Capitolo 1

Le reti logiche

1.1 Introduzione

Come ben sappiamo, a differenza di *ENIAC*, i computer moderni sono realizzati da un'enorme quantità di circuiti elettronici. Trattandosi di eletronica digitale, a livello hardware è possibile lavorare con due livelli fondamentali:

- 1: alto, asserito; associato alla tensione di alimentazione *Vdd*.
- 0: basso, negati; associato alla massa (tensione vicina a 0).

Nel passaggio da un livello all'altro si presenta lo stato *transitorio*, in cui la tensione assume valori intermedi fra zero e Vdd.
In particolare, i **circuiti logici** (formati dalle **porte logiche**) vengono utilizzati per trasformare alcuni valori logici in ingresso in altri valori logici in uscita. I circuiti logici si differenziano principalmente in:

- *Combinatorie*: l'uscita dipende unicamente dal valore di ingresso (non sono dotati di memoria).
- *Sequenziali*: l'uscita dipende dal valore di ingresso e dalla storia degli ingressi precedenti (sono dotati di memoria, detta "stato" della rete).

È possibile riassumere il comportamento di una rete logica attraverso una *tabella di verità*, che descrive i diversi output al variare dei diversi input.

1.2 L'algebra di Boole

Esisotno principalmente tre operazioni di base:

- **AND**: $A \cdot B$; produce 1 se entrambi gli operandi sono 1, 0 altrimenti.
- **OR**: $A + B$; produce 0 se entrambi gli operandi sono 0, 1 altrimenti.
- **NOT**: $\overline{A}$; inverte il valore logico.

1.2.1 Regole di semplificazione

Di seguito vengono riportate alcune regole di semplificazione:

- Identità: $A + 0 = A$, $A \cdot 1 = A$

- Regola "zero e uno": $A + 1 = 1$, $A \cdot 0 = 0$
- Regola dell'inversa: $A + \overline{A} = 1$, $A \cdot \overline{A} = 0$
- Regola commutativa: $A + B = B + A$, $A \cdot B = B \cdot A$
- Regola associativa: $A + (B + C) = (A + B) + C$, $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- Regola distributiva: $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$, $A + (B \cdot C) = (A + B) \cdot (A + C)$
- De Morgan: $\overline{A \cdot B} = \overline{A} + \overline{B}$, $\overline{A + B} = \overline{A} \cdot \overline{B}$

Proprio grazie alla regola di De Morgan, si può affermare che la porta **NAND** (not AND) e la porta **NOR** (not OR) sono universali. Ciò significa che utilizzando unicamente porte NAND (rispettivamente NOR) è possibile costruire tutte le altre porte logiche.

1.2.2 La tavola di verità e i mintermini

Input			Output		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Tabella 1.1: Esempio di tabella di verità

Si può verificare che:

$$D = A + B + C$$
$$F = A \cdot B \cdot C$$
$$E = (A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (B \cdot C \cdot \overline{A})$$

In particolare, l'ultima espressione è stata ottenuta come somme di prodotti attraverso i cosiddetti *mintermini*. Per ciscun 1 presente nella colonna E, si scrive il prodotto degli input (asseriti se pari a 1, negati se pari a 0); il risultato dell'espressione è pari alla somma di tali prodotti.

1.3 Le porte logiche

A seguire vengono rappresentate, attraverso i corrispettivi simboli, le porte logiche implementate attraverso gli operatori booleani fondamentali:

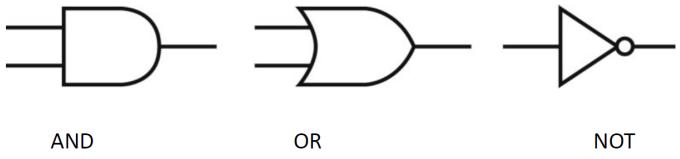
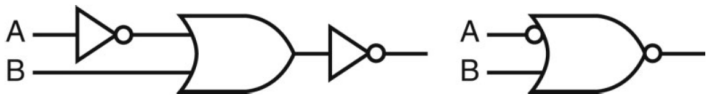


Figura 1.1: Rappresentazione di AND, OR e NOT

Si ricorda inoltre che è possibile combinare fra loro diverse porte logiche:



$$\overline{\overline{A} + B}$$

Figura 1.2: Combinazione di porte logiche

1.4 Alcuni circuiti degni di nota

1.4.1 Decoder

L'ingresso del *decoder* svolge il ruolo di selettore: ricevendo in input il valore n, si accenderà l'n-esima uscita. A seguire l'implementazione di un decoder a 3 bit:

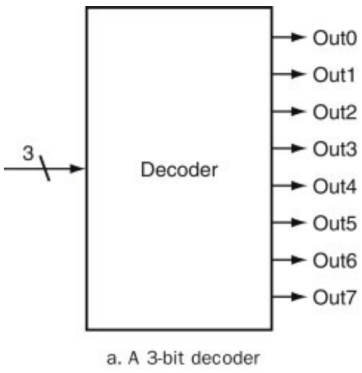


Figura 1.3: Decoder a 3 bit

Input			Output							
In2	In1	In0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Tabella 1.2: Tabella di verità per un decoder a 3 bit

1.4.2 Multiplexer

Prevede un input con valore S utilizzato come selettore. L'output del *multiplexer* è il valore dell'input S-esimo.

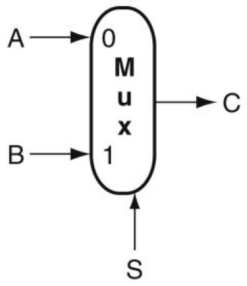


Figura 1.4: Multiplexer a 1 bit

È possibile costruire un multiplexer attraverso un decoder:

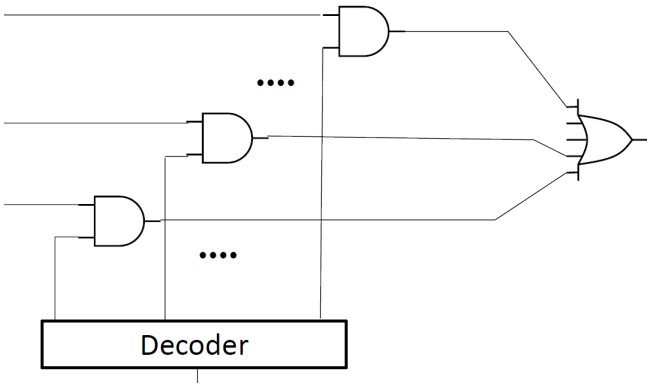


Figura 1.5: Multiplexer a n vie

Molto spesso, per operare con dati complessi, si compongono degli array di elementi elementari. Segue un esempio di come sia possibile costruire un multiplexer con *bus* a 32 bit utilizzando un array di multiplexer a 1 bit.

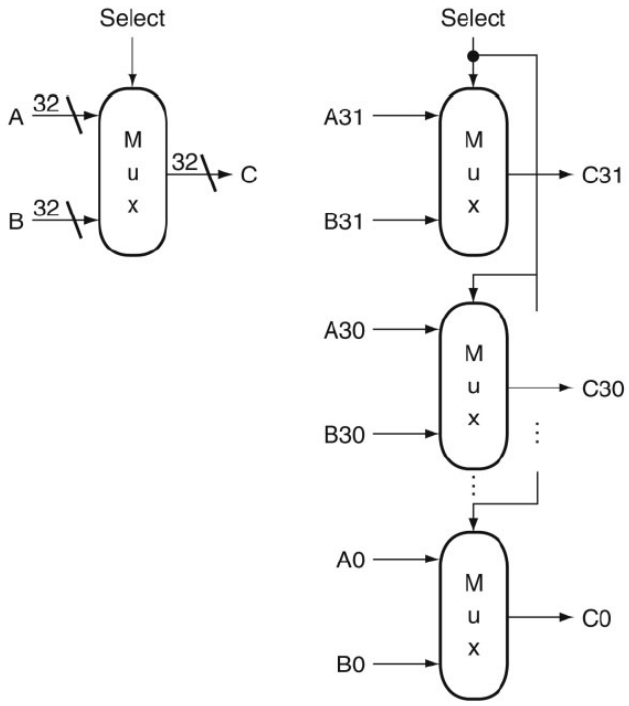


Figura 1.6: Multiplexer a 32 bit

1.4.3 Programmable Logic Array (PLA)

Il *programmable logic array* si compone di due strutture: una barriera di AND e da una barriera di OR. La dimensione totale del *PLA* è data dalla somma del piano AND (numero di mintermini) e del piano OR (numero di uscite).
Inoltre, il PLA implementa porte logiche solamente per le configurazioni che producono 1 in uscita. In aggiunta, se un mintermine è condiviso tra varie uscite, lo si può riutilizzare. Successivamente seguono due diverse implementazioni della rete logica descritta nel paragrafo 1.2.2.

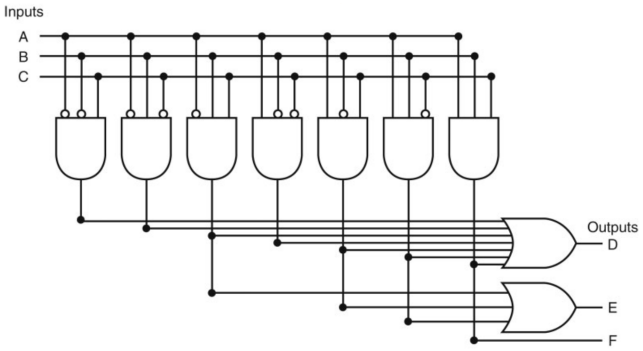


Figura 1.7: Esempio di rete logica

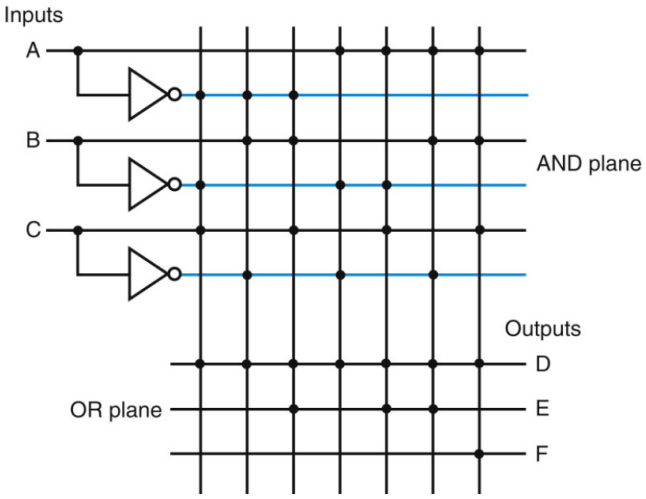


Figura 1.8: Esempio di rete logica implementata attraverso PLA

1.5 Il concetto di *costo*

Come appena visto, le reti logiche possono essere implementate in maniera differente. Si definisce *costo* di una rete logica la somma del numero di porte e del numero di ingressi della rete. Quando si analizza una rete, è importante saperne trovare l'implementazione con costo minimo. Principalmente si può procedere in due modalità:

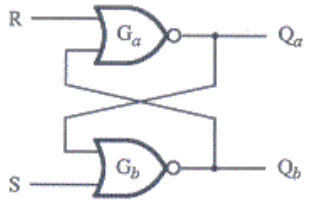
- regole di semplificazione matematiche dovute all'algebra di Boole (vedi paragrafo 1.2.1).
- metodi basati su rappresentazioni grafiche (mappe di *Karnaugh*), nei casi più semplici.

1.6 Le reti sequenziali

Come si è visto, le reti combinatorie non hanno memoria degli stati del passato: in ogni istante di tempo l'uscita dipende solamente dagli ingressi nell'istante considerato. Tuttavia, in certe applicazioni è necessario introdurre una vera e propria memoria nel sistema.
La memoria in una rete logica si ottiene attraverso una *retroazione*, che consiste nel ridirezionare alcune porte di uscita in ingresso ad altre porte del medesimo circuito, in maniera tale da formare un anello.
Seppur questo meccanismo offra delle potenzialità enormi, l'analisi e la sintesi di una rete logica è molto più complessa rispetto a quella delle reti combinatorie.

1.6.1 Latch S-R

A seguire un'implementazione di una rete sequenziale *latch S-R* (*set-reset*) costruita attraverso due porte NOR:



(a) Network

S	R	Q_a	Q_b	
0	0	0/1	1/0	(No change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

(b) Truth table

Figura 1.9: Esempio di latch a porte NOR

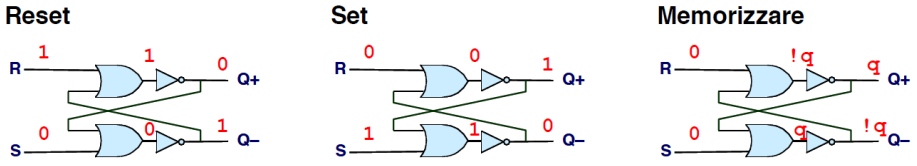


Figura 1.10: Operazioni di un latch a porte NOR

Ma, considerando che i segnali non si propagano in tempo nullo, ciò potrebbe causare dei problemi circa la correttezza del segnale stesso. La soluzione è quella di aggiungere un ulteriore segnale, in grado di temporizzare il circuito: il *clock*. Un latch dotato di temporizzazione viene chiamato *gated latch*.

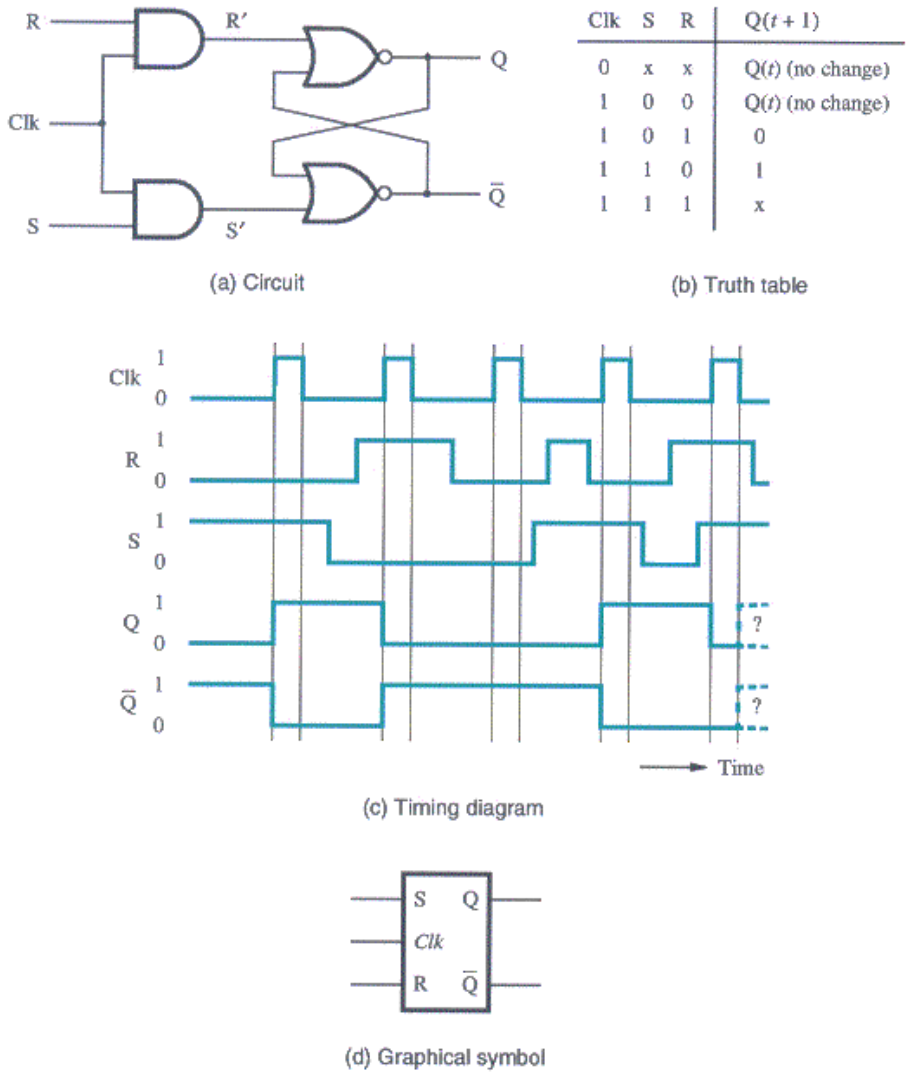


Figura 1.11: Esempio di gated latch

Il difetto principale di un latch R-S è quello di avere uno stato indecidibile: l'uscita non può essere nota con certezza quando entrambi gli ingressi sono a 1.

Una soluzione è implementata attraverso *latch* tipo D: gli ingressi al circuito sono ottenuti da un'unica variabile (di cui se ne fa il negato). Altrimenti, attraverso circuiti *flip-flop master-slave* è possibile ottenere che l'uscita del circuito commuti esattamente al termine dell'impulso di clock.
In particolare, i latch di tipo D vengono utilizzati per implementare i registri della CPU.