Progettazione di Sistemi Digitali

Simone Lidonnici

5 aprile 2024

Indice

1
Numeri binari

I **numeri binari** sono numeri composti solo dalle cifre 0 e 1. Per convertirli in base decimale bisogna moltiplicare ogni cifra del numero binario per 2^i, in cui i è la posizione partendo da destra e contando da 0.
Esempio:
$1010_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 10_{10}$

Con N cifre in binario possiamo scrivere 2^N valori con range $[0, 2^N - 1]$

1.1 Numeri esadecimali

I **numeri esadecimali** si utilizzano per comodità perché è molto facile convertire i numeri binari in esadecimali.
Per convertire i numeri binari in esadecimale basta raggrupparli in gruppi da 4 cifre e scriverli con il numero corrispondente da 0 a 15 (i numeri dal 10 al 15 sono rappresentati dalle lettere A,B,C,D,E e F).
Esempio:
$10100110_2 \implies 1010 = A, 0110 = 6 \implies A6_{16}$

Per convertirli in decimale si utilizza lo stesso principio dei numeri binari ma con 16 al posto di 2.
Esempio:
$4AF = 15 \cdot 16^0 + 10 \cdot 16^1 + 4 \cdot 16^2 = 1199$

Le cifre di un numero esadecimale convertite in binario e decimale:

Decimale	Binario	Esadecimale
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Ogni cifra di un numero binario viene chiamata **bit** e vengono raggruppati in gruppi da 8 chiamati **byte**. Il bit più a sinistra in un byte è quello più significativo e quello più a destra quello meno significativo.

1.2 Operazioni con numeri non decimali

1.2.1 Somme e sottrazioni

Le operazioni di somma e sottrazioni si effettuano in modo normale ma al posto di avere il riporto a 10 si ha a 16 o a 2.
Esempi:
$1011 + 0011 = 1110$
$3A09 + 1B17 = 5520$

Di solito i sistemi utilizzano un numeri di bit fisso e se un numero eccede il numero di bit massimo viene chiamato **overflow** e i bit in eccesso vengono scartati.
Esempio:
$1001 + 1100 = 10101 = 0101$ sbagliato

1.2.2 Moltiplicazioni

Per eseguire una moltiplicazione si moltiplica il primo numero per ogni bit del secondo spostandosi a sinistra di una posizione ogni volta.
Esempio:

$$\begin{array}{rc}
0101 & \times \\
0111 & = \\
\hline
0101 & + \\
01010 & + \\
010100 & + \\
000000 & = \\
\hline
100011 &
\end{array}$$

1.3 Numeri binari con il segno

Per scrivere un numero con il segno in binario si utilizza il bit più a sinistra per identificare il segno:

- 0 significa positivo
- 1 significa negativo

Esempio:
$1110 = -6$
$0110 = 6$

Con N cifre in binario possiamo scrivere 2^{N-1} valori con range $[-(2^{N-1}) - 1, 2^{N-1} - 1]$

1.3.1 Complemento a due

Per poter eseguire facilmente addizioni i numeri binari con segno si scrivono utilizzando un altro metodo: il **complemento a due**.
Il primo bit (a sinistra) viene considerato negativo mentre gli altri positivi.
Esempi:
$1001 = -1 \cdot 2^3 + 1 \cdot 2^0 = -7$
$01101 = 2^3 + 2^2 + 2^0 = 8 + 4 + 1 = 13$

Per invertire (passare da positivo a negativo o viceversa) un numero in complemento a due bisogna invertire gli 0 con gli 1 e poi sommare 1.
Esempio:
$27 = 011011$
$-27 = 100100 + 1 = 100101$

Se vengono sommati due numeri con lo stesso segno potrebbe succedere che il risultato esca dal range a causa di un'overflow e bisogna aggiungere un bit.
Esempio:
$0110 + 01010 = 10001 \implies$ sbagliato perché dalla somma di due numeri positivi è uscito un numero negativo, quindi va aggiunto un bit uguale a 0 all'inizio $\implies$ 010001

1.4 Estendere il numero di bit

1.4.1 Numeri senza segno

Se durante un'operazione (somma o moltiplicazione) il risultato eccede il numero di bit massimo e si causa un overflow bisogna estendere il numero di bit. Per farlo basta aggiungere tutti 0 davanti agli operandi.
Esempio:
$1010 + 1000 = 10010 \implies$ 1 non viene considerato quindi aggiungiamo uno 0 davanti ad entrambi gli operandi:
$01010 + 01000 = 10010$

1.4.2 Complemento a due

Per estendere il numero di bit di un numero in complemento a due si aggiunge a sinistra la prima cifra significativa ripetuta.
Esempi:
$1010 = 111010$
$0111 = 000111$

1.4.3 Shiftare i numeri binari

Se moltiplichiamo o dividiamo un numero binario per una potenza di 2 possiamo spostare le cifre a destra o sinistra di quanti posti quanto l'esponente di 2 nella potenza.
Esempi:
$1 \cdot 2^2 = 100$
$1000 \cdot 2^2 = 100000$
$1010/2^3 = 1$ (il resto non viene considerato visto che stiamo lavorando con numeri interi)
$10000/2 = 11000$ (nei numeri negativi quando si divide si aggiungono degli 1 all'inizio)

1.5 Numeri binari con la virgola

I numeri binari con la virgola si possono scrivere in diversi modi. Nel più classico semplicemente continuiamo a sommare potenze di 2 anche per la parte decimale, solamente con esponente negativo.
Esempi:
$0.5_{10} = 2^{-1} = 0.1_2$
$3.75_{10} = 3 + 2^{-1} + 2^{-2} = 11.11_2$

Un altro metodo è il **fixed point** in cui si decide preventivamente dove sia la virgola e semplicemente si scrive il numero in modo normale.
Esempio:
01101100(virgola al 4 posto) $= 0110.1100 = 6.75$

1.6 Floating point

Numeri binari in floating point

Un numero binario secondo la notazione **floating point** si scrive in modo simile alla notazione scientifica, cioè:

$$\pm M \cdot B^E$$

In cui:

- M = mantissa, numero con una sola cifra prima della virgola
- B = base
- E = esponente

La scrittura standard è a 32 bit di cui 1 per il segno, 8 per l'esponente e 27 per la mantissa. All'esponente bisogna aggiungere 127 (quindi per scrivere 5 dovremo scrivere 132) e nella mantissa non dobbiamo considerare il numero prima della virgola perché è sempre 1. Per convertire un numero binario in questa notazione bisogna:

1. Convertire il numero in binario senza segno
2. Scrivere il numero in notazione scientifica
3. Completare i 32 bit in modo opportuno

Vista la lunghezza di questi numeri, vengono solitamente scritti in esadecimale.

Esempio:
228 = 11100100

$$11100100 = 1.11001 \cdot 2^7 = \overbrace{0}^{\text{segno}}\ \overbrace{10000110}^{\text{esponente+127}}\ \overbrace{11001000000000000000000}^{\text{mantissa}} = 0x43640000$$

Nella notazione floating point esistono dei numeri speciali che si scrivono con dei bit specifici:

Numero	Segno	Esponente	Mantissa
0	indifferente	00000000	00. . . 00
∞	0	11111111	00. . . 00
$-\infty$	1	11111111	00. . . 00

Oltre questi esistono una serie di numeri chiamati **numeri denormalizzati** che hanno esponente uguale a 0 e mantissa diversa da 0 e serbono per scrivere numeri con esponente minore del minimo possibile. In questi numeri si considera la cifra prima della virgola come 0 e non 1. Con questa estensione in floating point abbiamo:

- Numero massimo: 2^{126}
- Numero minimo normalizzato: 2^{-126}
- Numero minimo denormalizzato: 2^{-149}

Del floating point esistono altre due versioni con diversi bit:

- Half-precision: 16 bit, 1 per il segno, 5 per l'esponente e 10 per la mantissa. L'esponente va inserito sommato di 15.
- Half-precision: 32 bit, 1 per il segno, 11 per l'esponente e 52 per la mantissa. L'esponente va inserito sommato di 1023.

1.6.1 Operazioni con floating point

Per eseguire la **somma** tra due numeri in floating point:

1. Convertirli in binario e scriverli in notazione scientifica
2. Cambiare l'esponente minore per renderli uguali
3. Sommare le mantisse
4. Riscrivere il risultato in notazione scientifica
5. Arrotondare se non bastano i bit
6. Riscrivere nel formato floating point

Esempio:
$0x3FC00000 + 0x40500000$
$0x3FC00000 = 001111111100\ldots00 = 1.1. \cdot 2^0 = 0.11 \cdot 2^1$
$0x40500000 = 01000000010100\ldots00 = 1.101 \cdot 2^1$
$0.11 \cdot 2^1 + 1.101 \cdot 2^1 = 10.011 \cdot 2^1 = 1.0011 \cdot 2^2$
$1.0011 \cdot 2^2 = 0100000010011000\ldots00 = 0x40980000$

Per eseguire la **moltiplicazione** tra due numeri in floating point:

1. Convertirli in binario e scriverli in notazione scientifica
2. Sommare gli esponenti
3. Moltiplicare le mantisse
4. Riscrivere il risultato in notazione scientifica
5. Arrotondare se non bastano i bit
6. Riscrivere nel formato floating point

Esempio:
$(1.1 \cdot 2^{10}) \cdot (1.0110 \cdot 2^{11})$
$2^{10} + 2^{11} = 2^{21}$
$1.0110 \cdot 1.1 = 10.0001$
$10.001 \cdot 2^{21} = 1.0001 \cdot 2^{22} = 01001010100001000\ldots00 = 0x4A840000$

2

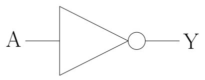
Porte logiche

Le **porte logiche** sono delle funzioni che hanno un output e possono avere:

- 1 input: not, buffer
- 2 o più input: and, or, xor ...

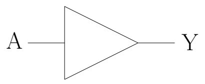
2.1 Tipi di porte logiche

Ci sono molti tipi diversi di porte logiche:
NOT: $Y = \overline{A}$



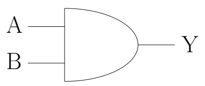
A	Y
0	1
1	0

BUFFER: $Y = A$



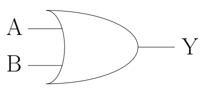
A	Y
0	0
1	1

AND: $Y = A \cdot B$



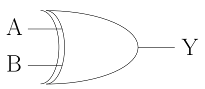
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR: $Y = A + B$



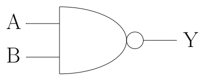
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR: $Y = A \oplus B$



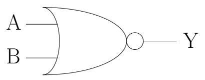
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NAND: $Y = \overline{A \cdot B}$



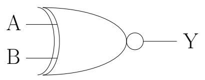
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR: $Y = \overline{A + B}$



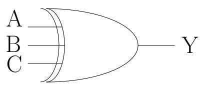
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR: $Y = \overline{A \oplus B}$



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

XOR3: $Y = A \oplus B \oplus C$



A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

2.2 Circuiti logici

Componenti di un circuito logico

Un **circuito logico** è composto da:

- nodi: input, output e nodi interni
- circuiti elementari

I circuiti sono di due tipi:

1. **Combinatori**: non hanno memoria e gli output sono definiti solo dagli input attuali
2. **Sequanziali:** hanno memoria e gli output sono definiti dagli input attuali e precedenti

I circuiti combinatori hanno 3 regole principali:

1. Tutti i circuiti elementari sono combinatori

2. Ogni nodo è un input o si collega ad esattamente un output
3. Non ci sono cicli

2.3 Equazioni booleane

Nelle equazioni booleane utilizzeremo delle definizioni precise:

- **Complemento**: opposto di una variabile ($\overline{A}$)
- **Literal**: una variabile o un suo complemento (A o $\overline{A}$)
- **Implicante**: prodotto di literal (AB o $B\overline{C}$)
- **Minterm**: prodotto che utilizza tutte le variabili di input (ABC o $\overline{A}B\overline{C}$)
- **Maxterm**: somma che utilizza tutte le variabili di input ($A + B + C$ o $\overline{A} + B + \overline{C}$)

2.3.1 Somma di prodotti (SOP)

Tutte le equazioni possono essere scritte come somma di prodotti.
Se abbiamo la tabella ci basta sommare i minterm che danno come risultato 1. Il risultato può anche essere scritto come sommatoria specificando l'indice dei minterm sommati.
Esempio:

A	B	Y	Minterm	Indice del minterm
0	0	0	$\overline{A}\overline{B}$	0
0	1	1	$\overline{A}B$	1
1	0	0	$A\overline{B}$	2
1	1	1	AB	3

In questo caso: $Y = \overline{A}B + AB = B$
Possiamo anche scrivere: $Y = \sum(1, 3)$

2.3.2 Prodotto di somme (POS)

Tutte le equazioni possono essere scritte come prodotto di somme.
Se abbiamo la tabella ci basta sommare i maxterm che danno come risultato 0. Il risultato può anche essere scritto come produttoria specificando l'indice dei minterm sommati.
Esempio:

A	B	Y	Maxterm	Indice del maxterm
0	0	0	$A + B$	0
0	1	1	$A + \overline{B}$	1
1	0	0	$\overline{A} + B$	2
1	1	1	$\overline{A} + \overline{B}$	3

In questo caso: $Y = (A + B) + (\overline{A} + B) = B$
Possiamo anche scrivere: $Y = \prod(0, 2)$

2.4 Completezza delle porte logiche

Le porte logiche **NAND** e **NOR** sono dette **funzionalmente complete** perché grazie a queste possono essere create tutte le altre porte logiche.

3

Algebra booleana

Dualità degli assiomi

Ogni assioma e teorema dell'algebra booleana può essere scritto sia con gli AND ($\cdot$) che con gli OR ($+$). Per farlo bisogna invertire i $+$ con i $\cdot$ e gli 0 con gli 1.
La versione alternativa dell'assioma o teorema viene chiamato **duale**.

3.1 Assiomi e Teoremi

L'agebra booleana ha diversi assiomi:

Assioma	Duale
$B = 0 \iff B \neq 1$	$B = 1 \iff B \neq 0$
$\overline{0} = 1$	$\overline{1} = 0$
$0 \cdot 0 = 0$	$1 + 1 + = 1$
$1 \cdot 1 = 1$	$0 + 0 = 0$
$0 \cdot 1 = 1 \cdot 0 = 0$	$1 + 0 = 0 + 1 = 1$

Ci sono anche diversi teoremi:

Assioma	Duale
$B \cdot 1 = B$	$B + 0 = B$
$B \cdot 0 = 0$	$B + 1 = 1$
$B \cdot B = B$	$B + B = B$
$\overline{\overline{B}} = B$	
$B \cdot \overline{B} = 0$	$B + \overline{B} = 1$
$B \cdot C = C \cdot B$	$B + C = C + B$
$B \cdot C \cdot D = B(C \cdot D)$	$B + C + D = B + (C + D)$
$B(C + D) = (B \cdot C) + (B \cdot D)$	$B + (C \cdot D) = (B + C) \cdot (B + D)$
$B(B + C) = B$	$B + (B \cdot C) = B$
$(B \cdot \overline{C}) + (B \cdot C) = B$	$(B + \overline{C}) \cdot (B + C) = B$
$B(1 + C) = B$	$B + (0 \cdot C) = B$

3.2 Semplificare un'equazione

Presa un'equazione booleana, semplificarla significa ridurla al numero minimo di implicanti tramite teoremi o assiomi (detta anche **minimizzazione**). **Esempi:**
$\overline{A}B + A = B + A$

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

Teorema di De Morgan

Il **teorema di De Morgan** dice che se abbiamo un'equazione booleana con una somma o una moltiplicazione negata ($\overline{AB}$ oppure $\overline{A+B}$), possiamo invertire l'operazione e negare i literal. Cioè:

$$\overline{AB} = \overline{A} + \overline{B}$$
$$\overline{A+B} = \overline{A} \cdot \overline{B}$$

Esempio:
$Y = \overline{\left(A + \overline{BD}\right) \cdot \overline{C}} = \overline{A + \overline{BD}} + \overline{\overline{C}} = \overline{A}BD + C$

4
Circuiti logici

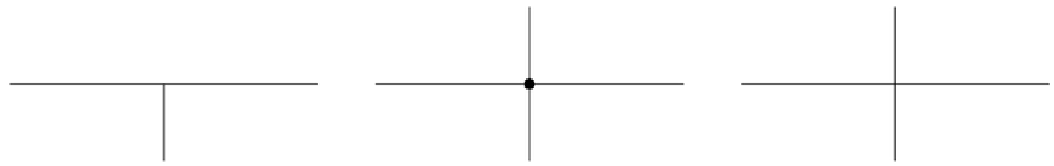
4.1 Regole dei circuiti

I circuiti logici, quando vengono disegnati, hanno diverse regole:

- Gli input vengono scritti a sinistra o in alto
- Gli output vengono scritti a destra o in basso
- Qualsiasi posta va da sinistra a destra

Inoltre vengono definite delle regole quando di intrecciano i fili:

- Se i fili formano una T sono collegati
- Se i fili formano una X con un punto sono collegati
- Se i fili formano una X senza punto non sono collegati



Nei primi due casi i fili sono collegati e nel terzo no

4.1.1 Circuiti con più output

Possono esserci circuiti che hanno più di un output, un esempio è il circuito a priorità, che ha questa tabella:

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	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	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

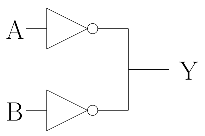
Molti valori in questa tabella non vengono considerati nel calcolare l'output quindi possiamo riscrivere la tabella in modo semplificato:

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	**X**	0	0	1	0
0	1	**X**	**X**	0	1	0	0
1	**X**	**X**	**X**	1	0	0	0

X vuol dire **don't care**, cioè che il valore non cambia il risultato

4.1.2 Contention

La **contention** è una situazione che viene causata quando in un circuito si incontrano due valori diversi. Si indica con X (da non confondere con il don't care) e solitamente implica un problema.
Esempio:

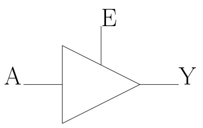


A	B	Y
0	0	1
0	1	**X**
1	0	**X**
1	1	0

4.1.3 Floating

Il **floating** è una situazione che viene causata quando un filo è disattivato o non c'è corrente, si indica con Z.
Un esempio è il **tristate buffer**, un buffer che è attivo solo quando un parametro E è uguale

a 1:



E	A	Y
0	0	**Z**
0	1	**Z**
1	0	0
1	1	1

4.2 Mappe di Karnaugh (K-maps)

Le espressioni booleane possono essere semplificate in modo grafico utilizzando le **mappe di Karnaugh**. Questa mappa conterrà in ogni riquadro una riga della tabella della verità. Una volta che abbiamo scritto gli 0 e 1 nella tabella possiamo semplificarli in due modi:

- Utilizzando i minterm e gli 1 secondo le regole:
  1. Bisogna cerchiare ogni 1 almeno una volta
  2. Ogni cerchio deve contenere un numero di 1 pari ad una potenza di 2 (1,2 o 4)
  3. Ogni cerchio deve essere preso il più grande possibile
  4. Un cerchio può contenere don't care se servono a creare un cerchio più grande
  5. Alla fine ogni cerchio rappresenterà un prodotto tra le variabili cerchiate escludendo quelle che all'interno del cerchio appaiono sia negate che normali
- Utilizzando i miaxterm e gli 0 secondo le regole:
  1. Bisogna cerchiare ogni 0 almeno una volta
  2. Ogni cerchio deve contenere un numero di 0 pari ad una potenza di 2 (1,2 o 4)
  3. Ogni cerchio deve essere preso il più grande possibile
  4. Un cerchio può contenere don't care se servono a creare un cerchio più grande
  5. Alla fine ogni cerchio rappresenterà una somma tra le variabili cerchiate escludendo quelle che all'interno del cerchio appaiono sia negate che normali

Esempio:

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

Si scrive 11 vicino a 01 perchè possono essere raggruppati solo riquadri che differiscono per solo una variabile. In questo caso avremo:

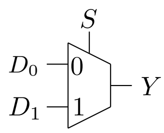
$$Y = (\overline{A} + \overline{B})(\overline{B} + C + D)(\overline{A} + \overline{C} + \overline{D})(A + B + C + \overline{D})$$

5
Blocchi combinatori

I blocchi combinatori sono un insieme di porte logiche che eseguono delle determinate operazioni specifiche, i due tipi più importanti sono **multiplexer (mux)** e **decoder**.

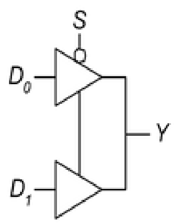
5.1 Multiplexer (Mux)

I multiplexer selezionano uno dei possibili N input (numero potenza di 2) tramite un numeri di selezionatori pari a $\log_2 N$. Anche il mux, come il NOR e il NAND permette di creare tutte le altre porte logiche. Viene disegnato:



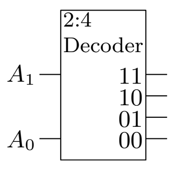
$$Y = \overline{S}D_0 + SD_1$$

Se lo volessimo disegnare con le porte logiche classiche sarebbe:

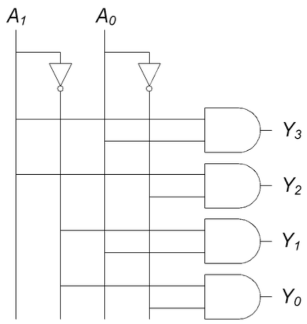


5.2 Decoder

I decoder hanno N input (numero potenza di 2) e un output per ogni combinazione possibile degli input, per un totale di 2^N output. Segue una politica **one-hot**, cioè che solo un output può essere attivo (uguale a 1) alla volta. Viene disegnato:



Se lo volessimo disegnare con le porte logiche classiche sarebbe:



I decoder possono essere usati per creare funzioni logiche partendo dai minterm.

5.3 Teorema di Shannon

Teorema di Shannon

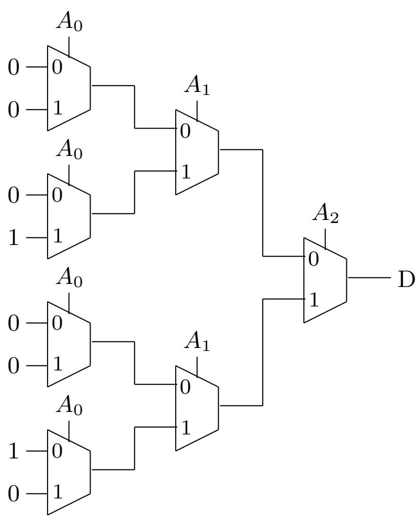
Data una funzione f con n variabili, questa può essere semplificata nella somma di due funzioni ad $n-1$ variabili moltiplicate per i due possibili valori della variabile:

$$f(x_1, x_2, ..., x_n) = x_1 \cdot f(0, x_2, ..., x_n) + \overline{x_1} \cdot f(0, x_2, ..., x_n)$$

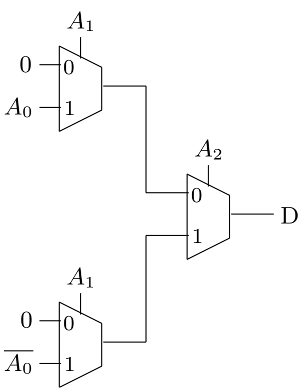
Questo teorema si può applicare ai mux per scrivere qualsiasi funzione.

Esempio:

A_2	A_1	A_0	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

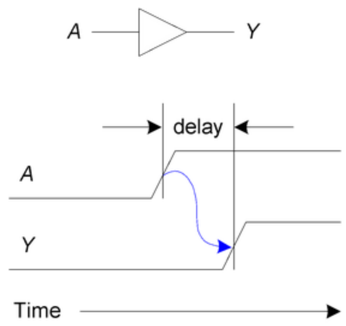


Semplificandolo otteniamo:



5.4 Delay

Il **delay** è il tempo da quando cambia un input fino a quando cambia l'output corrispondente. Viene calcolato da quando l'input è a metà del cambiamento fino a quando l'output è a metà del cambiamento.

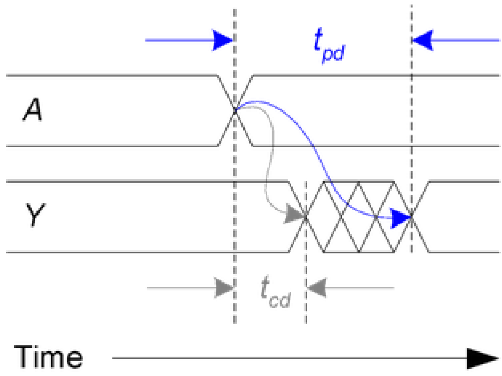


Il delay può essere causato da:

- capactà e resistenze del circuito
- limitazioni della velocità della luce

Ci sono due tipi di tempo che possiamo calcolare:

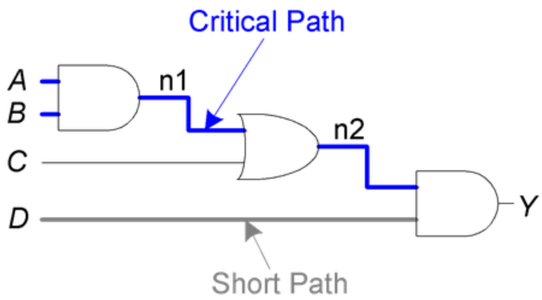
- **Propagation delay** (t_{pd}): tempo massimo per effettuare il cambiamento
- **Contamination delay** (t_{cd}): tempo minimo per effettuare il cambiamento



Il propagation delay si calcola sommando tutti i tempi massimi delle porte logiche del percorso più lungo mentre il delay minimo sommando tutti i tempi minimi delle porte logiche del percorso più corto.
I motivi per cui t_{pd} e t_{cd} sono diversi sono:

- diversi tempi di innalzamento o discesa
- multipli input e output, alcuni più veloci di altri
- rallentamenti dovuti alla tempeatura

Esempio:



In questo caso:

- $t_{pd} = 2 \cdot t_{pd\_AND} + t_{pd\_OR}$
- $t_{cd} = t_{cd\_AND}$

6
Circuiti sequenziali