Architettura degli Elaboratori
6 - Complementi sulle Reti Combinatorie

Zeynep KIZILTAN

Dipartimento di Scienze dell'Informazione
Università degli Studi di Bologna

Anno Accademico 2007/2008

Sommario

Porte NAND e NOR

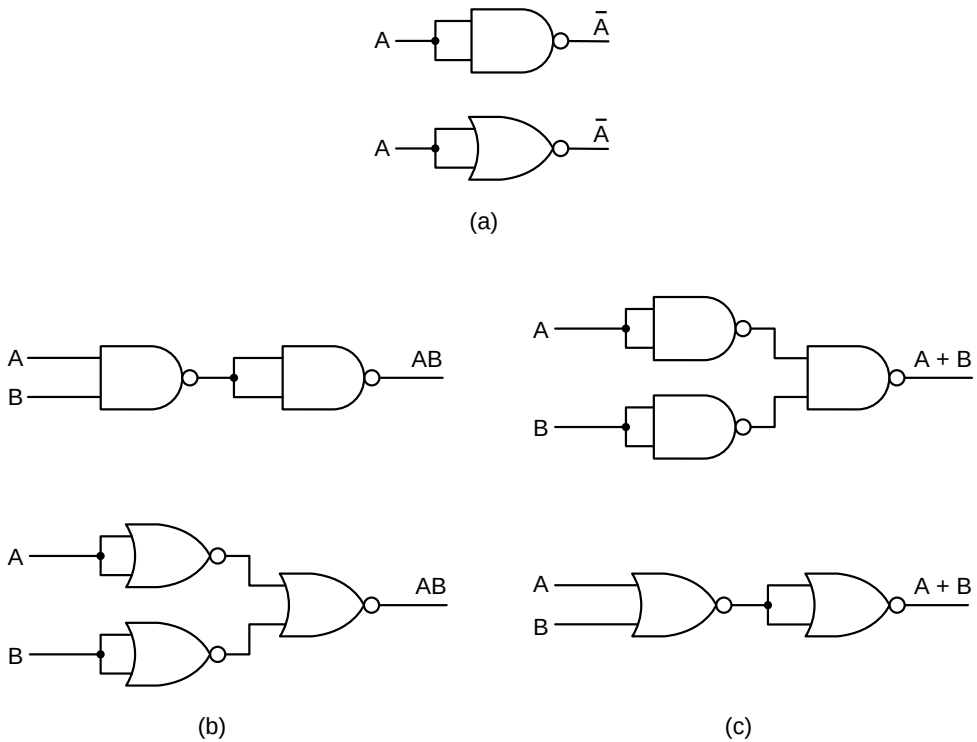
- Siccome le funzioni booleane sono usualmente espresse in termini di operazioni AND, OR, e NOT, risulta naturale la loro implementazione usando le porte logiche AND, OR e NOT.
- Tuttavia è di interesse pratico utilizzare altre porte logiche che implementano nuove, differenti operazioni logiche.
- Alcune porte utili nella progettazione sono le porte NAND e NOR.
- Abbiamo visto che le porte NAND e NOR necessitano di due transistor ciascuna, mentre le porte AND e OR ne richiedono tre.
- Per questa ragione, molti calcolatori sono basati sulle porte NAND e NOR.

Porte NAND e NOR

- Inoltre, le porte NAND e NOR hanno un proprietà interessante: sono **universali**.
  - Ogni rete combinatoria può essere trasformata in una rete equivalente contenente solo la porta NAND o, in alternativa, contenente solo la porta NOR.
- Per provare questa affermazione, basta mostrare che gli operatori AND, OR, e NOT possono essere realizzati utilizzando un solo tipo di porta logica (NAND o NOR).

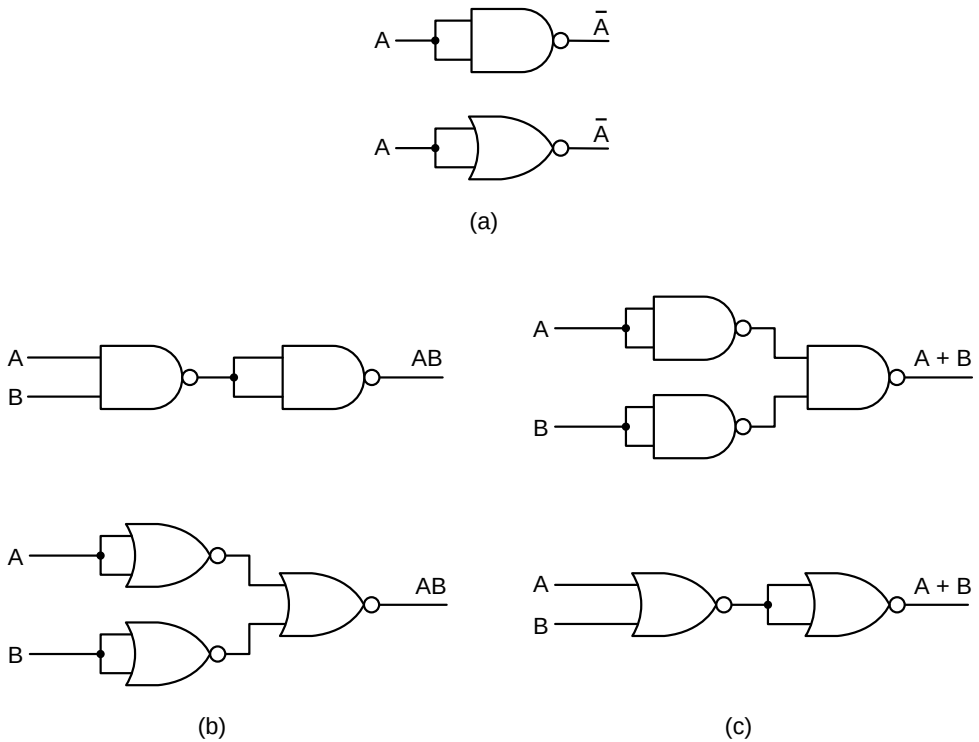
Porte NAND e NOR

- Nella figura (a), l'operazione NOT è ottenuta da una porta NAND ai cui due ingressi è applicato lo stesso segnale.
- In tal caso, si ottiene $\overline{AA} = \overline{A} + \overline{A} = \overline{A}$.



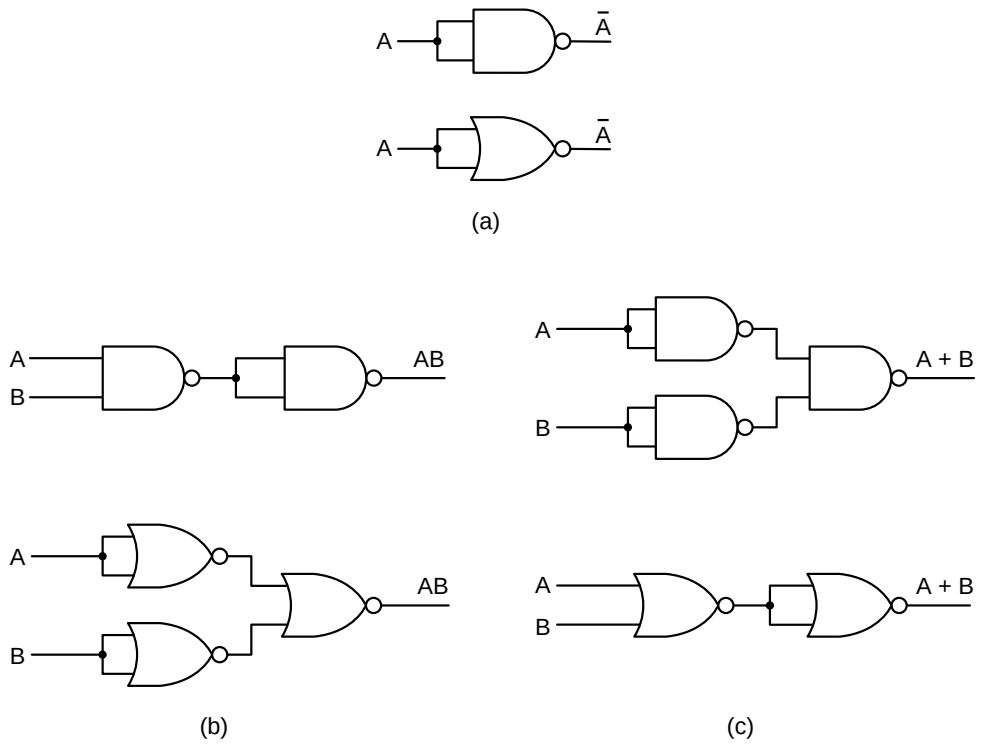
Porte NAND e NOR

- Nella figura (b), l'operazione AND è ottenuta da due porte NAND: la prima svolge l'operazione di NAND e la seconda complementa il segnale.
- In tal caso, si ottiene $\overline{\overline{AB}} = AB$.



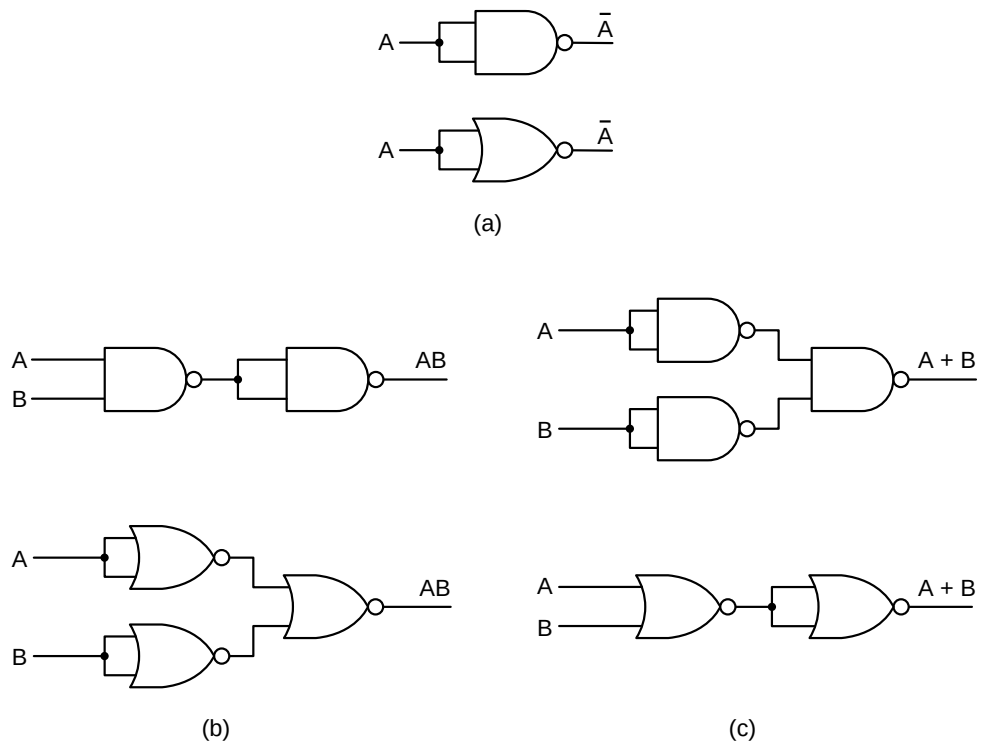
Porte NAND e NOR

- Nella figura (c), l'operazione OR è ottenuta utilizzando una porta NAND con un'altra porta NAND su ogni ingresso.
- In tal caso, si ottiene $\overline{\overline{A}\,\overline{B}} = A + B$.



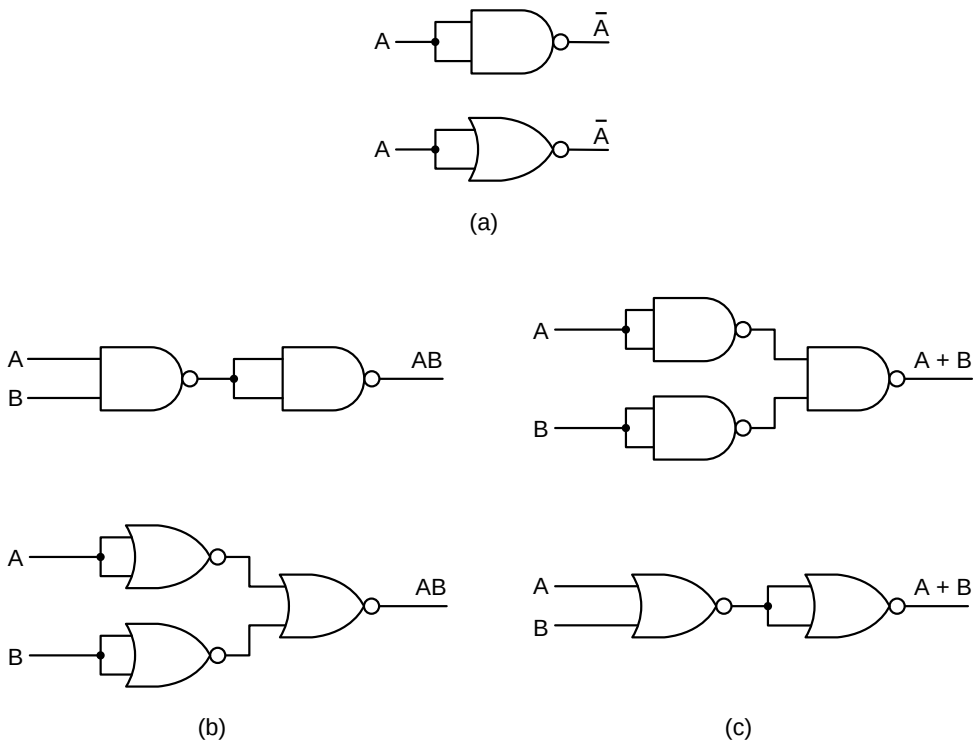
Porte NAND e NOR

- La porta NOR è anch'essa una porta universale che può essere utilizzata per implementare qualunque funzione booleana.
- Il loro uso è simile a quello delle porte NAND.
- Nella figura (a), l'operazione NOT è ottenuta da una porta NOR ai cui due ingressi è applicato lo stesso segnale.
- In tal caso, si ottiene $\overline{A + A} = \overline{A}\,\overline{A} = \overline{A}$.



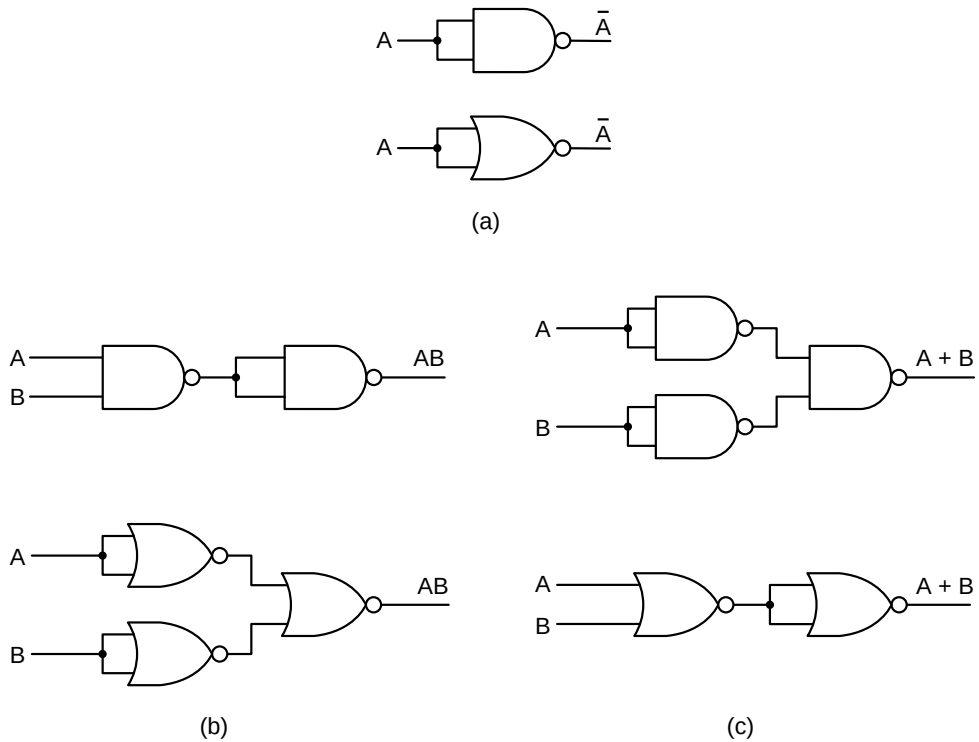
Porte NAND e NOR

- Nella figura (b), l'operazione AND è ottenuta con una porta NOR i cui ingressi sono entrambi complementati da altre porte NOR.
- In tal caso, si ottiene $\overline{\overline{A} + \overline{B}} = AB$.



Porte NAND e NOR

- Nella figura (c), l'operazione OR è ottenuta da due porte NOR: la prima svolge l'operazione di NOR e la seconda complementa il segnale.
- In tal caso, si ottiene $\overline{\overline{A + B}} = A + B$.

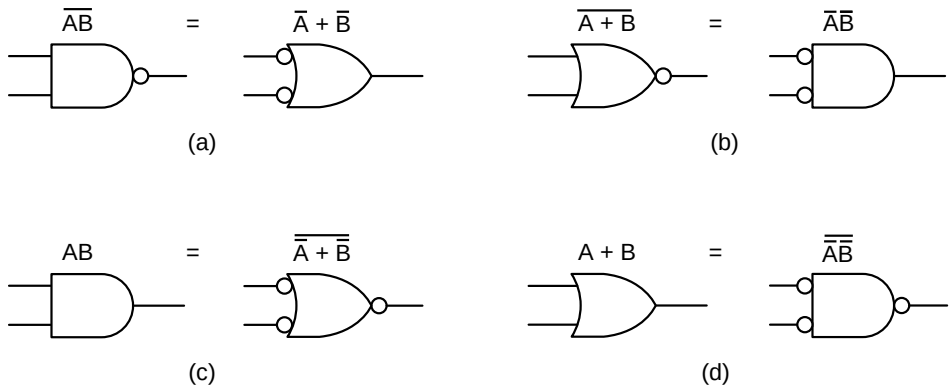


Universalità

- Esistono tecniche diverse per dimostrare l'**universalità** delle porte NAND e NOR, ossia per dimostrare che per ogni funzione booleana F esistono:
  - una rete combinatoria contenente soltanto porte NAND;
  - una rete combinatoria contenente soltanto porte NOR.
- Una tecnica per implementare una funzione booleana utilizzando solo le porte NAND (o NOR) consiste nel:
  1. realizzarla inizialmente mediante le porte NOT, AND, e OR;
  2. successivamente, sostituire tutte le porte dai circuiti equivalenti che usano solo le porte NAND (o NOR).
- Questa tecnica mostra che esiste sempre una possibile implementazione.
- Però, i circuiti ottenuti non sono ottimali, nel senso che non utilizzano il minimo numero possibile di porte logiche.

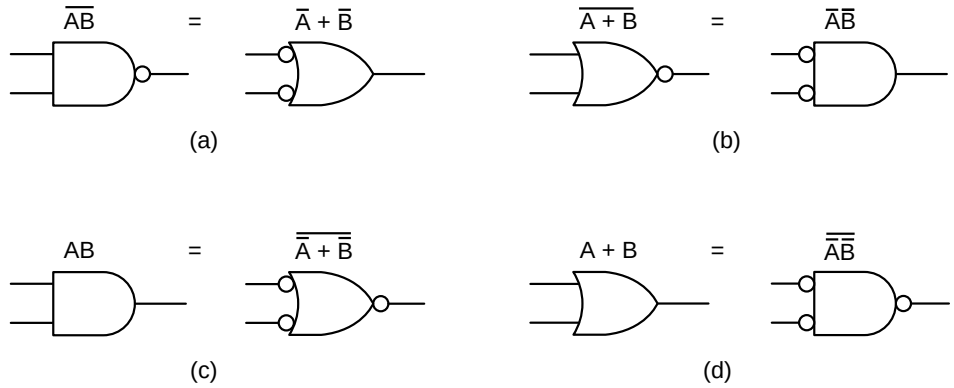
Universalità: Un Tecnica Generale

- Un'altra tecnica sfrutta una notazione alternativa per le porte NAND e NOR.
- Applicando il teorema di De Morgan si possono ottenere i seguenti simboli per le porte NAND e NOR:



- Una porta NAND (detta AND-NOT) è equivalente ad una porta OR con gli ingressi invertiti (detta NOT-OR), come si vede in Figura (a).
- Analogamente, una porta NOR (detta OR-NOT) è equivalente ad una porta AND con gli ingressi invertiti (detta NOT-AND), come si vede in Figura (b).

Universalità: Un Tecnica Generale



- Negando entrambe le forme della legge di De Morgan si ottengono le Figure (c) e (d) che mostrano due rappresentazioni equivalenti delle porte AND e OR.
- Esistono simboli analoghi per le forme a più variabili delle legge di De Morgan.

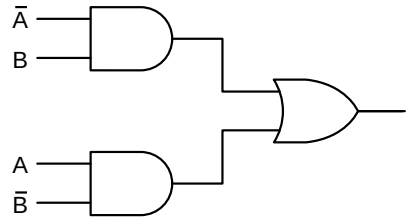
Universalità: Un Tecnica Generale

- Una tecnica generale per implementare una funzione booleana F utilizzando solo le porte NAND consiste nel:
  1. costruire una rete combinatoria per F tramite porte AND-OR;
  2. convertire tutte le porte AND in porte NAND con simbolo AND-NOT;
  3. convertire tutte le porte OR in porte NAND con simbolo NOT-OR;
  4. controllare tutte le complementazioni presenti nel diagramma: per ogni pallino non bilanciato da un altro pallino lungo la stessa connessione, inserire una porta NOT (in forma di porta NAND) o complementare il letterale in ingresso rispetto alla sua versione originale.
- Questa tecnica è facilmente adattabile per implementare una funzione con sole porte NOR:
  - convertire ogni porta OR in NOR con simbolo OR-NOT;
  - convertire ogni porta AND in NOR con simbolo NOT-AND;
  - inserire una porta NOT (in forma NOR) per bilanciare i pallini.

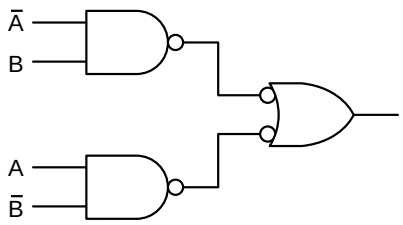
Universalità: Un Tecnica Generale

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

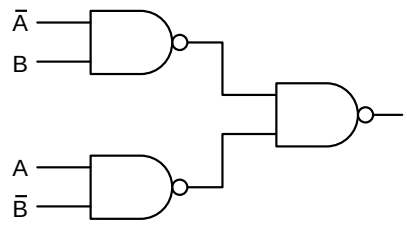
(a)



(b)



(c)

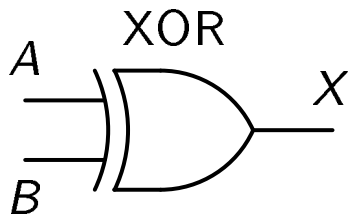


(d)

- La tabella di verità di una funzione (Figura (a)).
- L'implementazione AND-OR (Figura (b)).
- L'implementazione NAND (Figure (c) e (d)).

Porte XOR e XNOR

- Con **XOR** e **XNOR** indicheremo sia porte logiche che operazioni binarie sull'Algebra di Boole (indicate con $\oplus$ e $\odot$).
- $A \oplus B$ è definito come $A\overline{B} + \overline{A}B$
- Intuitivamente, $A \oplus B = 1$ se una sola tra A e B vale 1.



A	B	X	$\overline{X}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

$$X = A \oplus B \quad \overline{X} = A \odot B$$

- L'operatore XNOR è il complemento dello XOR:

$$A \odot B = \overline{A \oplus B} = AB + \overline{A}\,\overline{B}$$

- Intuitivamente, $A \odot B = 1$ se A e B hanno valori coincidenti.
- La porta logica XNOR si indica con un pallino all'uscita della porta XOR.

Porte XOR e XNOR

- Le seguenti identità valgono per l'operatore XOR:

$$X \oplus 0 = X \qquad X \oplus 1 = \overline{X}$$
$$X \oplus X = 0 \qquad X \oplus \overline{X} = 1$$
$$X \oplus \overline{Y} = \overline{X} \oplus Y \quad \overline{X} \oplus Y = X \oplus \overline{Y}$$

  che possono essere verificate con la teballa di verità o con la corrispondente espressione booleana.

- Inoltre, l'operazione $\oplus$ gode delle proprietà associativa e commutativa. Quindi:

$$A \oplus B = B \oplus A$$
$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

Funzioni Pari e Dispari

- Generalizzando l'operazione XOR a più ingressi, l'espressione booleana:

$$E_n = A_1 \oplus A_2 \oplus \ldots \oplus A_n.$$

  vale 1 quando il numero di variabili cui si assegna 1 è dispari.
- La funzione di verità catturata da E_n è detta **funzione dispari**.
- Se si osserva la mappa di Karnaugh per la funzione dispari, si nota subito che la sua implementazione in somma di prodotti (o in prodotto di somme) risulta estremamente inefficiente.
  - Serve una quantità esponenziale di porte AND, OR e NOT.
  - Utilizzando la porta XOR, basta una quantità lineare di porte.
- Il complemento della funzione dispari è detto **funzione pari** e può facilmente implementata usando la porta XNOR.
- Le funzioni per pari e dispari risultano molto utili nei sistemi che richiedono codici a controllo e correzione di errore (e.g., la generazione e il controllo di parità).

Reti Combinatorie Elementari

- Finora, abiamo visto come si implementa tabelle dei verità e altri semplici circuiti utilizzando singole porte logiche.
- In realtà, i componenti elementari sono generalmente rappresentati dai moduli contenenti un certo numero di porte.
- Analizzeremo ora questi blocchi elementari e vedremo come vengono utilizzati.

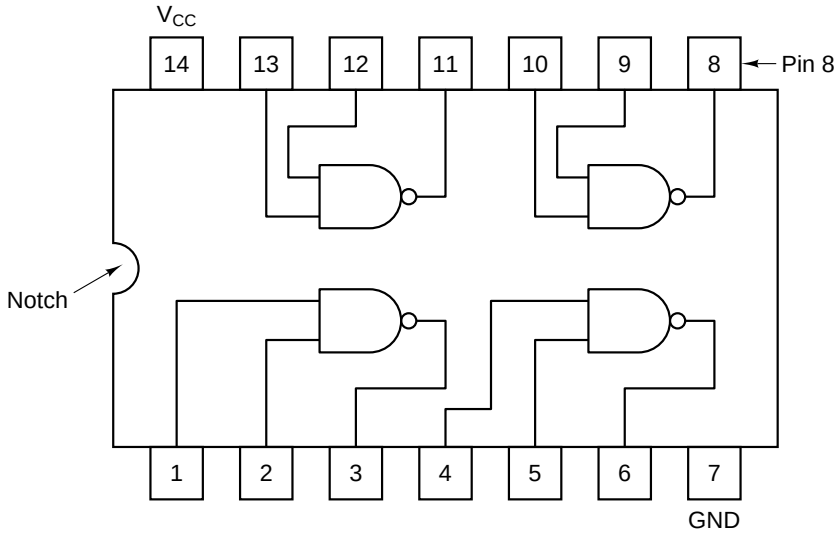
Circuiti Integrati

- Le porte logiche non sono vendute individualmente, ma in unità chiamate **circuiti integrati** (cui spesso ci si riferisce con i termini **IC** oppure **chip**).
- Un IC è un quadrato di silicio di lato pari a $5mm$, su cui sono posizionate alcune porte logiche.
- Uno o più IC sono poi inseriti in contenitori rettangolari di plastica o ceramica.
- Sul lato più lungo vi sono due file parallele di contatti (pin), inseribili in una presa o saldabili su schede di circuiti stampati.
- Ciascun pin è collegato a un ingresso o a un'uscita di una porta logica oppure all'alimentazione o alla terra.

Circuiti Integrati

- I chip possono essere classificati in base al numero di porte che contengono:
  - Circuiti **SSI** hanno da 1 a 10 porte.
  - Circuiti **MSI** hanno da 10 a 100 porte.
  - Circuiti **LSI** hanno da 100 a 100.000 porte.
  - Circuiti **VLSI** hanno più di 100.000 porte.
- Per gli scopi del corso le porte logiche sono **ideali**, nel senso che l'uscita appare non appena viene applicato il segnale in ingresso.
  - In realtà i chip hanno un ritardo temporale finito, chiamato **ritardo della porta**.

Chip SSI: Esempio

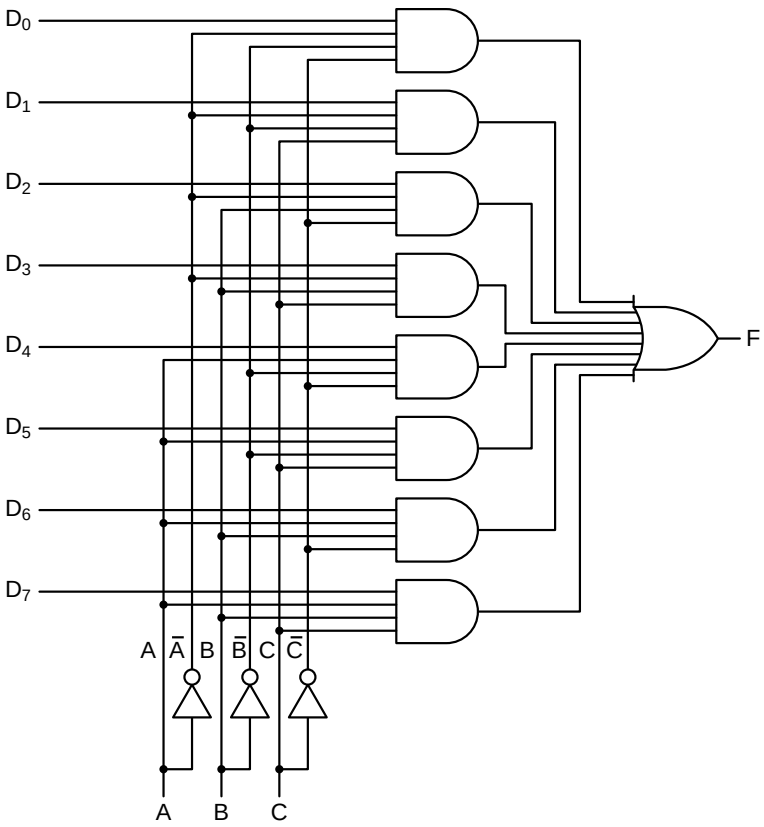


- La figura mostra lo schema di un comune chip SSI con quattro porte NAND.
- Siccome ogni porta ha due ingressi e un'uscita, sono necessari 12 pin per le porte.
- Inoltre, il chip necessita della tensione (V_{cc}) e della terra (GND) che sono condivise da tutte le porte.

Multiplexer

- Un **multiplexer** è una rete combinatoria con:
  - 2^n ingressi per i dati;
  - n ingressi di controllo;
  - una singola uscita.
- Gli input di controllo permettono di **selezionare** uno tra i possibili ingressi per i dati:
  - ad ogni sequenza di valori di verità lunga n corrisponde un numero binario compreso tra 0 e $2^n - 1$.
- Il dato selezionato dagli ingressi di controllo diventa l'unica uscita.
- L'inverso del multiplexer è il **demultiplexer** che:
  - ha un ingresso di dato, n ingressi di controllo, e 2^n uscite;
  - redirige l'unico ingresso verso una delle uscite in base ai valori degli ingressi di controllo.

Multiplexer: Esempio



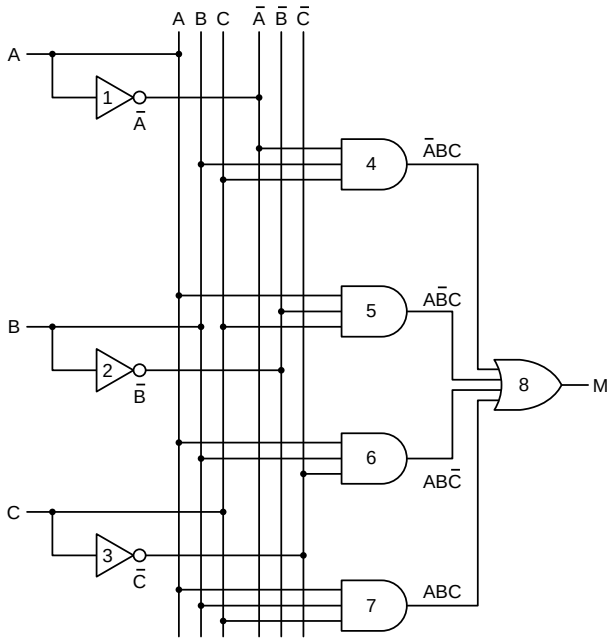
- La figura mostra l'implementazione di un multiplexer.
- Ogni dato d'ingresso è collegato a una porta AND.
- Solo una porta AND è abilatata secondo i segnali d' ingresso.
- La porta abilitata i genera l'output D_i che diventa anche l'output del multiplexer.
- E.g., se $A = B = C = 0$:
  - la prima porta AND è attivata, producendo l'uscita D_0;
  - tutte le altre porte sono disattivate, producendo l'uscita 0;
  - quindi, $F = D_0$.

Multiplexer: Esempio

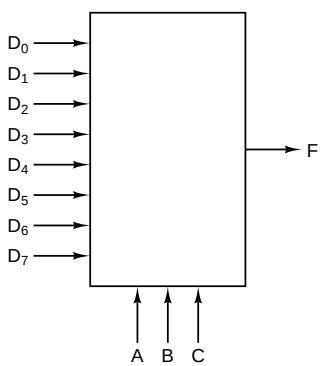
- Assegnando valori costanti agli ingressi per i dati, si può costruire una rete combinatoria che implementi qualunque funzione booleana di n variabili.
- E.g., si consideri la funzione definita dalla tabella di verità di sotto. Nell'implementazione tramite un multiplexer (la figura a destra):
  - le variabili della funzione sono presenti agli ingressi di controllo;
  - i valori della funzione sono presenti (V_{cc}/la terra per i valori 1/0) agli ingressi dei dati, in corrispondenza di ciascuna combinazione di A, B, C nella tabella di verità.

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

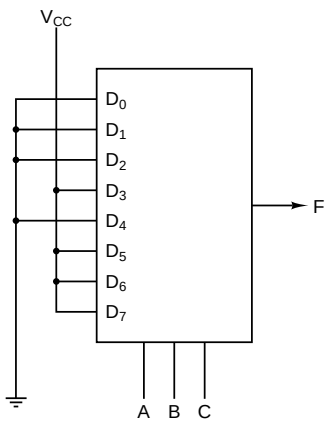
(a)



(b)



(a)

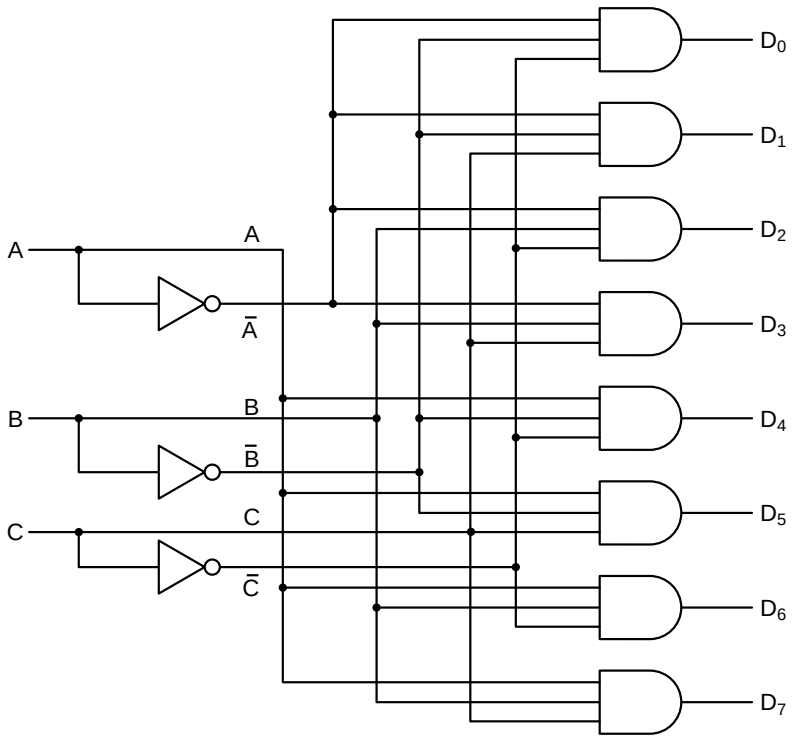


(b)

Decodificatori

- Un **decodificatore** (decoder) è una rete combinatoria con:
  - n ingressi;
  - 2^n uscite.
- Le uscite corrispondono ai 2^n mintermini di n variabili.
- Quindi, i valori in ingresso vengono utilizzati per selezionare una tra le uscite, che sarà l'unica ad essere impostata ad 1.
- Siccome qualunque funzione può essere espressa in forma di somma di mintermini, i decoder sono molto utili nella progettazione dei circuiti.
- Un decoder può utilizzare un **ingresso di abilitazione** (enable) E. Tale decodificatore è attivato:
  - quando $E = 1$, nel qual caso opera normalmente;
  - disattivato quando $E = 0$, nel qual caso genera 0 su tutte le uscite, qualunque sia il valore applicato agli ingressi.

Decodificatori: Esempio



- La figura mostra l'implementazione di un decoder.
- Ogni porta AND implementa un mintermine.
- E.g., se $A = B = C = 0$:
  - la prima porta AND ha il valore 1 all'uscita;
  - tuttle le altre porte AND hanno il valore 0 alle sue uscite.

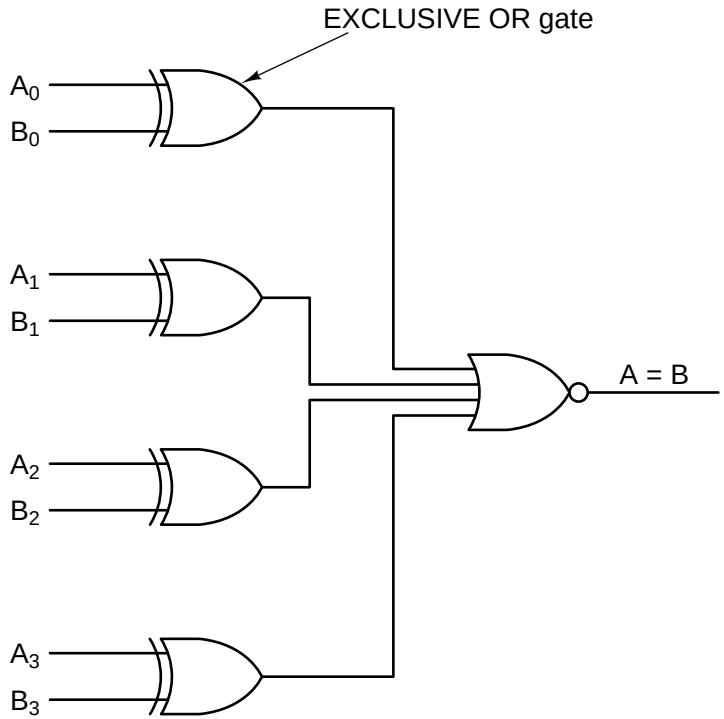
Codificatori

- L'inverso del decodificatore è il **codificatore** (encoder), con 2^n ingressi e n uscite.
- Le linee di uscita danno un codice binario corrispondente al valore di ingresso.
- E.g., con un 8x3 codificatore, l'output che corrisponde all'input 00000001 è 001.
- Si assume che, in qualunque conidizione, soltanto un ingresso valga 1.
- Allora, cosa succede se tutti gli ingressi sono uguali a 0? oppure se ci sono 2 ingressi con il valore 1?
- L'ambiguità viene risolta aggiungendo un'uscita con il compito di indicare che almeno uno degli ingressi sia uguale a 1.
  - Se due o più ingressi sono uguali a 1, ha precedenza l'ingresso con **priorità** maggiore.

Comparatori

- Un **comparatore** è una rete combinatoria con:
  - 2 sequenze di ingresso lunghe n (in totale $2n$ ingressi);
  - 1 uscita.
- Il valore dell'uscita è:
  - 0 se le due sequenze in input sono identiche;
  - 1 se le due sequenze sono diverse.
- È possibile realizzare un comparatore usando dei gate **XOR**.
  - Sappiamo che il gate XOR a 2 ingressi ha come output:
    - 0 se i due ingressi sono diversi;
    - 1 se i due ingressi sono uguali.
- Quindi, possiamo usare:
  - 1 gate XOR per confrontare 1 coppia di bit;
  - n gate XOR per confrontare n coppie bit;
  - la somma (OR) delle uscite degli n gate XOR per verificare che esse siano tutte a 0 (cioè se tutte le coppie sono uguali).
    - Solo in questo caso (sequenze uguali), OR restituisce 0.
  - In questo modo, la negazione della somma (NOR) restituisce 1 se i due ingressi sono uguali.

Comparatori: Esempio



- Ecco un comparatore a 4 bit, realizzato con 4 XOR e un NOR.
- Se le due parole sono uguali, le uscite degli XOR sono tutte 0.
  - Se tutte le uscite degli XOR sono 0, il NOR restituisce 1 ($A = B$).
- Se almeno una coppia di bit è diversa, e.g., $A_2 \neq B_2$, esiste una uscita di XOR a 1.
  - In questo caso, il NOR restituisce 0 ($A \neq B$).

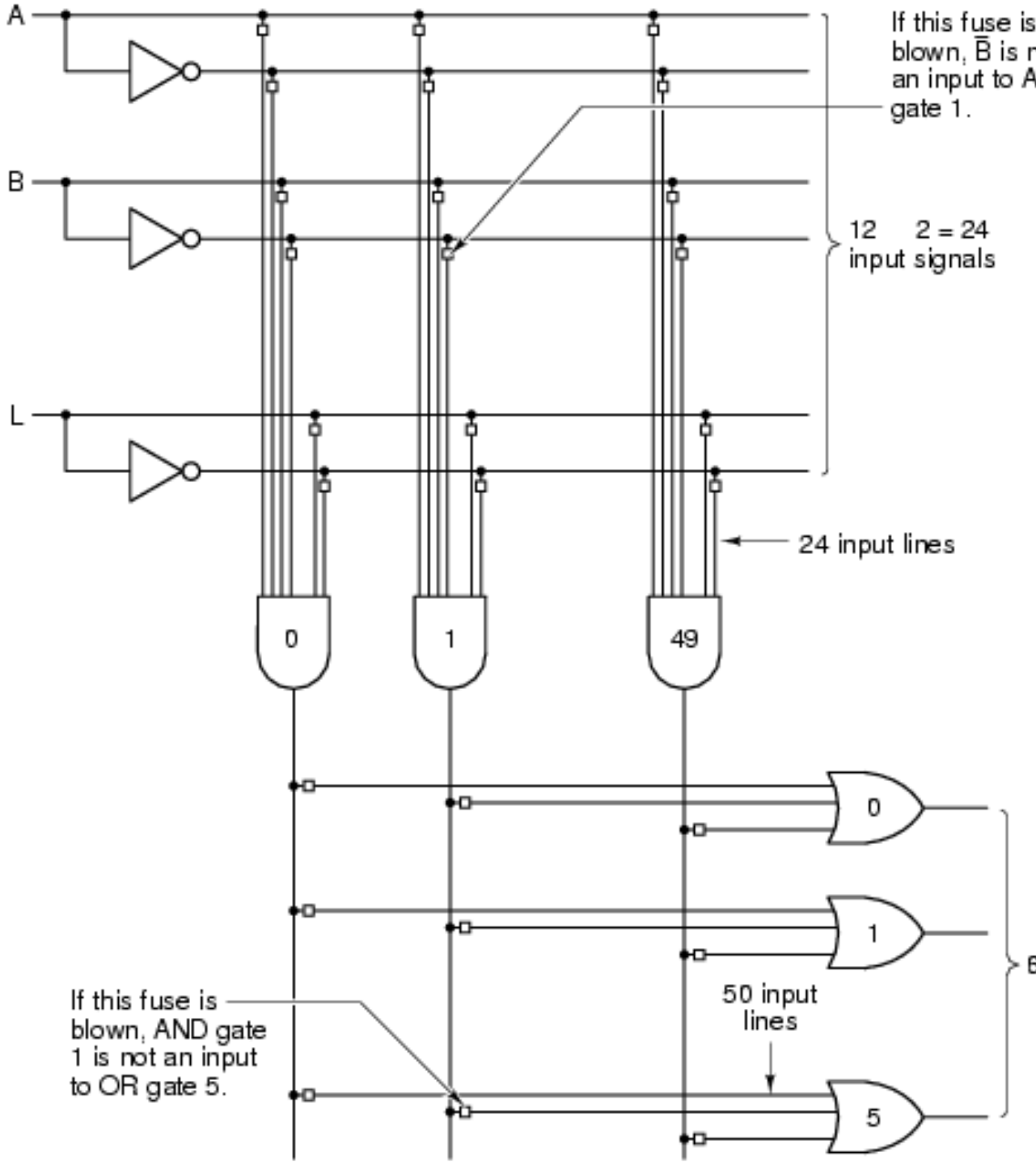
Array Logici Programmabili (PLA)

- L'**Array Logico Programmabile** (PLA) è un chip generale che permette di calcolare funzioni come **somme di prodotti**.
- Per fare ciò, il PLA usa dei gate AND e OR, più dei gate NOT per ottenere tutti i possibili letterali, prodotti e somme dagli ingressi.
- È **programmabile** perché i collegamenti tra letterali, gate AND, e gate OR, sono modificabili (una volta sola).
- Quando il PLA è nuovo:
  - tutti i letterali sono collegati a tutti gli AND;
  - tutte le uscite degli AND sono collegate a tutti gli OR.
- I collegamenti sono effettuati con dei **fusibili**.
- I fusibili possono essere bruciati per interrompere alcuni collegamenti.
- Quindi, per programmare un PLA, bisogna decidere quali fusibili bruciare, e quali no.

Struttura Interna dei PLA

- Un generico PLA ha k ingressi e può realizzare m funzioni.
  - Per ciascuna funzione abbiamo un segnale in uscita.
  - Quindi, il PLA ha in totale $k + m + 2$ pin (k ingressi, m uscite, 2 pin per tensione e terra).
- Internamente, il PLA contiene:
  - k porte logiche NOT, una per ciascun ingresso;
  - n porte logiche AND, una per ogni prodotto;
  - m porte logiche OR, una per ciascuna uscita.
- Ciascun prodotto coinvolge un certo numero di letterali.
  - Abbiamo $2 \times k$ possibili letterali distinti.
  - Quindi gli AND hanno $2 \times k$ ingressi ciascuno.
- Ciascuna funzione coinvolge un certo numero di prodotti.
  - Abbiamo n AND nel PLA.
  - Quindi, gli OR hanno n ingressi ciascuno.
- Vediamo un esempio di PLA con:
  - $k = 12$ ingressi (A, B, ..., L);
  - $n = 50$ possibili prodotti;
  - $m = 6$ uscite (0, 1, ..., 5).

- A sinistra abbiamo gli 12 ingressi A, B, . . . , L.
- A destra in basso abbiamo 6 uscite.
- A metà figura si vede l'array di 50 AND.
- Ciascun AND ha 24 linee di ingresso.
  - I quadratini bianchi sono i fusibili.
  - Se il fusibile indicato in alto è bruciato, $\overline{B}$ non è un input all'AND 1.
- Cascun OR ha 50 linee di ingresso.
  - Se il fusibile sotto è bruciato, AND 1 non è un input a OR 5.



PLA: Esempio

- Con un singolo PLA è possibile realizzare:
  - m somme
  - di al più n prodotti (in totale)
  - di al più k letterali.
- E.g., la funzione $F = ABC + \overline{A}BC + A\overline{B}C + AB\overline{C}$:
  - contiene una somma di 4 prodotti;
  - usa 3 diverse variabili (6 letterali).
- Può essere quindi realizzata con un PLA come quello visto prima, utilizzando:
  - 3 delle 12 ingressi;
  - 4 delle 50 porte AND (attivando solo le connessioni che corrispondono ai 4 prodotti);
  - una delle 6 porte OR (attivando solo la connessione che corrispondo alla somma).
- Si noti che, nello stesso PLA, si possono realizzare fino a 6 funzioni di questo tipo su 12 variabili distinte.

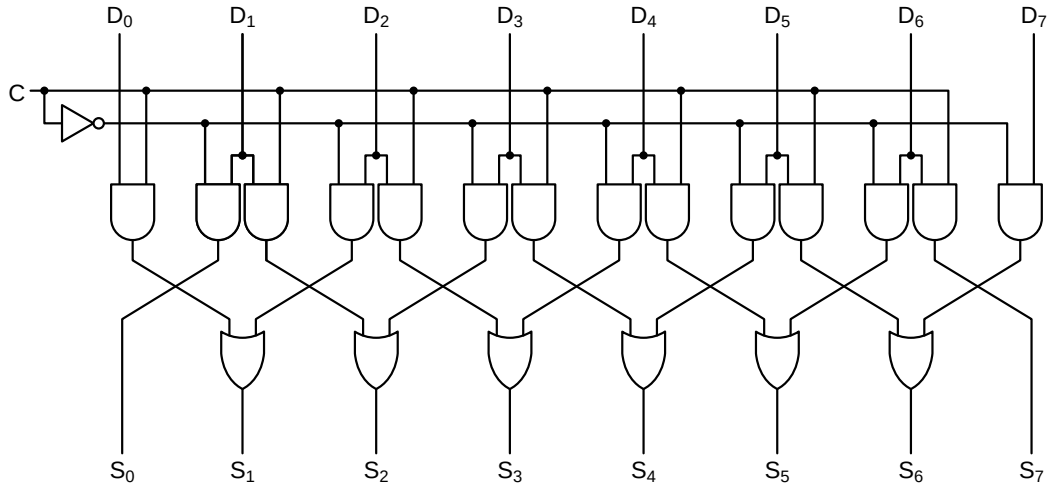
Circuiti per L'aritmetica

- Analizzeremo ora le reti combinatorie impiegate per eseguire calcoli aritmetici.

Registri a Scorrimento

- Il **Registro a Scorrimento** (**shifter**) è una rete combinatoria con:
  - n ingressi per i dati;
  - un ingresso di controllo;
  - n uscite.
- L'idea è quella di far scorrere **verso destra** o **verso sinistra** i valori in input.
  - Si inserisce un valore nuovo nel bit più destra (sinistra) e poi si spostano tutti gli altri bit di una posizione a sinistra (destra).
- E.g., $A_3A_2A_1A_0 \rightarrow 0A_3A_2A_1$ (right shift)
  $A_3A_2A_1A_0 \rightarrow A_2A_1A_00$ (left shift)
- L'input di controllo serve a determinare se lo spostomento debba avvenire verso destra o verso sinistra.

Registri a Scorrimento: Esempio



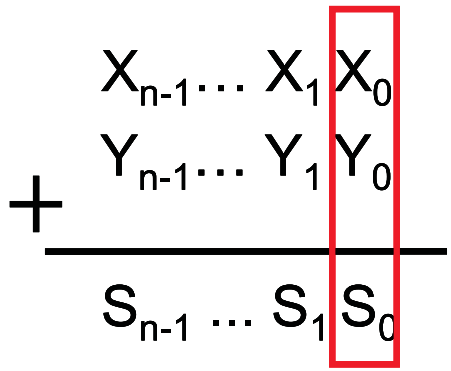
- La figura mostra l'implementazione di uno shifter.
- Per tutti i bit c'è una coppia di porte AND tranne che per le porte all'inizio e alla fine.
- $C = 1$: la porta a destra di ciascun D_i viene abilitata e quella a sinistra disabilitata, passando D_i a destra e 0 all'inizio.

  $S_0 = 0, S_1 = D_0 + 0, S_2 = D_1 + 0, S_3 = D_2 + 0, S_4 = D_3 + 0, S_5 = D_4 + 0, S_6 = D_5 + 0, S_7 = D_6 + 0$
- $C = 0$: la porta a sinistra di D_i viene abilitata e quella a destra disabilitata, passando D_i a sinistra e 0 alla fine.

  $S_0 = D_1 + 0, S_1 = D_2 + 0, S_2 = D_3 + 0, S_3 = D_4 + 0, S_4 = D_5 + 0, S_5 = D_6 + 0, S_6 = D_7 + 0, S_7 = 0$

Sommatore (Adder)

- Un sommatore binario è una rete combinatoria che dà in uscita la somma aritmetica di due numeri binari di n bit:



- Due blocchi logici di base permettono la realizzazione di un sommatore:
  - Half adder
  - Full adder
- essi riguardano la somma in una colonna.

Semisommatore (Half Adder)

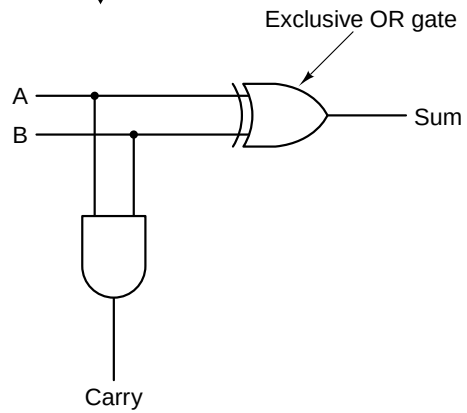
- Un **half adder** è una rete combinatoria con:
  - due ingressi;
  - due uscite.
- La prima uscita rappresenta la **somma** dei due bit in ingresso.
- La seconda uscita rappresenta il **riporto** generato dalla somma dei due bit in ingresso.

$$S = A \oplus B$$

$$C = AB$$

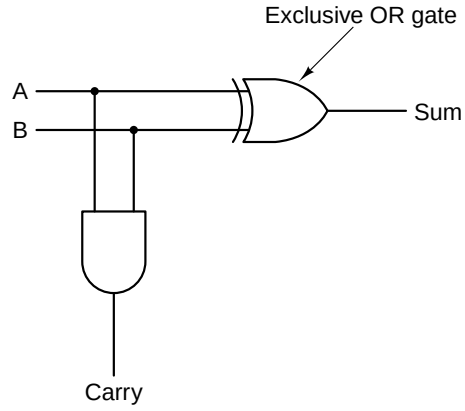
$\Uparrow$ $\Downarrow$

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Half Adder

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



- L'half adder non è in grado di eseguire correttamanete la somma di una sequenza di n bit (con $n \geq 1$), dato che:
  - non riesce a gestire il riporto che arriva dalle posizioni precedenti;
- Per far ciò è necessario un full adder.

Full Adder

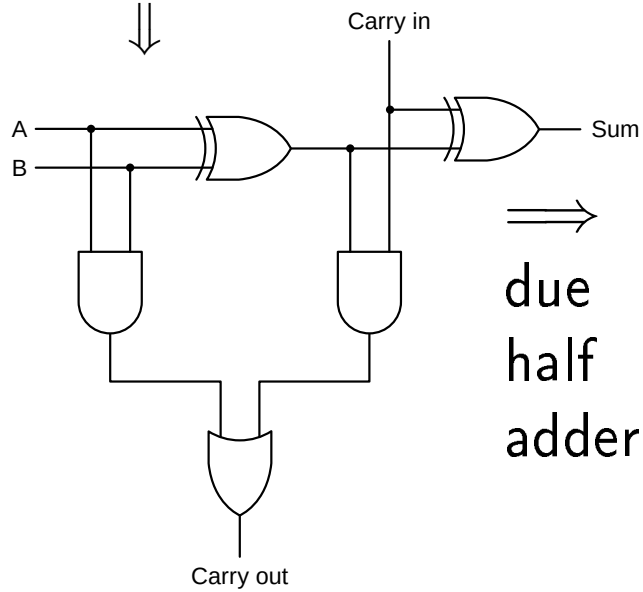
- Un **full adder** è una rete combinatoria con:
  - tre ingressi;
  - due uscite.
- La prima uscita rappresenta la **somma** dei tre bit in ingresso: due operandi e un **riporto in entrata**.
- La seconda uscita rappresenta il **riporto** generato dalla somma dei tre bit in ingresso.

$$S = A \oplus B \oplus Cin$$

$$Cout = AB + Cin(A \oplus B)$$

$\Uparrow$ $\Downarrow$

A	B	Carry in	Sum	Carry out
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

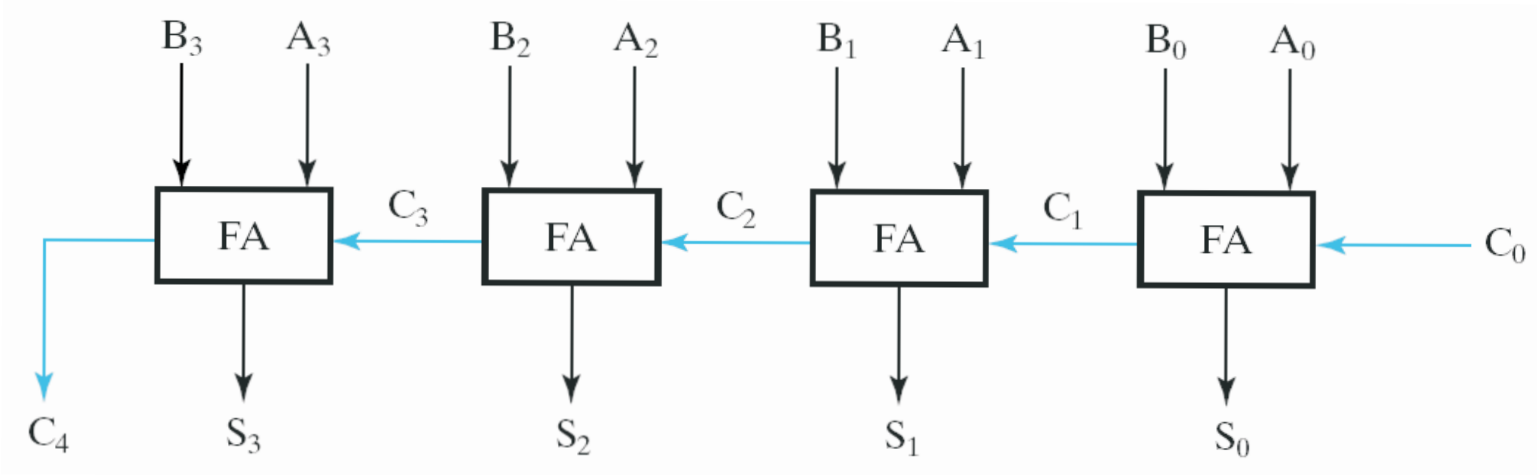


$\Longrightarrow$ due half adder

Sommatore a Propagazione di Riporto

- I full adder possono essere combinati per costruire un sommatore che può effettuare la somma tra due numeri binari di n bit: $A_{n-1}...A_1A_0+B_{n-1}...B_1B_0$.
- In questo modo, il sommatore utilizza n full adder in parallelo, ai cui ingressi sono applicati simultaneamente gli n bit di ciascun ingresso.
- I full adder sono connessi in cascata:
  - l'uscita riporto dell'uno è connessa all'ingresso riporto del successivo.
- Parliamo in questo caso di **sommatore a propagazione di riporto** dato che:
  - la somma non può essere completata finché il riporto non si sia propagato dal primo full adder all'ultimo.

Sommatore a Propagazione di Riporto: Esempio



- La figura mostra un sommatore a 4 bit che utilizza 4 full adder.
- Ogni full adder i ha:
  - come ingressi A_i e B_i (le variabili della colonna i-esima della somma), e il riporto C_i che viene dal full adder precedente;
  - come uscite S_i (la somma) e C_{i+1} (il riporto) che va al full adder successivo.
- Il riporto in ingresso al primo adder è fissato a 0.
- La somma è ottenuta dalla sequenza $C_4 S_3 S_2 S_1 S_0$.

Sommatore a Selezione di Riporto

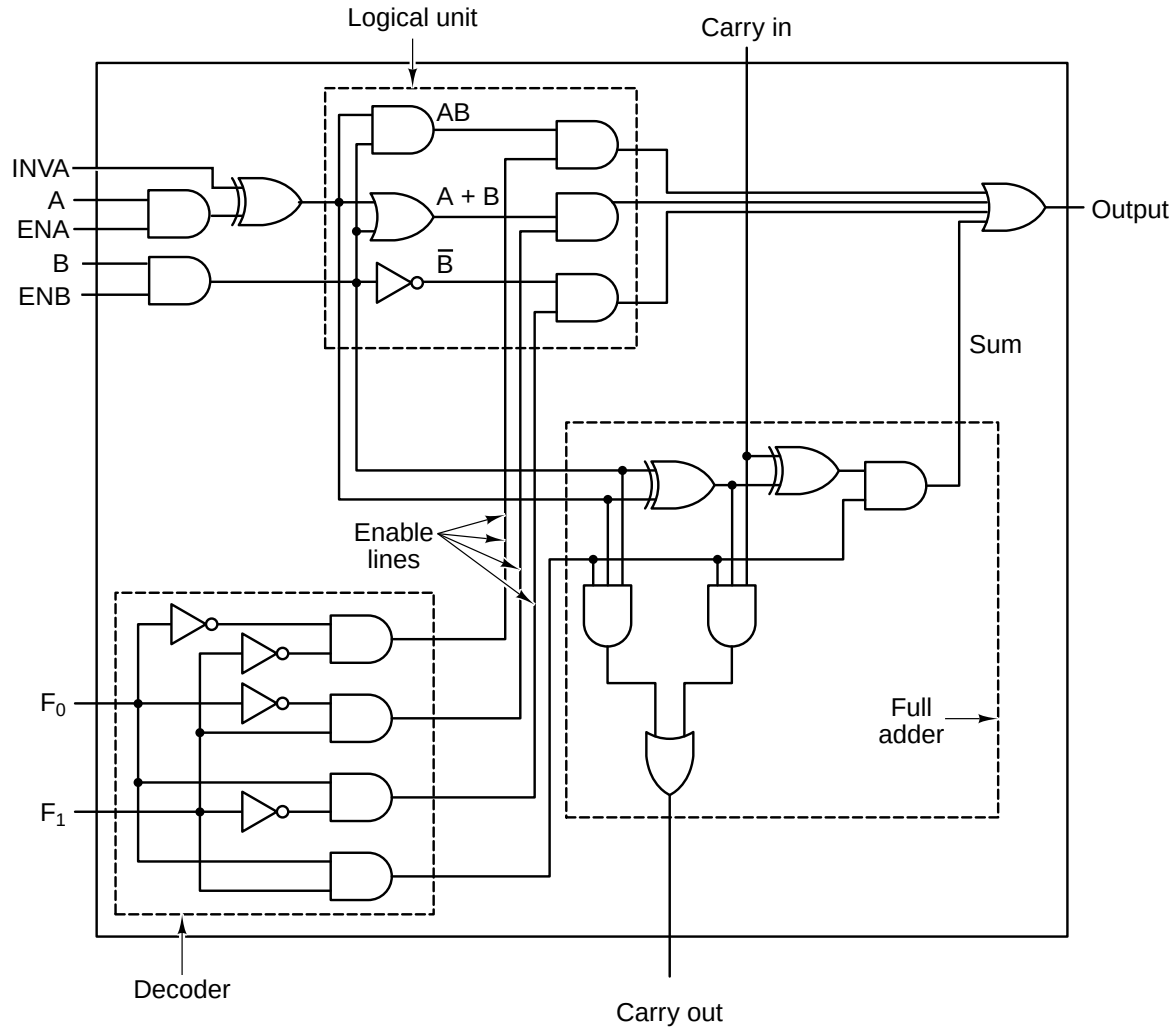
- Il sommatore già descritto è caratterizato da un ritardo di risposta dovuto alla propagazione dei riporti.
- Per fare un sommatore più veloce, possiamo dividere un sommatore a n bit in uno per gli $n/2$ bit meno significativi (T_2) e in un altro per gli $n/2$ bit più significativi (T_1).
- Poi, manteniamo due coppie di T_1 (T_{11} e T_{12}).
- A T_{11} viene collegato il valore 0 come riporto, mentre a T_{12} viene collegato il valore 1.
- Tutti e tre i sommatori possono ora cominciare nello stesso momento, anche se uno solo dei T_{11} darà il risultato corretto.
- Appena T_2 genera il riporto finale, è possibile selezionare il sommatore corretto fra T_{11} e T_{12}.
- Questo trucco permette di dimezzare il tempo richiesto e viene chiamato **selezione di riporto**.

Unità Aritmetico-Logiche

- Come sappiamo, la CPU contiene un'unità funzionale, chiamata **ALU** o **Unità Aritmetico-Logica**.
- L'ALU è in grado di effettuare operazioni sia logiche che aritmetiche. In particolare, può calcolare le operazioni seguenti tra 2 variabili booleane A e B:
  1. AB
  2. $A + B$ (OR)
  3. $\overline{B}$
  4. $A + B$ (somma aritmetica)
- Internamente, l'ALU è una rete combinatoria con:
  - n ingressi per i dati;
  - m ingressi di controllo;
  - k uscite.
- Gli ingressi di controllo servono a selezionare quale tra le operazioni possibili vada applicata ai dati.

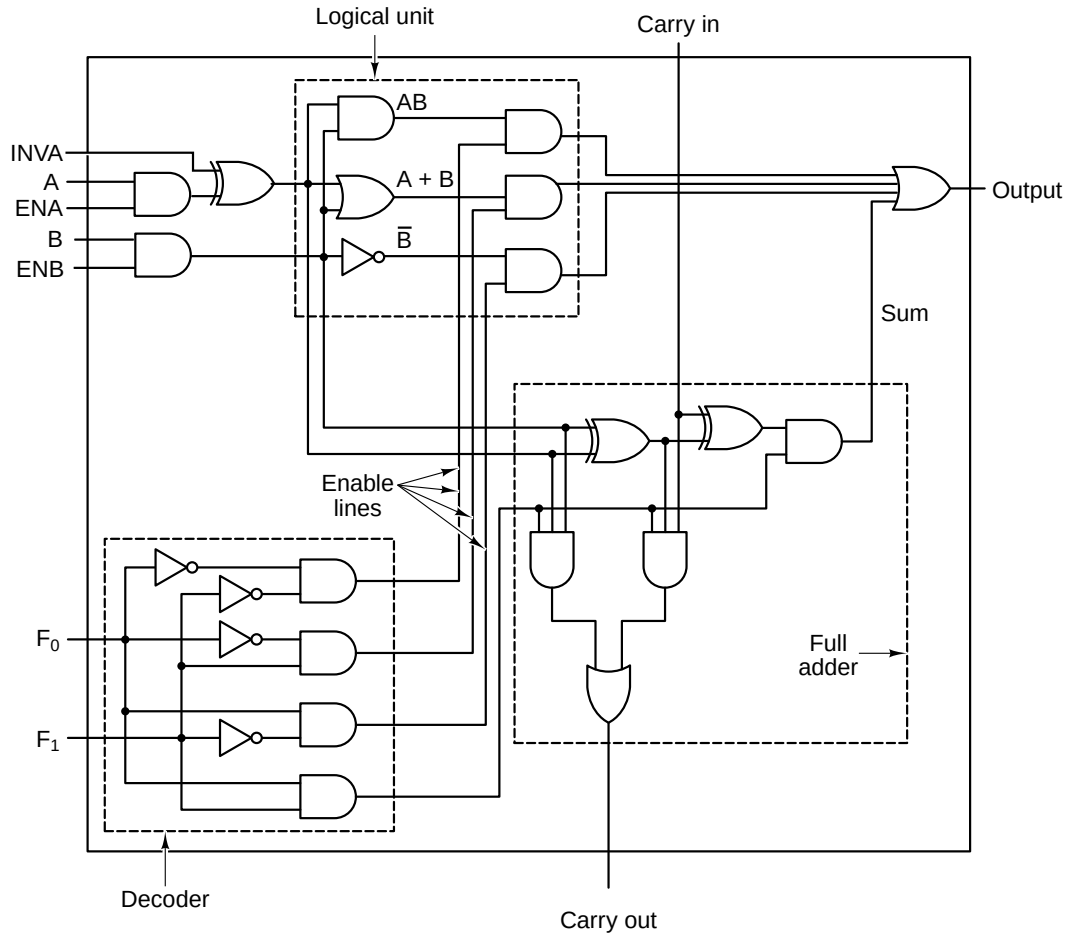
Una ALU a 1 Bit

- La figura mostra l'implementazione di un' ALU a 1 bit.
- Il circuito è composto da 3 parti:
  1. il settore in basso a sinistra (l'unità di selezione);
  2. il settore in alto a sinistra (l'unità logica);
  3. il settore in basso a destra (l'unità aritmetica).
- Ora analizziamo ogni parte...



Una ALU a 1 Bit

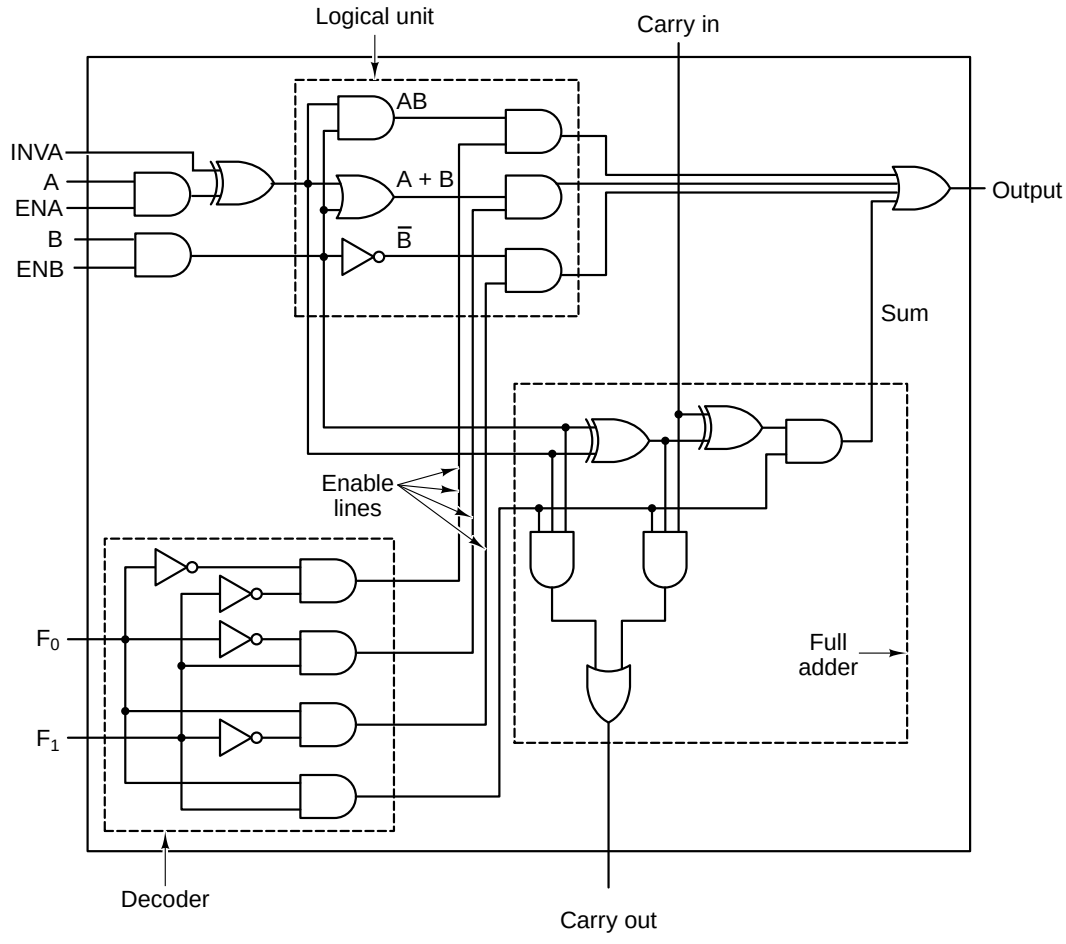
1. L'unità di selezione
   - Questa parte contiene un decoder a 2 bit.
   - In base ai segnali di controllo F_0 e F_1, il decoder genera i segnali di attivazione.
   - Secondo i valori di F_0 e F_1, una sola uscita del decoder vale 1.
   - Tale uscita attiva solo una delle operazioni AB, $A + B$ (OR), $\overline{B}$, e $A + B$ (somma).
   - Attraverso la porta OR, il risultato dell'operazione scelta passa all'output.



Una ALU a 1 Bit

2. L'unità logica

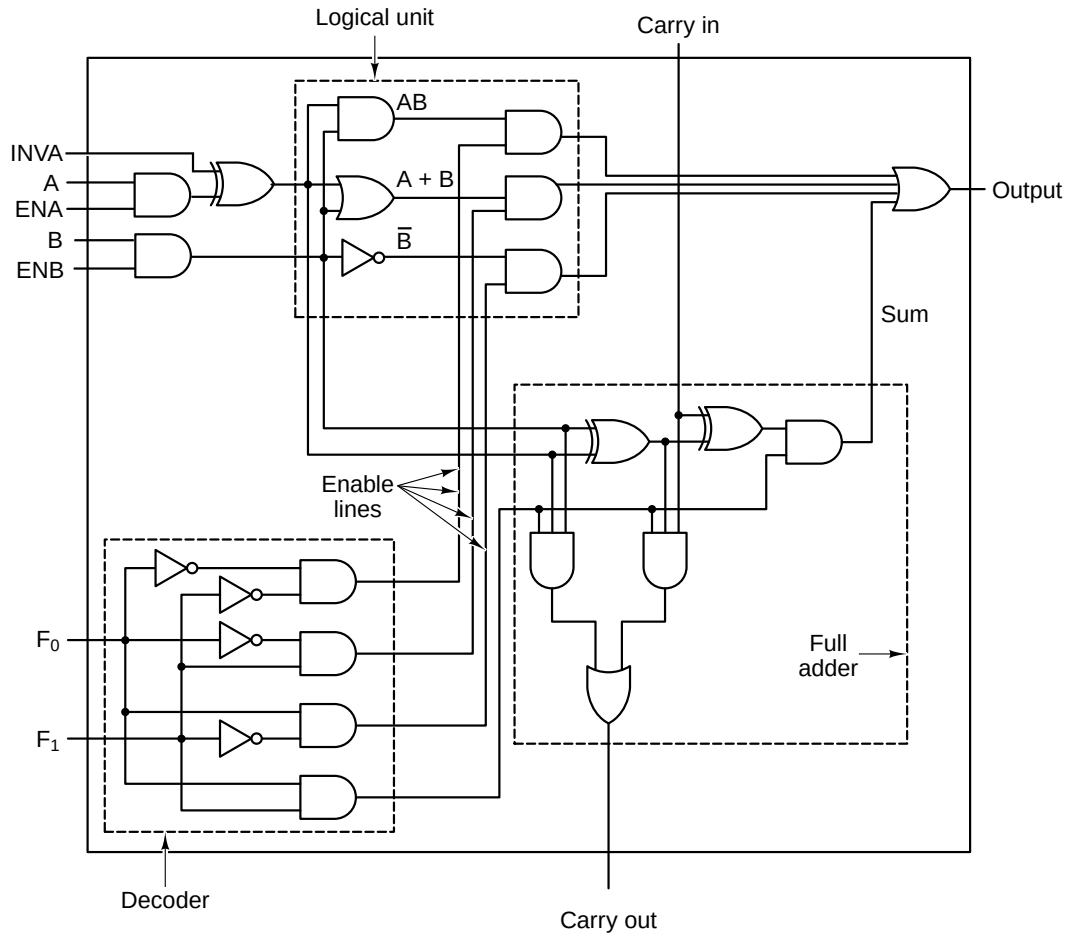
- Questa parte contiene le porte AND, OR, e NOT per calcolare AB, $A + B$, e $\overline{B}$.
- Le prime 3 uscite del decoder (E_0, E_1, E_2) attivano le porte.
  - Ciascuna porta è collegata a una porta AND.
  - Ciascuna di queste porte AND ha anche E_i come ingresso.
  - Quindi, una porta AND passa il valore dell'operazione corrispondente solo se $E_i = 1$.



Una ALU a 1 Bit

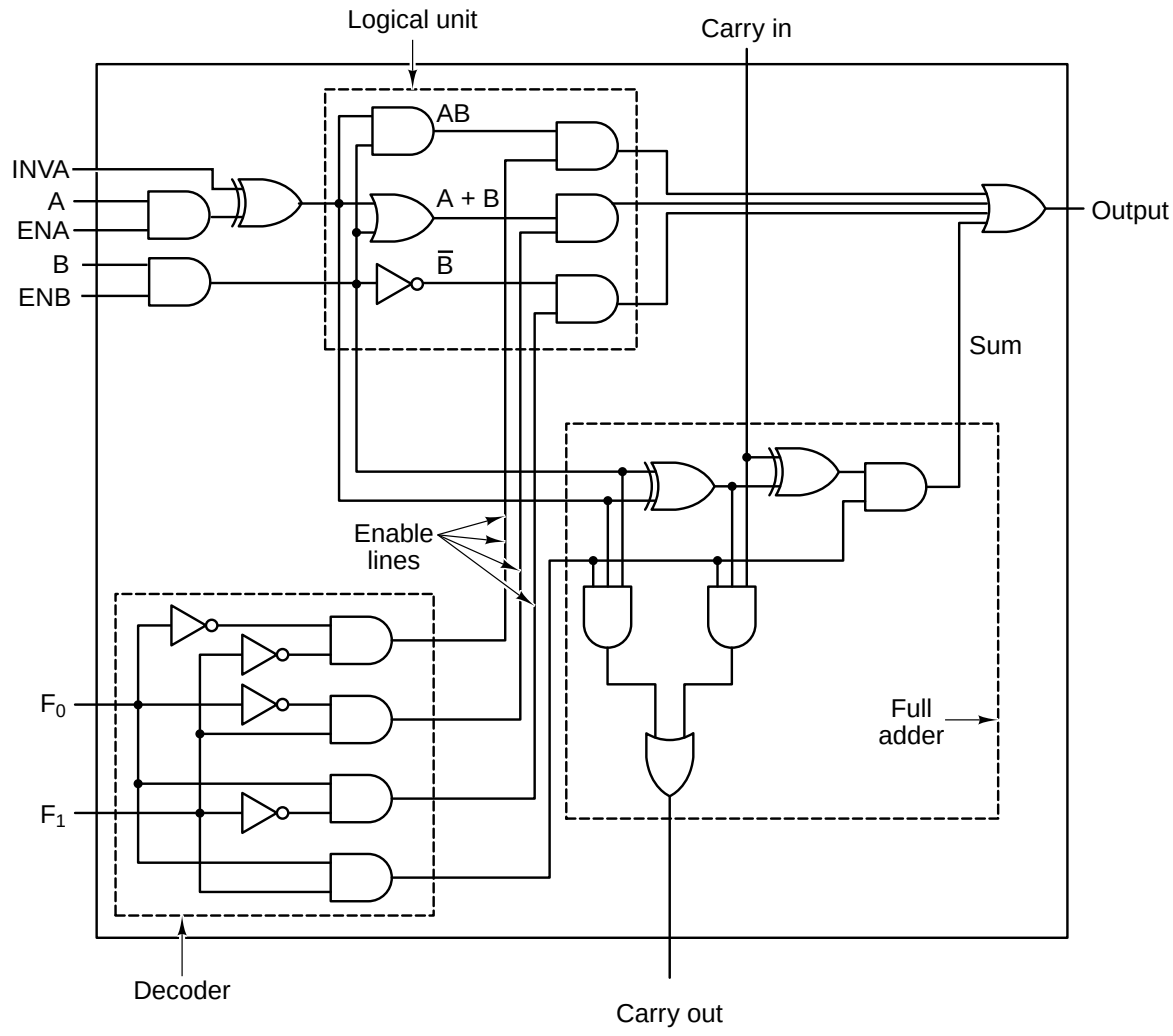
3. L'unità aritmetica

- Questa parte contiene un full adder per calcolare la somma $A + B$.
- L'ultima uscita del decoder (E_3) attiva il full adder.
  - E_3 è collegata alle porte AND.
  - La porta XOR è collegata a un'altra porta AND.
  - Questa porta AND ha anche E_3 come ingresso.
  - Quindi, il full adder funziona normalmente solo se $E_3 = 1$.

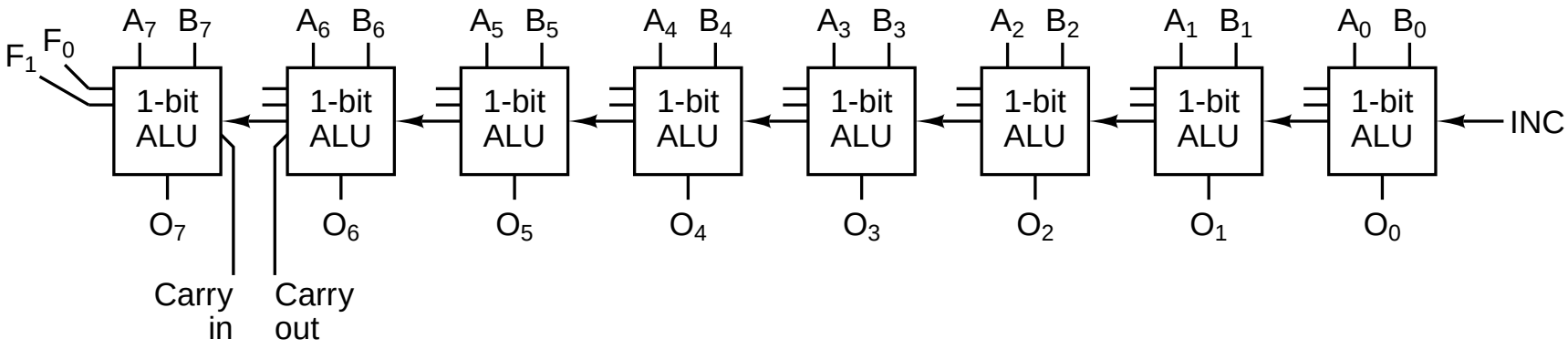


Una ALU a 1 Bit

- Oltre ad avere A e B come input, è anche possibile:
  - forzare A/B a 0 negando ENA/ENB e usando le porte AND;
  - ottenere $\overline{A}$ abilitando INVA e usando la porta XOR.



Una ALU a 8 Bit



- L'ALU a n bit è costruita unendo n ALU a 1 bit, ai cui ingressi di dati sono applicati simultaneamente gli n bit di ciascun ingresso.
- La figura mostra una ALU a 8 bit.
- Le ALU sono connesse in cascata:
  - l'uscita riporto dell'una è connessa all'ingresso riporto della successiva.
- Il riporto in ingresso (INC) alla prima ALU è fissato a 1 solo in certi casi, per rendere possibili operazioni come $A + 1$, $A + B + 1$ ...