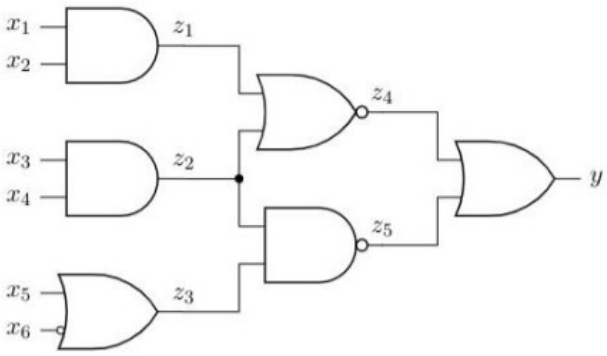
Lezione 10 Circuiti Combinatori 2

martedì 17 ottobre 2023 16:12

Andiamo a vedere come attraverso le porte logiche , possiamo passare da funzione booleana (impulsiva) a circuito logico . Si parte dai termini più interni (precedenza) e realizzo quella funzione come pezzo del circuito totale . Inversamente do un nome ad ogni uscita e poi le ricompongo . Vediamo un esempio :



$$y = z_4 + z_5$$

NOR

$$y = (z_1 \downarrow z_2) + (z_2 | z_3)$$

AND

$$y = ((x_1 \cdot x_2) \downarrow (x_3 \cdot x_4)) + ((x_3 \cdot x_4) | (x_5 + \overline{x_6}))$$

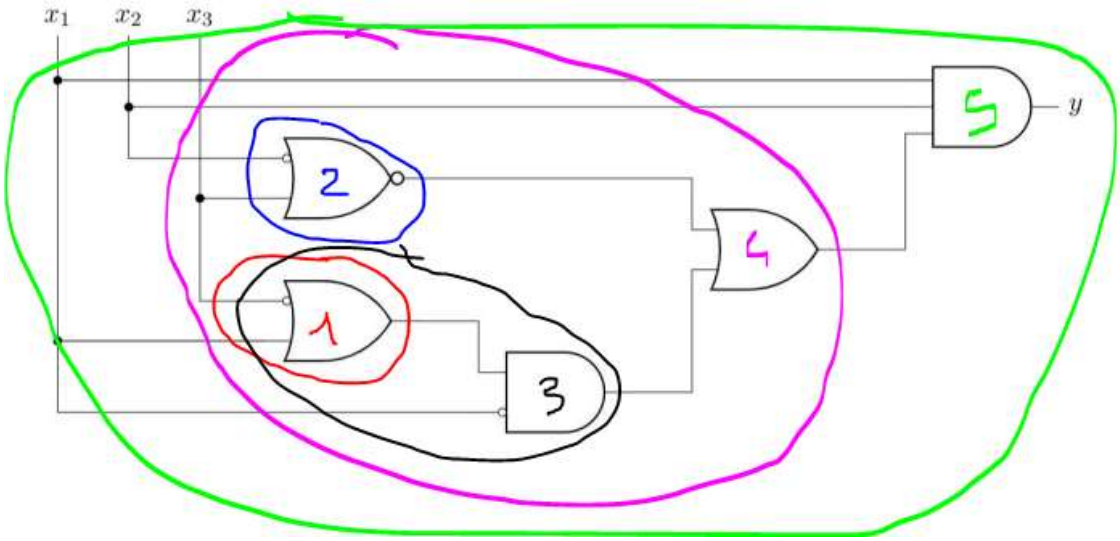
$$y = \overline{x_1 x_2 + x_3 x_4} + \overline{x_3 x_4 (x_5 + \overline{x_6})}$$

Esempio inverso : da funzione a circuito (usiamo la precedenza) :

$$y = x_1 x_2 ((\overline{x_2} \downarrow x_3) + (\overline{x_1}(x_1 + \overline{x_3})))$$

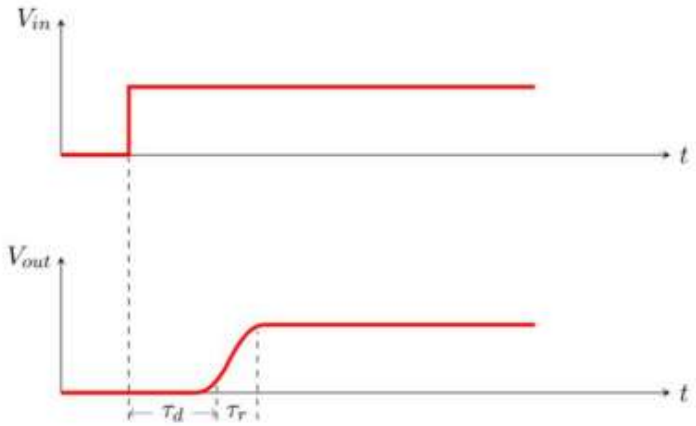
$1 = \overline{x_1 \oplus x_2}$ $\quad 3 = 1 \cdot \overline{x_1}$ $\quad 5 = y$

$2 = \overline{x_2 \oplus x_3}$ $\quad 4 = 2 + 3$



Sorge una domanda : quanto è stato "costruito bene/buono " questo circuito ? Andiamo a vedere il costo : nr porte logiche usate e tempo : velocità con la quale vengono processate le informazioni . In realtà ogni circuito ha un ritardo di commutazione , ovvero il tempo che la corrente percorre tutti i

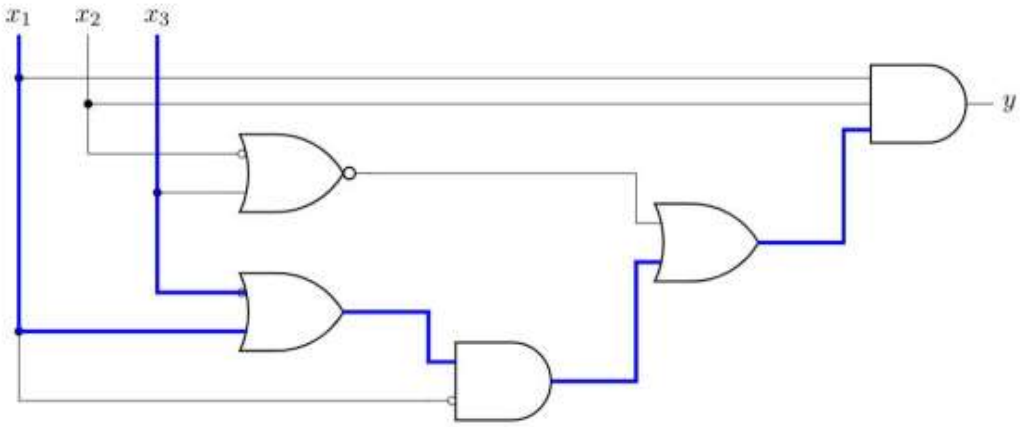
transistor (rete) :



τ_d: ritardo di commutazione (tempo per arrivare al 10% del valore finale)
τ_r: tempo di salita (tempo per arrivare al 90% del valore finale)

Di solito per ogni porta si considera un unico tempo : quello di propagazione , il quale corrisponde alla somma di entrambi .

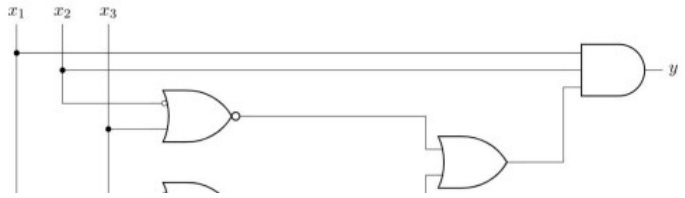
Per calcolarlo si usa il **critical path** : percorso più lungo per arrivare ad uscita : nel nostro caso sarà 4*tp : ogni livello aggiunge un ritardo :

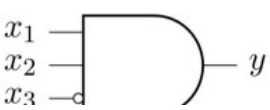


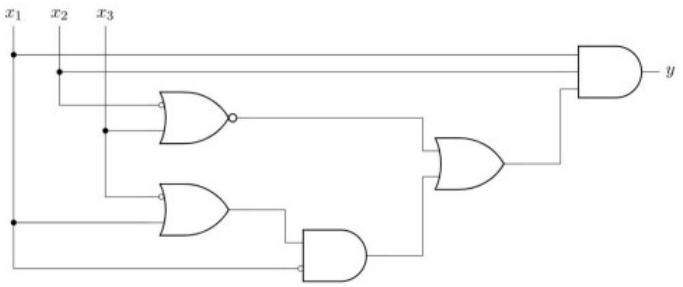
Per quanto riguarda invece il costo si vanno a vedere il numero di porte che vengono usate nel circuito, si vede anche il numero degli ingressi di ogni porta . Per ridurre la complessità del circuito devo usare le tecniche di minimizzazione viste finora :

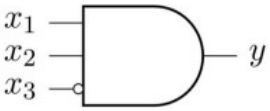
$$y = x_1x_2((\overline{x_2} \downarrow x_3) + (\overline{x_1}(x_1 + \overline{x_3})))$$
$$y = x_1x_2((x_2 \cdot \overline{x_3}) + (\overline{x_1}(x_1 + \overline{x_3}))) \text{ (definizione dell'operatore NOR)}$$
$$y = x_1x_2((x_2 \cdot \overline{x_3}) + (\overline{x_1}x_1 + \overline{x_1}\,\overline{x_3})) \text{ (proprietà distributiva)}$$
$$y = x_1x_2(x_2 \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_3}) \text{ (elemento inverso e elemento identità)}$$
$$y = x_1x_2\overline{x_3} + x_1\overline{x_1}x_2\overline{x_3} \text{ (proprietà distributiva e idempotenza)}$$
$$y = x_1x_2\overline{x_3} \text{ (elemento inverso e elemento identità)}$$

Quindi riassumendo :



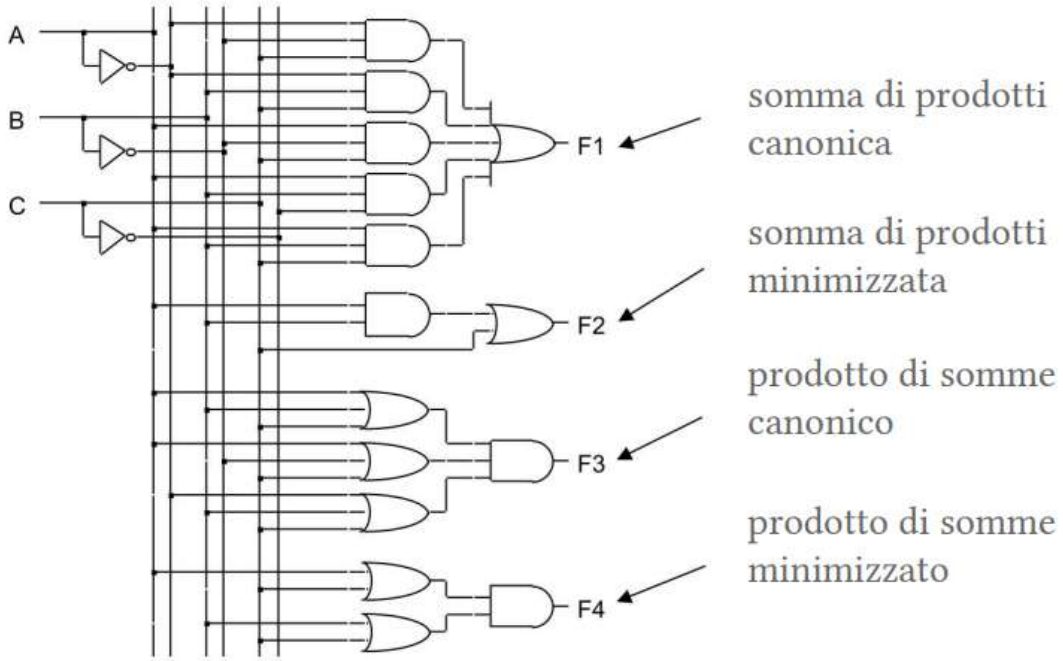






Tornando alle forme canoniche : non sempre sono la forma minima del circuito , ma ha il vantaggio di usare una rete a due livelli (and - or), quindi aumenta la velocità : è una buona approssimazione. In generale non c'è un metodo/modo univoco di rappresentare un qualunque circuito :

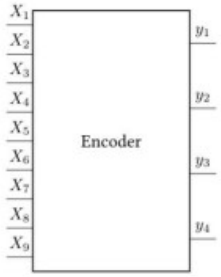
Esempio di realizzazione di $f = ab + c$



Andiamo a vedere ora i componenti fondamentali per realizzare funzioni booleane in hw :

1. **Codificatore (encoder)** : codifica binaria : associo parola a rappresentazione . Per ciascun elemento in input, genererà il relativo codice : in input al massimo 2^n bit :

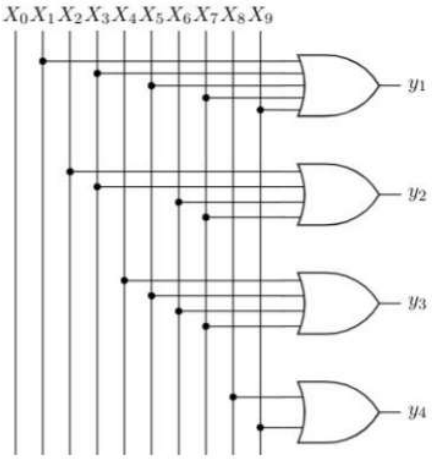
a.



b. Dato un qualunque ingresso, in uscita sarà data la codifica binaria di input.
c. Esempio : codificatore in formato BCD:

i.

Base 10	BCD	Base 10	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

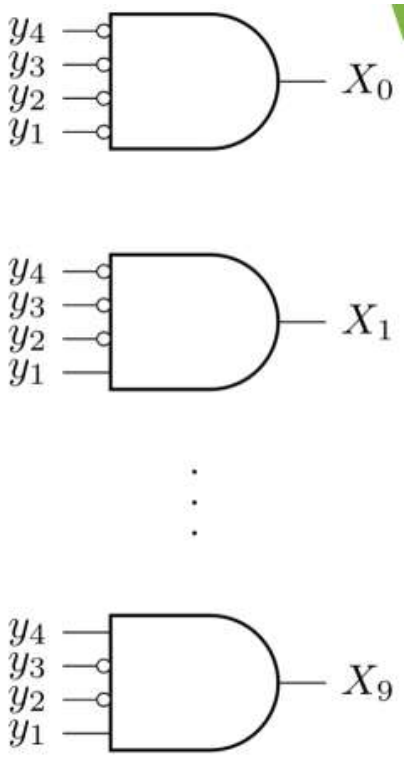


d. Quindi ragiono su ciascun bit in uscita come se fosse funzione booleana indipendente : **studio il bit meno significativo vale 1 , e torno a ritroso : da ultimo bit fino al primo bit meno significativo . Si parte a ritroso per il vettore di variabili booleana (y4,y3y2,y1).**
e. Vediamo un esempio : codificare il "3" : vediamo x3 su quali porte va : su y1 e y2 ,mentre y3 e y4 non ci va : quindi il codice sarebbe 1100 , ma visto che lo prendiamo a ritroso : 0011

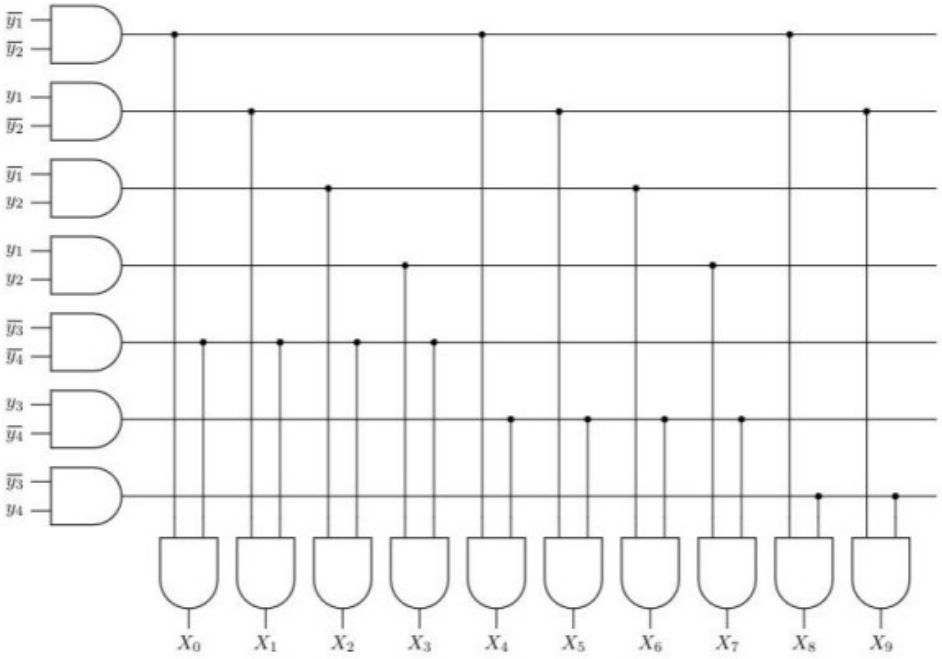
2. **Decodificatore (decoder)** : data una parola binaria, attiva una ed una sola linea di uscita :

a.

$$X_0 = \overline{y_1}\,\overline{y_2}\,\overline{y_3}\,\overline{y_4}$$
$$X_1 = y_1\,\overline{y_2}\,\overline{y_3}\,\overline{y_4}$$
$$\vdots$$
$$X_9 = y_1\,\overline{y_2}\,\overline{y_3}\,y_4$$

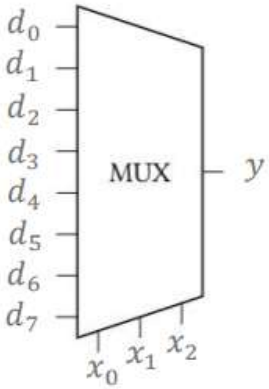


b.



3. **Multiplexer** : dispositivo elettronico che permette di scegliere tra più funzioni uno solo da propagare in output : attiva porzioni differenti del mio processore: Permette di implementare tutte le funzioni booleane

a.



b. Con d si intendono i segnali (variabili booleane in ingresso), mentre con (x0,x1,x2) si intendono i segnali di abilitazione

c.

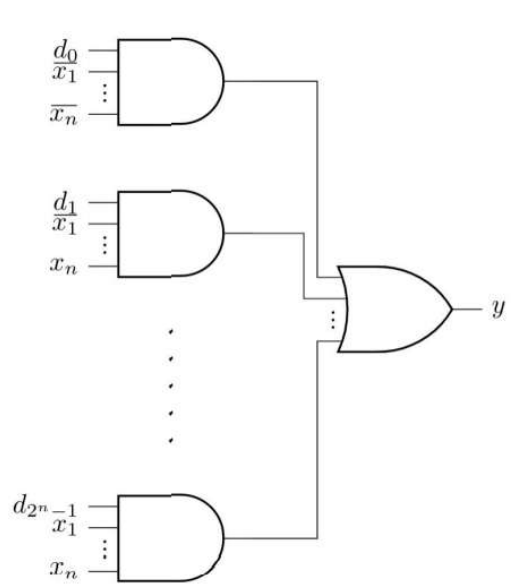
x	y
0	d_0
1	d_1
2	d_2
3	d_3
4	d_4
⋮	⋮
2^{n-1}	$d_{2^{n-1}}$

d. $$y = \sum_{i=0}^{2^n-1} d_i m_i$$

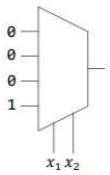
   i. Con vettore x assume valore d(i) se e solo se x=i

e. Andiamolo a vedere circuitalmente :

   i.

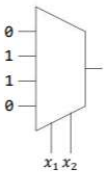
   ii. 

   iii. Vediamo in dettaglio che il mux può realizzare tutte le funzioni :

1. 

AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

4. Demultiplexer (demux) : dispositivo duale del mux : circuito che dato un singolo valore in input , lo trasferirisce su una riga dell'output : questa riga viene definita dalla configurazione degli enable :

a.