Progettazione di sistemi digitali

1-Numeri binari

I numeri binari sono numeri composti solamente dalle cifre 0 e 1 ripetute.

Per convertire i numeri binari in decimali bisogna moltiplicare ogni cifra per 2^i in cui i è la posizione in cui si trova la cifra (partendo da destra e contando da 0) e sommarle tra loro.

Esempio:

$1010_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 0 + 2 + 0 + 8 = 10_{10}$

Con N cifre di un numero binario:

- Si possono scrivere 2^N valori
- Con un range: [0 ; $2^N - 1$]

1.1-Numeri esadecimali

Per comodità si utilizzano i numeri esadecimali, che utilizzano cifre tra 0 e F, questo perché permettono di raggruppare le cifre di un numero binario a gruppi di 4 (nibble).

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 del 2.

Esempio:

$4AF = 15 \cdot 16^0 + 10 \cdot 16^1 + 4 \cdot 16^2 = 1199$

Numero in decimale	Numero in binario	Numero in 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 raggruppate a gruppi di 8 chiamati byte in cui quello più a sinistra è quello più significativo e l'ultimo a destra quello meno significativo.

1.2-Operazioni con numeri non decimali

1.2.1-Somme e sottrazioni

Le operazioni si eseguono allo stesso modo di quelle decimali semplicemente il riporto non si effettua quando si arriva a 10 ma a 16 o 2.

Esempi:

1011 + 0011 = 1110

3A09 + 1B17 = 5520

Di solito i sistemi utilizzano un numero di bit fisso e se il numero eccede i bit predefiniti si chiama overflow.

Esempio:

9 + 12 = 1001 + 1100 = 10101 = 0101 = 5 è sbagliato

1 non viene contato perché eccede il numero di bit

1.2.2-Moltiplicazioni:

Si moltiplica il primo numero per ogni bit del secondo numero spostandosi di una posizione ogni volta

$$0101 \cdot 0111 = 0101 + 01010 + 010100 + 000000 = 0100011$$

1.3-Numeri con il segno

Per scrivere i numeri con il segno in binario si utilizza un bit per identificare il segno (quello più a sinistra):

- 0 è un numero positivo
- 1 è un numero negativo

Esempio:

1110 = -6

0110 = 6

Con N cifre di un numero binario con segno:

- Si possono scrivere 2^{N-1} valori
- Con un range: $[-(2^{N-1}) - 1; 2^{N-1} - 1]$

1.4-Complemento a due

Per fare le addizioni però i numeri si scrivono con 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 un numero non basta più cambiare la prima cifra ma bisogna invertire gli 1 con gli 0 e aggiungere 1.

Esempio:

27 = 011011

-27 = 100100 +1 = 100101

Se si sommano due numeri con lo stesso segno è possibile che la somma esca dal range a causa dell'overflow.

Esempio:

7 + 10 = 0111 + 01010 = 10001 = -15 è sbagliato e bisogna aggiungere un bit con uno 0 all'inizio = 010001 = 17 è giusto

-16 + (-2) = 10000 + 11110 = 101110 = 14 è sbagliato e bisogna aggiungere un bit per l'1 all'inizio 101110 = -18 è giusto

1.5-Estendere il numero di bit

1.5.1-Senza segno:

Se durante una somma o una moltiplicazione il numero di bit eccede il massimo e c'è un overflow, bisogna estendere il numero di bit. Per farlo basta aggiungere tutti 0 davanti al numero.

N bit·N bit = 2N bit

Esempio:

1010 + 1000 = 10010 = 1 non viene considerato per l'overflow quindi estendiamo il numero di bit= 01010 + 01000 = 10010

1.5.2-Complemento a due:

Per estendere il numero di bit di un numero con il complemento a due si aggiunge a sinistra tante volte la prima cifra significativa.

Esempi:

1010 = 111010

0111 = 000111

1.5.3-Shiftare i numeri binari

Se dividiamo o moltiplichiamo per una potenza di 2 possiamo spostare le cifre a destra o sinistra di quanti posti quanto l'esponente della potenza.

Esempi:

$00001 \cdot 2^2 = 00100 = 4 = 1 \cdot 2^2$

$1000 \cdot 2^2 = 100000 = -64 = -16 \cdot 2^2$

$01010/2^3 = 00001 = 1 = 10/2^3$ il resto non viene considerato

$10000/2^1 = 11000 = -8 = -16/2$ nei numeri negativi quando si divide si aggiungono 1 all'inizio

1.6-Numeri con la virgola

Per scrivere in binario un numero con la virgola possiamo continuare a sommare potenze di due anche per la parte decimale.

Esempi:

$0,5_{10} = 2^{-1} = 0,1_2$

$3,75_{10} = 3 + 2^{-1} + 2^{-2} = 11,11_2$

C'è un altro metodo, cioè moltiplicare per due la parte decimale e poi sottrarre 1 e aggiungere 1 al numero in binario per ogni volta che si sottrae 1 e 0 se il numero decimale è minore di 1, si continua fino a che la parte decimale non è 0.

Esempi:

$0,75_{10} \cdot 2 = 1,5_{10} - 1 = 0,5_{10} \implies 0,1_2 \implies 0,5_{10} \cdot 2 = 1–1 = 0 \implies 0,11_2$

$0,4 \cdot 2 = 0,8 < 1 \implies 0,0_2 \implies 0,8 \cdot 2 = 1,6 – 1 = 0,6 \implies 0,01_2 \implies 0,6 \cdot 2 = 1,2 - 1 = 0,2 \implies 0,011_2 \implies 0,2 \cdot 2 = 0,4 < 1 \implies 0,0110_2 \implies 0,4 \cdot 2 = 0,8 \implies$ siamo tornati a 0,8 quindi il numero è periodico $\implies 0,4_{10} = 0,0110_2$

Ci sono altri due modi per rappresentare i numeri con la virgola:

1.6.1-Fixed point

Si dà per scontato dove sia la virgola (viene deciso prima e ci si deve mettere d'accordo)

Esempi:

$01101100 = 108 \implies 0110,1100 = 6,75 = 108/16$

$7,5 = 0111,1000$

Anche i numeri decimali possono essere scritti con il segno o con il complemento a due

Segno:

Si aggiunge 1 all'inizio se è negativo o 0 se è positivo

$-7,5 = 1111,1000$

Complemento a due:

Per scrivere un numero decimale negativo in binario si usa il positivo, si inverte e poi si aggiunge 1 alla fine (non 1 come valore ma 0,0…..1)

$-5,5 \implies 5,5 = 0101,1 \implies 10100 + 1 = 10101 \implies 1010,1$

1.7-Floating point

Si scrive molto simile alla notazione scientifica cioè:

$\pm M \cdot BE$

M = mantissa, numero con solo una cifra prima della virgola

B = base

E = esponente

La scrittura standard è composta da 32 bit, 1 per il segno, 8 per l'esponente e 23 per la mantissa. All'esponente bisogna aggiungere 127 (quindi per scrivere 5 dovremo scrivere 132) e nella mantissa non bisogna considerare la cifra prima della virgola perché è sempre un 1.

Per scrivere un numero con questa notazione bisogna:

1. Convertire il numero in binario senza segno
2. Scrivere il numero in notazione scientifica binaria
3. Completare i 32 bit in modo opportuno non considerando la cifra prima della virgola e aggiungendo 127 all'esponente

Esempio:

228 = 11100100

$11100100 = 1,11001 \cdot 2^7 \implies$
$0(\text{segno})10000110(\text{esponente} + 127)11001000000000000000000(\text{mantissa}) \implies$
$0\ 10000110\ 11001000000000000000000$

Per scrivere questi numeri in modo compatto si scrivono in esadecimale

Esempio:

$01000011011001000000000000000000 = 0x43640000$

$-58,25 = 111010,01 = 1,1101 \cdot 2^5 = 11000010011010000000000000000000 = 0xC2690000$

Numeri speciali:

Numero	Segno	Esponente	Mantissa
0	indifferente	00000000	00000000000...
∞	0	11111111	00000000000...
–∞	1	11111111	00000000000...
NaN (not a number)	indifferente	00000000	M≠0

1.7.1-Numeri denormalizzati:

I numeri denormalizzati hanno esponente uguale a 0 e mantissa diversa e servono per scrivere numeri minori dell'esponente minimo che si può avere.

In questi numeri la mantissa non inizia più con 1,… ma con 0,…

Numero massimo: 2^{126}

Numero minimo normalizzato: 2^{-126}

Numero minimo denormalizzato: 2^{-149}

Del floating point esistono altre due versioni.

1.7.2-Half precision:

- Ha solamente 16 bit, di cui 1 per il segno, 5 per l'esponente e 10 per la mantissa
- L'esponente viene sottratto di 15

Numero massimo: $2^{16}–2^{5}$

Numero minimo normalizzato: 2^{-14}

Numero minimo denormalizzato: 2^{-24}

I numeri speciali funzionano come il floating point.

1.7.3-Double precision:

- Ha 64 bit, di cui 1 per il segno, 11 per l'esponente e 52 per la mantissa
- L'esponente viene sottratto di 1023

Numero minimo: 10^{-308}

Numero massimo: 10^{308}

1.7.4-Approssimazioni

I numeri binari si possono approssimare in diversi modi:

- eccesso
- difetto
- verso lo 0 (è uguale al difetto se il numero è positivo e all'eccesso se è negativo)
- più vicino (si sceglie il più vicino tra eccesso e difetto)

Esempio:

approssimiamo 1,578125 con 3 bit

1,578125 = 1,100101

- eccesso: 1,101 = 1,625
- difetto: 1,100 = 1,5
- verso 0: 1,100 = 1,5
- più vicino: 1,101 = 1,625 perché 1,625 è più vicino di 1,5

1.8-Operazioni con floating point

1.8.1-Somma:

- estrarre in numeri e scriverli in notazione scientifica binaria
- cambiare l'esponente minore per renderli uguali
- sommare le mantisse
- riscrivere in notazione scientifica binaria
- arrotondare se non bastano i bit
- riscrivere nel formato floating point

Esempio:

0x3FC0000 + 0x40500000

$0x3FC00000 = 0011111111000000\ldots = 1,1 \cdot 2^0$

$0x40500000 = 0100000001010000\ldots = 1,101 \cdot 2^1$

$1,1 \cdot 2^0 = 0,11 \cdot 2^1$

$0,110 \cdot 2^1 + 1,101 \cdot 2^1 = 10,011 \cdot 2^1$

$10,011 \cdot 2^1 = 1,0011 \cdot 2^2 = 0100000010011000000\ldots = 0x40980000$

1.8.2-Moltiplicazione:

- estrarre in numeri e scriverli in notazione scientifica binaria
- sommare gli esponenti
- moltiplicare le mantisse
- riscrivere in notazione scientifica binaria
- arrotondare se non bastano i bit
- riscrivere nel formato floating point il risultato

Esempio:

$1,1 \cdot 2^{10} \cdot 1,0110 \cdot 2^{11}$

$1,0110 \cdot 1,1 = 10,0001$

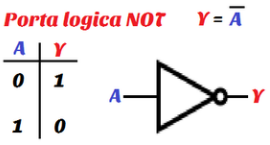
$2^{10} + 2^{11} = 2^{21} \implies 10,0001 \cdot 2^{21} = 1,00001 \cdot 2^{22} = 01001010100001000000\ldots = 0x4A840000$

2-Porte logiche

Le porte logiche hanno un output e possono avere:

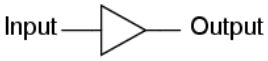
- 1 input: not, buffer
- 2 input o più: and, or, xor ecc…

NOT



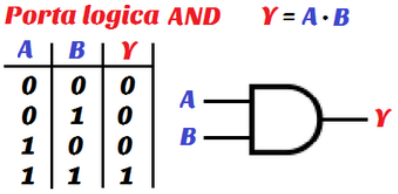
BUFFER

"Buffer" gate



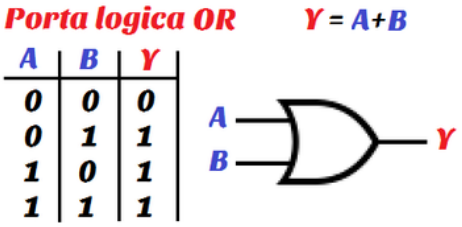
Input	Output
0	0
1	1

AND



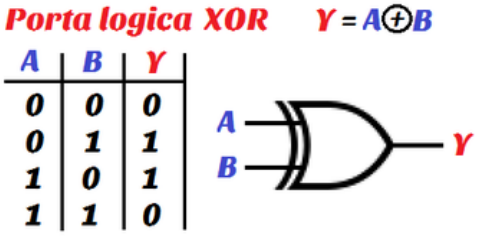
Può essere scritta come una moltiplicazione: A and B = A·B

OR

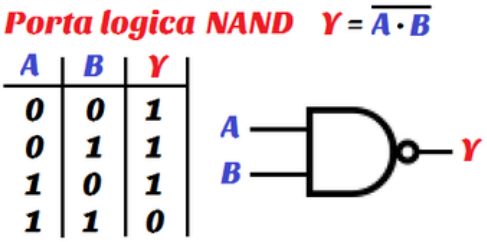


Può essere scritta come una somma: A or B = A+B

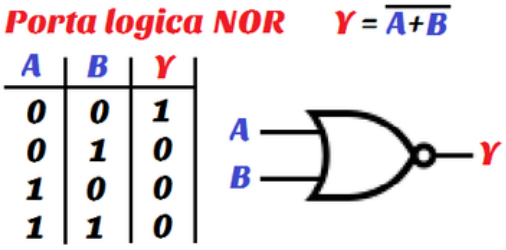
XOR



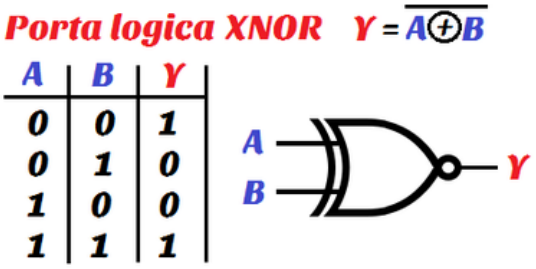
NAND



NOR



XNOR



AND3

Vero solo se tutte e tre sono veri

OR3

Vero quando uno o più sono veri

XOR3

Vero quando il numero di 1 è dispari

2.1-Circuiti logici

Un circuito logico è composto da:

- input
- output
- caratteristiche funzionali
- caratteristiche temporali

Un circuito ha:

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

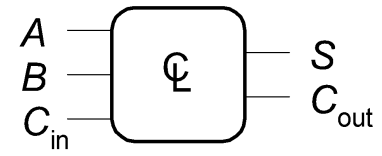
I circuiti sono di due tipi:

- combinatori logici: non hanno memoria e gli output sono definiti solo dagli input correnti
- sequenziali logici: hanno memoria e gli output sono definiti dagli input correnti e precedenti

Regole dei circuiti combinatori:

- Tutti i circuiti elementari sono combinatori
- Ogni nodo è un input o si collega ad esattamente un output
- non ci sono cicli

2.2-Equazioni booleane



$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Definizioni:

Complemento: il contrario di una variabile = $\overline{A}$

Literal: la variabile o il suo complemento = A o $\overline{A}$

Implicante: prodotto di literal = $A \cdot B$ o $B \cdot \overline{C}$

Minterm: prodotto che usa tutte le variabili di input = $A \cdot B \cdot C$ o $\overline{A} \cdot B \cdot \overline{C}$

Maxterm: somma che utilizza tutte le variabili di input = A+B+C o $\overline{A}$+ B + C

2.2.1-Somma di prodotti(SOP):

Tutte le equazioni possono essere scritte come somma di prodotti.

A	B	Y	minterm	Nome del minterm
0	0	0	$\overline{A}\,\overline{B}$	m_0
0	1	0	$\overline{A}B$	m_1
1	0	0	$A\overline{B}$	m_2
1	1	1	AB	m_3

Ogni riga ha un minterm che in quella riga vale 1.

Y = F(A,B) = somma dei minterm uguali a 1 $\implies Y = \overline{A}B + AB = B(\overline{A} + A) = B$

Può anche essere scritto come: $\sum (1, 3)$ indicando il pedice dei minterm sommati.

2.2.2-Prodotto di somma(POS):

Tutte le equazioni possono essere scritte come prodotto di somma.

A	B	Y	maxterm	Nome del maxterm
0	0	0	A+B	M_0
0	1	0	$A + \overline{B}$	M_1
1	0	0	$\overline{A} + B$	M_2
1	1	1	$\overline{A} + \overline{B}$	M_3

Ogni riga ha un maxterm che in quella riga vale 0.

Y = F(A,B) = prodotto dei maxterm uguali a 0 $\implies Y = (A + B) \cdot (\overline{A} + B) = \overline{A}A + AB + \overline{A}B + BB = AB + \overline{A}B + B = B$

Può anche essere scritto come: $\prod (0, 2)$ indicando il pedice dei maxterm moltiplicati.

3-Algebra booleana

Assiomi e teoremi per semplificare l'algebra.

Ci sono solo le variabili 1 e 0.

Dualità di assiomi e teoremi:

- I teoremi con OR e AND possono essere interscambiati tra loro modificando i + in · e gli 0 in 1

3.1-Assiomi:

Assioma	Duale
B = 0 se B≠1	B = 1 se B≠0
$\overline{0} = 1$	$\overline{1} = 0$
0·0 = 0	1 + 1 = 1
1·1 = 1	0 + 0 = 0
0·1 = 1·0 = 0	1 + 0 = 0 + 1 = 1

3.2-Teoremi:

Teorema	Duale
B*1 = B	B + 0 = B
B*0 = 0	B + 1 = 1
B*B = B	B + B = B
$\overline{\overline{B}}$= B	$\overline{\overline{B}} = B$
B·$\overline{B}$= 0	B +$\overline{B}$= 1
B·C=C·B	B+C=C+B
B·C·D=B(C*D)	B+C+D=B+(C+D)
B(C+D)=(B·C)+(B·D)	B+(C·D)=(B+C)·(B+D)
B·(B+C)=B	B+(B·C)=B
(B·$\overline{C}$)+(B·C)=B	(B+$\overline{C}$)·(B+C)=B
B(1+C)=B·1	B+(0·C)=B+0

3.3-Semplificare un’equazione

Ridurre un’equazione al numero minimo di implicanti, dove ogni implicante compare il numero minimo di volte (detta anche minimizzare un’equazione).

Esempi:

P$\overline{A}$+A=P+A

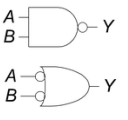
PA+$\overline{A}$=P+$\overline{A}$

A+AP=A

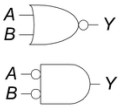
P$\overline{A}$+PA=P

3.3.1-Teorema de Morgan

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



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



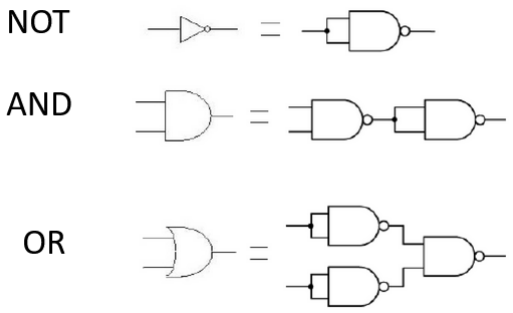
Esempi:

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

$$Y = \overline{\overline{AC}E + \overline{D} + B} = \overline{\overline{AC}E} \cdot D \cdot \overline{B} = \left(AC + \overline{E}\right) \cdot D\overline{B} = A\overline{B}CD + \overline{B}D\overline{E}$$

3.4-Completezza

Le porte NAND (e NOR) sono funzionalmente complete, cioè grazie a quelle si possono creare tutte le altre porte.



4-Circuiti

4.1-Regole dei circuiti

- Gli input si mettono a sinistra o in alto
- Gli output si mettono a destra o in basso
- I gate vanno da sinistra a destra

Fili che si intrecciano:

- se formano una giunzione a T sono collegati
- se c’è un punto sopra la giunzione sono collegati
- se si incrociano ad X senza puntini non sono collegati

SI SI NO

4.1.1-Circuiti con più input

Circuito a priorità

Se è attivo l’input più importante gli altri non sono considerati.

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 non vengono considerati, quindi possiamo scrivere la tabella in maniera semplificata.

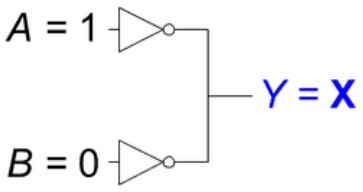
A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	**0**	**0**	**0**	**0**
0	0	0	1	**0**	**0**	**0**	**1**
0	0	1	X	**0**	**0**	**1**	**0**
0	1	X	X	**0**	**1**	**0**	**0**
1	X	X	X	**1**	**0**	**0**	**0**

X = don’t care, cioè il valore non viene considerato nell’equazione

4.1.2-Contention

Succede quando in un circuito si incontrano due risultati diversi, uno con 0 e uno con 1. Si indica con X e solitamente implica un problema.

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

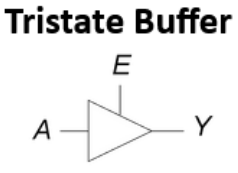


4.1.3-Floating

Si ha quando un filo è disattivato e non c’è corrente (diverso da 0).

Si scrive con Z.

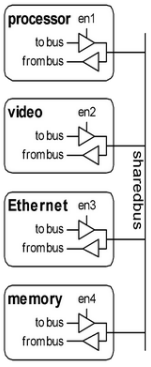
Un esempio è il tristate buffer, un buffer che è attivo solo quando E è 1.



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

4.1.4-Tristate busses

Z è usato nei tristate busses, dove diversi driver sono collegati ad un solo filo, ma solo uno alla volta è attivo.



4.2-Mappe Karnaugh(K-maps)

Le espressioni booleane possono essere semplificate combinando i termini.

Con le mappe K possiamo semplificarle graficamente. Ogni riquadro indica una riga della tabella della verità e un minterm. Il risultato verrà scritto in forma SOP.

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

Y \ AB / C	00	01	11	10
0	1	0	0	0
1	1	0	0	0

Y \ AB / C	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	ABC	$A\bar{B}C$

Ogni minterm della tabella può essere semplificato con quelli vicino(anche da destra a sinistra), togliendo il literal che cambia valore.

Dopo aver posizionato gli 0 e 1 nella mappa dobbiamo cerchiare gli 1 secondo delle regole:

- devo ricoprire tutti gli 1 almeno una volta
- ogni cerchio deve contenere un numero di 1 pari ad una potenza di 2 (1,2 o 4)
- bisogna prendere il cerchio più grande possibile
- il cerchio può contenere un don’t care (X) solo se serve a creare un cerchio più grande

Truth Table

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

K-Map

Y \ AB / C	00	01	11	10
0	0	1	0	0
1	0	1	1	0

Y = $\overline{A}B$+ BC

4.2.1-K-maps con 4 input

Si fa semplicemente aggiungendo una variabile in verticale.

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

Y / 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

Le mappe K possono anche essere usate per creare delle POS, prendendo gli 0 al posto degli uno e moltiplicando i maxterm.

Y / CD \ AB	00	01	11	10
00	A+B+C+D	A+B'+C+D	A'+B'+C+D	A'+B+C+D
01	A+B+C+D'	A+B'+C+D'	A'+B'+C+D'	A'+B+C+D'
11	A+B+C'+D'	A+B'+C'+D'	A'+B'+C'+D	A'+B+C'+D'
10	A+B+C'+D	A+B'+C'+D	A'+B'+C'+D	A'+B+C'+D

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

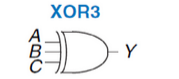
Y / CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

Y=(A'+B')(B'+C+D)(A'+C'+D')
(A+B+C+D')

4.3-Logica combinatoria multi-livello

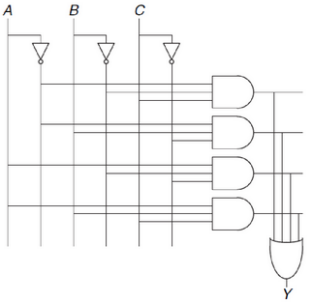
La logica in SOP/POS è una logica a due livelli, di solito però si creano circuiti a più di due livelli, perché utilizzano meno spazio.

XOR3



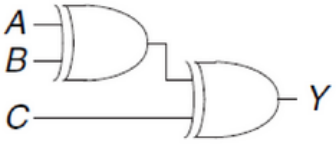
$Y = A \oplus B \oplus C$

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



A livello di circuiti lo XOR a molti ingressi è molto più efficiente farlo con

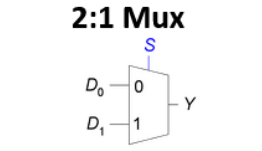
(A ⊕ B)⊕ C rispetto che A⊕ B⊕ C.



5-Blocchi combinatori

5.1-Multiplexer(mux)

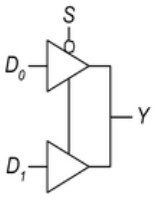
Selezione uno di N input da far uscire come output. Il numero di selezionatori necessario è $\log_2 N$.

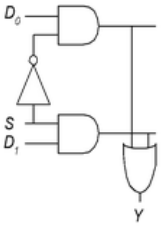


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

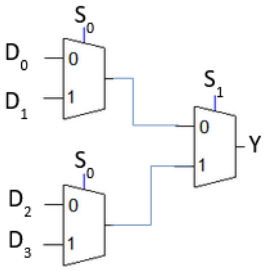
S	Y
0	D_0
1	D_1

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



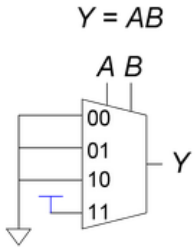


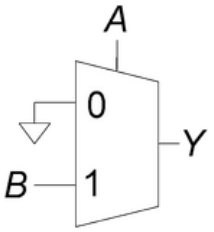
5.1.1-4 ingressi:



S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Con i mux posso fare anche tutte le altre funzioni logiche, sia con due input che con uno.

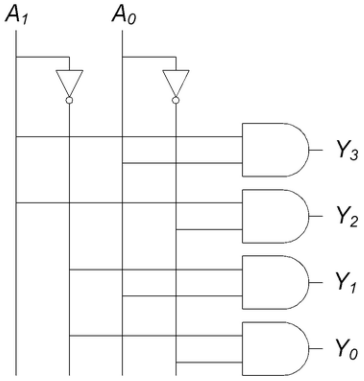


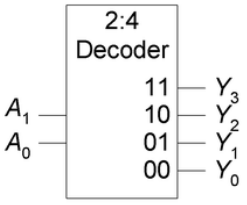


5.2-Decoder

Ha N input e 2^N output.

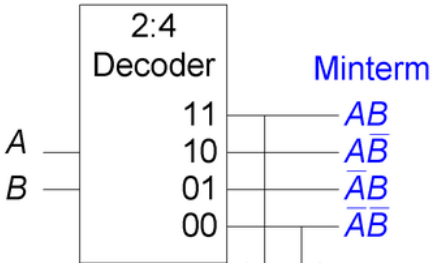
One-hot: solo un output è attivo alla volta





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

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



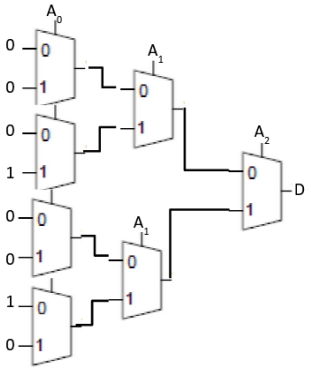
5.3-Teorema di Shannon

Data una funzione con n variabili, possiamo semplificarla in 2 mux e una funzione con n-1 variabili.

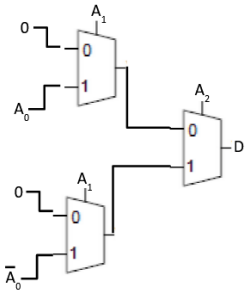
$f(x_1, x_2, \ldots, x_n) = x1 \cdot f(0, x_2, \ldots, x_n) + \overline{x_1} \cdot f(1, x_2, \ldots, x_n)$

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



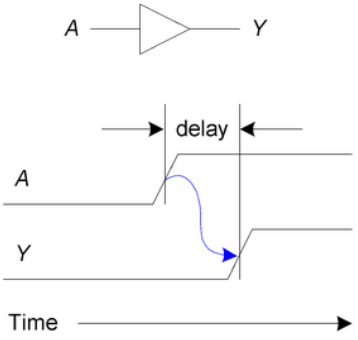
Semplificandolo con il teorema di Shannon diventa:



5.4-Delay

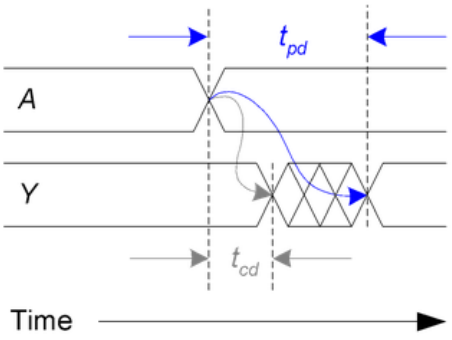
Il delay è il tempo da quando cambia un input fino a quando cambia l'output.

Viene considerato da quando l'input è a metà del cambiamento fino a quando l'output è a metà del cambiamento.

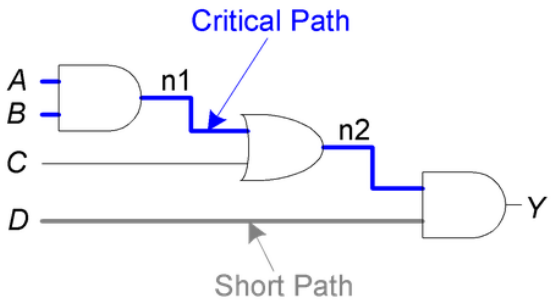


Ci sono due tipi di tempi che possiamo calcolare:

- Propagation delay = tempo massimo: t_{pd}
- Contamination delay = tempo minimo: t_{cd}



Il delay minimo si calcola sommando tutti i tempi massimi delle porte logiche del percorso più lungo e il delay minimo sommando tutti i tempi minimi delle porte logiche del percorso più corto.



$t_{pd} = 2t_{pd\_AND} + t_{pd\_OR}$

$t_{cd} = t_{cd\_AND}$

Il delay può essere causato da:

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

Motivi per cui t_{cd} e t_{pd} sono diversi:

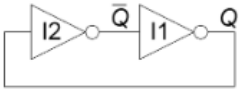
- diversi tempi di innalzamento e discesa
- multipli input e output, alcuni più veloci di altri
- rallentamenti dovuti alla temperatura

6-Circuiti sequenziali

I circuiti sequenziali posseggono una memoria per immagazzinare i precedenti valori, quindi l'output dipende anche dai valori che ha in memoria. Rappresenta l'insieme di valori passati racchiudendoli nello stato del sistema, cioè le informazioni su un circuito necessarie per spiegare i comportamenti futuri. Lo stato del sistema viene memorizzato in blocchetti chiamati Latches e Flip-Flops, che immagazzinano un bit dello stato.

6.1-Circuito bistabile

Circuito con 2 output ma nessun input.

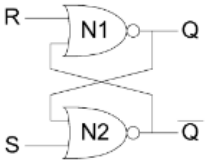


Se Q = 0 $\implies$ $\overline{Q}$= 1

Se $\overline{Q}$= 0 $\implies$ Q = 1

Può immagazzinare 1 bit di stato.

6.2-SR (Set/Reset) Latch



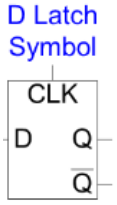
Consideriamo i 4 casi:

S	R	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	Q vale come il valore precedente	$\overline{Q}$ vale come il valore precedente
1	1	0	0

L'ultimo caso non è possibile perché $Q \neq \text{NOT}\overline{Q}$

Per risolvere questo caso si usa il D Latch.

6.3-D Latch



Ha due input, CLK che controlla quando l'input cambia e D che è l'input.

Se CLK = 1 allora D passa il suo valore a Q

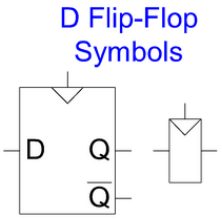
Se CLK = 0 allora Q mantiene il valore precedente

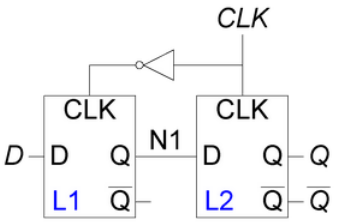
CLK	D	Q	$\overline{Q}$
0	X	Q precedente	$\overline{Q}$ precedente
1	0	0	1
1	1	1	0

6.4-Flip-Flop

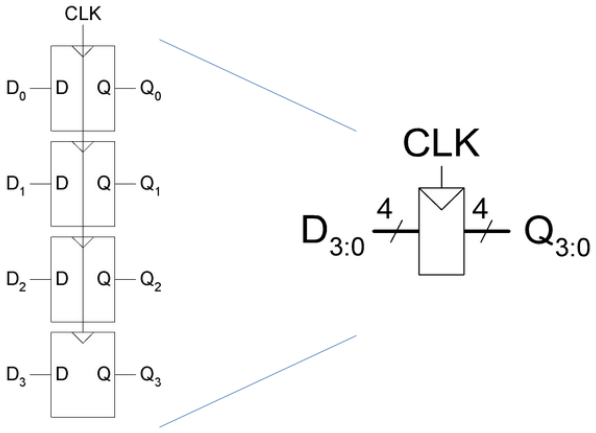
Un flip-flop memorizza il valore quando il clock passa da 0 a 1 (non da 1 a 0).

Può essere anche creato con dei D Latch.



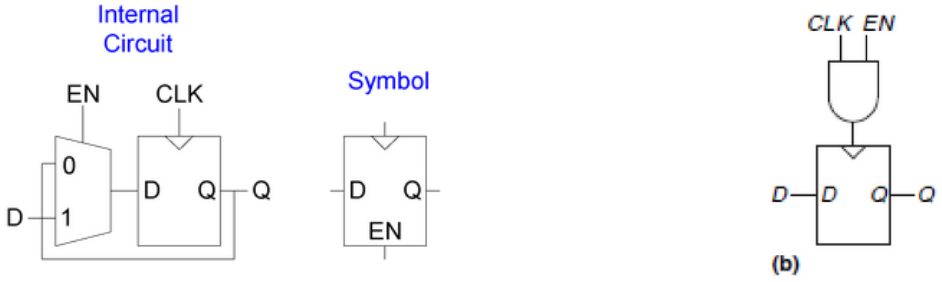


6.4.1-Flip-Flop multipli



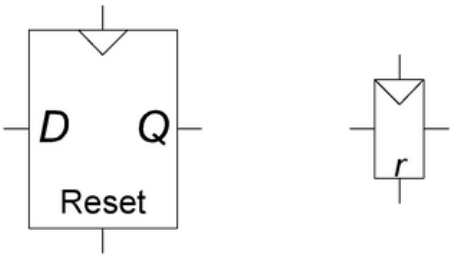
6.4.2-Flip-Flop attivabili

- Quando en=0: D passa a Q
- Quando en=1: Il flip-flop torna normale



6.4.3-Flip-Flop resettabili

- Quando R=1: Q diventa 0
- Quando R=0: Il flip-flop torna normale

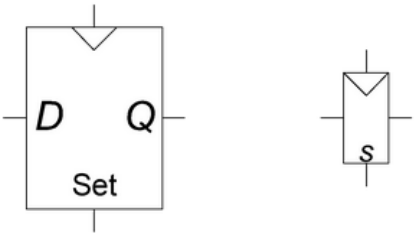


Ce ne sono di due tipi:

- sincroni: resetta Q a 0 quando il clock passa da 0 a 1
- asincroni: resetta Q a 0 appena R=1

6.4.4-Flip-Flop settabili

- Quando S=1: Q diventa 1
- Quando S=0: Il flip-flop torna normale



6.5-Logica sequanziale sincrona

- I cicli vengono interrotti da dei registri che contengono lo stato del sistema
- Lo stato cambia quando il clock passa da 0 a 1

Regole:

- Ogni circuito è un registro oppure un circuito combinatorio
- Almeno un elemento è un registro
- Tutti i registri hanno lo stesso clock
- Ogni circuito ha almeno un registro

7-Finite State Machine(FSM)

Composta da:

Registri:

- memorizzano lo stato corrente
- cambiano lo stato memorizzato quando il clock si attiva

Logica combinatoria:

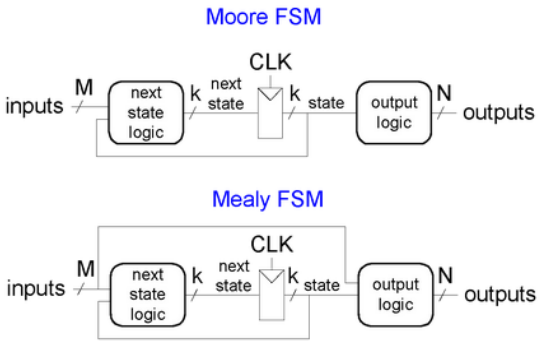
- calcola il prossimo stato
- calcola gli output

Nelle FSM lo stato successivo è definito da quello precedente.

Ci sono due tipi di FSM:

- Moore FSM: output dipende solo dallo stato corrente

- Mealy FSM: output dipende dallo stato corrente e dagli input

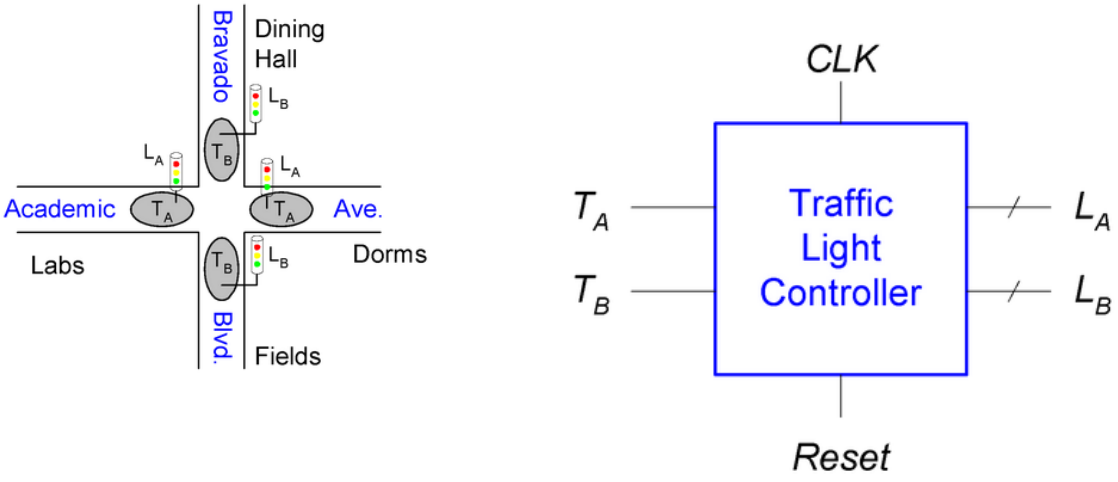


Esempio di FSM

Controllo del traffico:

Sensori: Ta, Tb sono True quando c'è traffico

Luci: La, Lb



7.1-Diagramma di Transizione

Stati: cerchi

Transizioni: archi

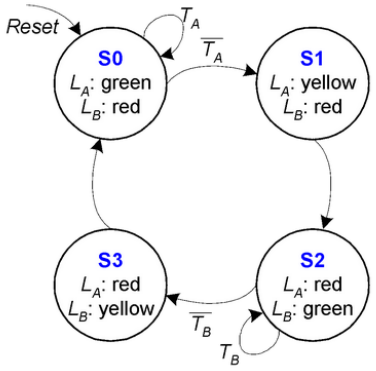


Tabella di transizione

Stato attuale	Input	Input	Prossimo stato
S	Ta	Tb	S’
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Tabella codificata:

S	Codifica
S0	00
S1	01
S2	10
S3	11

Stato attuale	Input	Input	Input	Stato successivo	Stato successivo
S1	S0	Ta	Tb	S’1	S’0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

Tabella degli output:

Output	Codifica
green	00
yellow	01
red	10

Stato attuale	Stato attuale	Output	Output	Output	Output
S1	S0	La1	La0	Lb1	Lb0
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

7.2-Codifica degli stati di una FSM

Può essere binaria:

- ogni numero binario rappresenta uno stato
- Es. 4 stati: 00, 01, 10, 11

O one-hot:

- 1 bit per ogni stato

- solo un bit è uguale a 1
- richiede più flip-flop
- Es. 4 stati: 0001, 0010, 0100, 1000

7.3-Differenza tra Moore e Mealy

- Moore ha bisogno di più stati di Mealy

Se l'output di Mealy fosse rallentato da un flip-flop, avrebbe gli output sincronizzati con Moore, questo:

- evita output "glitchati" (sbalzi di valore veloci)
- evita che le macchine comunicanti vadano in una condizione di corsa (l'output glitchato di una modifica l'output dell'altra creando una reazione a catena)

Esempio:

Una macchina che legge una stringa binaria e sorride se c'è 01.

Moore:

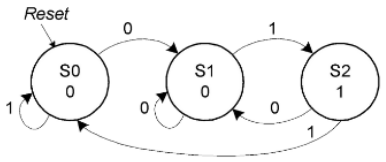


Tabella di trascrizione:

S	Codifica
S0	00
S1	01
S2	10

S1	S0	I	S'1	S'0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

S1 = S0cotA

S0 = $\overline{A}$

Tabella degli output:

S1	S0	Y
0	0	0
0	1	0
1	0	1

Y=S1

Mealy

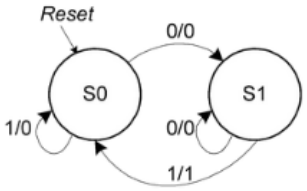


Tabella di trascrizione e output:

S	Codifica
S0	0
S1	1

S0	I	S'0	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

Y = S0·A

7.3.1-Passare da Moore a Mealy

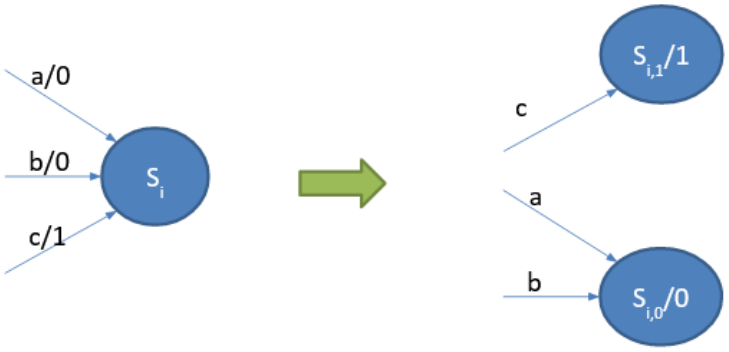
Per passare da una FSM Moore a Mealy bisogna:

- spostare gli output dagli stati agli archi
- ogni arco entrante in uno stato deve avere l'output dello stato
- se due stati hanno gli stessi archi uscenti, possono essere uniti in un solo stato

7.3.2-Passare da Mealy a Moore

Per passare da una FSM Mealy a Moore bisogna:

- spostare gli output dagli archi agli stati
- se uno stato ha due archi entranti con output diversi, bisogna dividerlo in due stati



7.4-Stati irraggiungibili

Sono stati che non possono essere raggiunti e solitamente si usano come stati di inizio o di reset. Sono presenti nelle FSM che hanno un numero di stati non potenza di 2.

Procedura per il design di una FSM:

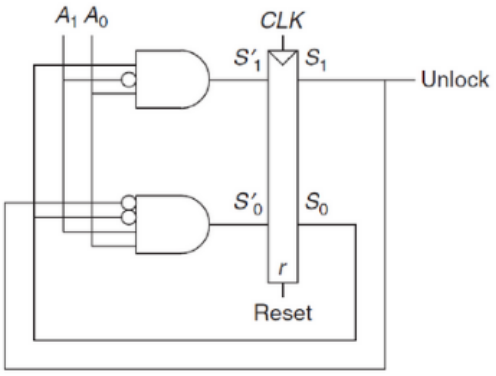
1. Identifica input e output
2. Disegna il diagramma di transizione
3. Scrivi la tabella di trascrizione
4. Codifica gli stati
5. Per Moore: Riscrivi la tabella di trascrizione con gli stati codificati e fai quella degli output

   Per Mealy: Riscrivi la tabella di trascrizione e degli output con gli stati codificati
6. Scrivi l'equazione booleana per la next state logic e per gli output
7. Disegna il circuito

Derivare una FSM da un circuito:

1. Esamina circuiti, input, output e bit di stato
2. Scrivi le equazioni booleane degli stati e degli output
3. Crea la tabella della next state logic
4. Riduci la tabella eliminando gli stati irraggiungibili
5. Assegna a ogni combinazione di bit il nome di uno stato
6. Riscrivi la tabella con i nomi degli stati
7. Disegna il diagramma di stato
8. Descrivi a parole

Esempio:



Equazioni:

S'1 = $\overline{A_1} A_0 S_0$

S'0 = $A_1 A_0 \overline{S_1}\ \overline{S_0}$

Tabella:

S1	S2	A1	A0	S'1	S'0	Y
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	1	0	1
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	1	0	1
1	1	1	0	0	0	0
1	1	1	1	0	0	0

Tabella ridotta con nome degli stati:

S1	S0	Stato
0	0	S0
0	1	S1
1	0	S2

S	A	S'	Y
S0	0	S0	0
S0	1	S0	0
S0	2	S0	0
S0	3	S1	0

S	A	S’	Y
S1	0	S0	0
S1	1	S2	1
S1	2	S0	0
S1	3	S0	0
S2	X	S0	0

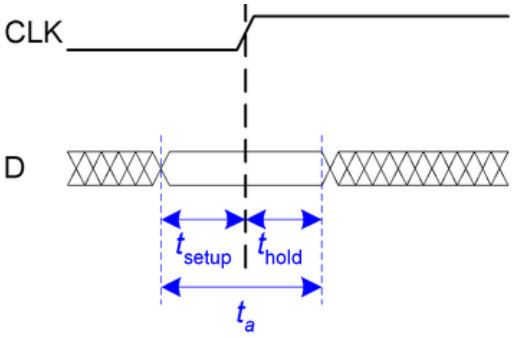
8-Timing

I flip-flop salvano il valore corrente secondo il clock, ma il valore deve essere stabile in quel momento se no si entra in un caso di metastabilità.

La metastabilità è quando ci si trova in uno stadio intermedio tra 0 e 1 da cui prima o poi si uscirà tornando a 0 o a 1.

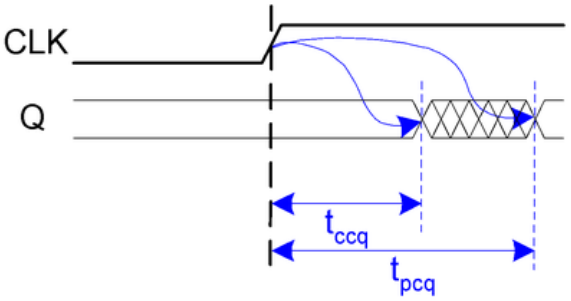
Input:

- Periodo di setup(tsetup): periodo prima del clock in cui il valore deve essere stabile
- Periodo di hold(thold): periodo dopo il clock in cui il valore deve essere stabile
- Periodo di apertura: somma del periodo di setup e di hold



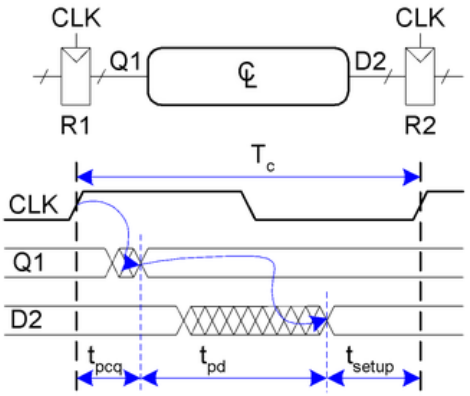
Output:

- Delay di propagazione(tpcq): tempo massimo in cui siamo sicuri che l’output diventi stabile.
- Delay di contaminazione(tccq): tempo minimo in cui l’output potrebbe essere stabile



Disciplina dinamica

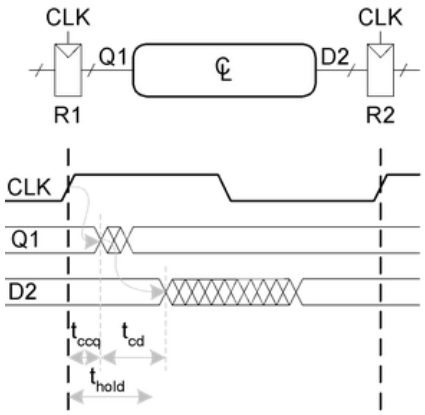
Il delay tra i registri ha un massimo e un minimo, che dipende dai circuiti.



I valori non possono cambiare per tutta la durata dei processi del primo registro, del circuito e del tempo di setup, quindi il tempo di clock dovrà essere maggiore:

Tc >= tcpq + tpd + tsetup

tpd <= Tc – (tpcq + tsetup)

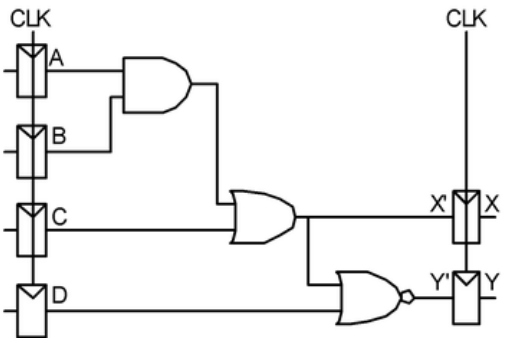


I valori non possono cambiare neanche prima del tempo di hold, che dovrà essere minore dei tempi minimi:

thold < tccq + tcd

tcd > thold – tccq

Es.



t

ccq

= 30 ps

tpcq = 50 ps

tsetup = 60 ps

thold = 70 ps

tpd = 35 ps per gate

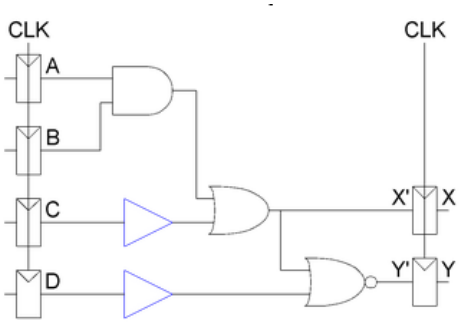
tcd = 25 ps per gate

Tc >= (50+(35*3)+60) = 215 ps

thold < tccq + tcd

70 < 30 + 25 => falso

Per ovviare a questo problema si aggiungono dei buffer.

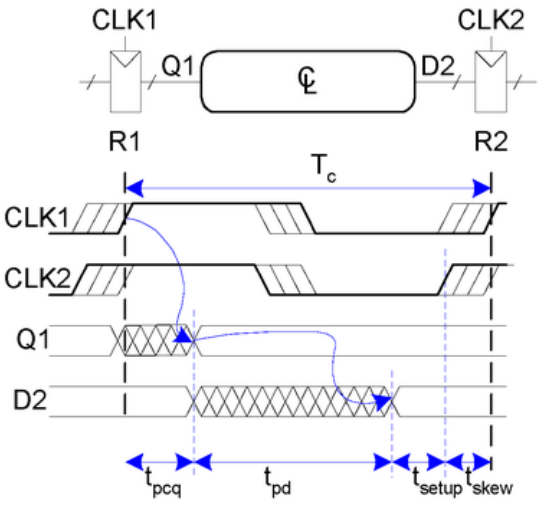


Clock stew

Il clock non arriva a tutti i registri allo stesso tempo.

Skew è la differenza tra il clock in due parti del circuito.

Per controllare che i tempi siano rispettati va sempre considerato il caso peggiore.



Quindi ora:

Tc >= tpcq + tpd + tsetup + tskew

thold + tskew < tccq + tcd

Parallelismi

Ci sono due tipi di parallelismo:

- parallelismo spaziale: un hardware duplicato fa più operazioni alla volta
- parallelismo temporale: le operazioni sono divise in più stage (chiamato pipeline)

Definizioni:

Token: gruppo di input processati

Latency: tempo che impiega per processare un token

Throughput: numero di token processati in un intervallo di tempo

Es.

5 min per fare i biscotti

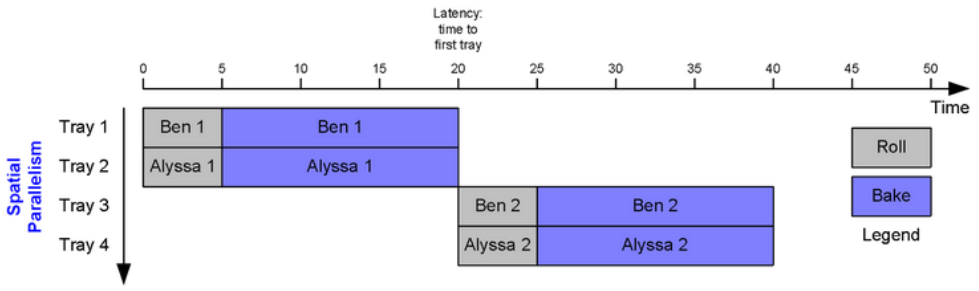
15 min per cuocerli

Senza parallelismo:

- Latency: 20 min
- Throughput: 3/h

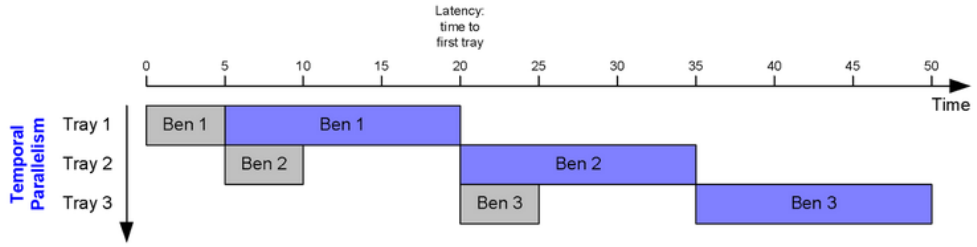
Parallelismo spaziale (un'altra persona fa altri biscotti nello stesso momento):

- Latency: 20 min
- Throughput: 6/h



Parallelismo temporale (mentre i biscotti cuociono inizia a prepararne altri):

- Latency: 30 min
- Throughput: 4/h

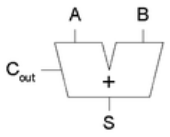


9-Blocchi costruttivi

Adder

1-bit:

Half Adder

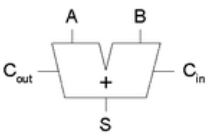


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C_{out} = AB$

Full Adder



C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S = A \oplus B \oplus C_{in}$

$C_{out} = AB + AC_{in} + BC_{in}$

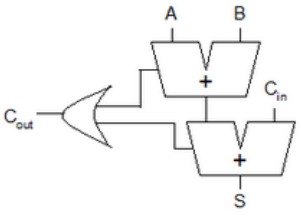
A,B = numeri: 1 bit

Cin = riporto in entrata: 1 bit

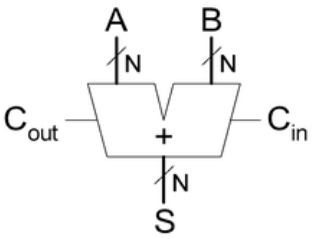
S = somma: 1 bit

Cout = riporto in uscita: 1 bit

Un full adder può anche essere creato con 2 half adder:



Multibit:



A, B = numeri: N bit

Cin = riporto in entrata: 1 bit

S = somma: N bit

Cout = riporto in uscita: 1 bit

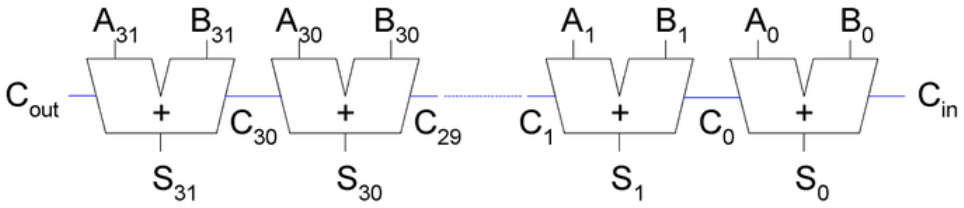
Ce ne sono di diversi tipi:

- Ripple-carry (lento)
- Carry-lookahead (veloce)
- Prefix (velocissimo)

Il carry-lookahead e il prefix sono più veloci ma occupano più porte logiche.

Ripple-carry:

Unisce adder a 1 bit tra di loro e il riporto passa attraverso l'intera catena.



tripple = NtFA

Carry-lookahead:

Per velocizzare rispetto al ripple-carry calcola direttamente il Cout.

Ci = ai*bi + (ai + bi)(ai-1*bi-1 + (ai-1 + bi-1)ci-1)

Gi = ai*bi = riporto generato nel bit i

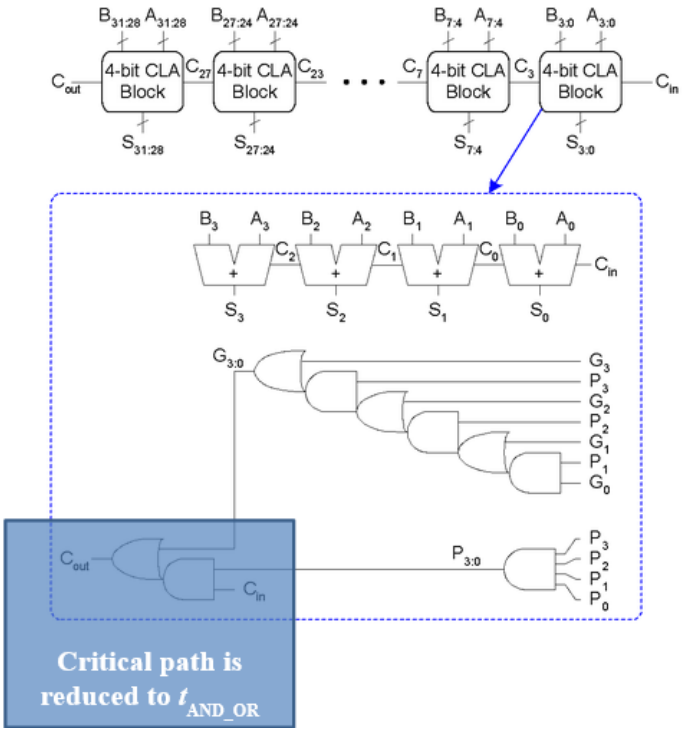
Pi = ai + bi = riporto propagato dal bit prima del bit i

Ci = Gi + Pi*Ci-1

Pi:j = Pi*Pi-1...Pj = riporto propagato dal bit j fino al bit i

Gi:j = Gi + Pi*Gi-1:j = riporto generato in un bit tra j e i

Per creare degli adder a molti bit(N) si usano blocchi più piccoli a k bit



Cosa fa l'adder:

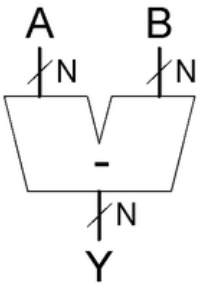
1. Calcola Gi e Pi per tutti i k bit
2. Calcola Gi:j e Pi:j per tutti i blocchi
3. Cin si propaga attraverso tutti i blocchi(nelle stesso momento delle somme)
4. Calcola la somma degli ultimi k bit

tCLA = tPG + tPG_BLOCK + ((N/K) – 1)tAND_OR + ktFA

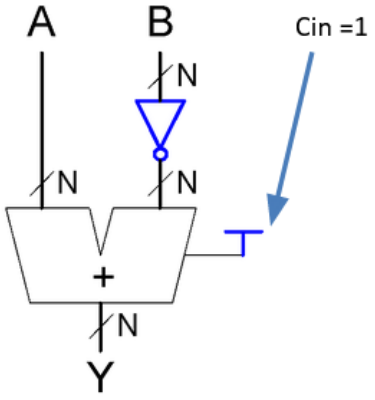
tPG = tempo per generare tutti i Gi e Pi

tPG_BLOCK = tempo per generare tutti i Gi:j e Pi:j

Subtracter

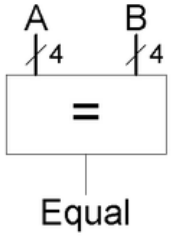


Si può anche fare usando un adder:

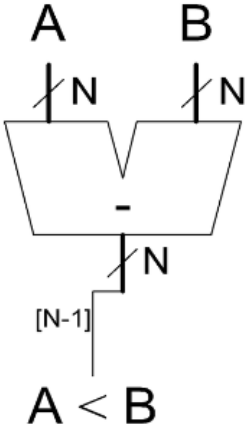


Equality

Esce 1 se è vero e 0 se è falso



Less than



Shifter

Eseguono divisioni o moltiplicazioni per 2.

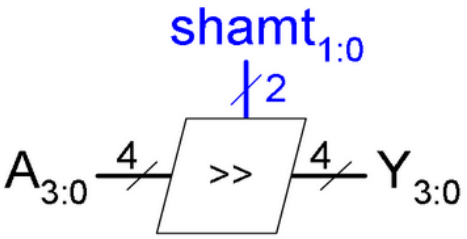
Shifter logico:

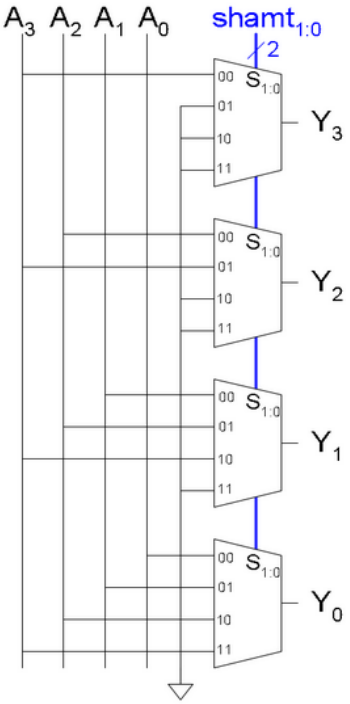
- Si usa per i numeri binari senza segno
- Quando shifta riempie gli spazi vuoti con 0

Es.

11001>>2 = 00110

11001<<2 = 00100 sbagliato perché esce fuori dal range





Shifter aritmetico:

- Si usa per i numeri binari in complemento a 2
- Se divide riempie gli spazi vuoti con il bit più significativi

Es.

11001>>2 = 11110

11001<<2 = 00100 sbagliato perché esce fuori dal range

Rotatori:

- Fanno ruotare i bit (cioè se un bit esce a sinistra torna a destra e viceversa)

Es.

11001 ROR2 = 01110

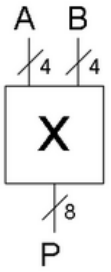
11001 ROL2 = 00111

Multipliers

Il prodotto parziale è calcolato moltiplicando un singolo bit del primo numero per il secondo numero. Sommando i prodotti parziali si ottiene il risultato.

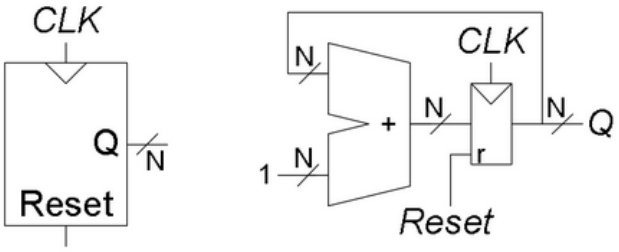
```
    0101
x   0111
-------
    0101
   0101
  0101
+ 0000
-------
  0100011
```

5 x 7 = 35



Counter

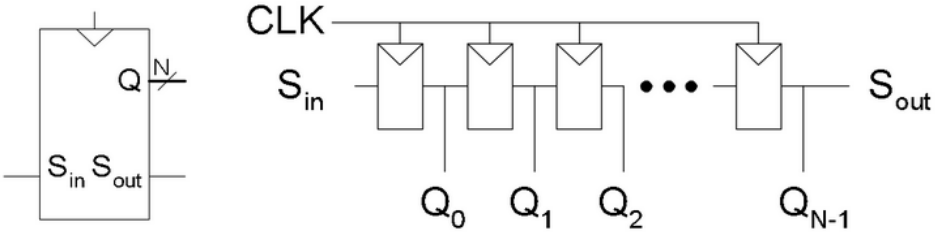
Aumenta ad ogni clock e cicla attraverso i numeri: 00, 01, 10, 11, 00…



Shift registers

Shiftano il numero di un bit ad ogni clock e inseriscono un bit di input.

Converte un input singolo in un output parallelo.



Shifter parallelo:

Quando load = 1, funziona come un normale registro ad N bit

Quando load = 0, funziona come uno shift register

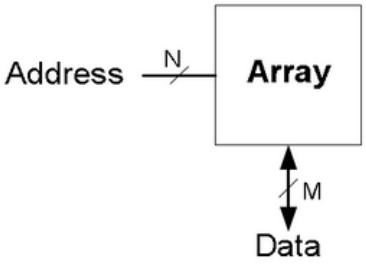
Funziona come convertitore da input singolo a output parallelo (Da Sin a Q0:N-1) o sa input parallelo a output singolo (Da D0:N-1 a Sout)

10-Memoria

Array

Immagazzinano grandi quantità di dati.

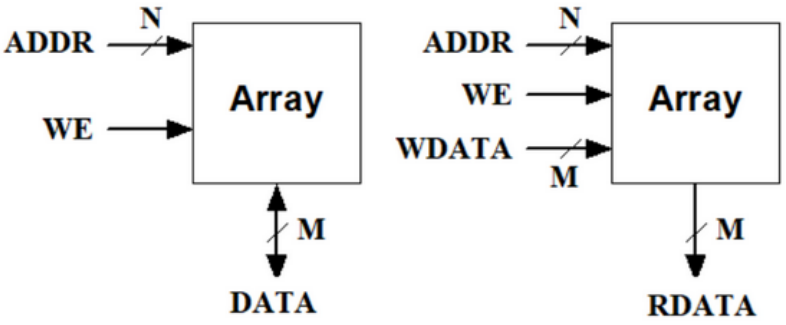
Vengono letti o scritti M bit per ogni indirizzo a N bit.



Sono delle celle con 2N righe e M colonne.

Un array di memoria di 2NxM ha:

- N bit indirizzo in input
- M bit bidirezionali di dati oppure M readdata bit e M writedata bit
- Un segnale WE (Write Enable)



wordline: legge la riga se uguale a 1

bitline: uguale al bit delle righe in cui wordline è 1

data: output dei bit

Si dividono in 2 tipi principali:

RAM(Random Access Memory) = volatile

ROM(Read Only Memory) = non volatili

RAM

Ci sono due tipi di RAM:

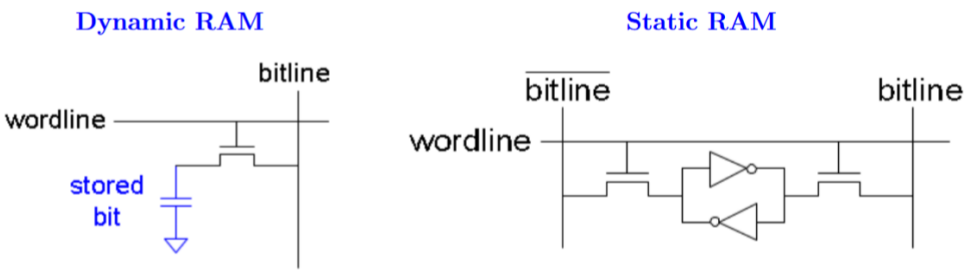
- DRAM (Dynamic Access Memory)
- SRAM (Static Access Memory)

DRAM

I bit vengono storati in un capacitor

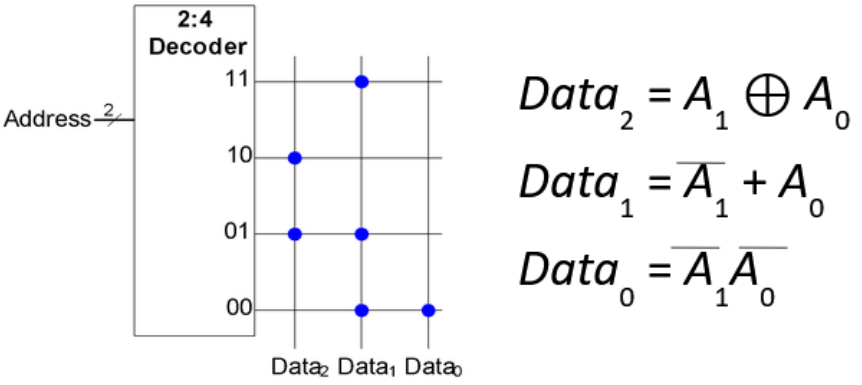
È dinamica perché deve essere riscritta periodicamente e dopo che viene letta:

- La perdita di carica nel capacitor fa diminuire il valore
- leggere un bit distrugge quello storato

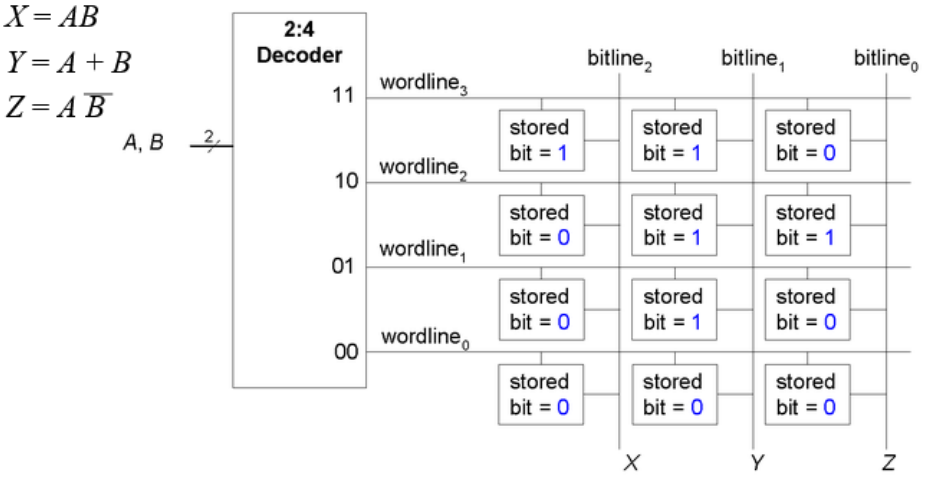


Logica con ROM

Si possono usare i pallini per indicare le celle che contengono 1.

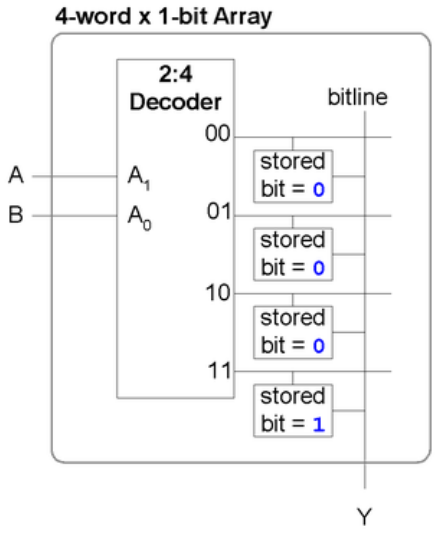


Logica con altri tipi di memorie



LUT (Lookup tables)

Sono memorie con solo un bit in output.

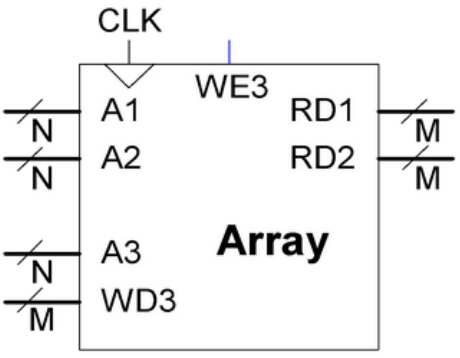


Memorie multiporta

Gli indirizzi e le uscite sono accoppiati.

Le più comuni sono a 3 porte:

- 2 porte di lettura
- 1 porta di scrittura



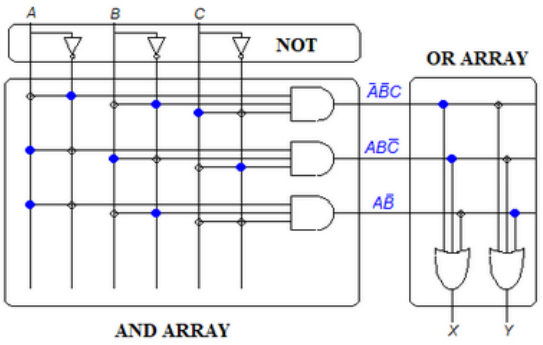
PLA (Programmable Logic Array)

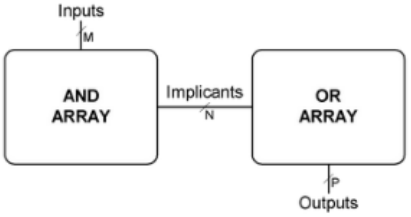
Sono composti da array AND seguiti da array OR. Possono eseguire solo logica combinatoria. Hanno connessioni interne fisse.

Hanno:

- M input
- P output
- 3 stadi interni:

1. Uno stadio di inversioni degli ingressi con dei not
2. Una matrice di AND
3. Una matrice di OR

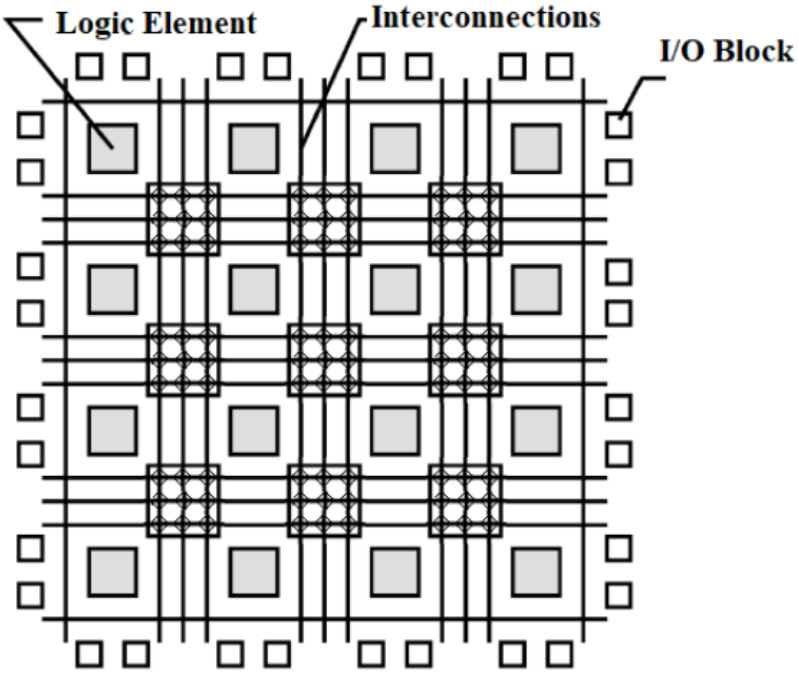




FPGA (Field Programmable Gate Array)

Sono composti da:

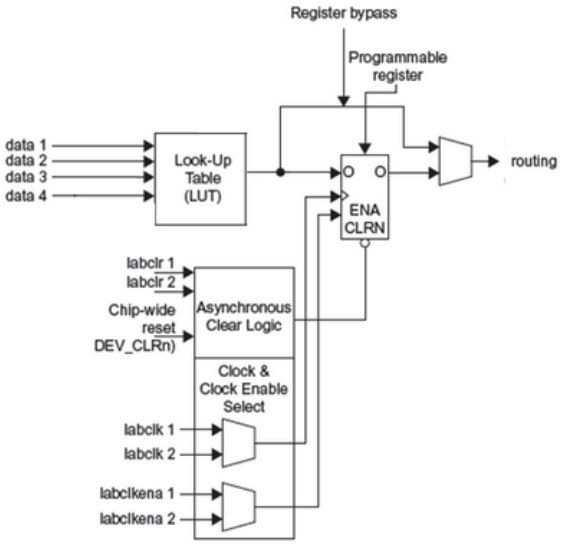
- Elementi logici
- Blocchi I/O: interfacce con l'esterno
- Connessioni programmabili: collegano gli elementi logici con i blocchi I/O
- Possono contenere altri blocchi come RAM o multipliers



Elementi logici

Sono composti da:

- LUT (a 4 ingressi): eseguono la logica combinatoria
- Flip-Flop: eseguono la logica sequenziale
- Multiplexer: collegano LUT e Flip-Flop



ALU

L'alu esegue:

- addizioni
- sottrazioni
- porte AND
- porte OR

  Input:

A, B = numeri

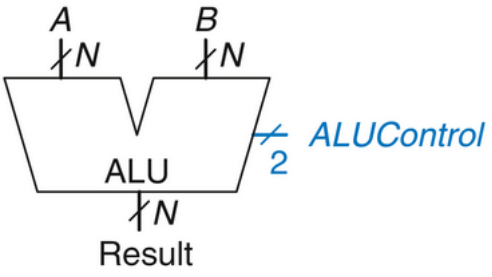
ALUControl = decide che operazione deve eseguire

output:

Result

Flag (N, Z, C, V): output che controllano determinate cose

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



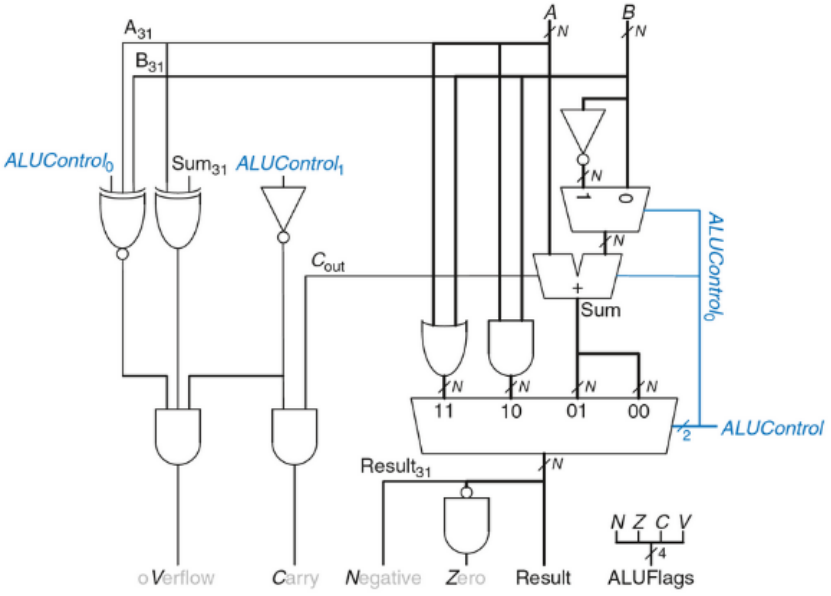
Flag	Description
N	*Result* is Negative
Z	*Result* is Zero
C	Adder produces Carry out
V	Adder oVerflowed

N = Y31

Z = $\overline{Y + Y}$

C = Cout * $\overline{\text{ALUControl}_{1}}$

V = $\overline{\text{ALUControl}_{1}}$ * ($\overline{A_{31} \oplus B_{31} \oplus \text{ALUControl}_{0}}$)*(A31 ⊕ Y31)



11-SystemVerilog

Simboli:

~ = NOT

& = AND

| = OR

[3:0] = indica che il dato è un array a x bit

11.1-Creare un modulo

```
module(definisce il modulo) nome(input logic a,b,c, output logic y)
    assign(return) y=~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule(chiude il modulo)
```

11.2-Richiamare un modulo:

```
nomemodulo nomeistanza(a,b,c,y)
```

Non si può usare una istanza più di una volta.