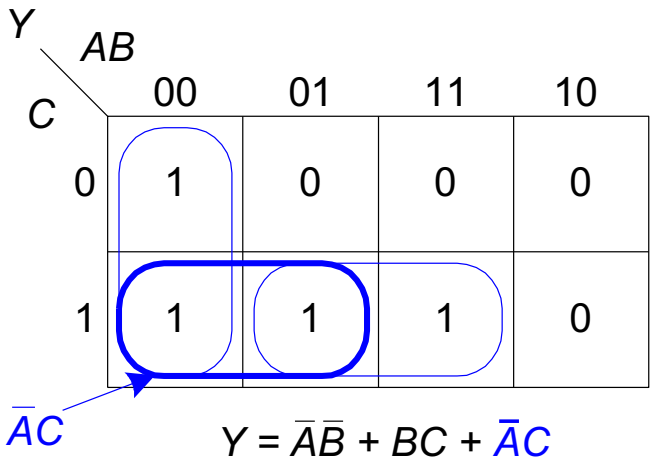
Glitch Example (cont.)

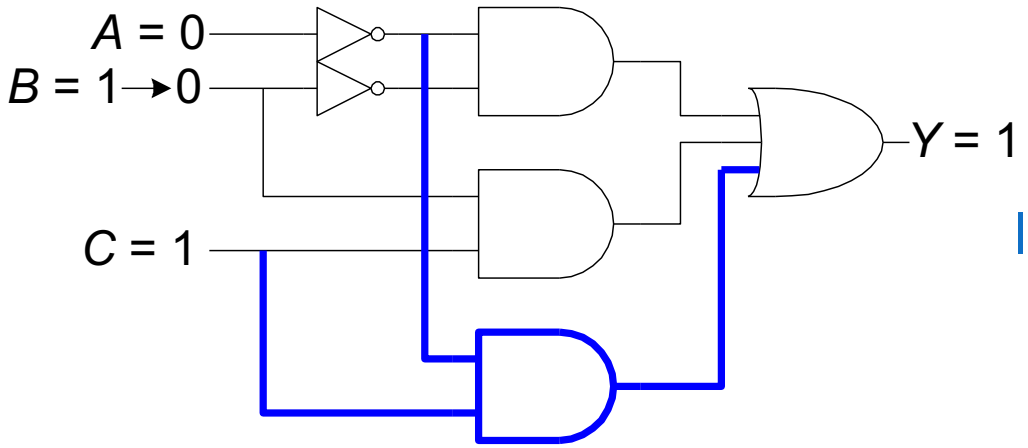




- Cosa Succede quando A = 0, C = 1, B falls
- Lo short path passa attraverso 2 porte (And e OR)
- Il critical Path passa attravero un inverter (NOT) e 2 porte (AND e OR)
- Quando B decade l'output n2 viene generato prima di n1
- Finchè n1 non diventa = "1" l'input alla porta OR sono "2"
- E quindi l'output Y passa da "0" prima di diventare "1"

Fixing the Glitch





- Si aggiunge un cerchio che copre gli implicanti primi.
- Lo si può vedere come il teorema del consenso o termine ridondante.

T11: BC+!BD+CD = BC+!BD

T11': (B+C)(!B+D)(C+D) =(B+C)(!B+D)

In questo caso il decremento di B quando A =0 e C=1non causa glitch perchè la porta AND restituisce 1 attraverso la transizione