Architetture dei Calcolatori

Introduzione alle Reti Logiche

Prof. Francesco Lo Presti

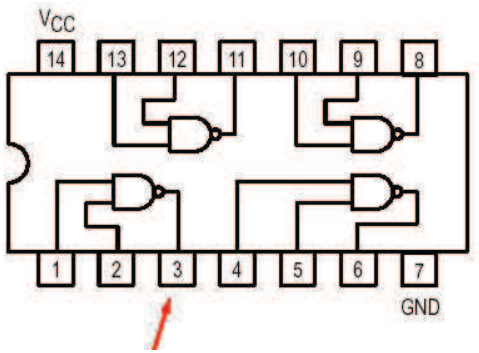
Funzioni Riusabili

- Quando possible, realizzare un sistema complesso decomponendolo in un insieme di blocchi funzionali riusabili
- Questi blocchi funzionali sono
  - Verificati e ben documentati

Funzioni e Blocchi Funzionali

- Le funzioni implementate sono quelli ricorrenti/utili nella progettazione
- Il circuito logico che implementa una data funzione è detto blocco funzionale
- Nel passato (fino agli anni '70) i blocchi funzionali erano realizzati come circuiti integrati a piccola scala (small-scale-integrated (SSI)), media scala (medium-scale integrated (MSI), e larga scala (large-scale-integrated (LSI)).
- Di seguito sono spesso semplicemente realizzati all'interno di circuiti VLSI (very-large-scale-integrated) o ULSI (ultra-large-scale-integrated)

Circuito Reale: Esempio SSI



Pin spacing is 0.1" x 0.3"; chip is about 15mm long

- 74LS00 – 4 porte NAND a due ingressi
- Small scale integration (SSI)

Circuiti Integrati

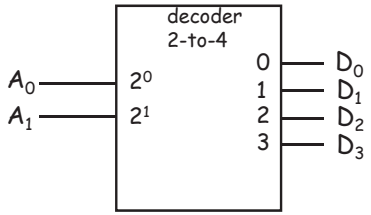
- Scala di integrazione

- (Small) SSI: 1-10 gates
- (Medium) MSI: 10-500 gates
- (Large) LSI: 500-20.000 gates
- (Very Large) VLSI: 20.000-1.000.000 gates
- (Ultra Large) ULSI: >1.000.000

Decoder (n-to-2^n)

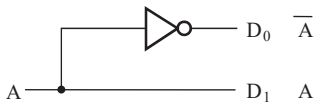
- Converte *n* ingressi in una delle 2^n uscite, i.e., si attiva l'uscita i-esima corrispondente all'intero i che corrisponde alla alla configurazione degli n ingressi.

Un esempio



Esempi di Decodificatore

A	D_0	D_1
0	1	0
1	0	1

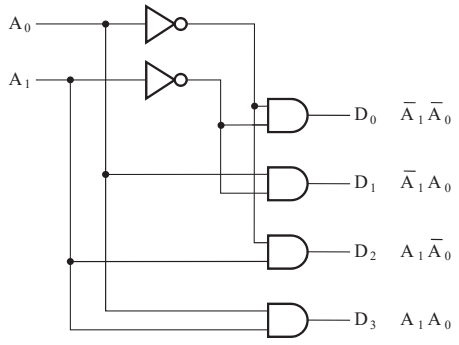


Decoder 1-to-2

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

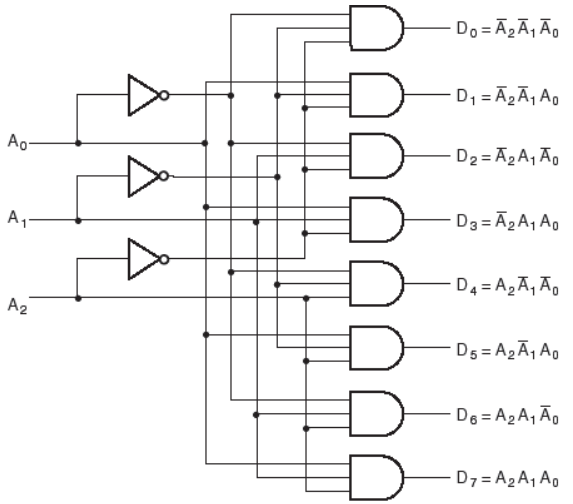
(a)

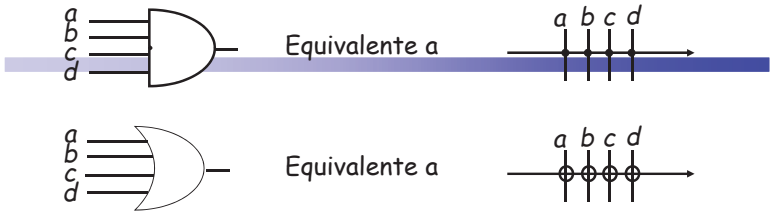
Decoder 2-to-4

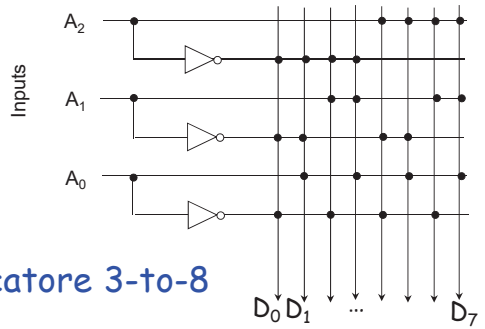


(b)

Decoder 3-to-8





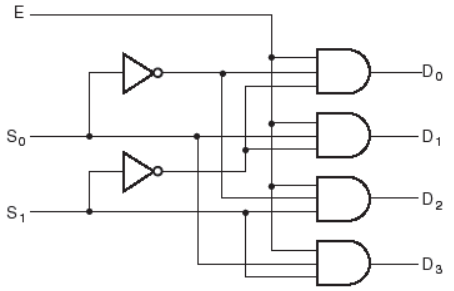


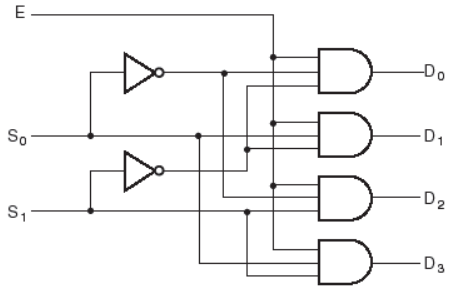
Decodificatore 3-to-8

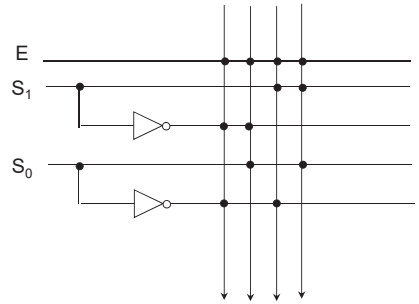
Decoder con ingresso Enable

- Uscita del Decoder è 0 se ingresso Enable=0
- Tabella di verità
  - Notare l'uso delle X per denotare 0/1

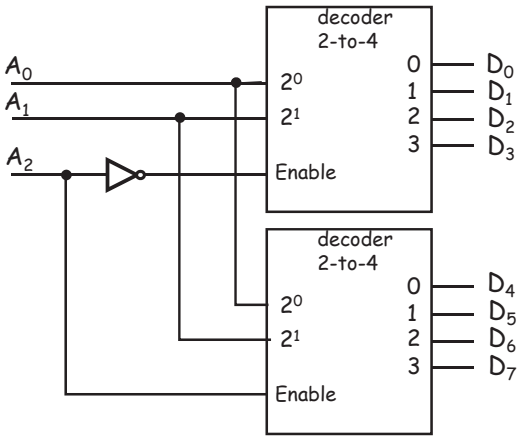
EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1







Decoder 3-to-8 realizzato tramite due decoder 2-to-4 con enable



Implementazione di funzione booleana tramite Decoder e porte OR

- Implementazione di *m* funzioni di n variabili con:
  - 1 decodificatore *n*-to-2^n
  - *m* porte OR, una per ogni uscita
- Approccio 1: Per ogni funzione in uscita
  - Determina i mintermini
  - Combina in OR i mintermini
- Approccio 2: Per ogni funzione in uscita
  - Determina la tabella di verità
  - Combina in OR le uscite del decoder che corrispondono ai alle combinazione degli ingressi per cui la funzione assume valore 1

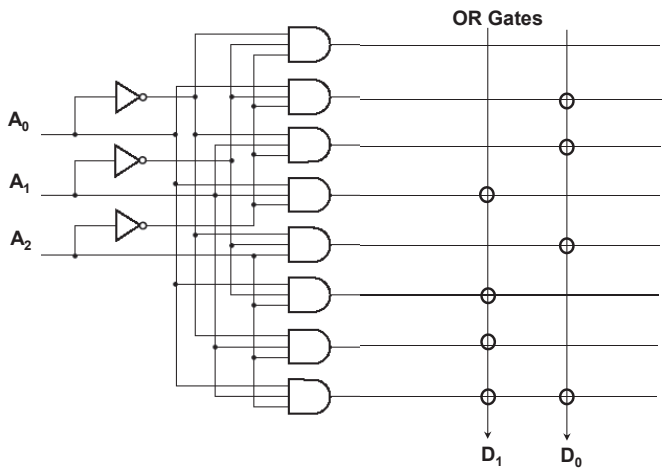
Esercizio

- Usando un decoder e porte OR, realizzare la seguente rete combinatoria
  - INPUT: 3 variabili booleane
  - OUTPUT: il numero di 1 ingresso (codificato in binario)

Soluzione: Tabella di Verità

A_0	A_1	A_2	D_1	D_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Soluzione: Tabella di Verità



Selezione

- Un Circuito che effettula la selezione è caratterizzato da:
  - Un insieme di linee in ingresso
  - Una singola linea in uscita
  - Un insieme di line di controllo per selezionare una delle linee ingresso
    - ✓ L'uscita sarà uguale al valore della linea in ingress selezionata
- I ciruiti logici che effettuano la selezione di uno degli ingress sono detti multiplexers

Multiplexer (Mux) 2^n-to-1

- 2^n ingressi -- 1 uscita
- *n* linee di controllo, per selezionare l'ingresso da "inviare" in uscita

Esempio: 4-to-1 mux

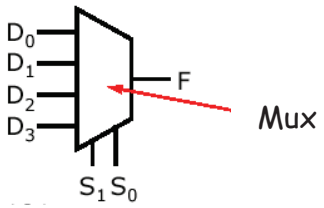
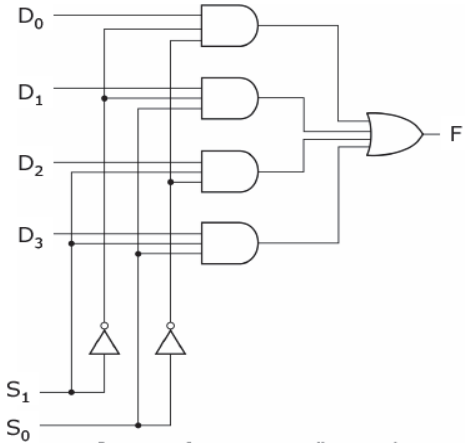


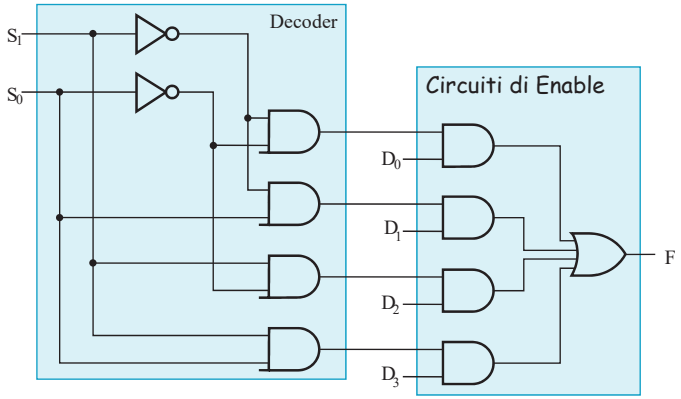
Tabella di Verita

S_1	S_0	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Circuito logico di un Multiplexer 4-to-1



Circuito logico di un Multiplexer 4-to-1 con Decoder



De-multiplexer (Demux)

- 1 ingresso -- 2^n uscite --
- *n* controlli, che selezionano quale uscita va posta uguale all'ingresso (tutte le altre a 0)

  Esempio: 1-to-4 demux

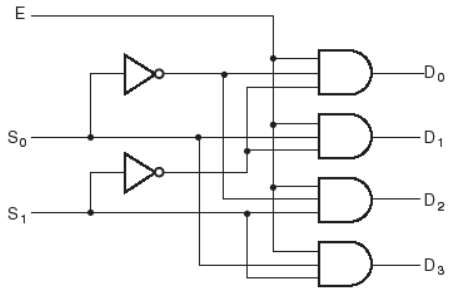
  ingresso: E, controlli: S_0 , S_1

  uscite: D_0 , D_1 , D_2 , D_3

S_0	S_1	D_0	D_1	D_2	D_3
0	0	E			
1	0		E		
0	1			E	
1	1				E

Tabella di Verità

Circuito logico di un Demux a 1-to-4



E' uguale al circuito di un decoder con enable

Memorie a Sola Lettura
Read Only Memories (ROMs)

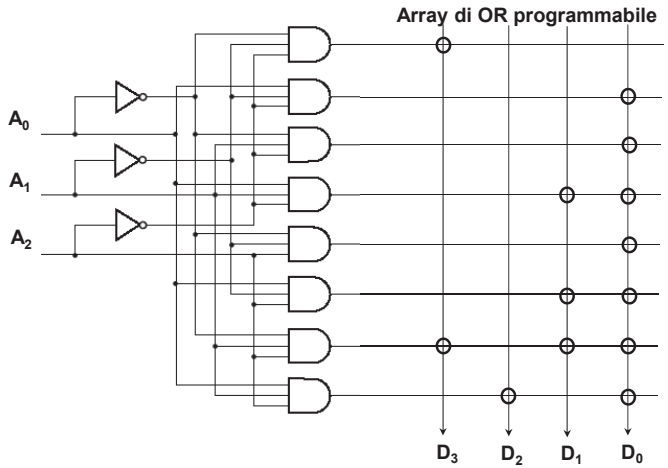
- Si consideri una rete combinatoria con n ingressi e m uscite!
- Può essere vista come una memoria: gli ingressi corrispondono agli indirizzi e i dati letti alle uscite
- Le informazioni sono "memorizzate" nel circuito
- 2^n parole di *m* bit

A_0	A_1	A_2	D_3	D_2	D_1	D_0
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	0	1
0	1	1	0	0	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	1
1	1	0	1	0	1	1
1	1	1	0	1	0	1

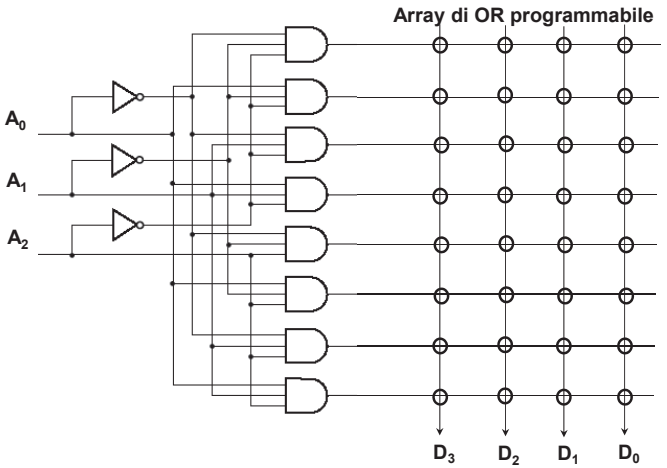
ROM: il circuito

- Le ROM sono realizzate con un decoder seguito da un modulo che combina in OR i mintermini che realizzano la funzione desiderata (matrice di OR)
- Primo Modulo: un Decoder
  - Array fisso di AND con 2^n uscite che implementa tutti i possibili mintermini
- Secondo Modulo:
  - Array di OR "programmibile" con m linee di uscta the realizza la somma logica dei mintermini necessari

ROM: implementazione



...una ROM non ancora programmata...



Programmable Logic Array (PLA)

- Una PLA per la realizzazione di somme di prodotti consiste di
  - Un modulo che combina gli ingressi in AND (array programmabile di porte AND), seguito da
  - Un secondo modulo che combina i produtti in OR per realizzare le funzioni desiderate (array programmabile di OR
- Una PLA con decoder come primo modulo corrisponde ad una ROM

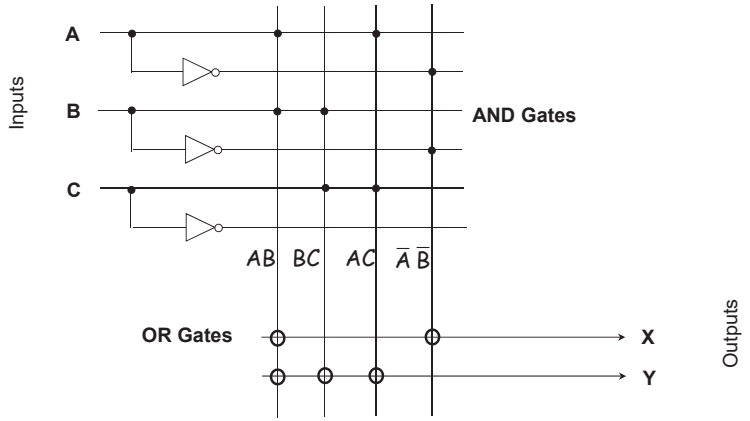
Esempio

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

$X = AB + \overline{A}\,\overline{B}$

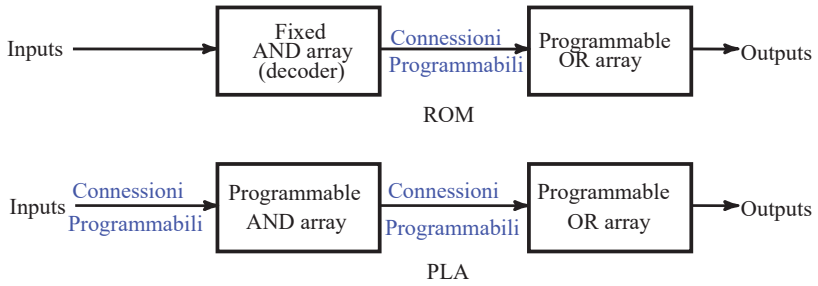
$Y = AB + BC + AC$

The PLA implementation



Configurazioni Programmabili

- ROM: un array fisso di porte AND e un array programmibile di porte OR
- PLA: un array programmbile di porte AND e un array programmibile di porte OR



Somma Binaria

Riporti	1	0	1	1	0	0	
Addendo-1		1	0	1	1	0	+
Addendo-2		1	0	1	1	1	=
Sum	1	0	1	1	0	1	

Blocchi Funzionali: Addizionatore

- Blocchi Funzionali:
  - *Half-Adder* (HA), addizionatore a 2 input
  - *Full-Adder* (FA), addizionatore a 3 input
  - *Ripple Carry Adder*, un circuito per realizzare la somma binaria
  - *Carry-Look-Ahead Adder* (CLA), un addizionatore con struttura gerarchica per migliorare le prestazioni

Half-Adder (Semi Addizionatore)

- Un circuito a 2 ingressi, 2 uscite che realizza i seguenti calcoli:

X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

- Il semi addizionatore somma 2 bit e restituisce in uscita il risultato della somma
- Il risultato è espresso come il bit di somma S e il bit di riporto (carry) C

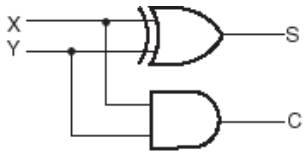
Half-Adder (Semi Addizionatore)

- Somma di due cifre binarie (senza riporto in ingresso)

Tabella di Verità

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Circuito Logico



Full-Adder (Addizionatore Completo)

- Addizionatore Completo è simile al semi addizionatore, ma include un bit di riporto in ingress (carry-in) dagli stadi inferiori. Come il semi-addizionatore calcola il bit di somma e il bit di riporto.
  - Se il carry-in (Z) è pari a 0, si comporta come il semi-addizionatore:

Z	0	0	0	0
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

  - Se il carry- in (Z) è 1:

Z	1	1	1	1
X	0	0	1	1
+ Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

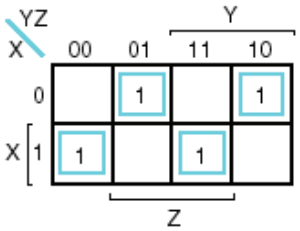
Full-adder

- Z rappresenta il carry in

Tabella di Verità

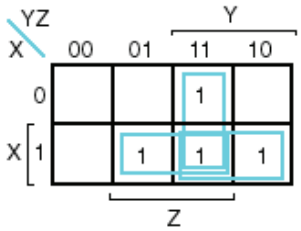
X	Y	Z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Mappa di Karnaugh per il Full Adder



$S = \overline{X}\,\overline{Y}\,Z + \overline{X}\,Y\,\overline{Z} + X\,\overline{Y}\,\overline{Z} + XYZ$

$= X \oplus Y \oplus Z$



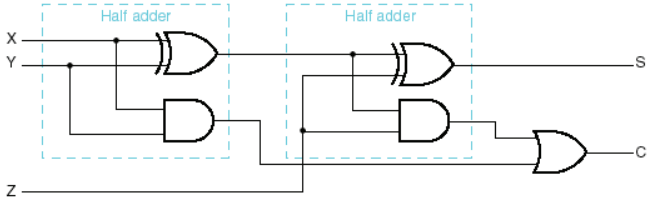
$C = XY + XZ + YZ$

$= XY + X\overline{Y}Z + \overline{X}YZ$

$= XY + Z\cdot(X\overline{Y} + \overline{X}Y)$

$= XY + Z\cdot(X \oplus Y)$

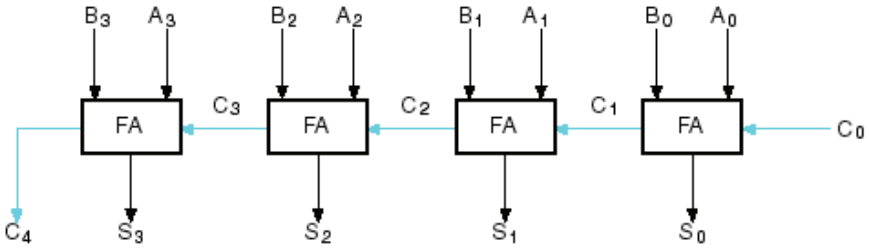
Circuito logico di un Full Adder



Addizionatore Binario

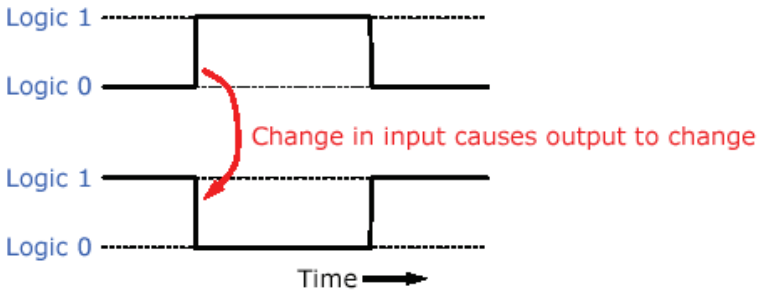
- Deve permettere la somma tra due numeri binary di n bit
- L' addizionatore binario a *n*-bit si può realizzare collegando tra loro n full adder

  Schema logico addizionatore **ripple-carry**



…Comportamento ideale dei circuiti

- Consideriamo un inverter (porta NOT)

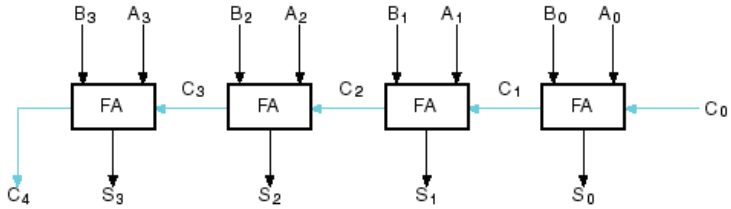


…e comportamento reale

- Ritardo di propagzione: tempo che intercorre tra una modifica in ingresso e la corrispondnete modifica in uscita (**ritardo di porta**)
- Tempo di discesa (Fall time) : tempo richiesto a che il segnale in uscita scenda dal livello alto a quello basso
- Tempo di salita (Rise time) : tempo richiesto per una commutazione dal livello basso a quello alto

Propagazione del Riporto

- I segnali devono propagarsi dagli ingressi alle uscite per essere validi
- Le uscite S e C di un singolo full adder sono validi dopo un tempo t_{prop} di ritardo di porta dopo che i valori in ingressi sono stabili
- t_{prop} dipende dalla tecnologia impiegata (ordine dei ns)
- In un addizionatore binario a n bit l'ultimo riporto è valido dopo $t_{prop} \cdot n$ dopo che gli input sono stabili
- Per valori grandi di n questo ritardo può diventare inaccettabile



Addizionatore "Carry look-ahead"

- Ogni riporto è funzione degli ingressi A e B
  - Ogni riporto può essere quindi calcolato con un circuito a due livelli
- Idea: pre-calcolare tutti i riporti usando un circuito logico a due livelli

Soluzione

- Scivere un espressione generale per il riporto
  - Quando si genera il riporto?
  - Quando un riporto in ingresso si propaga in uscita?

a_i	b_i	c_i	s_i	c_{i+1}
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

Espressione Generale

- Espressione generale per l' (i+1)-esimo riporto
  - $c_{i+1} = a_i b_i + c_i (a_i + b_i) = g_i + c_i p_i$
  - $g_i = a_i b_i \rightarrow$ genera il riporto
  - $p_i = a_i + b_i \rightarrow$ propaga il riporto
- Iteriamo l'espressione per c_i

Espressione Generale (2)

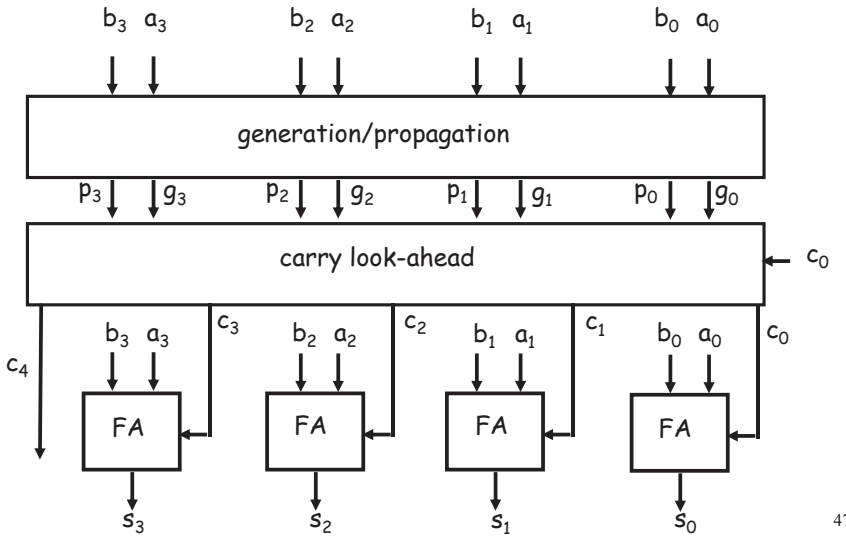
$$
\begin{aligned}
c_{i+1} &= g_i + p_i c_i \\
&= g_i + p_i(g_{i-1}+c_{i-1}p_{i-1}) = g_i+p_i g_{i-1}+p_i p_{i-1} c_{i-1} = \\
&= g_i+p_i g_{i-1}+p_i p_{i-1}(g_{i-2}+c_{i-2}p_{i-2}) \\
&= g_i+p_i g_{i-1}+p_i p_{i-1} g_{i-2}+p_i p_{i-1} p_{i-2} c_{i-2} \\
&= g_i+p_i g_{i-1}+p_i p_{i-1} g_{i-2}+p_i p_{i-1} p_{i-2} g_{i-3}+p_i p_{i-1} p_{i-2} p_{i-3} g_{i-4}+\cdots \\
&\cdots \\
&= \cdots
\end{aligned}
$$

- c_i dipende da c_0 e da p_j, g_j per $j<i$

Espressioni per i riporti di un addizionatore a 4-bit adder

- $c_1 = g_0 + p_0 c_0$
- $c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$
- $c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$
- $c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$

Architettura dell'addizionatoe con Carry Look-Ahead



Problema pratico…

- $c_1 = g_0 + p_0 c_0$
- $c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$
- $c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$
- $c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$

C'è un limite dovuto al **fan-in** di un circuito
(numero massimo di ingressi di una porta logica)