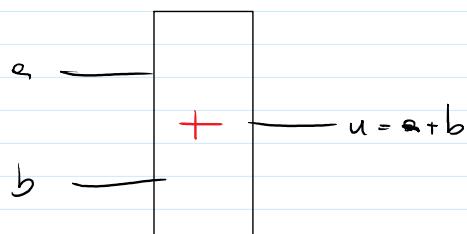


Dispositivo che permette di effettuare calcoli su rappresentazioni binarie

- SOMMATORI (Full ADDER)

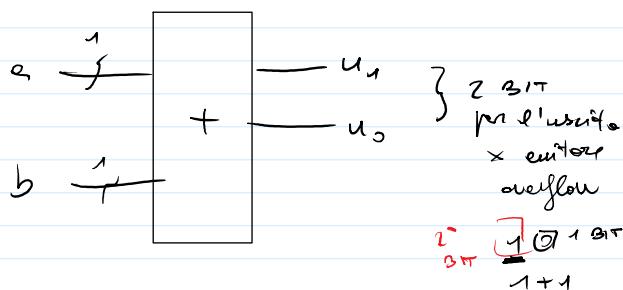


N.B. Deve tenere conto di overflow: valori in ingresso e uscita per evitare overflow

(guarda il max valore ottenibile con i punti BIT occupati)

Rappresento le somme  $a+b$ :

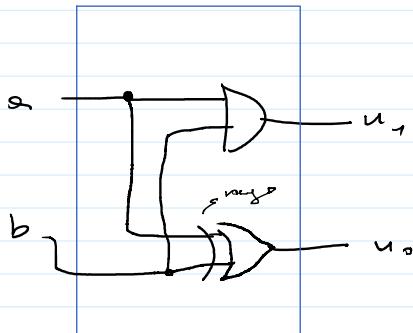
Trovare da vedere per vedere i possibili valori



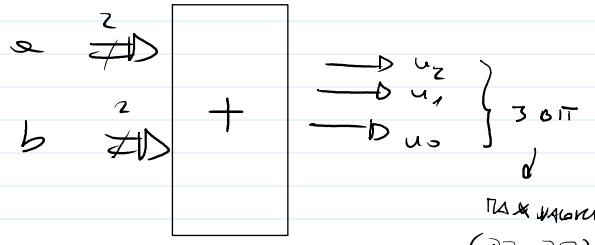
a	b	$u_1$	$u_0$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$1+1=2 \rightarrow 10$  Binary  
" AND " XOR

Forme grafiche (circuito logico)



Se questi



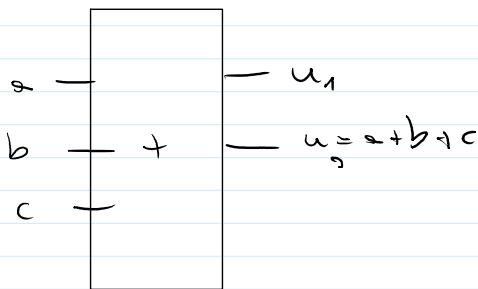
MAX NUMBER  
 $(2^2 + 2^2)$

$= 6$

$\leq 6$

$= 6$

$a_1$	$a_0$	$b_1$	$b_0$	$u_2$	$u_1$	$u_0$
0	0	0	0	0	0	0
0	1	0	1	0	0	1
1	0	0	1	0	0	1
1	1	0	0	1	1	0

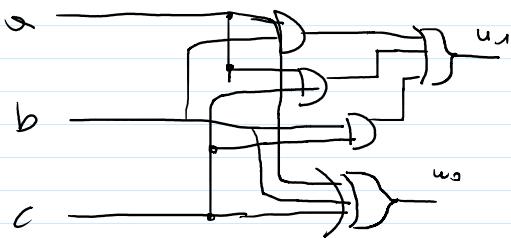


sono  
due fili  
di uscita  
(2bit)

a	b	\	c	0	1
0	0	0	0	0	1
0	1	0	1	1	0
1	1	1	0	1	1
1	0	0	1	0	1

scappigliazioni possibili  
(tre coppie di n. addossati su u1 e u2)

Rappresento le somme in  
Circuito logico:



(M1)

u1, u2

$$u_1 = a \cdot b + b \cdot c + a \cdot c$$

negli altri casi  $u_1 = 0$

un brano uno  
di questi per  
avere  $u_1 = 1$

per  $u_2$  ho un zigzag che non permette di  
avere coppia di addossati  
→ esiste una funzione che fa zigzag

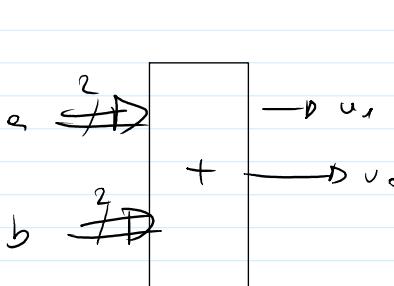
e lo XOR

$$u_2 = a \text{ XOR } b \text{ XOR } c$$

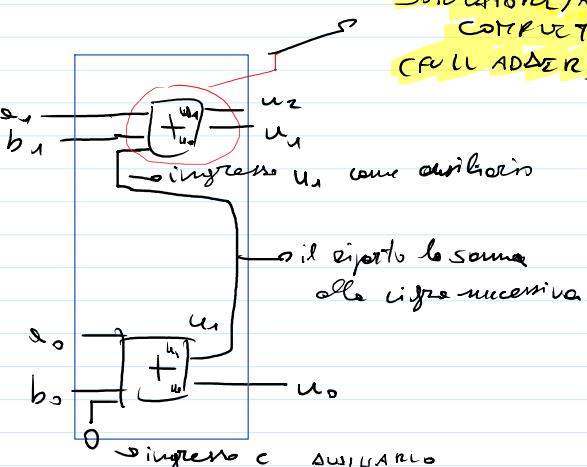
quando uno fra a, b o c è vero

gli altri due sono falsi

Per avere sommatori a più bit passo generalmente il modello:



IMPLEMENTAZIONE



Per rappresentazioni a 8 bit riplico

8 volte il singolo sommatore (full adder)

corrisponde a un "modulo"

→ la complessità del circuito cresce al crescere del numero di bit = VANTAGGIO

- + . 1. un . 1. uno solo digitato risultato 10 cifre di cui - VANTAGGIO

→ la complessità dei circuiti viene ora ridotta con numero di bit - VANTAGGIO

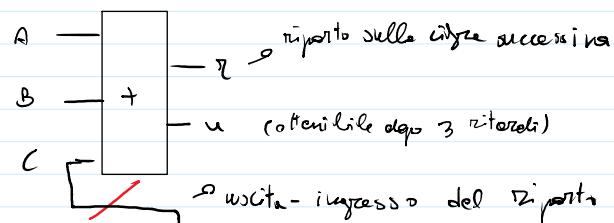
→ tempi di produzione del risultato ELEVATI (il ritardo di uscita = Svantaggio)  
del risultato varia in base alle tecnologie impiegate)

Il ritardo diventa grande, come nell'esempio, collego l'uscita di un modulo con l'ingresso di un altro modulo; il ritardo della prima uscita si somma al ritardo dell'uscita successiva (= il ritardo cresce col numero di bit impiegati)

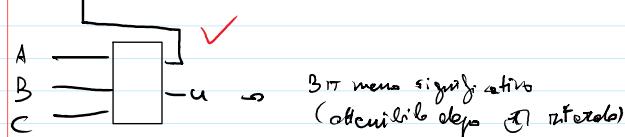
→ se sommetto a 64 bit invece metà delle velocità di calcolo di un sommetore a 32 bit (ma il 64 bit puo' permettere calcoli con bit maggiori)

Dunque bisogna scendere a "compromessi" in base alle esigenze;  
il sommetore in figura produce meno calore (= meno consumo di energia)  
MA non e' il più veloce. La velocità infatti si traduce in costi maggiori poiché i calcolatori sono più grandi e pesanti.

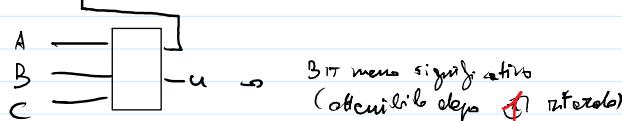
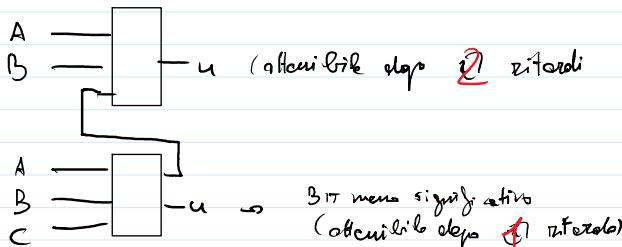
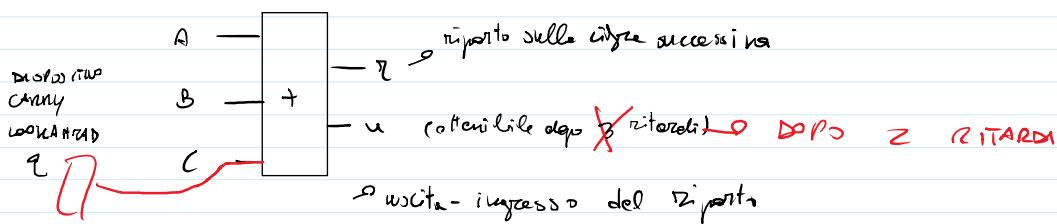
Fun ADDER



per velocizzare bisogna ottimizzare  
la trasmissione del ritardo



Ritardo calcolare



Secondo il sistema

**CARRY  
LOOKAHEAD**

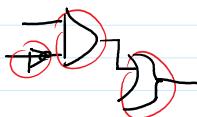
il ritardo max dell'unità  
è pari a 2 ritardi  
e non pelle totale del  
sommatore nel complesso

↓  
segue la logica  
a 3 livelli

Sulle TAV. DI VERITÀ (AND, OR, NOT)

osservo:

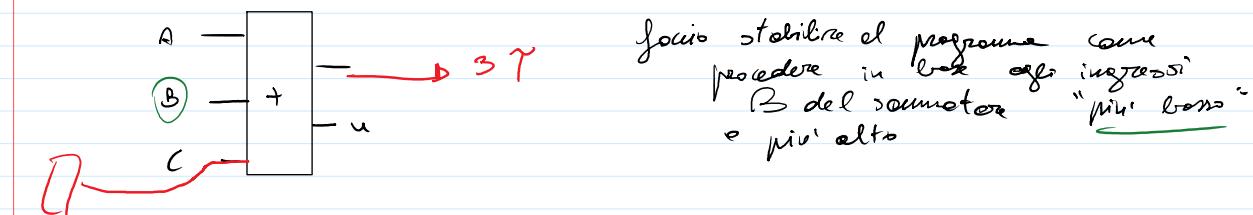
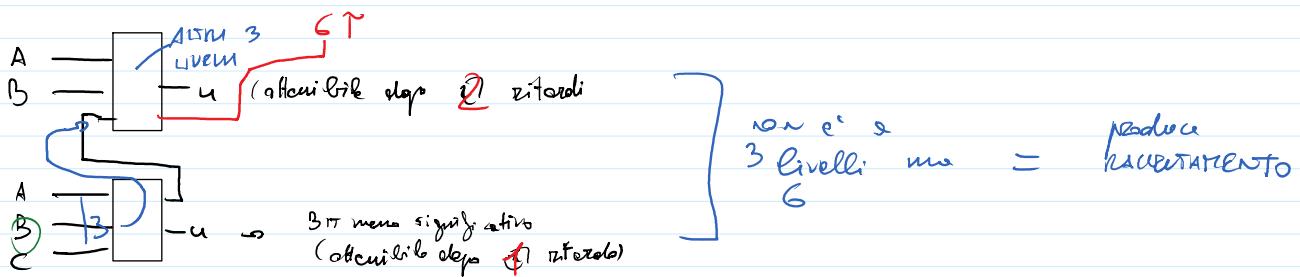
- ingressi negati = meno
- funzioni AND fra ingressi
- funzioni OR



impiego in  
tempo T (tempo)  
per calcolare

al massimo poss. incontrare  
3 operazioni elementari

Dunque il max intervallo di tempo, indipendentemente dalle funzioni usate,  
sarà 3T



RITARDO =  $\Delta t$  → modulo aggiuntivo con logica a 3 livelli (riporto da un ritardo  $3T$ , "offro" il tempo impiegato nella cefone dei riporti)

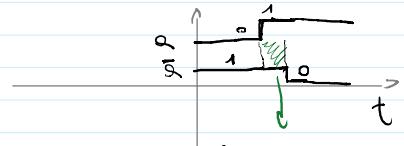
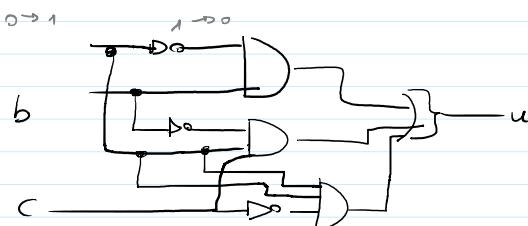
$\underline{f(a,b,c)}$

a	b	c	u	valore 1 class
0	0	0	0	
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	0	

applico proprietà algebraiche

$$f : (\bar{a} \cdot b) \bar{c} + (\bar{a} \cdot b) \cdot c + a \bar{b} \bar{c} + a \bar{b} c$$

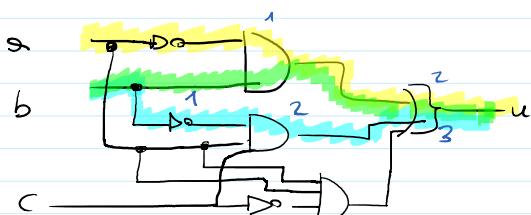
$$\Rightarrow \bar{a} \cdot b (\bar{c} + c) = \bar{a} \cdot b$$



Se io applico delle variazioni senza tener conto dei tempi del dispositivo, non vengono restituiti il valore corretto delle funzioni

il cambio di valore avviene dopo un certo  $T$

Osserviamo che:



portando da qualsiasi punto  $a, b, c$  in centro al massimo 3 operazioni.

→ LOGICA A 3 LIVELLI

e al suo interno il full adder usa tale sistema

(il vantaggio si nota a partita di 3 bit superiori)

ES

$\square \rightarrow RITARDO \quad 5T$

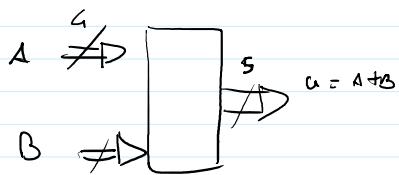
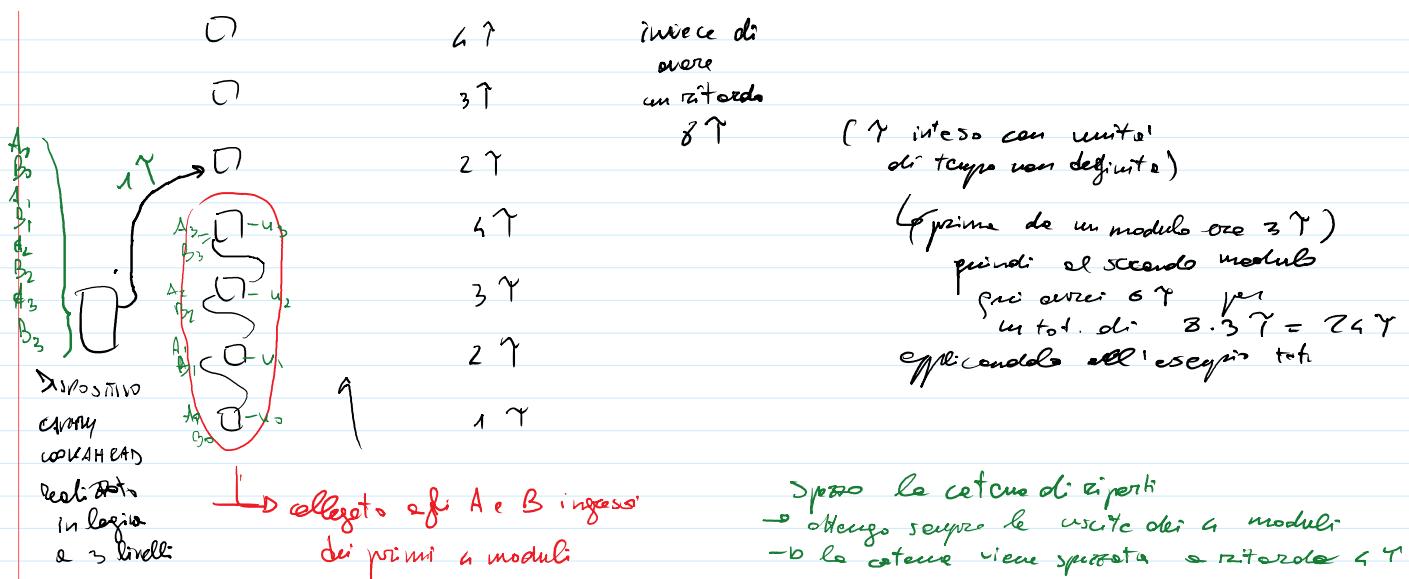
$\square$

$4T$

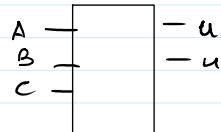
invece di avere un ritardo

$\square$

$3T$

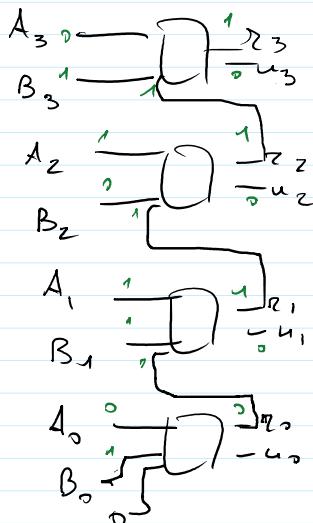


FULL ADDER



	$A_3 A_2 A_1 A_0$
A	0 1 1 0
B	1 0 1 1
$\gamma u$	0 0 0 1

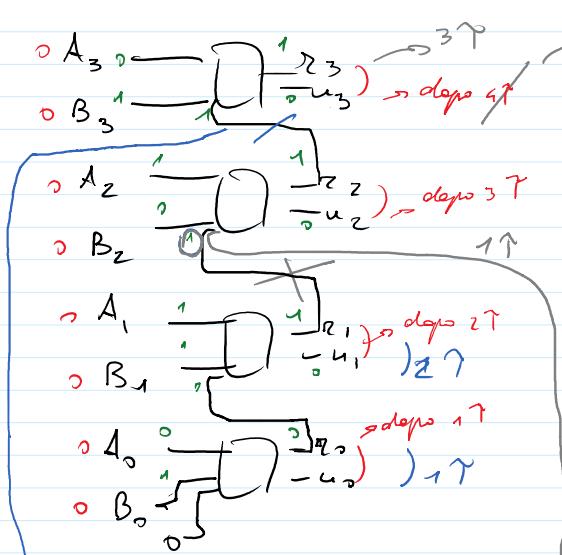
Ripato  $R_3$



Funziona bene con delle rappresentazioni fissate

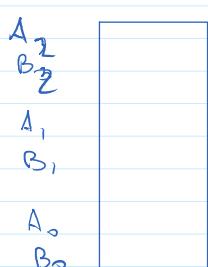
	$A_3 A_2 A_1 A_0$
A	0 1 1 0
B	1 0 1 1
$\gamma u$	0 0 0 1

Ripato  $R_3$

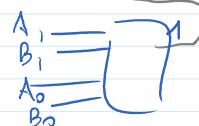


Se cambio configurazione con tutti zero, l'uscita finale giunge dopo una serie di ritardi  
 ↓  
 faccio altrettante in un tempo  $1\gamma$  il risultato (1) invece che dom  $2\gamma$

N.B. Se ciò non bastasse passo  
inserire un altro dispositivo



ottenendo al massimo  
2 ritardi.  
(2T)



produrre in un tempo 1T  
il risultato  
invece che dopo 2T

- Velocità maggiore
- = dispositivi più costosi
- (permettono un notevole risparmio)

Lo passo orzivale e ritardato  
complessivo 1T, costituendo (mappa di Karnaugh esclusa)  
una funzione che restituisce  
tutte le uscite dati tutti gli ingressi

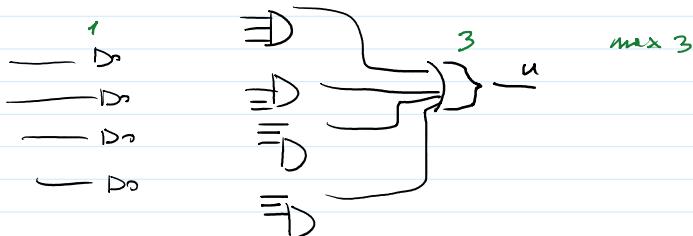
N.B. Nella logica a 3 livelli le funzioni che descrivono le uscite possono essere uguali ma anche diverse.

Rappresentare tutte le combinazioni possibili per avere  $u=1$  da f

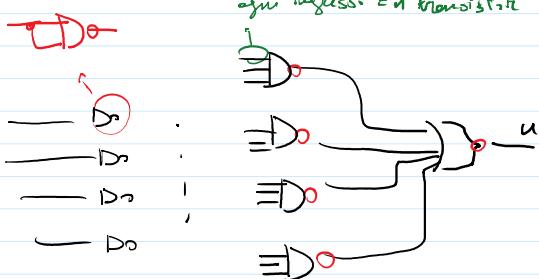
a	b	c	d	0011
0	0	0	0	0 1 1 0
0	0	1		0 0 0 0
0	1			0 1 0 1
1	1			0 0 0 0
1	0			1 1 0 0

$$a \cdot b \cdot \bar{c} + \bar{b} \cdot \bar{c} \cdot d + \bar{a} \cdot b \cdot \bar{c} \cdot d + \bar{a} \cdot b \cdot \bar{c} \cdot \bar{d}$$

non tiene conto di d



ELETTRONICA  
ogni ingresso è un transistor



La tavola di verità rimane  
INVARIATA se ussi solo NAND  
utilizzabili

Le funzioni sono realizzabili in modo alternativo secondo un principio di dualità:

- identifico celle con zero e uno
- esigo dei prodotti di somma (qui ho fatto scelta come di precedenti)
- cerco alla fine del circuiti una funzione NOR

ri cade nella forma normale congiuntiva o disgiuntiva

→ PRODOTTI DI SORRE: congiunzioni di disgiunzioni  
= AND = OR

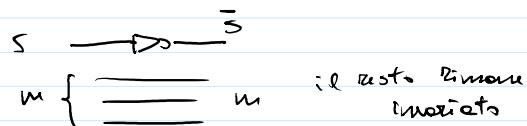
→ SORRE DI PRODOTTI: disgiunzioni di congiunzioni  
= OR = AND

La rappresentazione binaria in base al codice (esponentiale) che si vuole usare:

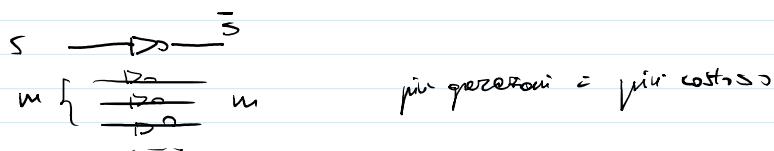
- **Modulo e segno**



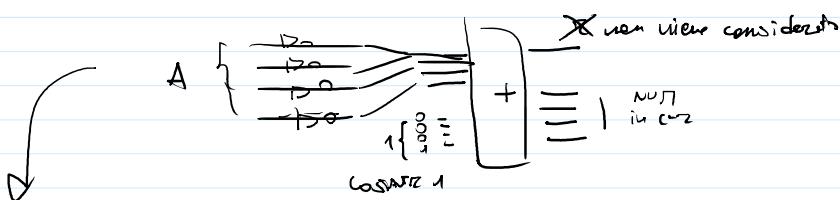
Circuito di cambio segno; bisogna invertire il bit di segno



- **COMPONENTE A 1**; inverto tutti i bit della rappres. mod e segno

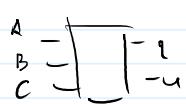


- **COMPONENTE A 2**; C-1 + sommatore a 4 bit



**Non c'è certo il modo più efficiente:**

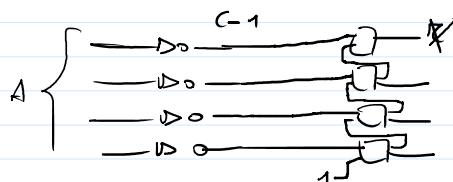
- **FULL ADDER**



(2021 - ADDIZIONATORE)  
HALF ADDER



Potrei ottimizzare con  
l'HALF-ADDER:

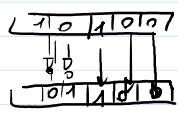


Sono gli eventuali ripetuti, ignorando  
quello finale

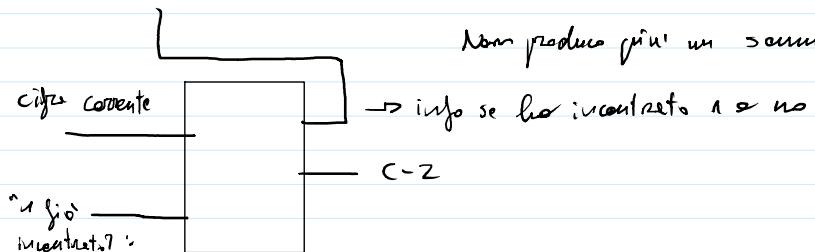
(ha lo stesso problema di ritardo di prima:  
ritardo collegato agli ingressi, ritardo  
lineare in base al numero di ingressi)

631T → ritardo 6T, 32BIT → 32T  
4 ingressi

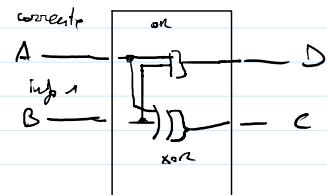
- Un algoritmo alternativo ci consente di sommare le cifre una per una, prendendo la cifra meno significativa (da dx verso sx), trasformando i bit. Se il valore della 1<sup>a</sup> cifra è zero si passa alla cifra dopo e la copia. Se trovo 1 la copio ma le cifre successive sono trasformate.



direttamente C-2, resta passare per il codice intermedio di C-1



$\begin{matrix} 0 & 1 \\ \text{no} & \text{si} \end{matrix} \rightarrow D \text{ inizializzo a zero (non ho ancora incontrato nessun uno)}$



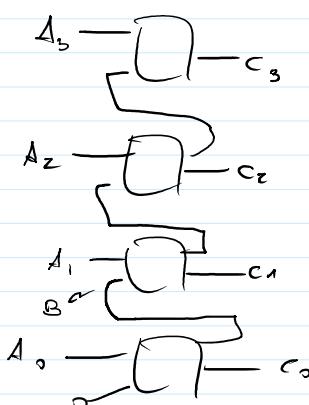
non è un sommatore, se vogliamo fare una realizzazione a 3 livelli (usando solo AND, NOT e OR) oppure in logica NAND

tenendo conto che  $x \oplus y = \bar{x} \cdot \bar{y} + \bar{x} \cdot y + x \cdot \bar{y}$

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

1 già incontrato  
 corrisponde a OR  
 non incontrato 1  
 corrisponde a XOR

Dunque in una rappresentazione a 4 bit

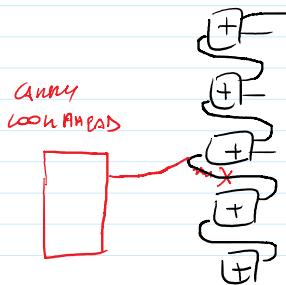


Stessa struttura della costruzione di HALF-ADDER ma i dispositivi non sono sommatori ma dispositivi con un circuito disegnato sopra.

Anche qui ho il problema dei ritardi:

Stavolta c'è di negare e sommare, ma calcolo tutto con uno XOR:

stesso elemento ha le stesse caratteristiche, inoltre aminuisce i diversi compromessi di efficienza e costi.



Come già visto, tale catena rallenta l'unità finale.

Il **carry lookahead** ci permette di ottenere il risultato da un dispositivo con ritardo ridotto.

CARRY LOOK AHEAD (2 bit per ingresso)

$$\delta(a_1, a_0, b_1, b_0)$$

$$\begin{array}{r} \begin{array}{cccc} & b_1 & 0 & 0 & 1 & 1 \\ a_1, a_0 & \diagdown & b_0 & 0 & 1 & 1 & 0 \end{array} \\ \hline \begin{array}{c} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{array} \end{array}$$

Diagram showing the calculation of the carry look-ahead function  $\delta(a_1, a_0, b_1, b_0)$  for the addition of two 3-bit numbers. The result is shown as a 4-bit binary number. The carry look-ahead terms  $a_1 \cdot b_1$ ,  $a_1 \cdot a_0 \cdot b_0$ , and  $b_1 \cdot b_0 \cdot a_0$  are highlighted with red circles and arrows pointing to them.

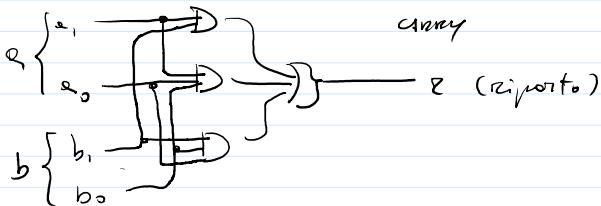
su 3 bit  
(caso, esito  
overflow)



mi dicono quando il  
riporto va a occupare la  $z^3$   
posizione  
(cioè per le 3° cifre per:  
valori 4, 5, 6 ( $n > 3$ ))

Procedo con la risoluzione minima

$$a_1 \cdot b_1 + a_1 \cdot a_0 \cdot b_0 + b_1 \cdot b_0 \cdot a_0$$



Per questo dispositivo ho una realizzazione su 2 livelli di logica (2 operazioni AND).

Il livello "mentre" sarebbe quello delle regestrie, che per ora vengono ingegnate nel circuito.

2 livelli = più veloce di un dispositivo a 3 livelli

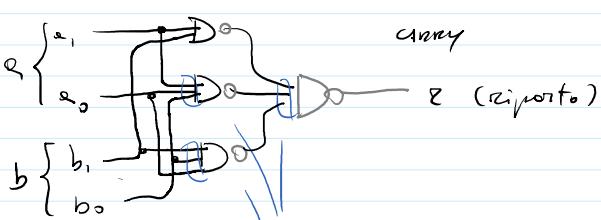
2E

2E RITARDO

Logici NAND (usando le leggi di De Morgan)

$$a_1 \cdot b_1 + a_1 \cdot a_0 \cdot b_0 + b_1 \cdot b_0 \cdot a_0$$

2 livelli di logica  
(2 NAND)



più ingressi = più consumi

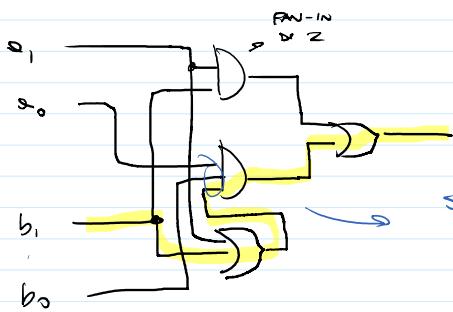
Stesso risultato procedendo  
ma col NAND posso semplificare;  
tolgo le differenze fra OR e AND.  
La registrazione pose una difficoltà  
in meno rispetto alle operazioni  
positive (costo meno ed è più veloce)

(ALTERNATIVA)

Allora posso trasformare la mia espressione algebrica, ricorrendole rispetto alla forma normale:

$$a_1 \cdot b_1 \rightarrow (a_0 \cdot b_0) \cdot (a_1 + b_1)$$

non è più una forma pura di somme di prodotti.



Voglio minimizzare il FAN-IN (numero di ingressi) di ogni funzione

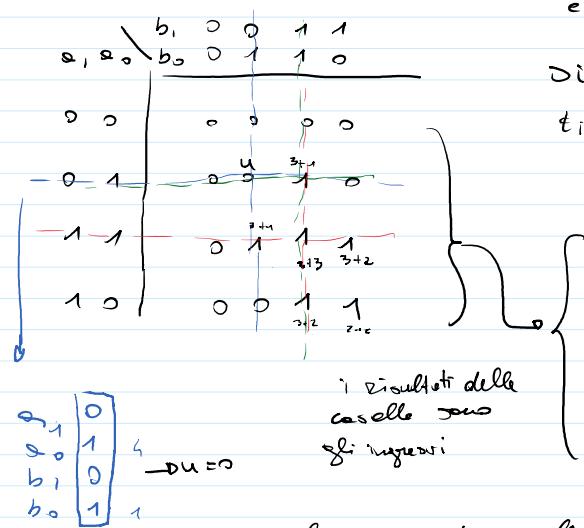
Rispetto alla f. normale ottengo lo vantaggio di avere 3 livelli di logica (osservando solo da b1)

Lo si osserva il max (1°) di funzioni elementari incontrabili per arrivare all'uscita partendo dagli ingressi.

- Se uso la forma normale so che il livello di logica è  $\leq 3$ , (al massimo NOT, AND, OR)

- Con un metodo diverso non posso obiettare e più quant'è il limite del livello di logica

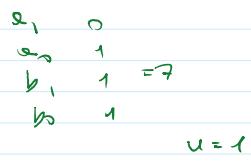
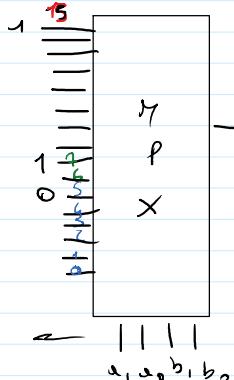
E' possibile trovare una componente generica che mi permette di realizzare una qualunque funzione e i suoi ingressi, senza notazione algebrica e risoluzione minima?



Le interpretiamo come un numero ( $= 5$ )

gli ingressi di controllo li imposto con gli ingressi principali.

Di, si può usare un modulo generico di tipo scrittore con 16 ingressi ( $2^4$ )



e così via ricopri tutti gli ingressi dell'MPX

Potrei realizzare qualsiasi funzione, basta comandare le contacti 1 e 0, in ingresso.

Potrei inoltre realizzare dispositivi programmabili (avendo comandi di programma posso aggiornare le costanti)

Ad esempio usando degli interruttori posso scegliere se l'ingresso  $a_0 = 0$  o  $1$ , potendo definire qualsiasi funzione di 4 ingressi:

una volta scelti gli ingressi (programmando il dispositivo), mi tengo la funzione creata in modo irreversibile (non posso vedere altre funzioni).

A meno che non voglia usare una memoria riscrivibile (= + costi), ma poco efficiente su

grandi calcolatori.

Se vogli si può usare il  $\text{N}P\text{X}$  con gli interruttori come carry look-ahead  
ha vantaggi e vantaggi:

- utilizzabile anche come dispositivo per funzioni diverse ulteriori al carry lookahead
- logica + TPL LEVEL (ritardo maggiore a livello 2)
  - ↳ non ci perde se poi comunque fa realizzare dispositivo a livello 3

N.B. Se cambio i fili

a <sub>1</sub>	0
b <sub>1</sub>	0
a <sub>0</sub>	1
b <sub>0</sub>	1

ottengo valori da inserire negli ingressi differenti

! Poss. lavorare con più ingressi. Attenzione a ingressi usibili ( $\Rightarrow 11$ ) =  $2^5 = 32$   
crece EXPONENZIALE!

## Circuiti numerici (comparatore)

venerdì 30 ottobre 2020 17:24

$$\text{Ingressi } a, b \quad a = b ? \begin{cases} 0 & a \neq b \\ 1 & a = b \end{cases}$$

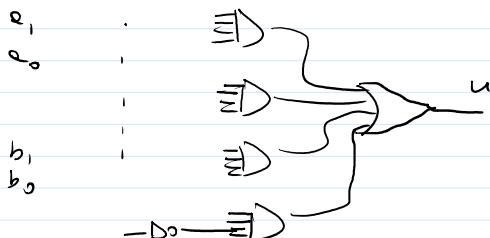
2 BIT ingresso

	$b_1$	0	0	1	1
	$b_0$	0	1	1	0
0 0		1	0	0	0
0 1		0	1	0	0
1 1		0	0	1	0
1 0		0	0	0	1

o celle adiacenti

reduzione minima:

$$\bar{a}_1 \cdot \bar{a}_0 \cdot \bar{b}_1 \cdot \bar{b}_0 + \bar{a}_1 \cdot a_0 \cdot \bar{b}_1 \cdot b_0 + a_1 \cdot a_0 \cdot b_1 \cdot b_0 + a_1 \cdot \bar{a}_0 \cdot b_1 \cdot \bar{b}_0$$



logica 3 livelli  
(no dei NOT, AND, OR)

con maggiori bit deve trovare una semplificazione facendo riduzione agli elementi della diagonale.

Dunque non sono raggruppabili, non essendo adiacenti

es. con 3 bit a ingresso allei AND e OR con 6 ingressi c'è uno.

però via via più efficienza aumentando i bit (impossibile a 32 bit e ca 67%)

Passo dunque ragionare su una soluzione modulare (siolog su una cifra binaria)

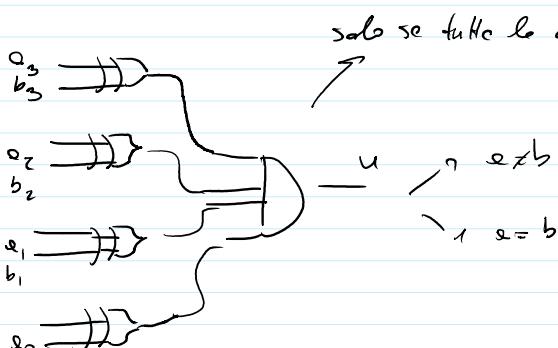
$a_0 \setminus b_0$	0	1
0	1	0
1	0	1

Riconosco la funzione negazione di XOR



Su fronte di bit maggiori, basta confrontare le coppie di bit:  $a_1$  con  $b_1$  ecc...

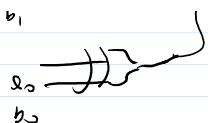
4 BIT



solo se tutte le cifre sono uguali ; due numeri  $a$  e  $b$  sono uguali

Il vantaggio è di aver una crescita lineare di funzioni XNOR

Se considero XOR come f. elementare (bit da controllo) la funzione è a 2 livelli.



$\rightarrow$  considero XOR come f. elementare  
(il ritardo circuitorio) la funzione è: è 2 livelli.  
bisogna verificare che XOR abbia ugual ritardo  
dell'AND, non ne sono sicuro, ma posso dire  
che il ritardo rimane costante  
(velocità 32 BIT = velocità 64 BIT, dipende  
da cosa c'è dentro)

- $a > b$  ( $\Rightarrow$  altri comparatori)

$$\begin{cases} 0 \quad a \leq b \quad (\text{es } b \text{ falso}) \\ 1 \quad a > b \end{cases}$$

Disturbo Hamming = 1

2 BIT

	b <sub>1</sub>	b <sub>0</sub>	r <sup>max</sup> numero
1	0 0 1 1		
2, 8 0	0 1 1 0		

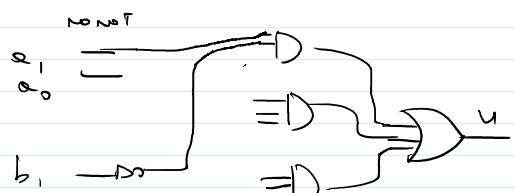
	0 0	0 1	1 1	1 0
0 0	0 0 0 0			
0 1	1 0 0 0			
1 1	1 1 0 0	1		
1 0	1 1 0 0	1	1	

Prima di confrontarsi bisogna stabilire  
come sono codificati i numeri (senza segno,  
con segno) e sono circuiti diversi.

• NUMERI NERI SONO DEGLI SU 2 BIT

$$\rightarrow a_1 \cdot \bar{b}_1 + a_0 \cdot \bar{b}_1 \cdot \bar{b}_0 + a_1 \cdot a_0 \cdot \bar{b}_0$$

i valori e i circuiti  
de a e b non  
combinano le unità  
(escluso a = b se allora,  
tengono determinanti)

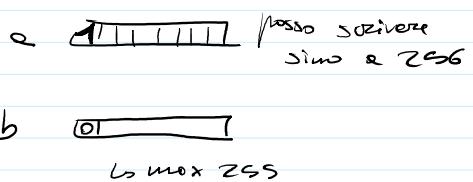
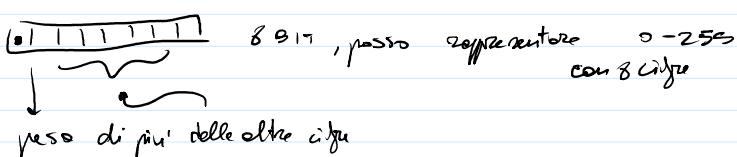


3 LIVELLI

Con più ingressi (multi, es 64 BIT) è notevolmente più difficile.

Provo con una rappresentazione modulare (esplico più moduli)

Questo volta non posso separare le cifre, in questi casi le cifre più significative  
"possono" di più rispetto alle scuse di quelle meno.



$$a > b$$

Se invece trovo:

$$\begin{array}{c} a \\ \hline 1 1 1 1 1 \end{array} \quad \text{max}$$

ordo più concorde velocemente  
che  $a > b$  è falso

Se le cifre più significative  
sono diverse fra loro  
posso rispondere velocemente  
se  $a > b$  o no

se reguoli bene