

LSS: Reti Logiche: circuiti combinatori

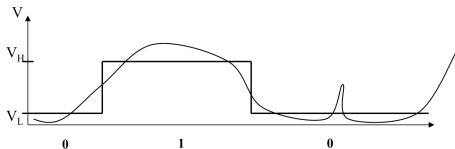
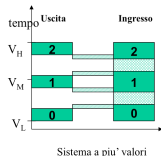
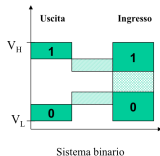
Piero Vicini

A.A. 2017-2018

Argomenti:

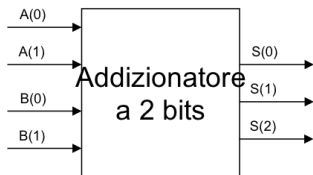
- Codici e aritmetica
- Operatori dell'algebra booleana
- Minimizzazione e sintesi di funzioni
- Esempi di implementazione hardware di circuiti combinatori

- Computers operano con segnali elettrici con valori discreti di potenziale
- Significativi soltanto 2 valori (intervalli) di potenziale elettrico (high/low, true/false, 1/0)
- Gli elementi elettronici binari sono semplici per definizione (interruttori...)
- Algebra di Boole permette di modellare il funzionamento dei circuiti elettronici binari.



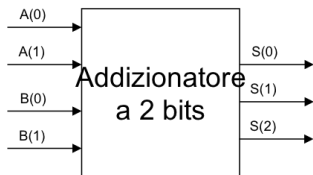
- Un *blocco logico* e' un circuito elettronico con linee in input e output a cui sono associate variabili binarie.
- Il circuito calcola *funzioni logiche* come combinazioni di operazioni algebriche booleane sulle variabili in input.
- Due classi: circuiti *combinatori* e circuiti *sequenziali*

- Un *blocco logico* e' un circuito elettronico con linee in input e output a cui sono associate variabili binarie.
- Il circuito calcola *funzioni logiche* come combinazioni di operazioni algebriche booleane sulle variabili in input.
- Due classi: circuiti *combinatori* e circuiti *sequenziali*

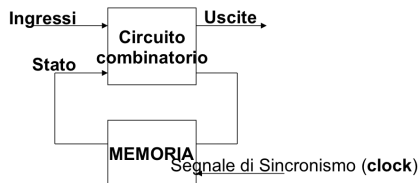


L'uscita di un circuito combinatorio e' determinata completamente dal valore istantaneo della combinazione degli ingressi

- Un *blocco logico* e' un circuito elettronico con linee in input e output a cui sono associate variabili binarie.
- Il circuito calcola *funzioni logiche* come combinazioni di operazioni algebriche booleane sulle variabili in input.
- Due classi: circuiti *combinatori* e circuiti *sequenziali*



L'uscita di un circuito combinatorio e' determinata completamente dal valore istantaneo della combinazione degli ingressi



Un circuito sequenziale e' un sistema composto da circuiti combinatori e *elementi di memoria* in cui le uscite sono una funzione del valore degli ingressi e dello *stato passato* del circuito.

- La funzione logica di un circuito combinatorio e' completamente descritta e specificata da una *tavola della verita'*

- Dati n bits di ingresso, il numero delle configurazioni possibili degli ingressi e' 2^n
- La tavola possiede quindi 2^n righe con valore delle uscite per quella particolare combinazione degli ingressi.

| A(1) | A(0) | B(1) | B(0) | C(2) | C(1) | C(0) |
|------|------|------|------|------|------|------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 |

- La rappresentazione decimale di numeri interi positivi e' una rappresentazione **posizionale** in base **10**

$$(288)_{10} = 2 * 10^2 + 8 * 10^1 + 8 * 10^0$$

- il cui intervallo va da 0 a $10^N - 1$, con N uguale all'estensione della rappresentazione

- La rappresentazione decimale di numeri interi positivi e' una rappresentazione **posizionale** in base **10**

$$(288)_{10} = 2 * 10^2 + 8 * 10^1 + 8 * 10^0$$

- il cui intervallo va da 0 a $10^N - 1$, con N uguale all'estensione della rappresentazione

- In perfetta equivalenza anche la rappresentazione binaria e' posizionale ma in base **2**. Dato un numero a n bits:

$$(x)_2 = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

- dove l'intervallo e' $0 : 2^n - 1$
- A 32 bits l'intervallo va da 0 a +4.294.967.295

- Esempio:

$$\begin{aligned} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2 \\ & = 0 + \dots + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 \\ & = 0 + \dots + 8 + 0 + 2 + 1 = 11_{10} \end{aligned}$$

- Lo stesso ragionamento vale anche per la rappresentazione di numeri frazionari.
- In base 10:

$$(0.571)_{10} = \dots + 5 * 10^{-1} + 7 * 10^{-2} + 1 * 10^{-3}$$

- In base 2:

$$(0.00101)_2 = \dots + 0 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} + 0 * 2^{-4} + 1 * 2^{-5}$$

- La rappresentazione esadecimale e' un codice la cui base e' 16
- E' una rappresentazione compatta per stringhe di bit

| | | | | | | | |
|---|------|---|------|---|------|---|------|
| 0 | 0000 | 4 | 0100 | 8 | 1000 | c | 1100 |
| 1 | 0001 | 5 | 0101 | 9 | 1001 | d | 1101 |
| 2 | 0010 | 6 | 0110 | a | 1010 | e | 1110 |
| 3 | 0011 | 7 | 0111 | b | 1011 | f | 1111 |

- Ogni gruppo di 4 bits viene rappresentato da una cifra *Hex*
- Esempi:

C1A0 C1A0 = 1100 0001 1010 0000 1100 0001 1010 0000

- La rappresentazione esadecimale e' un codice la cui base e' 16
- E' una rappresentazione compatta per stringhe di bit

| | | | | | | | |
|---|------|---|------|---|------|---|------|
| 0 | 0000 | 4 | 0100 | 8 | 1000 | c | 1100 |
| 1 | 0001 | 5 | 0101 | 9 | 1001 | d | 1101 |
| 2 | 0010 | 6 | 0110 | a | 1010 | e | 1110 |
| 3 | 0011 | 7 | 0111 | b | 1011 | f | 1111 |

- Ogni gruppo di 4 bits viene rappresentato da una cifra *Hex*
- Esempi:

C1A0 C1A0 = 1100 0001 1010 0000 1100 0001 1010 0000

???? ???? = 1111 1110 1101 1110 1100 1110 1100 1010

- Per convertire da base *qualsiasi* a base 10 sfruttiamo il fatto che la rappresentazione e' posizionale:

$$(427)_8 = 4 * 8^2 + 2 * 8^1 + 7 * 8^0 = 4 * 64 + 2 * 16 + 7 = (279)_{10}$$

- Per convertire da base 10 a base *qualsiasi* procediamo per divisioni successive:

$$35 : 2 = 17 \text{ resto } 1$$

$$17 : 2 = 8 \text{ resto } 1$$

$$8 : 2 = 4 \text{ resto } 0$$

$$4 : 2 = 2 \text{ resto } 0$$

$$2 : 2 = 1 \text{ resto } 0$$

$$1 : 2 = 0 \text{ resto } 1$$

$$(35)_{10} = (100011)_2$$

Numeri *signed* complemento a 2

- Dato un numero ad n bit:

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

- l'intervallo e' -2^{n-1} a $2^{n-1} - 1$

- Esempio:

- $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
 $= -1 * 2^{31} + 1 * 2^{30} + \dots + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0$
 $= -2.147.483.648 - 2.147.483.644 = -4_{10}$

- A 32 bits l'intervallo va da $-2.147.483.648$ a $+2.147.483.647$

Numeri *signed* complemento a 2 (continua)

- Il bit 31 e' il bit di segno: 1 per negativi, 0 per positivi
- la rappresentazione non e' completa i.e. intervallo non simmetrico: $-(-2^{n-1})$????
- I numeri non-negativi hanno la stessa rappresentazione unsigned e 2s-complement (utile per aritmetica...)
- Alcuni numeri specifici:
 - 0 : 0000 0000 ... 0000
 - -1 : 1111 1111 ... 1111
 - Massimo numero negativo: 1000 0000 ... 0000
 - Massimo numero positivo: 0111 1111 ... 1111

Per passare dal numero positivo al suo equivalente negativo (e viceversa) si effettua un'operazione di complementazione e somma (+1) numeri negativo

- *Complementare* un numero binario significa invertirne tutti i bit

$$x + \bar{x} = 1111 \dots 111_2 = -1$$

$$\bar{x} + 1 = -x$$

- Esempio negazione di +2

- $+2 = 0000 \ 0000 \dots 0010_2$

- $-2 = 1111 \ 1111 \dots 1101_2 + 1$
 $= 1111 \ 1111 \dots 1110_2$

Abbiamo imparato alle scuole elementari che addizione e sottrazione si eseguono così'

Addizione

$$\begin{array}{r} 111111 \\ 1011011000000 + \\ 1011011000000 = \\ \hline 100010001000000 \end{array}$$

Sottrazione (come in base 10 ma occhio ai "prestiti")

$$\begin{array}{r} 111 \\ 1000 - \quad (8) \\ 0101 = \quad (5) \\ \hline 0011 \end{array}$$

Moltiplicazione

$$\begin{array}{r} 1111 * \\ 1111 = \\ \hline 1111 \\ 1111 \\ 1111 \\ 1111 \\ \hline 11100001 \end{array}$$

Estensione di segno: se $N < 0$ si riempie a sinistra di 1 altrimenti di 0

$$\begin{array}{r} 0100 = 4 \\ 00000100 = 4 \end{array}$$

$$\begin{array}{r} 1100 = -4 \\ 11111100 = -4 \end{array}$$

- Se la rappresentazione e' in complemento a 2, addizione e sottrazione si riducono alla sola addizione trascurando il bit MSB del risultato
- La rappresentazione e' limitata: dobbiamo fare attenzione agli overflow...

$$\begin{array}{r}
 +7 \quad 0 \ 1 \ 1 \ 1 \ + \\
 -3 \quad 1 \ 1 \ 0 \ 1 \ = \\
 \hline
 +4 \ (1) \ 0 \ 1 \ 0 \ 0
 \end{array}$$

$$\begin{array}{r}
 \quad \quad \quad 1 \ 1 \\
 +7 \quad 0 \ 1 \ 1 \ 1 \ + \\
 +3 \quad 0 \ 0 \ 1 \ 1 \ = \\
 \hline
 \quad \quad \quad (0) \ 1 \ 0 \ 1 \ 0 \ \text{?????}
 \end{array}$$

- Sottrazione $\rightarrow a - b = a + (-b)$

$$6 - 2 = (6) + (-2)$$

$$-4 - 2 = (-4) + (-2)$$

$$-4 - 5 = (-4) + (-5)$$

$$\begin{array}{r}
 1 \ 1 \\
 0 \ 1 \ 1 \ 0 \ + \quad (+6) \\
 1 \ 1 \ 1 \ 0 \ = \quad (-2) \\
 \hline
 (1) \ 0 \ 1 \ 0 \ 0 \quad (+4)
 \end{array}$$

$$\begin{array}{r}
 \quad \quad \quad 1 \\
 1 \ 1 \ 0 \ 0 \ + \quad (-4) \\
 1 \ 1 \ 1 \ 0 \ = \quad (-2) \\
 \hline
 (1) \ 1 \ 0 \ 1 \ 0 \quad (-6)
 \end{array}$$

$$\begin{array}{r}
 1 \ 1 \ 0 \ 0 \ + \quad (-4) \\
 1 \ 0 \ 1 \ 1 \ = \quad (-5) \\
 \hline
 (1) \ 0 \ 1 \ 1 \ 1 \quad (+7) \ \text{?????}
 \end{array}$$

Overflow compare quando il risultato dell'operazione non puo' essere rappresentato nel formato degli operandi (es 32 bits) ovvero quando il bit di segno non e' coerente con quello aspettato dal tipo di operazione e dal segno degli operandi

- No overflow quando si addizionano un numero positivo ed uno negativo...
- No overflow quando in una sottrazione i segni degli operandi sono uguali

| Operazione | Operando A | Operando B | Sign Result per Ovf |
|------------|------------|------------|---------------------|
| $A + B$ | ≥ 0 | ≥ 0 | < 0 |
| $A + B$ | < 0 | < 0 | ≥ 0 |
| $A - B$ | ≥ 0 | < 0 | < 0 |
| $A - B$ | < 0 | ≥ 0 | ≥ 0 |

$$R = (-1)^S \times 1.M \times b^E$$

- In virgola mobile (i.e. *floating point*) il numero reale perde la sua esattezza algebrica ma si rappresenta con
 - un bit di *segno* **S**,
 - un'ordine di grandezza (*esponente*) **E**
 - un numero arbitrario di cifre significative (*mantissa*) **M**

$$R = (-1)^S \times 1.M \times b^E$$

- In virgola mobile (i.e. *floating point*) il numero reale perde la sua esattezza algebrica ma si rappresenta con
 - un bit di *segno* **S**,
 - un'ordine di grandezza (*esponente*) **E**
 - un numero arbitrario di cifre significative (*mantissa*) **M**
- La rappresentazione in virgola mobile si puo' usare con qualsiasi base e con qualsiasi rappresentazione di esponente e mantissa
- Esempi:
 $1.27E^{12} = 1.27 \times 10^{12}$
 $12,433 \times 10^{-4}$, $11.10010 \times 10^{11101}$, $1110.10110 \times 10^{-10}$
- Di conseguenza c'e' bisogno di accordarsi su uno standard

- Nei computers si usano abitualmente rappresentazioni IEEE da 32 bit (*singola precisione*) e 64 bit (*doppia precisione*)
- La singola precisione (SP) ha 7 cifre decimali significative ed un range di $10^{\pm 38}$
- La doppia precisione (DP) ha 16 cifre decimali significative ed un range di $10^{\pm 308}$

Numero float = +15213.0

$$15213_{(10)} = 11101101101101_{(2)} = 1.1101101101101 \times 2^{13}$$

Mantissa:

$$M = 1.1101101101101$$

$$\text{frac} = 1101101101101\ 0000000000$$

Esponente:

$$E = 13$$

$$\text{bias} = 127$$

$$E = 127 + 13 = 140 = 10001100$$

Rappresentazione in SP

$$0\ 10001100\ 110110110110100000000000$$

Algebra Booleana e porte logiche

La funzione logica di un circuito combinatorio e' completamente descritta e specificata da una *Equazione Logica*

- Le variabili (i segnali...) in input e output sono *variabili logiche*.
- la funzione e' la combinazione dei 3 *operatori fondamentali* dell'algebra booleana
 - **OR** ($A + B$) out = 1 se *almeno un input* = 1
 - **AND** ($A * B$) out = 1 se *tutti gli input* = 1
 - **NOT** (\bar{A}) out = inverso dell'input

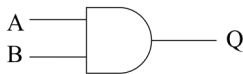
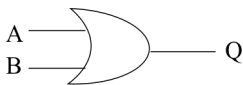
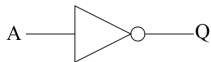


Table: Operatore NOT

| A | Q |
|---|---|
| 0 | 1 |
| 1 | 0 |

Table: Operatore OR

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Table: Operatore AND

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Proprieta' degli operatori:

| | | |
|--------------|-------------------|-------------------|
| Identita': | $A + 0 = A$ | $A * 1 = A$ |
| Nulla: | $A + 1 = 1$ | $A * 0 = 0$ |
| Idempotente: | $A + A = A$ | $A * A = A$ |
| Inverso: | $A + \bar{A} = 1$ | $A * \bar{A} = 0$ |

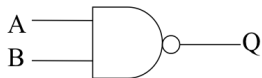
Proprieta' dell'algebra:

| | | |
|---------------|-----------------------------------|-----------------------------------|
| Commutativa: | $A + B = B + A$ | $A * B = B * A$ |
| Associativa: | $A + (B + C) = (A + B) + C$ | $A * (B * C) = (A * B) * C$ |
| Distributiva: | $A * (B + C) = (A * B) + (A * C)$ | $A + (B * C) = (A + B) * (A + C)$ |

Leggi di De Morgan:

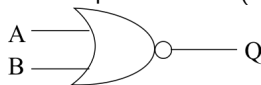
$$\overline{(A + B)} = \bar{A} * \bar{B} \quad \overline{(A * B)} = \bar{A} + \bar{B}$$

NAND: e' l'operatore NOT(AND)



| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

NOR: e' l'operatore NOT(OR)



| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

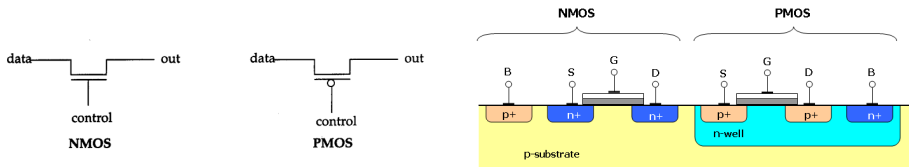
Si puo' dimostrare che il NAND(NOR) e' un'operatore universale i.e e' l'unico necessario per implementare qualsiasi funzione logica

$$\bar{A} = \bar{A} + 0 = \overline{A * 1}$$

$$A + B = \overline{\overline{A + B}} = \overline{\overline{A * 1} * \overline{B * 1}}$$

$$A * B = (A * B) + 0 = \overline{\overline{(A * B) + 0}} = \overline{\overline{A * B} * 1}$$

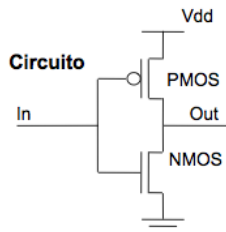
- Tecnologia *CMOS* (Complementary Metal Oxide Semiconductor) viene usata per realizzare transistor planari su silicio
- I principali vantaggi rispetto ad implementazione BJT sono la maggiore semplicità della tecnologia planare usata (che permette densità maggiori) e la potenza statica dissipata quasi nulla.



- Il comportamento è quello di un interruttore comandato dal gate Control
 - *NMOS* (N-Type Metal Oxide Semiconductor) transistor conduce se $C = 1$.
Resistenza infinita se $C = 0$
 - *PMOS* (P-Type Metal Oxide Semiconductor) transistor conduce se $C = 0$.
Resistenza infinita se $C = 1$

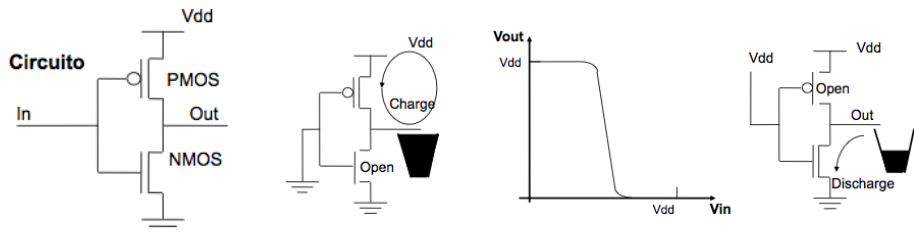
- In tecnologia CMOS un PMOS e' sempre accoppiato ad un NMOS.
- Si escludono quindi path *statici* tra VDD e GND \rightarrow potenza dissipata statica (quasi) nulla...
- Solo in presenza del cambiamento di stato del gate si apre un canale VDD-GND \rightarrow potenza dissipata dinamica bassa

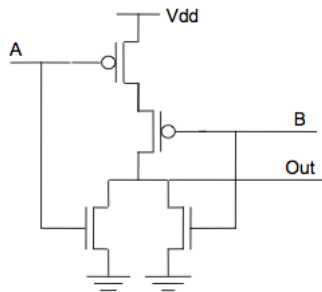
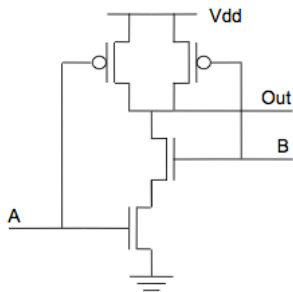
Es: Inverter

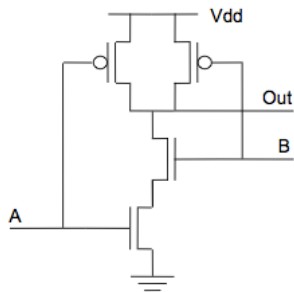


- In tecnologia CMOS un PMOS e' sempre accoppiato ad un NMOS.
- Si escludono quindi path *statici* tra VDD e GND \rightarrow potenza dissipata statica (quasi) nulla...
- Solo in presenza del cambiamento di stato del gate si apre un canale VDD-GND \rightarrow potenza dissipata dinamica bassa

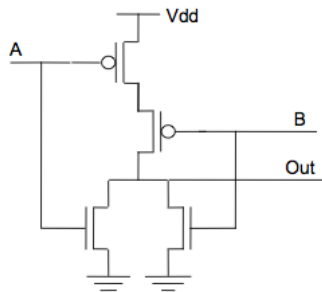
Es: Inverter



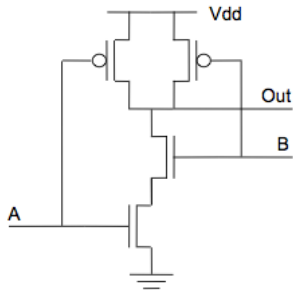




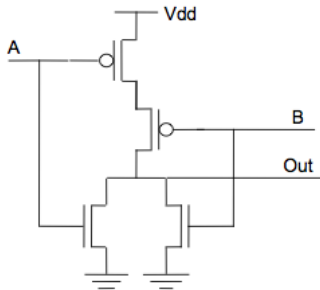
NAND



NOR



NAND



NOR

- Se in una particolare tecnologia il transistor PMOS e' piu' veloce
 - Meglio avere PMOS in serie
 - Porte NOR preferite per implementazione circuiti
- Se invece una particolare il transistor NMOS e' piu' veloce
 - Meglio avere NMOS in serie
 - Porte NAND preferite per implementazione circuiti

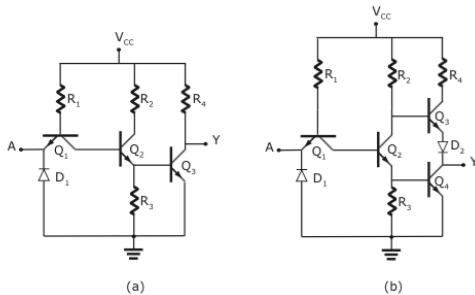


Figura 7.5: (a) Circuito NOT; (b) circuito NOT con uscita "totem pole".

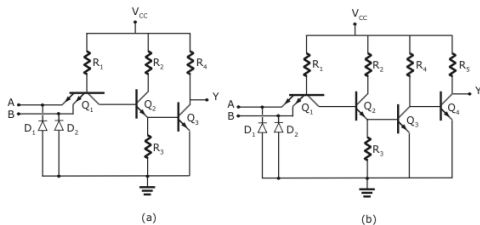
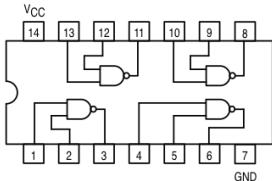


Figura 7.6: (a) Circuito NAND; (b) Circuito AND



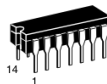
QUAD 2-INPUT NAND GATE

- ESD > 3500 Volts



SN54/74LS00

**QUAD 2-INPUT NAND GATE
LOW POWER SCHOTTKY**



**J SUFFIX
CERAMIC
CASE 632-08**

GUARANTEED OPERATING RANGES

| Symbol | Parameter | | Min | Typ | Max | Unit |
|-----------------|-------------------------------------|--------|------|-----|------|------|
| V _{CC} | Supply Voltage | 54 | 4.5 | 5.0 | 5.5 | V |
| | | 74 | 4.75 | 5.0 | 5.25 | |
| T _A | Operating Ambient Temperature Range | 54 | -55 | 25 | 125 | °C |
| | | 74 | 0 | 25 | 70 | |
| I _{OH} | Output Current – High | 54, 74 | | | -0.4 | mA |
| I _{OL} | Output Current – Low | 54 | | | 4.0 | mA |
| | | 74 | | | 8.0 | |

DC CHARACTERISTICS OVER OPERATING TEMPERATURE RANGE (unless otherwise specified)

| Symbol | Parameter | | Limits | | | Unit | Test Conditions |
|-----------------|---|--------|--------|-------|------|------|---|
| | | | Min | Typ | Max | | |
| V _{IH} | Input HIGH Voltage | | 2.0 | | | V | Guaranteed Input HIGH Voltage for All Inputs |
| V _{IL} | Input LOW Voltage | 54 | | | 0.7 | V | Guaranteed Input LOW Voltage for All Inputs |
| | | 74 | | | 0.8 | | |
| V _{IK} | Input Clamp Diode Voltage | | | -0.65 | -1.5 | V | V _{CC} = MIN, I _{IN} = -18 mA |
| V _{OH} | Output HIGH Voltage | 54 | 2.5 | 3.5 | | V | V _{CC} = MIN, I _{OH} = MAX, V _{IN} = V _{IH} or V _{IL} per Truth Table |
| | | 74 | 2.7 | 3.5 | | V | |
| V _{OL} | Output LOW Voltage | 54, 74 | | 0.25 | 0.4 | V | I _{OL} = 4.0 mA I _{OL} = 8.0 mA V _{CC} = V _{CC} MIN, V _{IN} = V _{IL} or V _{IH} per Truth Table |
| | | 74 | | 0.35 | 0.5 | V | |
| I _{IH} | Input HIGH Current | | | | 20 | μA | V _{CC} = MAX, V _{IN} = 2.7 V |
| | | | | | 0.1 | mA | V _{CC} = MAX, V _{IN} = 7.0 V |
| I _{IL} | Input LOW Current | | | | -0.4 | mA | V _{CC} = MAX, V _{IN} = 0.4 V |
| I _{OS} | Short Circuit Current (Note 1) | | -20 | | -100 | mA | V _{CC} = MAX |
| I _{CC} | Power Supply Current Total, Output HIGH | | | | 1.6 | mA | V _{CC} = MAX |
| | Total, Output LOW | | | | 4.4 | | |

Note 1: Not more than one output should be shorted at a time, nor for more than 1 second.

AC CHARACTERISTICS (T_A = 25°C)

- Ogni equazione logica puo' essere rappresentata in *forma canonica* tramite uso di operatori AND, OR, e NOT
- La forma canonica si deriva (ad es..) dalla tabella della verita' in forma di *Somma di Prodotti* (SP)

| A | B | C | E |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

- Per ogni entry uguale ad 1 dell'output si genera un prodotto *minterm* degli input dove gli input=0 sono negati
- Per ottenere l'equazione in forma S, si sommano i prodotti cosi' ottenuti

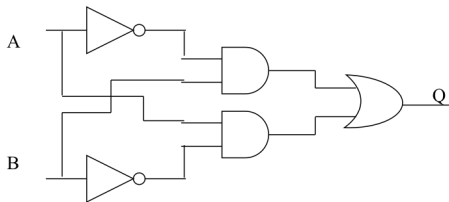
$$E = (\bar{A} * \bar{B} * C) + (A * B * \bar{C})$$

- Grazie alla proprieta' di *identita'* della somma logica ($A + 0 = A$) il contributo all'equazione logica viene solo dai minterm non 0

- A partire da un'equazione logica espressa come SP si puo' realizzare il circuito equivalente con variabili (segnali) invertite e non invertite che attraversano:
 - un livello di porte AND per i prodotti
 - un livello di porte OR per la somma dei prodotti

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$Q = (\bar{A} * B) + (A * \bar{B})$$



- Obiettivo della minimizzazione e' la riduzione del *costo* del circuito combinatorio in termini di numero di porte e variabili necessarie all'implementazione dell'equazione richiesta
- Si ottiene un circuito equivalente che generalmente e' piu' *piccolo* e con tempi di propagazione ridotti.
- Si puo' agire per ispezione utilizzando le proprieta' dell'algebra.

Esempio:

$$Q = \bar{A} * B + A * B \rightarrow \text{applico proprieta' distributiva}$$

$$Q = B * (\bar{A} + A) \rightarrow \text{applico inverso}$$

$$Q = B * 1 = B$$

- in questo caso la variabile *A* e' *DON'T CARE* cioe' non conta ai fini della definizione della equazione
- L'individuazione di variabili *DON'T CARE* e' l'obiettivo ultimo della minimizzazione

- Esempio: multiplexer a due input. Seleziona in uscita il valore di un ingresso scelto tra $2(N)$ diversi in base ai valori assunti dagli ingressi di *select* (S)

| S | A | B | Z |
|---|---|---|---|
| 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 | 0 |
| 1 | 1 | 1 | 1 |

- Scrivo la funzione in termini di somme di prodotti

$$Z = \bar{S}\bar{A}\bar{B} + \bar{S}A\bar{B} + S\bar{A}B + SAB \rightarrow \text{distributiva}$$

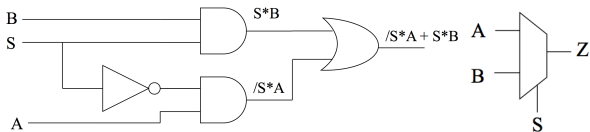
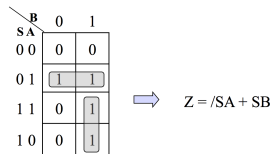
$$Z = \bar{S}A(\bar{B} + B) + SB(\bar{A} + A) \rightarrow \text{inverso}$$

$$Z = \bar{S}A + SB$$

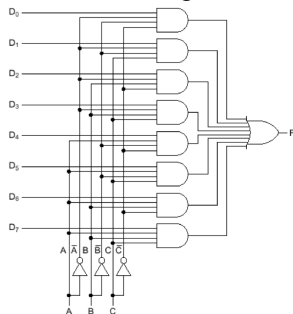
- 3 gate NOT, 8 gate AND e 3 gate OR \rightarrow 1 NOT, 2 AND, 1 OR !!!!!

Esempi di circuiti combinatori: Multiplexer

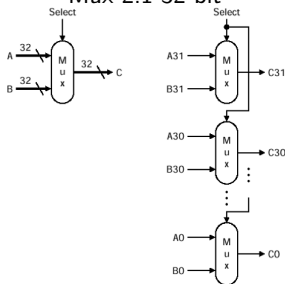
Multiplexer 2:1



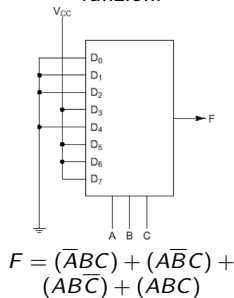
Mux 8:1 single bit



Mux 2:1 32 bit



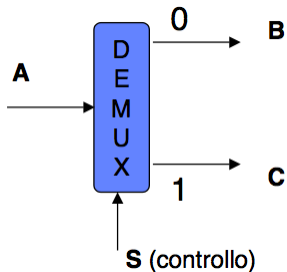
Mux 8:1 come gen funzioni



Esempi di circuiti combinatori: Demultiplexer

- Da 1 input a n output (scelti da select)
- $\log_2 n$ segnali di controllo

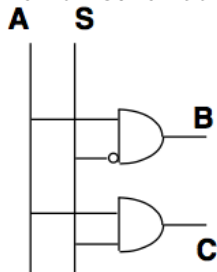
Simbolo del DeMux



| A | S | B | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

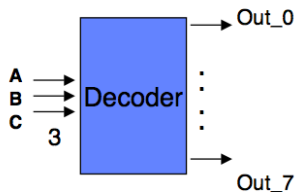
$$B = A\bar{S}, \quad C = AS$$

DeMux schematic

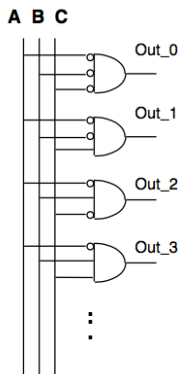


Esempi di circuiti combinatori: Decoder

- Componente con n inputs e 2^n outputs
- n input sono un numero unsigned
- Se $n = i$ allora solo i -esimo bit di output uguale a 1
- Lo abbiamo visto impiegato nel blocco di selezione della locazione in un Register File

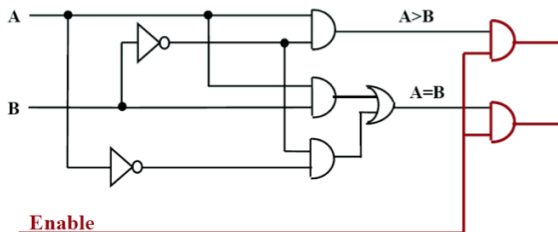


| A | B | C | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |



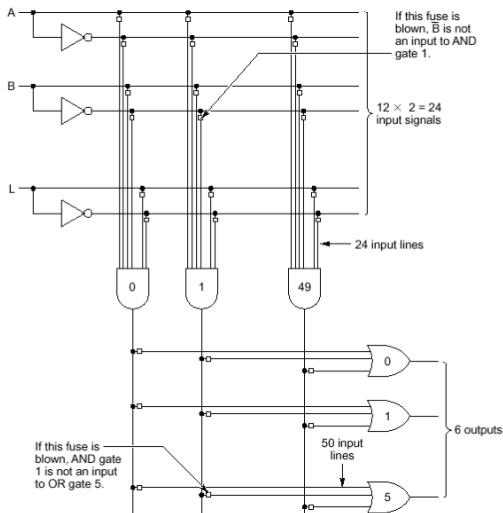
- Confronto di 2 *numeri* interi positivi
- In uscita $A = B$, $A > B$
- La condizione $A < B$?

| A | B | $A > B$ | $A = B$ |
|---|---|---------|---------|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



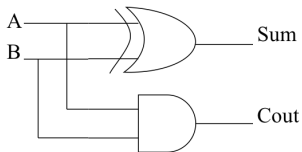
Programming Logic Array: componente utilizzato per generare funzioni qualsiasi in termini di somma di prodotti

- E' una struttura con n input, o output, m minterm esprimibili
- Strutture elettriche *fusibili* realizzano le connessioni i.e. i minterm
- La programmazione consiste nel decidere quali *fusibili* bruciare i.e. quali sono gli input ad ogni porta AND e quali quelli ad ogni port OR



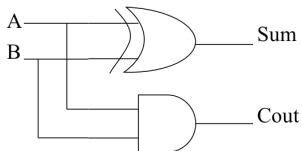
- Half-Adder circuito di addizione su numeri binari unsigned ad 1 bit (non completo....)

| A | B | Sum | Cout |
|---|---|-----|------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

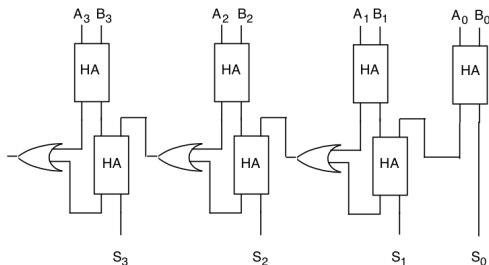


- Half-Adder circuito di addizione su numeri binari unsigned ad 1 bit (non completo....)

| A | B | Sum | Count |
|---|---|-----|-------|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

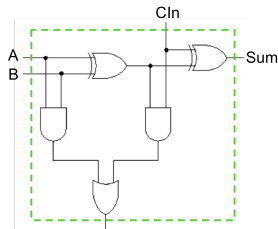


- Messi opportunamente in cascata realizzano addizionatori interi per parole a n bit implementando esattamente il metodo elementare *paper&pencil*



- Il *full-adder* o *sommatore completo* è un circuito elettronico che esegue addizioni su numeri binari unsigned ad 1 bit

| A | B | C_i | S | C_o |
|---|---|-------|---|-------|
| 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 |



SUM

| C_{In} | 0 | 1 |
|----------|---|---|
| AB | | |
| 00 | 0 | 1 |
| 01 | 1 | 0 |
| 11 | 0 | 1 |
| 10 | 1 | 0 |

$$S = (\overline{A}BC_i) + (\overline{A}B\overline{C}_i) + (ABC_i) + (ABC_i)$$

COut

| C_{In} | 0 | 1 |
|----------|---|---|
| AB | | |
| 00 | 0 | 0 |
| 01 | 0 | 1 |
| 11 | 1 | 1 |
| 10 | 0 | 1 |

$$C_o = BC_i + AB + AC_i$$