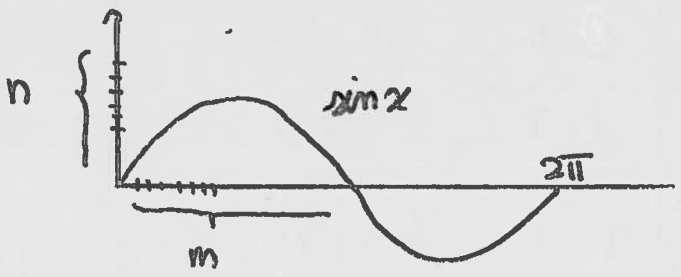
Una ROM è una look-up table (TABELLA di CONSULTAZIONE): un contenitore di informazioni già memorizzate. Il suo comportamento è assimilabile a quello di una memoria di dimensione $2^m \times n$, con

- m (# dei bit in ingresso) → risoluzione
- n (# dei bit in uscita) → accuratezza del risultato.

---

es. calcolatrice che calcola $\sin x$

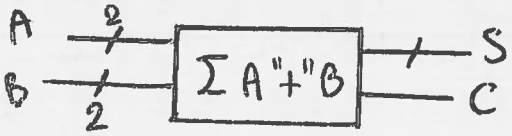


L'encoder è quella parte della ROM in cui si selezionano i minterm per implementare il comportamento desiderato

~ . ~

Ci stiamo occupando di come implementare funzioni aritmetiche e funzioni di tipo logico.
partire da

Per ora abbiamo visto il COMPARATORE, che, dati in ingresso due numeri, determina se sono $\leq$, $\geq$, ed il sommatore a due bit, che implementa la somma aritmetica A "+" B

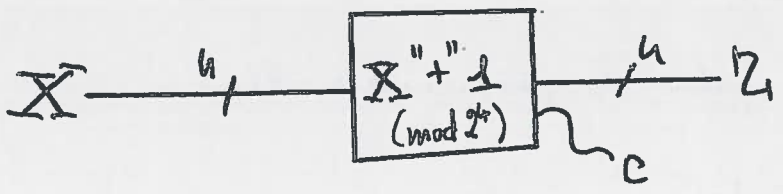


Quantizzazione: data una quantità

x_3	x_2	x_1	x_0	C	z_3	z_2	z_1	z_0
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0	0
0	1	0	0	0	0	1	0	1
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	1	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	1	0
1	0	1	0	0	1	0	1	1
1	0	1	1	0	1	1	0	0
1	1	0	0	0	1	1	0	1
1	1	0	1	0	1	1	1	0
1	1	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0

(*) 1

(1)



$C(x_3, x_2, x_1, x_0) = x_3 x_0 x_1 x_2$

forma canonica

Considerando le colonne $z_3\ z_2\ z_1\ z_0$, mi accorgo che nell'ultima riga avviene un overflow (1) (infatti il numero 15+1 non può essere rappresentato con 4 bit).
In aritmetica modulare, quando supero il limite della rappresentabilità c'è un troncamento: l'operazione realmente implementata in $\oplus_4$ è x "+" 1 (mod 16)

x_3	x_2	x_1	x_0		z_3	z_2	z_1	z_0
0	0	0	0	→	0	0	0	1
0	0	0	1	→	0	0	1	0
0	0	1	0	→	0	0	1	1
⋮					⋮			

Mi accorgo che in ogni riga della tabella c'è traccia di quale numero sono entrato a contare e di che è il suo successore

"Sapere a che punto sono entrati" è un concetto estraneo alle reti combinatorie (che nel momento in cui calcolano un risultato non tengono memoria dei calcoli precedentemente fatti).
Per calcoli più complessi, invece, è necessaria quella parte di memorizzazione che invece manca alle funzioni logiche combinatorie.

Oss. 1) Quando un bit passa da $0 \to 1$ o $1 \to 0$ parliamo di commutazione.

2) $a \oplus b = \bar{a}b + a\bar{b}$ infatti

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

analizziamo quel che avrete calcolato per calcolare

- z_0 commuta sempre, dunque

$$z_0 = \bar{x}_0$$

- $z_1 = x_1 \oplus x_0$ → commuta ogni due righe

- $z_2 = x_2 \oplus (x_1 x_0)$ → commuta ogni quattro righe

- $z_3 = x_3 \oplus (x_2 x_1 x_0)$

non è una F. canonica perché le variabili dovrebbero essere negate singolarmente

$$z_3 = \bar{x}_3 (x_2 x_1 x_0) + x_3 \overline{(x_2 x_1 x_0)}$$

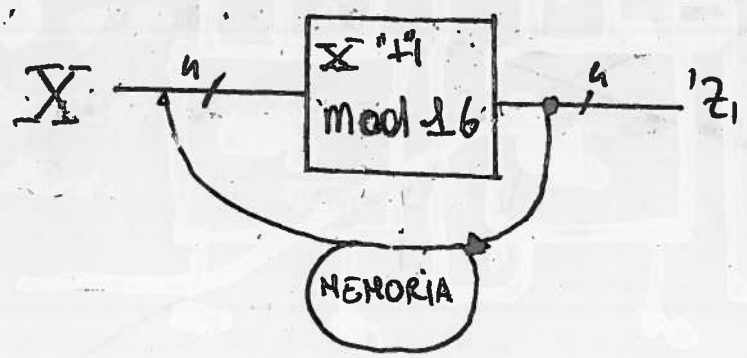
II - De Morgan

$$\bar{x}_3 (x_2 x_1 x_0) + x_3 \bar{x}_2 \underbrace{(x_1 + \bar{x}_1)}_{\text{1}} (x_0 + \bar{x}_0) + x_3 \bar{x}_1 (x_2 + \bar{x}_2)(x_1 + \bar{x}_1) +$$

1 = el. unico mancante

$$+ x_3 \bar{x}_0 (x_2 + \bar{x}_2)(x_1 + \bar{x}_1)$$

La funzione implementata in $\otimes_1$ è detta <u>contatore in avanti</u> (esiste anche il contatore all'indietro, detto <u>c. all'indietro</u>)
Si tratta di un esempio di MACCHINA SEQUENZIALE: non esiste alcuna rete combinatoria che abbia questo tipo di comportamento



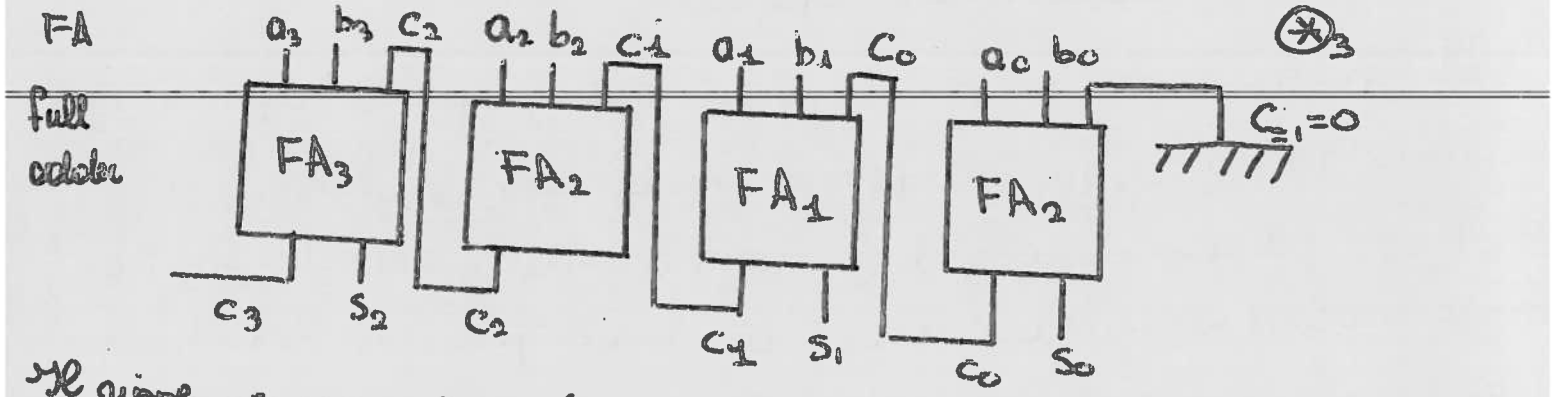
RIPPLE CARRY ADDER

Creiamo un circuito che calcola in serie i risultati: questo porta ad un vantaggio in termini di costo, perché memorizzare i risultati occupa spazio (2^m esiti).

Supponiamo di voler sommare fra loro parole da 4 bit

$(a_3\, a_2\, a_1\, a_0)_2$

$(b_3\, b_2\, b_1\, b_0)_2$



Il ripple-carry adder (sommatore a propagazione di riporto) tiene conto non solo dei bit analoghi, ma anche del riporto dalla somma precedente: il riporto si propaga da uno stato all'altro.

Ogni FA realizza una funzione combinatoria di questo tipo

a_i	b_i	c_{i-1}	c_i	S_i
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

La mappa di Karnaugh relativa a c_{i-1} è

$a_i b_i$ \ c_{i-1}	0	1
00	0	0
01	0	1
11	1	1
10	0	1

$$c_i = a_i b_i + b_i c_{i-1} + a_i c_{i-1}$$

⊛3 non è un sommatore efficiente.

Infatti un sommatore parallelo "lungo" impiega più tempo: i c_{i-1} all'inizio sono a 0, quindi pur se inizialmente dai sommatori usciranno (all'applicazione degli ingressi) dei risultati "sbagliati", che si stabilizzeranno solo dopo un tempo pari al tempo di FA 0 darà il primo risultato corretto al seguito dell'applicazione dei dati dopo un tempo τ; poiché FA 1 quando F_{A0} si sarà stabilizzato: in generale

FA_0	impiega	τ	→ uscite così per tutti gli altri
FA_1	—"—	2τ	calcoli: esistono tecniche per
FA_2	—"—	3τ	diminuire la dipendenza
FA_3	—"—	4τ	di ogni stadio dal precedente

a_i	b_i	c_{i-1}	c_i	$\bar{c}_i$		S_i
0	0	0	0	1	→	0
0	0	1	0	1		1
0	1	0	0	1		1
0	1	1	1	0		0
1	0	0	0	1		1
1	0	1	1	0		0
1	1	0	1	0		0
1	1	1	1	0	→	1

possibile variabile $c_i = s_i$

Es.

i) $x \to 0$ $\quad x \cdot 0 = 0$

ii) $x \to 1$ $\quad x + 1 = 1$

Vogliamo una variazione locale nella funzione $\circledast_a$

a_i	b_i	c_{i-1}	c_i	$\bar{c}_i \cdot (a_i + b_i + c_{i-1}) + (a_i b_i c_{i-1})$	s_i
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	1	1

Invece di ricavare separatamente c_i e s_i, in $\circledast_a$

1) calcolo c_i

2) ottengo s_i dalla combinazione di c_i con le variabili in ingresso