Videolezioni **Progettazione di Sistemi Digitali**

Docenti video:
Romeo Beccherelli - Istituto per la Microelettronica e Microsistemi (CNR-IMM)
Università Telematica Internazionale Uninettuno (Roma - Italia)

Lezione n. 1: Hardware digitale
- Hardware digitale
- Legge di Moore
- Chips standard
- Dispositivi logico-programmabili
- Chips custom
- Il processo di progettazione
- Schede di sviluppo

Lezione n. 2: Numeri Binari 1/2
- Rappresentazione dei numeri
- Numeri binari
- Conversione di base dei numeri
- Numeri binari con segno

Lezione n. 3: Numeri Binari 2/2
- Rappresentazioni Modulo-N
- Addizione e sottrazione di numeri binari con segno
- Codici binari
- Codici di Gray
- Codici ACSII

Lezione n. 4: Algebra booleana
- Contesto storico
- Gli operatori
- Teoremi
- Il diagramma di Venn

Lezione n. 5: Porte logiche e funzioni booleane 1/2
- Porte logiche
- Funzioni booleane

Lezione n. 6: Porte logiche e funzioni booleane 2/2
- Forme canoniche e forme standard
- Conversione fra forme canoniche
- Rappresentazione geometrica di funzioni booleane

Lezione n. 7: Semplificazione di funzioni booleane e mappe di Karnaugh
- Semplificazione di funzioni booleane
- Il metodo delle mappe di Karnaugh
- K-cubi e mappe di Karnaugh
- Implementazioni NAND e NOR

Lezione n. 8: Minimizzazione a livello di porte logiche
- Terminologia
- Minimizzazione algoritmica di funzioni booleane

Lezione n. 9: Circuiti Combinatori 1/3
- Circuiti combinatori
- Procedura d'analisi
- Procedura di progetto
- Funzione OR esclusivo

Lezione n. 10: Circuiti Combinatori 2/3
- Controllo di parità
- Sommatore e sottrattore binario
- Generatore di riporto
- Sommatore decimale

Lezione n. 11: Circuiti Combinatori 3/3
- Moltiplicatori
- Comparatori di grandezza
- Decodificatori
- Codificatori
- Multiplatori

Lezione n. 12: Circuti sequenziali sincroni 1/3
- Circuiti sequenziali
- latches
- D-type Flip-Flop

Lezione n. 13: Circuti sequenziali sincroni 2/3
- JK-type e T-type Flip-Flop
- Analisi di circuiti sequenziali sincroni

Lezione n. 14: Circuti sequenziali sincroni 3/3
- Assegnazione e riduzione degli stati
- Procedura di progetto

Lezione n. 15: Circuti sequenziali sincroni: registri e contatori
- Registri
- Registri a scorrimento
- Contatori "ripple"
- Contatori sincroni
- Altri contatori

Lezione n. 16: Memorie
- RAM
- RAM statiche (SRAM)
- DRAM
- SDRAM
- DDR

Lezione n. 17: Dispositivi logico-programmabili
- ROM
- Dispositivi logici combinatori
- Dispositivi logico-programmabili complessi
- Celle standard e matricidi porte
- FPGA

Lezione n. 18: Progetto al livello di trasferimento fra registri RTL
- Partizione di un sistema sequenziale in unità di controllo e unità di elaborazione
- Livello del trasferimento tra registri (RTL)
- Macchina algoritmica a stati finiti (ASM)

Lezione n. 19: Progetto al livello di trasferimento fra registri RTL - Esempi 1/2
- Struttura del bus
- Progetto dell'unità logica di controllo
- Progetto di un moltiplicatore binario sequenziale
- Progetto di un divisore fra interi

Lezione n. 20: Progetto al livello di trasferimento fra registri RTL - Esempi 2/2
- Progetto con decoder
- Progetto con multiplexer
- Progetto di un semplice processore

PROGETTAZIONE SISTEMI DIGITALI

Lezione 1 • HARDWARE DIGITALE

Hardware digitale. È un qualunque sistema idoneo a processare l'informazione codificata in modo ordinario ovvero ricevere, immagazzinare, manipolare e trasmettere le informazioni.

Legge di Moore. La densità dei transistori (a costo unitario minimo) raddoppia ogni due anni (1965). Tale legge è semplificata in: il numero dei transistori in un chip raddoppia ogni due anni.
A corollario della legge di Moore assistiamo che il costo degli impianti di fabbricazione (ponderie, foundry è il nome storico, anche se non si fonde niente).

Chip standard. Sono classificabili secondo diversi modi: per scala di integrazione, per famiglia logica, per caratteristiche.

- Scala di integrazione, ovvero il numero di transistori integrabili nel singolo chip. Abbiamo una bassa scala di integrazione (< 10, small scale Integration, SSI); media scala di integrazione (> 10 e $< 10^3$, Medium Scale Integration); grande scala di integrazione ($> 10^3$ e $< 10^4$,

Large Scale Integration, LSI); grandissima scala di integrazione ($> 10^4$ e $< 10^7$, Very Large Scale Integration, VLSI); estrema scala di integrazione ($> 10^7$, Ultra Large Scale Integration, ULSI).

- Famiglia logica. Logica transistore-transistore (Transistor-Transistor Logic, TTL); ad accoppiamento di emettitore (Emitter coupled Logic, ECL); metallo-ossido-semiconduttore complementare (Complementary Metal-Oxide-Semiconductor Logic, CMOS, HCMOS, HCTMOS, la cui differenze stanno nella tensione); nMOS ad alta densità (High-density nMOS, HnMOS, usata nei primissimi processori); bipolare MOS (Bipolar MOS Logic, BiCMOS).
- Caratteristiche. Di natura elettrica, corrispondente alla capacità di pilotare un certo numero di dispositivi (Fan-out), ma anche nel numero di ingressi disponibili (Fan-in) alla porta logica. Potenza dissipata e quindi calore dissipato. Ritardo di propagazione. Margine di rumore.

Dispositivi Logici Programmabili. Conosciuti come l'impianto di Passnedzone. Programmable Logical Devices (PLDs). Ci sono diverse famiglie/sigle: PLA, PAL, CPLDs, FPGA

Chips Custom. Realizzati espressamente per una certa funzionalità. Si intende un circuito integrato specifico per l'applicazione (Application Specific Integrated Circuits, ASIC). Tempi lunghi e alti costi.

12 Il Processo di Progettazione ... omissis.

Lezione 2 • I NUMERI BINARI I

Rappresentazione dei numeri. Per posizione, ovvero per ordine posizionale nei sistemi di numerazione

Numeri binari. Simboli 0 e 1.

Conversione di base dei numeri. È il resto di una divisione

Numeri binari con segno. Il bit più a sinistra, ovvero il bit più significativo, indica il segno, in dettaglio **0** per un numero positivo e **1** per un numero negativo. La rappresentazione della in ampiezza, con il segno + o −, non è formalizzabile.

compler. a 1 **Complemento a 1** è un altro modo per rappresentare numeri con segno, ovvero numeri positivi o negativi.
Nel complemento a 1 i valori negativi si ottengono complementando i valori positivi, bit a bit, cioè i bit, da 5_{10} a -5_{10} -> da 0101_2 a 1010_2 (bit di segno)
Analogamente per ottenere i valori positivi dei negativi

compl. a 2 **Complemento a 2** di un numero a n bit si ottiene o sottraendo il numero da 2^n oppure aggiungendo 1 al complemento a 1 del numero.

> complementare a 1 il numero, con segno, e aggiungere 1

$-12_{10} = 11110100$ -> inizio $12_{10} = 00001100$ modulo
(segno) ↓ compl. a 1 +1
11110011 +1
$-12_{10} = 11110100$

Altro esempio, da 5_{10} a -5_{10} con 8 bit

$5_{10} = 0000\ 0101_2$

zero

↓ complemento a 1 + 1

$1111\ 1010 + 1 =$

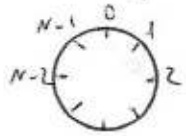
zero

$= -5_{10} = 1111\ 1011$

zero

Lezione 3 • I Numeri Binari II

Rappresentazione Modulo - N



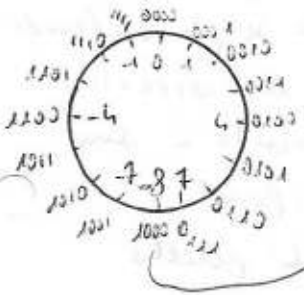
Rappresentazione dei numeri da 0 a N-1 su un cerchio, equipaziata. Esaurita la prima N-upla, la numerazione riparte da 0.

La somma in Modulo-N corrisponde a quella che si fa' fare con le dita, percorrendo in senso orario il cerchio.

Ad esempio $[1000_2 + 1110]_{mod\,16} = 0110_2$

e $[8_{10} + 14_{10}]_{mod\,16} = 7_{10}$

Reinterpretazione degli interi con segno, in valore da -8 a +7 nella rappresentazione in complemento a 2.



L'inizio da -8 permette una perfetta sequenzialità dei codici.

-8 in complemento a 2 è 1000, dato il numero binario 1000 in complemento a 2, per ottenere il numero decimale, che è negativo, si complementa a 1 e si somma 1, quindi in (ignorando il segno) abbiamo 0111, da cui è 8_{10} quindi -8_{10} in complemento a 2.

Addizione in complemento a due

- Somma rappresentazione n-bit
- Si ignora il riporto (carry-out)
- La somma è corretta se e solo se il risultato è nell'intervallo $[-2^{n-1}, +2^{n-1} - 1]$

Sottrazione in complemento a due

- Complementare il sottraendo a n-bit (quello sotto)
- Sommare il minuendo e il sottraendo complementato
- Ignorare il riporto
- La somma, ovvero il risultato, è corretta se è nell'intervallo $[-2^{n-1}, +2^{n-1}-1]$

L'overflow, ovvero quando il numero non può essere rappresentato con i bit disponibili.

- Per numeri senza segno occorre controllare il riporto, o carry out. Se c'è il riporto allora abbiamo un overflow, ovvero una condizione di errore.
- Per numeri con segno, rappresentati in complemento a due, l'overflow non si può verificare se gli addendi hanno segno opposto, cioè se i bit di segno sono diversi. Però, quando gli addendi, in complemento a due, hanno lo stesso segno, l'overflow si verifica se il segno della somma non è lo stesso di quello degli addendi.

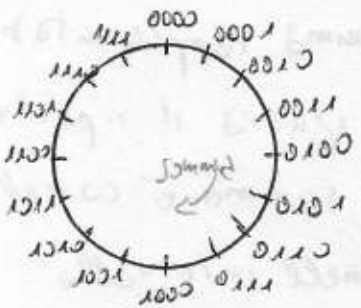
Estensione del segno. Per numeri positivi (0 davanti a sinistra) si aggiungono 0 a sinistra. Per numeri negativi (1 a sinistra), si aggiungono 1 a sinistra.

Codice binario — omissis.

Lez. 3 I NUMERI BINARI II

RAPPRESENTAZIONE MODULO-N

modulo-16 per
numeri in complemento
a 2



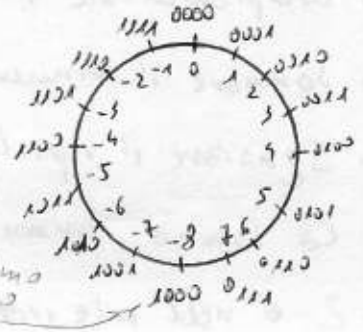
$[0111 + 0101]_{mod\,16} = 1100$ è una addizione come quelle fatte sommando con le dita

$[(7)_{10} + (5)_{10}]_{mod\,16} = (12)_{10}$

$[1001 + 1110]_{mod\,16} = 0111$

$[(9)_{10} + (14)_{10}]_{mod\,16} = (7)_{10}$

Reinterpretazione degli interi con segno da -8 a 7 nella rappresentazione in complemento a 2



è il primo numero negativo

ADDIZIONE E SOTTRAZIONE DI NUMERI BINARI CON SEGNO

9'40"

Addizione in complemento a due

- Somma rappresentazioni n-bit (ignorando il segno)
- Si ignora il riporto (carry-out)
- La somma è corretta se e solo se il risultato è nell'intervallo

$$[-2^{n-1}, +2^{n-1}-1]$$

nel caso della rappresentazione modulo 16 significa per -8 e +7

Esempi: $0001 + 0101 = 0110$; $+1 + 5 = 6$

$0100 + 1010 = 1110$; $+4 + (-6) = -2$

↳ questa è la rappresentazione in complemento a 2, per ottenere il valore ci sono due modi, il secondo è quello di fare il complemento a 1 e sommare 1, quindi $0110 \rightarrow$ compl. a 1 $\Rightarrow 1001 + 1 = 1010$

Il primo è $2^n - 0110$

Sottrazione in complemento a due

- Complementare il sottraendo a n-bit
- Sommare il minuendo e il sottraendo complementato
- Ignorare il riporto
- La somma, ovvero il risultato, è corretta se il numero 3.2 è nell'intervallo $[-2^{n-1}, +2^{n+1}-1]$

Esempi: $0110 + (+6) - 0010 = (-(+2)) \rightarrow +4$

$0110 + (+6) \rightarrow 1110 = -2 \rightarrow 0100 = +4$

complemento del sottraendo, $2^n - 1110$ oppure complemento a 1 e a 1 $+1$

$1101 + 1 = 1110$

$1110 + (-2) - 1011 = (-(-5)) \rightarrow +3$

$1110 + (-2) \rightarrow 0101 = +5 \rightarrow 0011$

(1) Riporto ignorato!

Come verificare l'insorgenza di overflow di numeri senza segno:

$0110 + (+6)$

$0100 = (+4)$

carry out $\rightarrow (1)0010$ $\quad (+10)$ ← overflow

Per numeri senza segno occorre controllare il riporto, o carry out. Se c'è => overflow, condizione di errore

---

Overflow di numeri con segno

I NUMERI SONO RAPPRESENTATI IN COMPLEMENTO A 2

L'overflow non si può verificare se gli addendi hanno segno opposto, cioè se i bit di segno sono diversi. IL CONTROLLO DI OVERFLOW NEL SEGNO PUÒ CONCLUDERE ESSERE FACILMENTE FATTO.

Esempi: $0110 + (+6)$

$0110 = (+4)$

$1010 \quad (+10)$

23 overflow

} 3 bit non sono sufficienti a rappresentare in complemento a 16 numeri più grandi di 7.

$$\begin{array}{r} 1110 \\ 1001 \\ \hline 10111 \end{array} \quad \begin{array}{r} -2 \\ -7 \\ \hline -9 \end{array}$$ OVERFLOW

overflow

1110 è il complemento a 2 di 4 bit.

questa è la rappresentazione di un numero positivo, in complemento a 2, su 4 bit, come gli addendi

Quando gli addendi (in complemento a 2) hanno lo stesso segno, l'overflow si verifica se il segno della somma non è lo stesso di quello degli addendi.

Estensione del segno

Se abbiamo un numero a 4 bit, 0101, esso può essere esteso ad 8 bit ponendo 0 davanti, ottenendo 0000 0101.

Se però il numero è negativo, allora il bit più significativo è 1, allora si estende la rappresentazione ripetendo 1, ad esempio dato 1110, esso si estende a 8 bit con 1111 1110; così facendo è come fare un giro del cerchio ed ogni bit è diverso in senso orario.

CODICI BINARI

Un codice binario a n bit è un gruppo di n bit che assume fino a 2^n distinte combinazioni di 1 e 0.

Non è necessario che tutte le combinazioni siano usate

Codifica binaria dei decimali (BCD)

Binary Coded Decimal

Possiamo codificare un numero decimale con 4 bit, non usando 6 combinazioni, ma avere una corrispondenza biunivoca tra il codice binario e il codice decimale rappresentato.

slide:

Decimal Digit	BCD 8421	2421	Excess 3	8, 4, -2, -1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111

Inutilizzati

Addizione BCD

$$\begin{array}{r} 0110 \\ +\ 0101 \\ \hline 1011 \end{array} \quad = \quad \begin{array}{r} 4 \\ +5 \\ \hline 9 \end{array} \quad ; \quad \begin{array}{r} 1000\ 8 \\ +\ 1001\ +9 \\ \hline 10001\ 17 \end{array}$$

↳ overflow

CODICE di GRAY

Numeri binari generati dall'hardware possono determinare ambiguità durante transizioni. È utile che un solo bit, come si alla volta per codici rappresentanti numeri consecutivi.

Codice Decimale	Codice binario $b_3\ b_2\ b_1\ b_0$	Codice di Gray $g_3\ g_2\ g_1\ g_0$
0	0 0 0 0	0 0 0 0
1	1	0 0 0 1
2	10	0 0 1 1
3	11	0 0 1 0
4	100	0 1 1 0
5	101	0 1 1 1
6	110	0 1 0 1
7	111	0 1 0 0
8	1000	1 1 0 0
9	1001	1 1 0 1
10	1010	1 1 1 1
11	1011	1 1 1 0
12	1100	1 0 1 0
13	1101	1 0 1 1
14	1110	1 0 0 1
15	1111	1 0 0 0

Codice ASCII
legge Aschi

OVERFLOW di numeri con segno per numeri in complemento a 2

- addendi di segno opposto: overflow non si verifica
- addendi dello stesso segno: overflow se il segno della somma non è lo stesso segno degli addendi

---

OVERFLOW di numeri senza segno

- occorre controllare il riporto, o carry out: se c'è il riporto allora si è verificato un overflow, che è una condizione di errore

---

OVERFLOW → il numero non può essere rappresentato con i bit disponibili

(ad esempio il numero 10 decimale, con 3 bit, in base con 3 bit, senza segno, so ottiene $a\ 7_{10} = 111_2$)

Lez. 4 Algebra Booleana

CONTESTO STORICO ...

L'algebra booleana è:

→ un sistema matematico deduttivo

→ un insieme di assiomi non dimostrati (o postulati)

→ un insieme di elementi : 0 , 1 algebra dei sistemi digitali commutativi

→ un insieme di operatori

GLI OPERATORI (OR, AND, NOT)

Disgiunzione

X OR Y vel, o congiunzione (aut, o esclusiva)

$X + Y$ (anche $X \vee Y$)

segno usato

$x + y = 0$ se $x = y = 0$ o logico

$x + y = 1$ altrimenti 1 logico

1 è "portante"

Congiunzione

X AND Y

$X \cdot Y$ (anche $X \wedge Y$)

segno usato

$X \cdot Y = 1$ se $X = Y = 1$

$X \cdot Y = 0$ altrimenti

0 è "portante"

Negazione NOT x

detto $\bar{x}$ (anche "y è l'inverso di x")

$x' = 0$ se $x = 1$

$x' = 1$ se $x = 0$

Assiomi dell'ALGEBRA BOOLEANA

- chiusura rispetto al + } il risultato del + e del · è appartenente all'insieme dei valori
- chiusura rispetto al · }
- 0 è l'elemento identità per + (OR): $x + 0 = x$
- 1 è l'elemento identità per · (AND): $x \cdot 1 = x$
- Commutatività + : $x + y = y + x$
  · : $x \cdot y = y \cdot x$
- · distributivo rispetto a + : $x \cdot (y + z) = x \cdot y + x \cdot z$
- + distributivo rispetto a · : $x + (y \cdot z) = (x + y) \cdot (x + z)$

Dualità di tutti gli assiomi

- Esistono almeno due elementi per cui $x \neq y$, lo 0 e l'1 nell'algebra di Boole
- Complemento : $x + x' = 1$, $x \cdot x' = 0$

Algebra a due valori

var. indip.			
x	y	x+y	x·y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

valori logici

non sono codici, ma valori logici.

• distributiva rispetto +

x	y	z	y+z	x·(y+z)	(x·y)	(x·z)	(x·y)+(x·z)
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

CVD

+ distributiva rispetto ·

x	y	z	y·z	x+(y·z)	x+y	x+z	(x+y)·(x+z)
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	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

CVD

ALCUNI TEOREMI dell'ALGEBRA BOOLEANA

→ $x + x = x$ (OR) $\qquad x \cdot x = x$ (AND)

→ $x + 1 = x$ $\qquad x \cdot 0 = 0$

→ Associatività

$x + (y + z) = (x + y) + z$

$x \cdot (y \cdot z) = (x \cdot y) \cdot z$

→ Teoremi di De Morgan

$(x + y)' = x' \cdot y'$

$(x \cdot y)' = x' + y'$

→ Assorbimento

$x + x \cdot y = x$ $\quad x(1 + y) = x$ (1, perché è l'operazione)

$x \cdot (x + y) = x$ $\quad \Rightarrow x + xy = x$, perché $= x(x+y)$

→ Combinazione

$x \cdot y + x \cdot y' = x$ $\quad x(y + y')$

$(x + y) \cdot (x + y') = x$

↳ $x + xy' + xy + y \cdot y'$ (x, 0)

y	y'
0	1
1	0

→ Consenso

$x \cdot y + y \cdot z + x' z = x \cdot y + x' \cdot z$

Forma duale $(x + y) \cdot (y + z) \cdot (x' + z) = (x + y) \cdot (x' + z)$

→ OR Esclusivo (EXOR, $\oplus$) "Sui bottoni, o l'uno o l'altro"

$x \cdot y' + x' \cdot y = 1$ se $x \neq 1$, $y \neq 1$

→ Equivalenza (X-NOR, NEXOR)

$x \cdot y + x' \cdot y'$

Prova di uno dei teoremi di De Morgan

x	y	x·y	(x+y)'	x'	y'	x'+y'
0	0	0	1	1	1	1
0	1	0	1	1	1	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

C.V.D.

IL DIAGRAMMA DI VENN

→ sono molto nella matematica per illustrare le operazioni nell'algebra degli insiemi.
Le operazioni che si utilizzano sono in sostanza l'unione e l'intersezione di elementi all'interno di un insieme.

→ Gli elementi nei diagrammi di Venn sono rappresentati dall'area racchiusa dal contorno.

→ Nell'algebra booleana ci sono solo due elementi, lo 0 e l'1.

Costante 1

Costante 0

→ Area nel contorno = 1

→ Area fuori contorno = 0

Variabile X → X

x' →

Intersezione

$x \cdot y$
AND

Unione

$x + y$
OR

Ad esempio:

$x + y'$

$x \cdot y + z$

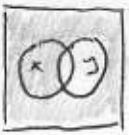
(l'AND ha la precedenza sull'OR)

Il teorema di De Morgan dimostrato con i diagrammi di Venn

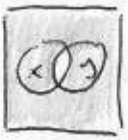
$(x \cdot y)' = x' + y'$

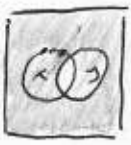
$x \cdot y$

x'



y'





$(x \cdot y)'$ $x' + y'$ C.V.D.

Dimostrare i teoremi in modo esauriente e con i diagrammi di Venn, quelli del consumo ecc. n.b.: potrebbero essere da 3 minuti di esame.

TEOREMA del CONSENSO

$$xy + x'z + yz = xy + x'z$$

x assente e complementata

In una espressione del tipo $xy + x'z + yz$ si dimostra che il termine yz è ridondante e può essere eliminato semplificando l'espressione originaria in $xy + x'z$

(ridondante)

Infatti se $yz = 1$ si deve avere $y = 1$ e $z = 1$ e perciò uno qualsiasi dei due termini xy e $x'z$ deve valere 1, sia che $x' = 1$ oppure $x = 1$.

Teorema del consenso $xy + x'z + yz = xy + x'z$

Dimostrazione

$$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') = \\ &= xy + x'z + xyz + x'yz = \\ &= xy + xyz + x'z + x'yz = \\ &= xy(1 + z) + x'z(1 + y) = \\ &= xy(1) + x'z(1) = \\ &= xy + x'z \end{aligned}$$

c. v. d.

Il termine ridondante yz è detto di consenso e rappresenta il consenso dei termini xy e $x'z$. In generale, dati due termini in cui una variabile compare in un termine e il complemento della stessa variabile compare nell'altro, il termine consenso è formato dal prodotto dei due termini in questione eliminando da esso la variabile e il suo complemento.

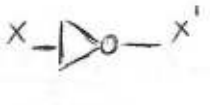
Ad esempio il consenso di xyz e $x'z$ è $(xz)(x'z) = xx'z$.

TEOREMI DI BASE e PROPRIETÀ dell'ALGEBRA BOOLEANA

Pag 43 Nano	postulato 2	$x+0=x$	$x \cdot 1 = x$
"	postulato 5	$x+x'=1$	$x \cdot x' = 0$
"	Teorema 1	$x+x=x$	$x \cdot x = 0$
"	Teorema 2	$x+1=1$	$x \cdot 0 = 0$
"	Teorema 3, involuzione	$(x')'=x$	✱
"	postulato 3, commutatività	$x+y=y+x$	$x\,y = y\,x$
"	Teorema 4, associatività	$x+(y+z)=(x+y)+z$	$x(y\,z)=(x\,y)\,z$
"	Postulato 4, distributività	$x(y+z)=xy+xz$	$x+y\,z=(x+y)(x+z)$
"	Teorema 5, De Morgan	$(x+y)'=x'y'$	$(xy)'=x'+y'$
"	Teorema 6, Assorbimento	$x+xy=x$	$x(x+y)=x$
	Teorema del consenso	$xy+x'z+yz=xy+x'z$	

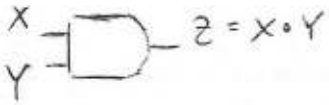
Lezione 5 • PORTE LOGICHE e FUNZIONI BOOLEANE

PORTE LOGICHE. Implementano gli operatori logici



NOT

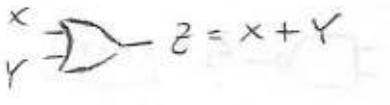
x	x'
0	1
1	0



AND

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

l'ingresso 0 è portante

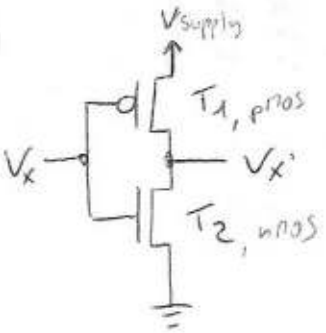


OR (disgiunzione)

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

l'ingresso 1 è portante

Porta NOT in implementazione CMOS, ovvero l'invertitore CMOS



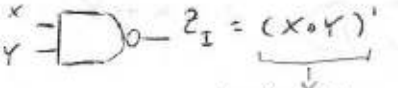
x	V_x	T_1	T_2	$V_{x'}$	x'
0	low	on	off	high	1
1	high	off	on	low	0

↓
Logica positiva

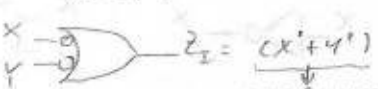
Invertitore CMOS

PORTA AND in implementazione CMOS

Per realizzare la porta AND si realizza la porta NAND in implementazione CMOS, alla quale si mette in cascata un invertitore CMOS.



n.b.:

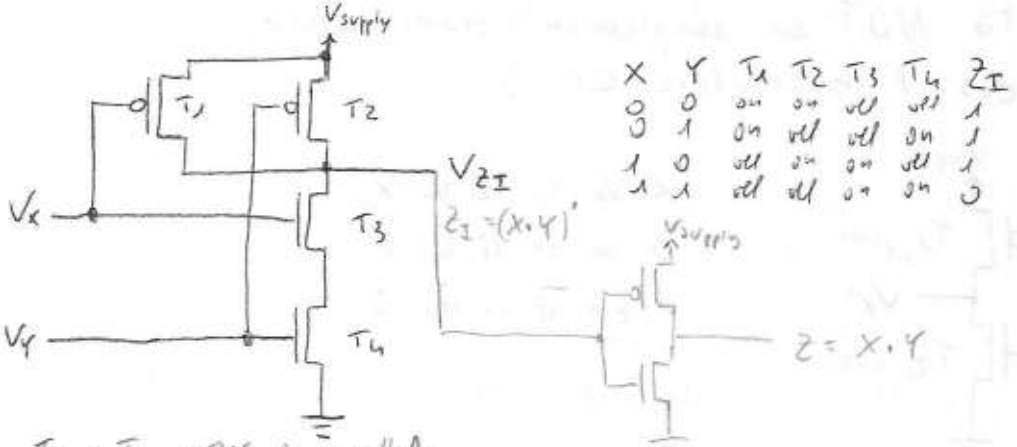


NAND

X	Y	Z	Z_I
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

$\Rightarrow I \Rightarrow$ invertita

Porta NAND in implementazione CMOS

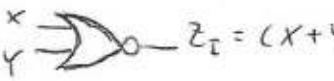


X	Y	T_1	T_2	T_3	T_4	Z_I
0	0	on	on	off	off	1
0	1	on	off	off	on	1
1	0	off	on	on	off	1
1	1	off	off	on	on	0

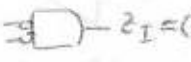
Ponendo un invertitore CMOS all'uscita V_{Z_I}, si ottiene una porta AND

PORTA OR in implementazione CMOS

Per realizzare la porta OR si realizza la porta NOR in implementazione CMOS, alla quale si mette in cascata un invertitore CMOS.



n.b.:

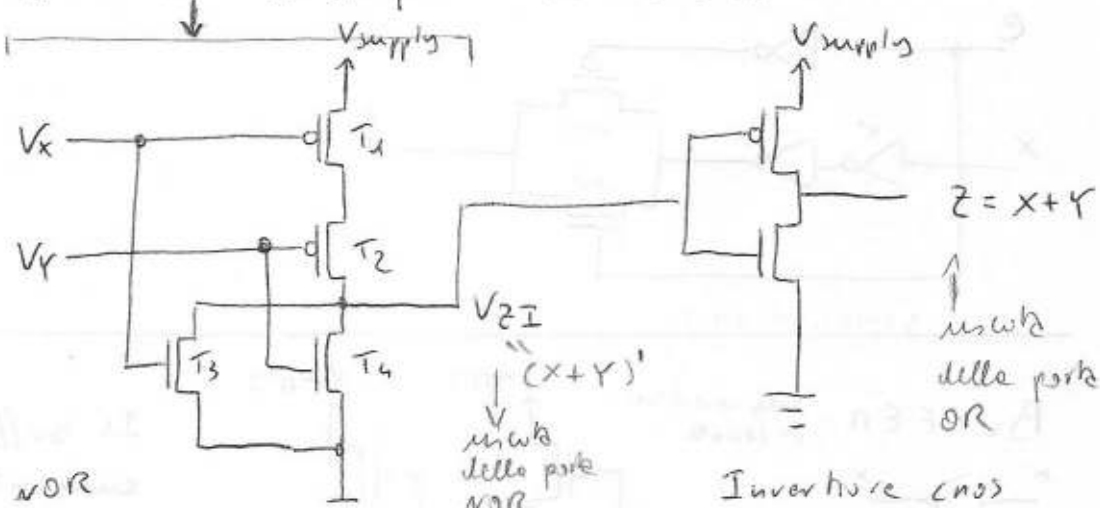


configurazione Invert-AND

NOR

X	Y	Z	Z_I
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Porta NOR in implementazione CMOS



NOR

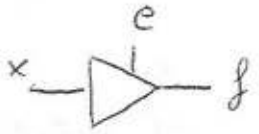
X	Y	T_1	T_2	T_3	T_4	Z_I
0	0	on	on	off	off	1
0	1	on	off	off	on	0
1	0	off	on	on	off	0
1	1	off	off	on	on	0

Invertitore CMOS

Ponendo un invertitore CMOS che ha per ingresso V_{ZI} si ottiene una porta OR

Buffer TRI-STATE (è una porta)

Il tri-state è un terzo stato, ad alta impedenza, che permette di isolare quello che sta a valle.

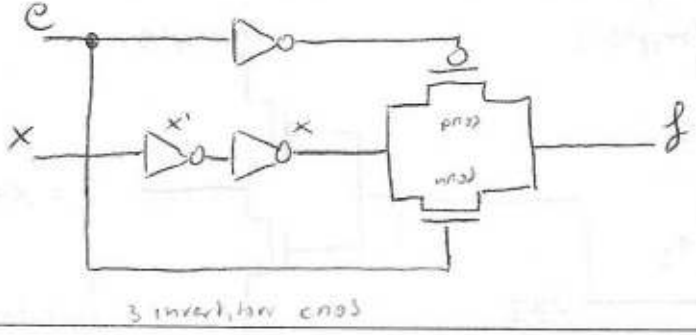


porta Buffer Tri-state

e	x	f
0	0	z
0	1	z
1	0	0
1	1	1

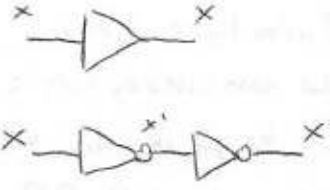
z è l'uscita ad alta impedenza, l'uscita f è isolata dall'ingresso e non se cosa c'è all'ingresso
e = 0 => l'abilitazione è bassa

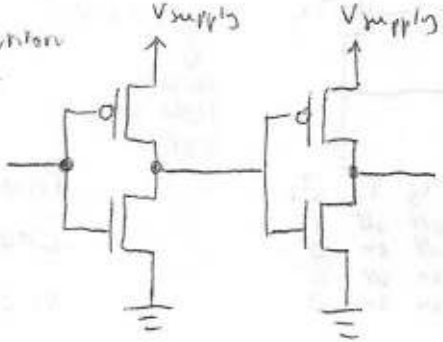
Buffer Tri-state in implementazione CMOS



3 inverting CMOS

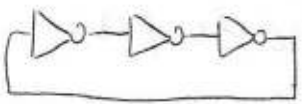
BUFFER = due inverters in cascata



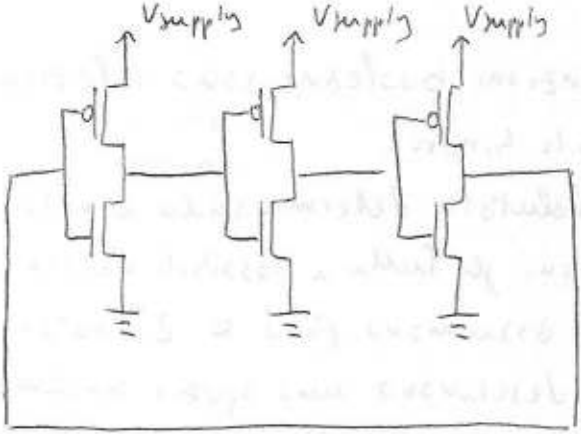


Il buffer aumenta la quantità di corrente pilotata dallo stadio

OSCILLATORE ad ANELLO



3 stadi NOT



Dispositivo utile, usato per misurare la velocità di propagazione delle porte. Chiudendo il circuito, esso si mette a oscillare ad una frequenza caratteristica che è data da una relazione legata alla somma dei tempi di propagazione. Può inoltre essere usato come elemento di memoria facendo oscillare uno stato logico attraverso tale circuito e, se prelevato con un opportuno sincronismo, è possibile prelevarlo anche in uscita.

FUNZIONI BOOLEANE

- Le funzioni booleane sono relazioni logiche fra variabili binarie
- sono valutate determinando i valori binari delle espressioni per tutti i possibili valori delle variabili
- n bit assumono fino a 2^n valori, da 0 a $2^n - 1$
- n bit descrivono uno spazio n-dimensionale B^n con $B = \{0, 1\}$
- ogni elemento dello spazio è un vettore $v \in B^n$
- In generale, ogni valore di una funzione booleana può essere:

  0 1 don't care (x) o tristate (Z)

  (complementare di un'uscita che non si verificano mai)

Le funzioni booleane possono essere espresse come sottospazio di B^n (vettori); come espressioni algebriche; come tabelle della verità; come rappresentazioni schematiche; con un linguaggio ad alto livello descrittivo dell'hardware (Hardware Description Language, HDL).

Funzioni completamente specificate sono definite se l'insieme don't care è vuoto e sono descritte dall'on-set (i valori della funzione sono 1) oppure dall'off-set (i valori della funzione sono 0).

LEZIONE 6 • PORTE LOGICHE e FUNZIONI BOOLEANE

FORME CANONICHE e FORME STANDARD

Forma canonica. Una funzione è in forma canonica se è espressa esclusivamente come somma di minterms (somma di prodotti) o come prodotto di Maxterms (Prodotto di somme)

letterale — minterm — Maxterm

Ad es. $f_1 = x'y'z + xy'z' + xyz = m_0 + m_4 + m_7$

$f_2 = (x + y + z') \cdot (x' + y + z) \cdot (x + y + z) = M_1 + M_4 + M_7$

Conversione fra forme canoniche

La tabella della verità di una funzione in forma canonica, è unica. La funzione è espressa come somma di prodotti oppure come prodotto di somme.

Abbiamo dunque che, data una funzione, vale sempre:

$f = \Sigma$ (minterms in cui $f = 1$) Σ = sommatoria di minterms

$f = \Pi$ (Maxterms in cui $f = 0$) Π = produttoria di Maxterms

ad esempio $f = \Sigma(1, 4, 7) \Rightarrow f = m_1 + m_2 + m_7$

e $f = \Pi(0, 2, 3, 5, 6) \Rightarrow f = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$

Quindi abbiamo che:

- Il complemento di una funzione espressa come somma di minterms è uguale alla somma dei minterms mancanti nella funzione originale.
- per convertire una forma canonica in un'altra, si scambiano i simboli Σ e Π e si elencano i numeri mancanti nella forma originale.

Forma standard. La funzione è espressa come somma di prodotti standard (SoP) oppure come prodotti di somme standard (PoS).

RAPPRESENTAZIONE GEOMETRICA delle FUNZIONI BOOLEANE

n variabili; poiché ogni variabile può assumere due soli valori.

per $n=1$ => spazio B^1, una retta

per $n=2$ => spazio B^2, un piano

per $n=3$ => spazio B^3

per $n=4$ => spazio B^4

K-CUBI

- Un K-cubo è un sottospazio o un sottoinsieme K-dimensionale di B^n con 2^k valori, ottenuto

fissando allo stesso valore $n-k$ variabili

- Non tutti gli spazi k-dimensionali di B^n sono k-cubi.
- Ci sono $\binom{n}{n-k} = \frac{n!}{(n-k)!\,k!}$ modi in cui n-k bit possono essere uguali
- Ci sono 2^{n-k} combinazioni
- n-k bit possono essere combinati
- $\frac{n!\,2^{n-k}}{(n-k)!\,k!}$ differenze k-cubi

Dai cubi ai termini prodotto. Occorre fare le sostituzioni

0 -> variabile complementata

1 -> variabile non complementata

Ignorare condizioni don't care

Dai cubi ai termini somma. Occorre fare le sostituzioni

0 -> variabile non complementata

1 -> variabile complementata

Ignorare condizioni don't care

LEZIONE 7 - SEMPLIFICAZIONE di FUNZIONI BOOLEANE e MAPPE di KARNAUGH

SEMPLIFICAZIONE di FUNZIONI BOOLEANE

Occorre applicare i postulati e i teoremi dell'algebra booleana:
(dimostrabili con la tabella delle verità)

- Proprietà distributiva

  $x \cdot (Y+Z) = x \cdot Y + x \cdot Z$ e $x + (Y \cdot Z) = (x+Y)(x+Z)$

- Teorema di De Morgan

  $(x+Y)' = x' \cdot Y'$ e $(x \cdot Y) = x' + Y'$

- Teorema dell'assorbimento

  $x + (x \cdot y) = x$ e $x \cdot (x+y) = x$

- Teorema della combinazione

  $x \cdot Y + x \cdot Y' = x$ e $(x+Y) \cdot (x+Y') = x$

Per la semplificazione si possono usare approcci euristici, rappresentativi. La soluzione ottima può non essere unica, anche se la soluzione ottima è impossibile a mano se circuiti troppo grandi.

IL METODO delle MAPPE di KARNAUGH

Permette un approccio più sistematico, la riduzione risulta e mano, al fine della semplificazione.

- Nella mappa ogni posizione della cella rappresenta una combinazione di condizioni di ingresso
- Le posizioni delle celle sono indirizzate con codice di Gray

- I contenuti delle celle rappresentano il valore della funzione
- La griglia è toroidale.

MAPPE di KARNAUGH e K-CUBI

ons(f) = on sets, le $f=1$
offs(f) = off sets, le $f=0$
dcs(f) = don't cares, le f è indefinita

IMPLEMENTAZIONE con NAND e NOR
in reminiscenza cnos delle funzioni booleane
vd. 7.6 ÷ 7.8 sugli appunti

Lezione 8

MINIMIZZAZIONE a LIVELLO di PORTE LOGICHE dei circuiti digitali

TERMINOLOGIA

• Un letterale è una istanza di una variabile in forma asserita o complementata: x o x'.

• Minterm, o prodotto fondamentale, è un prodotto di letterali: xy'z.

• Maxterm, o somma fondamentale, è una somma di letterali: x' + y + z.

• Ogni minterm e maxterm è uno 0-cubo, il verice di un n-cubo.

• **Primo implicante** è un termine prodotto ottenuto combinando il massimo numero possibile di celle adiacenti in una mappa di Karnaugh, ovvero il termine più semplificato possibile di un termine prodotto.

• Un termine primo implicante non è contenuto nessun'altra implicante.

• Primo implicante è un implicante che non può essere coperto da nessun altro implicante più generale costituito da un minor numero di letterali.

• **Primo implicante essenziale** è un primo implicante che copre una uscita della funzione che non può essere coperta da nessun'altra combinazione di implicanti.

• **On-set**, ons(f) di una funzione f è insieme delle valutazioni di f per cui f = 1.

• **Off-set**, ofs(f) di una funzione f è l'insieme delle valutazioni di f per cui f = 0.

• **Don't care set**, dcs(f) di una funzione f è l'insieme delle valutazioni per cui la funzione è indefinita.

• **Copertura** di una funzione è una collezione di implicanti che comprende tutti i vertici dell'ons(f), e non contiene i vertici dell'ofs(f).

Sia C una copertura, dunque: ons(f) $\subseteq$ C $\subseteq$ ons(f) U dcs(f)
Possono coesistere diverse coperture di una funzione.

• Una copertura definisce una specifica implementazione di f.

• Una copertura è non ridondante se la copertura è persa rimuovendo un implicante.

• Una copertura che comprende solo primi implicanti ha il minor costo di implementazione.

• Costo di una copertura è dato dal numero di porte più il numero di ingressi a tutte le porte.

Vincoli di progetto

• Fan-in (tipicamente ≤ 4)

• Fan-out (tipicamente ≤ 4). Dettato dalla quantità di corrente che lo stadio può pilotare a valle.

• Numero di livelli.

• Massimo tempo di propagazione ammesso.

• Dissipazione di potenza massima ammessa.

MINIMIZZAZIONE ALGORITMICA di FUNZIONI BOOLEANE

Finora l'approccio usato è stato di tipo terroristico prima, con l'uso delle mappe di Karnaugh poi.

Metodi migliori sono quelle di tipo algoritmi con, basati su regole ben precise, implementabili su un calcolatore che le

possa applicare in modo sistematico.

Procedura di minimizzazione

- Generare tutti i primi implicanti di f.
- Trovare l'insieme di primi implicanti essenziali (se esistenti).
- Insieme di primi implicanti essenziali è on(s) allora la copertura è stata trovata.
- Altrimenti, aggiungere primi implicanti non essenziali fino ad ottenere la copertura.
- Scegliere opportunamente questi primi implicanti per minimizzare il costo.
- Usare un approccio euristico

    - Aggiungere un primo implicante arbitrariamente a caso, poi altri primi implicanti fino a copertura.
    - Ripetere con diversa prima scelta.
    - Confrontare le copertura in termini di costo.
    - Scegliere quella con il minor costo parentesi e che soddisfi altri vincoli, ad esempio il tempo di propagazione oppure la dissipazione di potenza).

... esempi, vd. appunti.

LEZIONE 9

CIRCUITI COMBINATORI I

Prof. Romeo Beccherelli
38'23"

Circuiti Combinatori I
- Circuiti combinatori
- Procedura d'analisi
- Procedura di progetto
- La funzione OR esclusivo

Circuiti Combinatori II
- Controllo di parità
- Sommatore-sottrattore binario
- Generatore di riporto
- Sommatore decimale

Circuiti Combinatori III
- Moltiplicatori
- Comparatori di grandezza
- Decodificatori
- Codificatori
- Multiplatori

CIRCUITI COMBINATORI

- Costituiti da un'interconnessione di porte logiche
- Specificati logicamente con una funzione booleana
- Uscite ad ogni istante determinate solo dalla combinazione attuale degli ingressi

PROCEDURA DI ANALISI

- È fornito uno schematico
- Verificate sia combinatorio (nessun percorso di retroazione)
- derivate una espressione algebrica

oppure

- derivate la tabella della verità

Derivate una funzione Booleana

1. Etichettate con dei simboli tutte le porte d'uscita che sono direttamente funzione degli ingressi
2. Determinate la funzione Booleana per ogni porta d'uscita
3. Etichettate tutte le porte funzione degli ingressi e delle porte precedentemente etichettate con altri simboli
4. Determinate le funzioni Booleane realizzate da queste porte
5. Ripetete il processo ai passi 3 e 4 fino ad ottenere le uscite di tutto il circuito
6. Mediante sostituzioni ripetute delle funzioni così definite, ottenete le funzioni Booleane d'uscita in termini delle variabili di ingresso

Derivate la tabella della verità

1. Determinate il numero di ingressi nel circuito
2. Per gli n ingressi, formate le 2^n possibili combinazioni e enumerate i numberi binari fra 0 e $(2^n - 1)$ in una tabella
3. Etichettate le uscite di porte selezionate con dei simboli
4. Riempite la tabella della verità per le uscite che sono funzione delle sole variabili di ingresso
5. Ottenete la tabella della verità per le uscite di quelle porte che sono funzione dei valori precedentemente definiti fino a che tutte le colonne delle uscite siano determinate

Spunti di riflessione: Come entrano in gioco i tempi in un circuito combinatorio?

PROCEDURA DI PROGETTO

1.Dalle specifiche di progetto per il circuito, determinate il numero richiesto di ingressi e uscite e assegnate un simbolo a ciascuno

2.Derivate la tabella della verità che definisce la richiesta relazione fra ingressi ed uscite

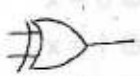
3.Ricavate la funzione Booleana semplificata/ottimizzata per ciascuna uscita come funzione delle variabili di ingresso

4.Disegnate il diagramma schematico e verificate la correttezza del progetto (manualmente o mediante strumenti di simulazione)

LA FUNZIONE OR ESCLUSIVO (XOR)

La sua definizione, simbolo $\oplus$, è:

$$X \oplus Y = X \cdot Y' + X' \cdot Y$$



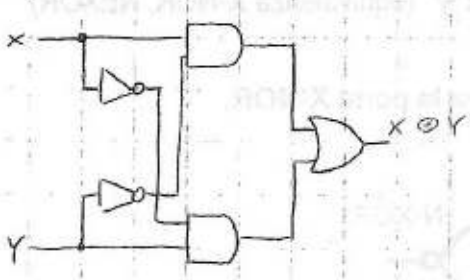
simbolo operatore or esclusivo, XOR

Tabella della verità

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

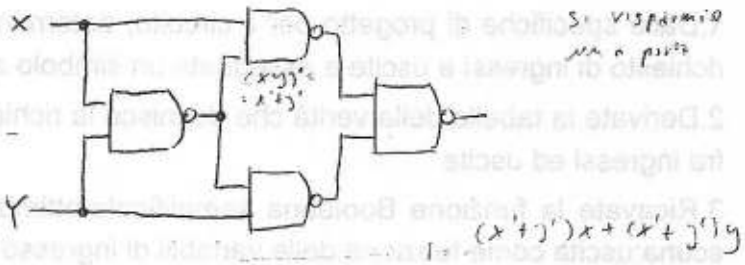
Se le due variabili di ingresso sono diverse allora XOR è 1, se sono uguali allora XOR è 0.

Implementazione



che può essere trasformata sostituendo le due porte AND con due porte NAND e la porta OR con una porta NAND, che è di più facile implementazione (in tecnologia CMOS).

Inoltre possiamo anche usare una porta NAND per negare x e y.



Caratteristiche della funzione XOR
(dimostrate con la tabella della verità)

$x \oplus 0 = x$

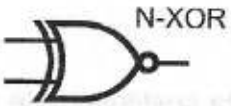
$x \oplus 1 = x'$

$x \oplus x = 0$

$x \oplus x' = 1$

$x \oplus y' = x' \oplus y = (x \oplus y)' = x \cdot y + x' \cdot y'$ (equivalenza X-NOR, NEXOR)

Dall'ultima caratteristica, si ricava la porta X-NOR,
il simbolo della porta X-NOR è:



Proprietà associativa di XOR:

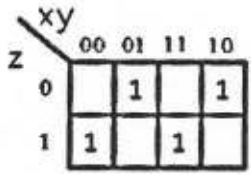
$x \oplus y \oplus z = (x \oplus y) \oplus z = x \oplus (y \oplus z) = (xy'+x'y)z'+(x \bullet y+x' \bullet y')'z =$

$= \sum (1, 2, 4, 7)$

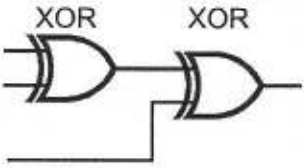
In $x \oplus y \oplus z$ possiamo osservare che vale 1 solo quando uno solo fra x, y, z ha valore 1, oppure quando valgono 1 tutti e tre. Vale 0 quando c'è uno zero.

Tabella della verità:

x	y	z	$x \oplus y \oplus z$
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



➡ **funzione dispari**

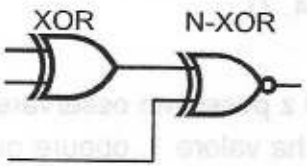


Implementazione di funzione dispari con input di 3 variabili

Funzione dispari con input di 3 variabili, si osserva che la funzione XOR contiene $2^n/2$ minterms i cui valori numerici binari contengono un numero dispari di 1.

La funzione N-XOR (o NEXOR), che è la funzione complementata della XOR: è una funzione pari, vale 1 quando in x, y, z c'è un numero pari di 1, caso notevole quello per cui la funzione vale 1 per x = y = z = 0.

A lato implementazione della funzione pari per un input a 3 variabili.



Lez. 9 Prof. Marco Bertacco

CIRCUITI COMBINATORI I

- Circuiti combinatori (combinazione di valori)
- Procedura di analisi
- Procedura di progetto
- La funzione OR esclusivo

I

- Controllo di parità
- Sommatore-sottrattore binario
- Generatore di parità
- Sommatore decimale

II

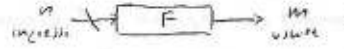
- Moltiplicatori
- Comparatori di grandezza
- Decodificatori
- Codificatori
- Multiplatori e demultiplexer

III

CIRCUITI COMBINATORI

→ Sono costituiti da una interconnessione di porte logiche

→ Sono specificati logicamente con una funzione booleana



→ Le uscite ad ogni istante sono determinate solo dalla combinazione attuale degli n ingressi.

PROCEDURA di ANALISI

→ E' fornito, in genere, uno schematico

→ Verificare che sia combinatorio (nessun percorso di retroazione)

→ Sarà derivata una espressione algebrica (booleana)

→ Sarà derivata la tabella oppure della verità

Derivazione della funzione booleana

1. Etichettare con dei simboli tutte le porte da usare che sono direttamente funzione degli ingressi
2. Determinare la funzione booleana per ogni porta da usare
3. Etichettare tutte le porte che sono funzione degli ingressi e delle porte precedentemente etichettate con altri simboli
4. Determinare la funzione booleana realizzata da queste porte
5. Ripetere il processo ai passi 3 e 4 fino ad ottenere le uscite di tutto il circuito
6. Mediante sostituzioni ripetute delle funzioni così definite, ottenere la funzione booleana d'uscita in termini delle variabili di ingresso

vd esempio slide 14 e 15

Derivazione della tabella della verità

1. Determinare il numero di ingressi nel circuito
2. Per gli n ingressi, formare le 2^n possibili combinazioni e enumerare i numeri binari fra 0 e $(2^n - 1)$ in una tabella
3. Etichettare le uscite di porte selezionate con dei simboli
4. Riempire la tabella della verità per le uscite che sono funzione delle sole variabili di ingresso.
5. Ottenere la tabella della verità per le uscite di quelle porte che sono funzione dei valori precedentemente definiti fino a che tutte le colonne delle uscite siano determinate.

vd slide 16

PROCEDURA DI PROGETTO

1. Dalle specifiche di progetto per il circuito, determinare il numero richiesto di ingressi e uscite e assegnare un simbolo a ciascuno
2. Derivare la tabella della verità che definisce la richiesta relazione fra ingressi ed uscite (in modo dinamico)
3. Ricavare la funzione booleana semplificata/ottimizzata per ciascuna uscita come funzione delle variabili di ingresso
4. Disegnare il diagramma schematico e verificare la correttezza del progetto (manualmente o mediante strumenti di simulazione).

Vd esempi slide 21÷25 [illegible] e Code converter sempl. a pag. 150 del libro-circuiti, e [illegible] nelle pagine a fianco.

LA FUNZIONE OR ESCLUSIVO, XOR

(vd. pag 103 libro-circuiti)

La sua definizione, simbolo $\oplus$, è:

$$X \oplus Y = X \cdot Y' + X' \cdot Y$$

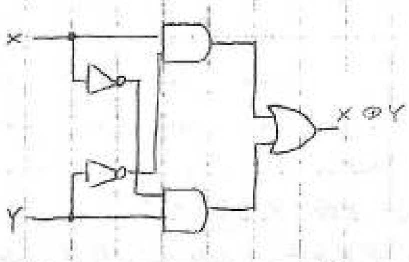
simbolo operatore or esclusivo, XOR

Tabella della verità

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

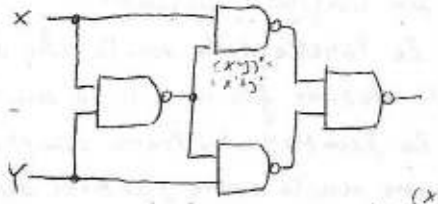
} se le due variabili di ingresso sono diverse allora XOR è 1, se sono uguali allora XOR è 0.

Implementazione



che può essere trasformata sostituendo le due porte AND

con due porte NAND e la porta OR con una porta NAND, che è di più facile realizzazione fisica. (Implementazione OR)
Inoltre possiamo anche usare una porta NAND per negare X e Y.



si risparmia una porta

$= (x'+y')x + (x'+y')y =$

$= xy' + x'y$

Caratteristiche della funzione XOR
(dimostrate con la tabella della verità)

$X \oplus 0 = X$

$X \oplus 1 = X'$

$X \oplus X = 0$

$X \oplus X' = 1$

$X \oplus Y' = X' \oplus Y = (X \oplus Y)' = X \cdot Y + X' \cdot Y'$

→ Equivalenza (X-NOR, NEXOR), simbolo

NEXOR

Proprietà associativa di XOR:

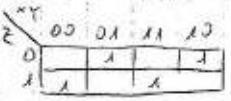
$X \oplus Y \oplus Z = (X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) =$

$= (xy' + x'y)z + (x \cdot y + x' \cdot y')'z =$ vd slide su e la tab. della verità

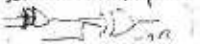
$= \sum(1, 2, 4, 7)$

In $X \oplus Y \oplus Z$ possiamo osservare che vale 1 solo quando uno solo fra X, Y, Z ha valore 1, oppure quando valgono 1 tutte e tre. Vale 0 quando c'è zero o due 1.
Abbiamo dunque, esprimendo come mintermi, lo "XOR":

$$X \oplus Y \oplus Z = (X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = \sum(1, 2, 4, 7)$$

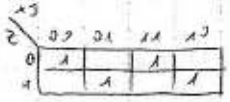


→ funzione dispari



XOR contiene $2^n/2$ mintermi i cui valori numerici binari contengono un numero dispari di 1

Duale mente, la funzione complementare è la NEXOR:



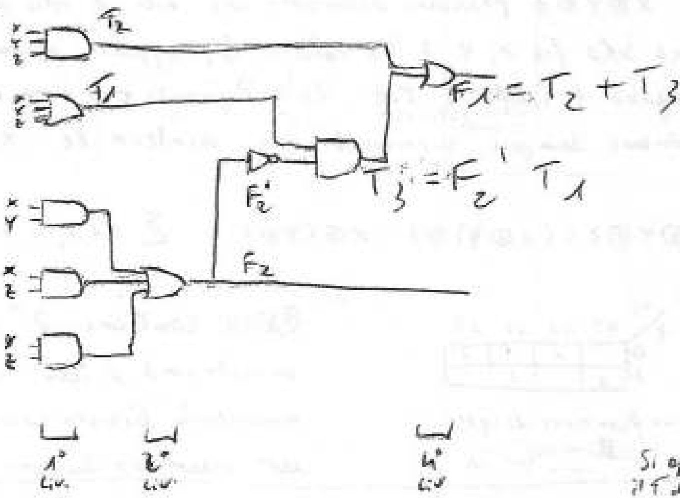
→ funzione pari, è una funzione che è 1 quando in input un numero pari di 1. (cioè quando in X, Y, Z ci sono un numero pari di 1)

Tabella della verità della funzione XOR: $X \oplus Y \oplus Z$

X	Y	Z	$X \oplus Y \oplus Z$
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

$X \oplus Y \oplus Z$ vale 1 quando uno solo fra X, Y, Z vale 1 oppure quando tutte e tre valgono 1.

Slide 12



Si applica 2 volte il T. di De Morgan o prescrizioni

Slide 13 $F_1 = F_2 + T_3 = xyz + F_2' T_1 = xyz + (xy + xz + yz)' \cdot (x+y+z)$

$= xyz + ((x'y') + (x'+z') + (y'+z')) \cdot (x+z+z) = \ldots$

$= xyz + x'y'z + xy'z' + x'yz'$

Slide 16 La tabella delle verità (unica!)

x	y	z	F2	F2'	T1	T2	T3	F1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Costruita [illegible]

Tutte le [illegible] 2^3

Lez 10

CIRCUITI COMBINATORI II

- Controllo di parità
- Sommatore - sottrattore binario
- Generatore di reparto
- Sommatore decimale

CONTROLLO DI PARITÀ

Il controllo di parità è l'operazione che controlla se il numero di bit di una operazione è un numero pari o dispari. Si fa questo perché durante la trasmissione dei dati spesso ci sono degli errori dovuti a rumore o interferenze.
Una delle procedure più semplici per determinare se il segnale ricevuto ha un errore è quello di aggiungere un bit di parità.

La funzione XOR può essere considerata un generatore di parità per 3 bit, infatti

Generatore 3-bit bit di parità = $x \oplus y \oplus z$

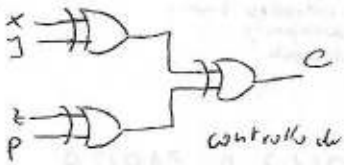
x	y	z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

generatore di parità per 3 bit, cioè il flusso di 4 bit per quanto delle anche della tastiera avrà sempre un numero pari di 1.
Chi legge il parity e non riceve un numero pari di 1 sa che si è verificato almeno un errore (max 3).

Controllo di parità su 4 bit, verrà usato per il bit più a sinistra del codice

Il messaggio è fatto da 3 bit ed un bit di parità

3 bit & P				controllo
x	y	z	P	x⊕y⊕z⊕P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



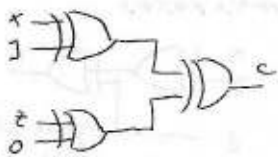
controllo di parità su 4 bit

XOR di 4 variabili, 2 e 2 poi tra i risultati

⇒ la n-pla x y z P contiene un numero dispari di 1

Possiamo rimpiazzare lo stesso circuito e, invece di considerare P come un ingresso, lo mettiamo a massa, quindi a zero e scriviamo l'uscita; questo in pratica è la realizzazione di un generatore di parità a 3 bit.

3 bit			P=0	controllo
x	y	z	P	C
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	



P=0 → generatore di parità per 3 bit implementato con controllo di parità per 4 bit.

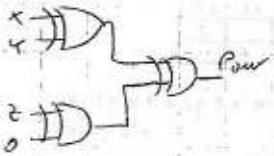
⇓

generatore di parità su 3 bit

3 bit P=0 Parity check, viene rilevata la parità su 3 bit

x	y	z	P	Pout
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

→ eliminazione di righe non necessarie



generatore di parità per 3 bit implementato con controllo di parità per 4 bit.

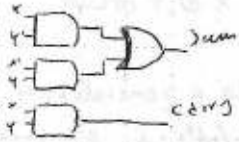
SOMMATORE - SOTTRATTORE BINARIO

op: Mezzo sommatore (half-adder)

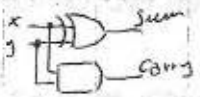
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$Sum = XY' + X'Y = X \oplus Y$

$Carry = XY$



equivalente a



circuito che implementa la funzione di somma e riporto di due numeri binari.

idem come sopra, con un XOR e un AND

SOMMATORE COMPLETO (full-adder)

Carry in: il riporto che viene in ingresso

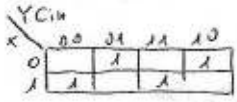
X	Y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$Sum = X \oplus Y \oplus C_{in}$

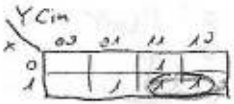
$Cout = C_{in}(XY' + X'Y) + XY =$

$= C_{in}(X \oplus Y) + XY$

Le mappe di Karnaugh del sommatore completo

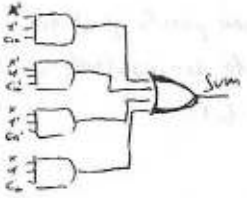


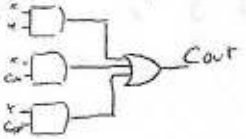
$Sum = x'y'C_{in} + x'yC_{in}' + xy'C_{in}' + xyC_{in}$



$Cout = XY + XC_{in} + YC_{in}$

forma standard (non ridotta)





Dunque il Full adder ha le seguenti funzioni

$Sum = X \oplus Y \oplus C_{in}$ $\quad Cout = C_{in}(X \oplus Y) + XY$

Andando a generalizzare considerando un certo numero di bit, allora, detto i l'i-esimo bit della somma è generato come

$S_i = A_i \oplus B_i \oplus C_i$ (bit i-esimo dei dati in ingresso; carry-in; somma bit i-esimo)

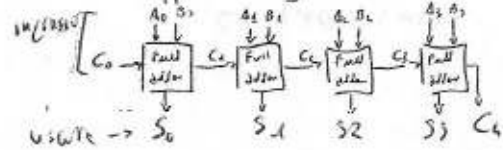
mentre il carry in uscita è

$C_{i+1} = C_{in}(A_i \oplus B_i) + A_i B_i$ (carry in)

è il carry in che entra nello stadio successivo

questo sommatore è detto "ripple carry" perché riporto ad onda, in quanto il carry, cioè il riporto, si propaga da uno stadio all successivo. Abbiamo dunque

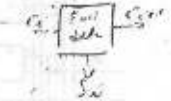
n-bit ripple carry adder

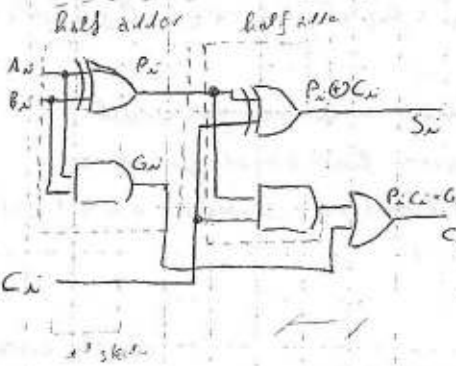


$S_i = A_i \oplus B_i \oplus C_i$

$C_{i+1} = C_i(A_i \oplus B_i) + A_i B_i$

Full adder, composto da due half adder, si notano come sono state introdotte due variabili, P_i e G_i quest'ultimo detto generatore del carry





$S_i = A_i \oplus B_i \oplus C_i$

$C_{i+1} = C_i (A \oplus B_i) + A \cdot B$

GENERATORE DI RIPORTO
(CARRY LOOK-AHEAD GENERATOR)

Con riferimento al circuito precedente abbiamo

$P_i = A_i \oplus B_i$

$G_i = A_i B_i$

$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_i C_i$

I termini di questi di $C_0 C_1 C_2$ e C_3 sono:

C_0 = Input carry

$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1$

$C_3 = G_2 + P_2 C_2$

Vogliamo calcolare le funzioni in cui le prestazioni dipendono soltanto dagli input C_0, C_1, C_2, C_3 in funzione degli ingressi, al fine di ridurre i tempi di

C_0 = Input carry

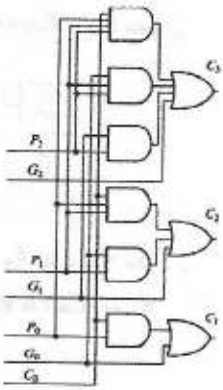
$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1 =$

$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

Possiamo rappresentare questa soluzione in uno schematico



C_0 = riporto in ingresso, è un bit di peso 0, 1, 2, 3

C_0 = riporto in ingresso, è un bit, non rappresentato

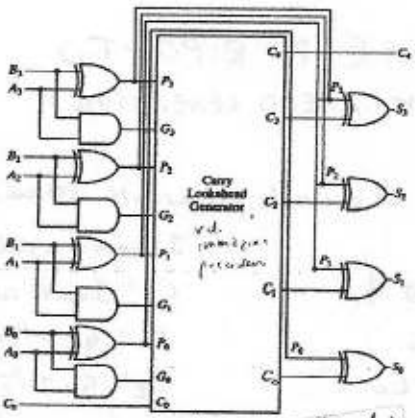
$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

Questo è lo schematico del carry look ahead generator, utilizzato nel sommatore a 4 bit sotto.

$P_i = A_i \oplus B_i$
$G_i = A_i B_i$
$S_i = P_i \oplus C_i$
$C_{i+1} = G_i + P_i C_i$



Questo circuito implementa un completo sommatore a 4 bit che non soffre dei problemi di propagazione attraverso i 4 stadi ma ha un tempo di propagazione indipendente dal numero di stadi

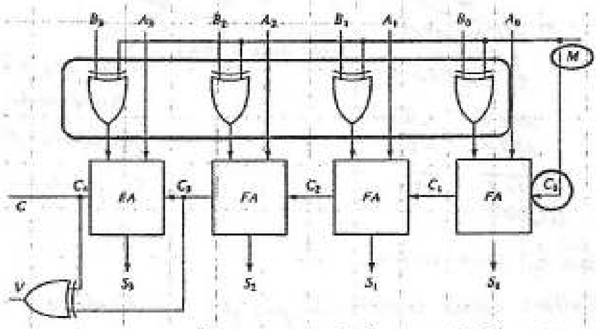
↑ aumentando gli stadi, la propagazione di questo circuito non cambia

Sottrazione in complemento a 2

-> Complementare il sottraendo a n-bit
-> Sommare il minuendo ed il sottraendo complementato
-> Ignorare il riporto
-> La somma è corretta se il risultato è nell'intervallo $[-2^{n-1} + 2^{n-1} - 1]$

IMPLEMENTAZIONE

Sottrattore binario



Si considera che si hanno 9 variabili (8 bit di ingresso più un carry in ingresso), quindi $2^9 = 512$ possibili mintermini da considerare che è una cosa non considerabile. Il full adder risolve egregiamente il problema

Somma se M=0 $\quad x \oplus 0 = x$

Sottrazione se M=1 $\quad x \oplus 1 = x'$

M è il segnale "modo", somma o sottrazione, a seconda del suo valore

Overflow di numeri senza segno

$$\begin{array}{r} 110 + \\ 100 = \\ \hline \text{carry out} \rightarrow (1)010 \end{array} \qquad \begin{array}{r} +6 \\ +4 \\ \hline 10 \end{array}$$

numeri a 3 bit

10 → overflow, con 3 bit max 111_2 = 7_{10}

→ overflow se il carry out = 1

→ Per numeri senza segno: overflow = Cn

Overflow di numeri con segno

L'overflow non si può verificare se gli addendi hanno segno opposto, cioè se i bit di segno sono diversi, si verifica se sono uguali:

$$\begin{array}{r} 0110 + \\ 0100 = \\ \hline 1010 \end{array} \qquad \begin{array}{r} +6 \\ +4 \\ \hline 10 \end{array} \text{ overflow} \qquad\qquad \begin{array}{r} 1110 + \\ 1001 = \\ \hline 10111 \end{array} \qquad \begin{array}{r} -2 \\ -7 \\ \hline -9 \end{array} \text{ overflow}$$

segno della somma

I numeri rappresentabili: [-8, +7]

segno della somma

numeri con segno: i segni della somma sono diversi da quelli degli addendi ⇒ overflow

Quando gli addendi (in complemento a 2) hanno lo stesso segno, l'overflow si verifica se il segno della somma non è quello

Degli addendi: quello è formalizzabile in

$$Overflow = An\,Bn\,Sn' + An'\,Bn'\,Sn$$

il segno degli addendi è positivo — Segno della somma — il segno degli addendi è negativo quello del risultato è positivo

I riporti, o carries

0110 +	6 +	1110	-2
0110 =	6	1001	-7
1100	12	0111	-9
01100		10000	

i riporti della somma, in uscita. In entrambi i casi si ha un overflow

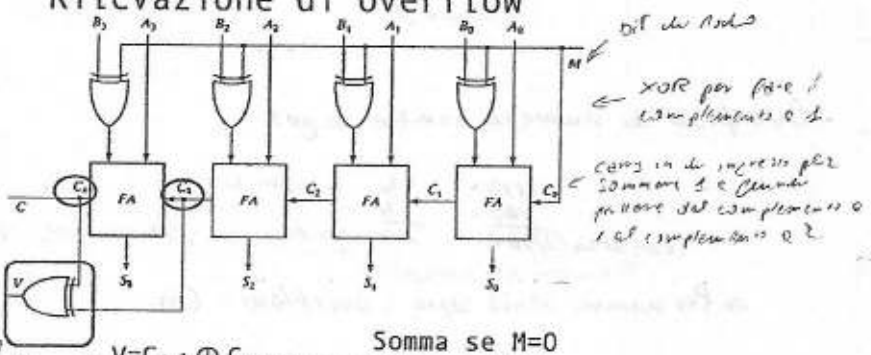
→ carries, i due bit sono diversi

Possiamo anche controllare i due riporti in modo più semplicistico:

$$Overflow = C_{n-1} \oplus C_n$$

, quindi se C_{n-1} e C_n sono diversi, allora overflow = 1

Rilevazione di overflow



bit di modo

← XOR per fare il complemento a 1

← carry in di ingresso per sommare 1 e quindi per avere il complemento a 1 del complemento a 2

Somma se M=0
Sottrazione se M=1

$V = C_{n-1} \oplus C_n$

XOR del carry out più significativo e del carry in del più significativo, che permette il controllo dell'overflow

controllo dell'overflow, si fa lo XOR del carry in (C_{n-1}) e del carry out (C_n) dell'ultimo full adder

SOMMATORE DECIMALE

→ Minimo nove bit di ingresso 4 bit + 4 bit degli addendi + 1 bit carry in

→ Cinque bit di uscita 4 bit della somma + carry out

→ Diverse rappresentazioni binarie dei decimali:

→ 2421

→ Excess-3

→ 8, 4, -2, -1

→ 8421 (aka BCD) DIVERSO

	Somma binaria				Somma BCD					
K	Z8	Z4	Z2	Z1	C'	Z8	Z4	Z2	Z1	Decimale
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9

Finché la somma binaria è ≤ 9 non c'è nessun problema.

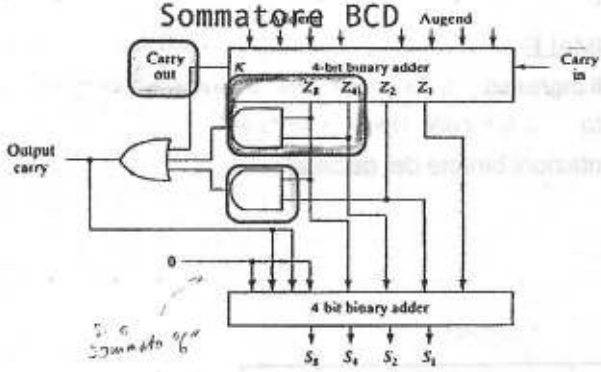
Al decimale 10 si ha un carry C cioè si ha un bit di carry e si scrive 0; all'11 si ha un 1 di carry e si scrive 1; al 12 si ha un 1 di carry e si scrive 2. Questo fino al 18 che è il massimo che si può avere sommando

	Somma binaria				Somma BCD					
K	Z8	Z4	Z2	Z1	C	Z8	Z4	Z2	Z1	Decimale
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18

Nella somma BCD si devono ricordare i bit Z8 e Z4 e i bit Z8 e Z2 che ci dicono che abbiamo superato 10.

$C = K + Z_8Z_4 + Z_8Z_2$

Sommatore BCD



$$C=K+Z_8Z_4+Z_8Z_2$$

CIRCUITI COMBINATORI III

Prof. Romeo Beccherelli
40'23"

Circuiti Combinatori I
- Circuiti combinatori
- Procedura d'analisi
- Procedura di progetto
- La funzione OR esclusivo

Circuiti Combinatori II
- Controllo di parità
- Sommatore-sottrattore binario
- Generatore di riporto
- Sommatore decimale

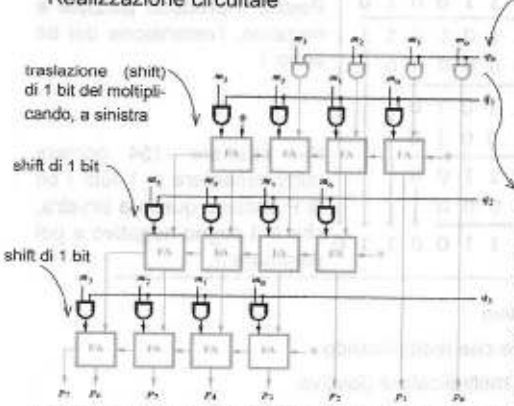
Circuiti Combinatori III
- Moltiplicatori
- Comparatori di grandezza
- Decodificatori
- Codificatori
- Multiplatori

MOLTIPLICATORI

Moltiplicando e moltiplicatore senza segno

Moltiplicando M (14)					1	1	1	0		
Moltiplicatore Q (11)					x	1	0	1	1	
Prodotto parziale 0					1	1	1	0	· (1)	ovvero 1110 x 1
	(shift left)	+	1	1	1	0				ovvero 1110 x 1
Prodotto parziale 1			1	0	1	0	1		· (2)	
		+	0	0	0	0				ovvero 1110 x 0
Prodotto parziale 2			0	1	0	1	0		·	
		+	1	1	1	0				ovvero 1110 x 1
Prodotto P (+154)		1	0	0	1	1	0	1	0	

Realizzazione circuitale



Ad ogni traslazione (shift) di 1 bit a sinistra, si moltiplica il moltiplicando per i bit di peso crescente del moltiplicatore fino ad ottenere i bit di peso più basso fino a quelli di peso più alto, da destra verso sinistra. Da due fattori di 4 bit si ha un prodotto di 8 bit, da p_0 a p_7.

q_0, il peso più basso, messo in AND con 4 bit del moltiplicatore

(1) operazione che produce il prodotto parziale di ordine 0

(2) prodotto parziale di ordine 1

Abbiamo degli AND logici fra i 4 bit del moltiplicando M con il bit meno significativo del moltiplicatore (di valore 1, il bit più a destra).

Poi un ulteriore serie di AND logici, sempre fra i 4 bit del moltiplicando M, ma questa volta con il 2° bit meno significativo del moltiplicatore Q, ovvero d_1, l'1 che è il secondo da destra.

Notare come i risultati parziali prima, o il prodotto finale, entrino in full adder. Alla fine si hanno 8 bit del risultato.

Numeri con segno: moltiplicando positivo

(in complemento a 2 => un bit aggiuntivo di estensione; con 5 bit si rappresentano numeri da -15 a +16)

```
Moltiplicando M   (+14)          0 1 1 1 0
Moltiplicatore Q  (+11)        x 0 1 0 1 1
                               -----------
Prodotto parziale 0            0 0 0 1 1 1 0
                             + 0 0 1 1 1 0
                               -----------
Prodotto parziale 1            0 0 1 0 1 0 1
                           + 0 0 0 0 0 0
                             -------------
Prodotto parziale 2          0 0 0 1 0 1 0
                         + 0 0 1 1 1 0
                           -------------
Prodotto parziale 3          0 0 1 0 0 1 1
                       + 0 0 0 0 0 0
                         -------------
Prodotto P        (+154)   0 0 1 0 0 1 1 0 1 0
```

In rosso i bit di segno.
L'estensione dei bit sono 0, a causa del segno positivo dei due fattori, M e Q.

Quindi l'operazione di moltiplicazione di numeri con segno, entrambi positivi, è in sostanza identica a quella dei numeri senza segno.

Numeri con segno: moltiplicando negativo

(in complemento a 2 => un bit aggiuntivo di estensione; con 5 bit si rappresentano numeri da -15 a +16)

```
Moltiplicando M   (-14)          1 0 0 1 0
Moltiplicatore Q  (+11)        x 0 1 0 1 1
                               -----------
Prodotto parziale 0            1 1 1 0 0 1 0
                             + 1 1 0 0 1 0
                               -----------
Prodotto parziale 1            1 1 0 1 0 1 1
                           + 0 0 0 0 0 0
                             -------------
Prodotto parziale 2          1 1 1 0 1 0 1
                         + 1 1 0 0 1 0
                           -------------
Prodotto parziale 3          1 1 0 1 1 0 0
                       + 0 0 0 0 0 0
                         -------------
Prodotto P        (-154)   1 1 0 1 1 0 0 1 1 0
```

In rosso i bit di segno.
Poiché il prodotto parziale è negativo, l'estensione dei bit sono 1.

Per ottenere -154 occorre complementare a 1 tutti i bit di P escluso quello a sinistra, che è il segno negativo e poi sommare 1.

Numeri con segno: moltiplicatore negativo

→ Complementare a 2 sia moltiplicatore che moltiplicando

→ Moltiplicare questi come nel caso di moltiplicatore positivo

→ Il prodotto avrà segno corretto!

→ In alternativa, poiché il prodotto logico è commutativo, scambiare di posto il moltiplicando ed il moltiplicatore

COMPARATORE DI GRANDEZZA

→ Determina la grandezza relativa di due numeri

→ Il risultato è specificato da alcune variabili che indicano se A > B, A = B o A < B

→ Per 2 numeri a n bit, 2^{2n} righe nella tabella della verità!

Poiché il numero di componenti usando la forza bruta sarebbe troppo elevato, analizziamo come funziona il confronto di numeri, ad esempio a 4 bit, che avrebbe una tabella della verità con 2^8 righe, ovvero 256 righe.

Confronto di numeri a 4 bit

Si considerano 2 numeri, A e B, di 4 bit e si analizza la soluzione che porta alla comparazione tra i due numeri. Vengono introdotte delle variabili x_i che sono l'equivalenza tra A e B; x_i è il complemento dell'OR esclusivo.

$A = A_3A_2A_1A_0$ $\qquad$ $B = B_3B_2B_1B_0$

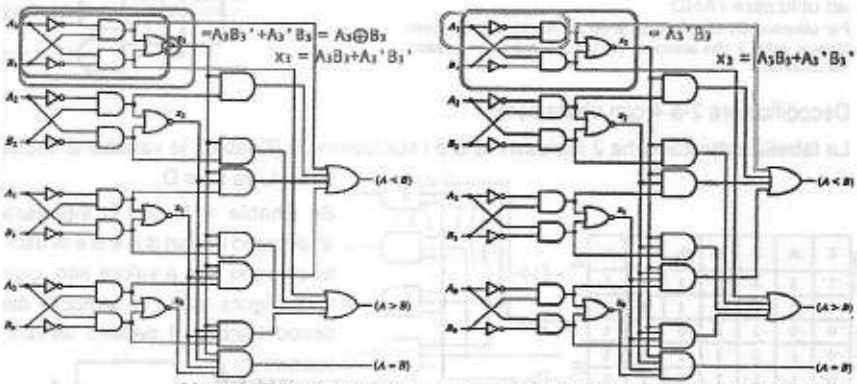
$x_i = A_iB_i + A_i'B_i'$ $\qquad$ per i = 0, 1, 2, 3 $\qquad$ n.b.: $x_i = A_iB_i + A_i'B_i'$ significa che $A_i = B_i$

$(A = B) = x_3x_2x_1x_0$

$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$ $\qquad$ n.b.: prendendo ad esempio A_3B_3', significa che se $A_3 = 1$ e $B_3 = 0$ allora $B_3' = 1$ e quindi $A_3B_3' =$ ed è forzante nell'OR

$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$

Il circuito di questa soluzione è:



^ Le variabili intermedie x_0, x_1, x_2 e x_3 vengono messe in alcuni AND, per cui se una variabile è 1 è quindi trovata l'equivalenza.

Notare che il blu esclude il pallotto della negazione, che è inclusa nell'arancio. In pratica in blu lo XOR fra A_3 e B_3 e in arancio il suo complemento.

Quindi x_3 è $(A_3 \oplus B_3)'$, per cui $x_3 = (A_3 \oplus B_3)' = A_3B_3 + A_3'B_3'$ (vd. 9.4 appunti)

DECODIFICATORI

→ Il decodificatore converte informazione binaria da n linee di ingresso ad un massimo di 2^n distinte linee di uscita

→ Genera $m \leq 2^n$ minterms; 2^n è il numero massimo per n linee

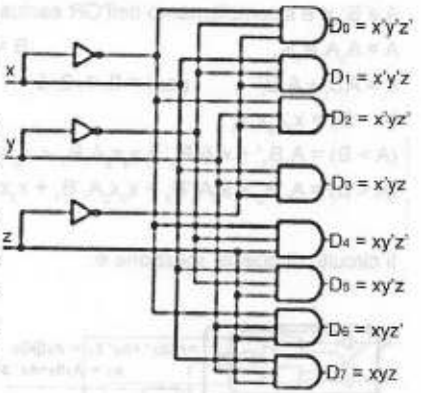
→ Ogni combinazione di ingressi asserisce una sola uscita

Decodificatore 3-a-8

Ingressi			Uscite							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_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

A lato l'immagine del decodificatore da tre a otto: 3 ingressi, 8 uscite, con una sola uscita per volta ad 1, le altre a 0.

Questo decodificatore in inglese è detto decoder three-to-eight, di cui sotto la sua rappresentazione circuitale.



Questo è un semplice circuito che produce in uscita un solo minterm asserito e tutti gli altri nulli.

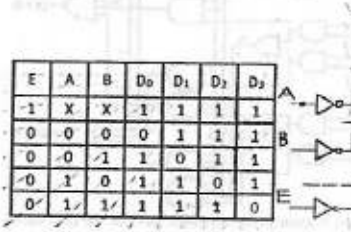
I NAND sono automaticamente disponibili in tecnologia CMOS quindi è più economico usare tutte le porte invertite invece di andare ad utilizzare l'AND.

Per ottenere un AND, produciamo un NAND con un invertitore a valle, il che aumenta di due il numero di transistor nel circuito.

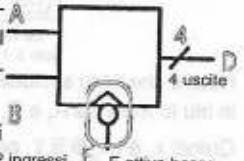
Decodificatore 2-a-4 con abilitazione

La tabella sottostante ha 2 ingressi A e B e l'abilitazione E (Enable); le variabili di uscita sono 4, da D_0 a D_4.

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

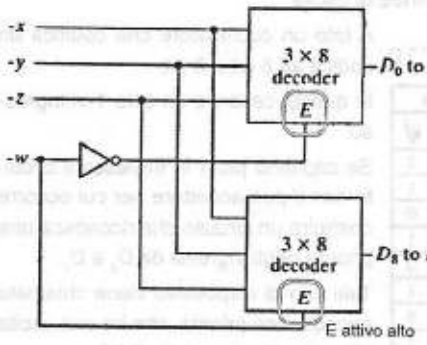


Se Enable = 1, non ci interessa quali siano i valori di A e B e le uscite saranno tute a valore alto, cioè 1. In figura, sotto, il simbolo del decodificatore. Il pallotto all'abilitazione indica che il segnale in ingresso è attivo basso. Tale pallotto indica che esiste in qualche modo un invertitore incorporato in ingresso.



Da questo tipo di decoder si possono realizzare decodificatori di

dimensioni maggiori, come sotto riportato.



Il decodificatore 3x8 avrà tre ingressi + 1 segnale di enable ed avrà 8 uscite.

Per pilotare il segnale di enable si prende il quarto bit di w, ovvero del codice in ingresso. In questi decoder si noti che l'enable è attivo alto (mancanza del pallotto; al posto del pallotto ci può essere una barra sopra oppure un apostrofo, simbolo di negazione. Questo per capire se il segnale è attivo basso o attivo basso).

Sono prodotti 8 bit della parte meno significativa con il decoder in alto ed 8 bit della parte più significativa con quello in basso. Quindi sono stati decodificati 4 bit producendo i 16 minterm possibili.

Full adder

x	y	C_{in}	Sum	C_{out}
0	0	0	0	0
0	1	1	1	0
0	0	0	1	0
0	1	1	0	1
1	0	0	1	0
1	1	1	0	1
1	0	0	0	1
1	1	1	1	1

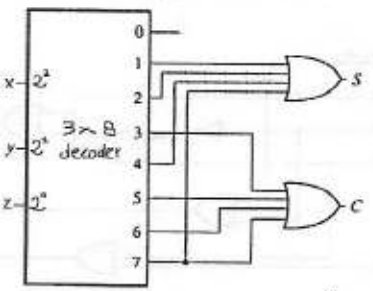
$Sum = x \oplus y \oplus C_{in} =$
$= \Sigma(1,2,4,7)$

$C_{out} = C_{in}(xy' + x'y) + xy =$
$= C_{in}(x \oplus y) + xy =$
$= \Sigma(3,5,6,7)$

n.b.: C_{in} è il riporto in input, C_{out} è il riporto in output

Si veda ora una implementazione del full adder con un decoder da 3 a 8 senza abilitazione, con due porte OR.

In questa situazione si nota l'utilità delle forme canoniche che ci permettono di utilizzare un dispositivo standard, un decoder, con quella che è una media scala di integrazione, semplicemente mettendo a valle OR si può produrre automaticamente qualunque funzione e da un singolo decoder possiamo produrre una molteplicità di funzioni semplicemente utilizzando una corrispondente molteplicità di porte OR correttamente connesse in uscita.



Decoder 3x8 che implementa il Full adder, ovvero la funzione Sum e la funzione Cout

CODIFICATORI

→ Un codificatore (encoder) svolge l'operazione inversa di un decodificatore (decoder)

→ Prende m ≤ 2^n linee di ingresso ed ha n linee di uscita.

Encoder da ottale a binario

Ingressi								Uscite		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	V
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

A lato un codificatore che codifica un codice ad 8 bit a 3 bit.

In questo caso c'è un solo 1 in ingresso.

Se capitano più 1 in ingresso il circuito non li può accettare per cui occorre costruire un circuito che riconosca una priorità degli ingressi da D_0 a D_7.

Tale tipo di dispositivo viene chiamato encoder con priorità, che ha una uscita V in più, che sta per "valido".

Encoder con priorità

Ingressi				Uscite		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

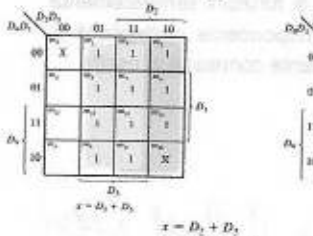
Si noti come la prima riga sia la codifica di "stato non valido" per cui sono stati codificati 5 stati.

Inoltre si noti l'importanza del bit più significativo.

Dalla tabella della verità si ricava la mappa di Karnaugh.

Il circuito che si ottiene è una forma standard, cioè una somma di prodotti e l'invertitore serve solo per produrre la variabile D_2 che abbiamo detto può a volte essere disponibile già in forma complementata.

mappa di Karnaugh

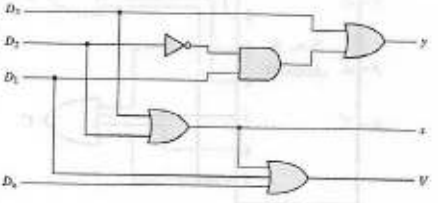


$x = D_2 + D_3$

$y = D_1 + D_3 D_2'$

$V = D_0 + D_1 + D_2 + D_3$

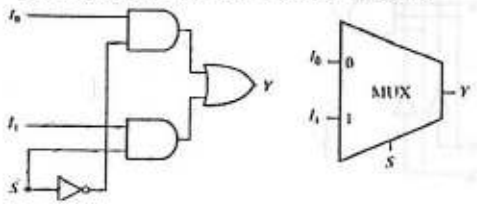
circuito



(→ V indica "valid", questa metodologia rende il codificatore (encoder) del tipo a priorità

Multiplatori (multiplexer) 2 in 1

- Seleziona una fra diverse linee
- La connette ad un uscita comune



Un multiplatore 2 in 1 seleziona una fra diverse linee e la connette ad una uscita comune.

S è il segnale di select.

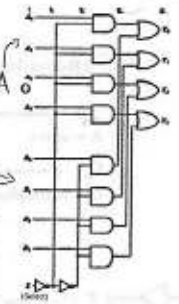
Y è il segnale di uscita.

Nel caso più semplice di due diverse linee, un multiplatore è realizzabile mettendo in AND un segnale di selezione opportunamente complementato su una delle due linee. In figura si tira fuori I0 se S = 0 o si tira fuori I1 se S = 1.

Si possono realizzare multiplexer multipli che pilotino, ad esempio, insiemi di piste contemporaneamente.

Multiplexer quadruplo 2 in 1

- S seleziona ingressi A o B e li connette alle uscite comuni



S	Uscita
0	seleziona A
1	seleziona B

Un multiplatore quadruplo 2 in 1 ha per uscita S che seleziona gli ingressi A o B e li connette alle uscite comuni.

A e B sono nibble; → parole di 4 bit, word è una parola a 32 bit

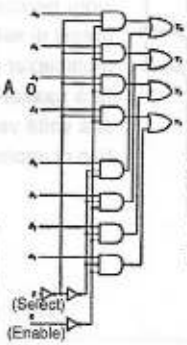
Y è l'uscita.

Y sarà A o B a seconda del valore di select.

Il circuito a lato può essere complicato ulteriormente con l'aggiunta di un segnale di abilitazione.

Multiplexer quadruplo 2 in 1

- S seleziona ingressi A o B e li connette alle uscite comuni
- E abilita multiplexer



E	S	Uscita
0	0	seleziona A
0	1	seleziona B
1	X	tutti 0

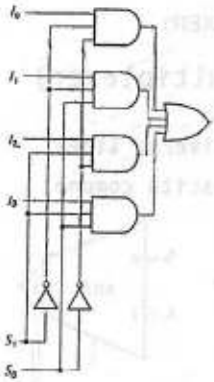
Questo circuito può essere complicato inserendo un circuito di enable, cioè di abilitazione, come a lato.

A seconda del valore di Enable, che è attivo basso, si selezionerà A o B.

Se Enable = 1, nessun AND è abilitato e in uscita si avranno tutti 0.

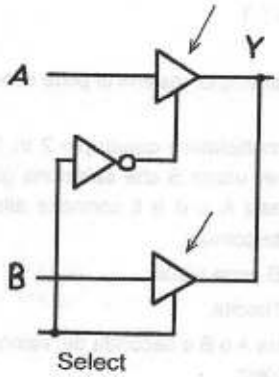
multiplexer 4 in 1

➡ S_0 e S_1 sono decodificati con gli AND per selezionare uno degli ingressi e connetterlo ad un ingresso comune



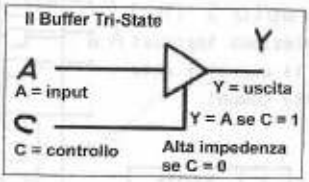
Multiplexer 4 in 1

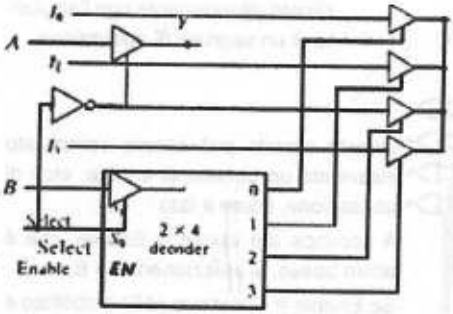
Possiamo realizzare multiplexer con una molteplicità di ingressi opportunamente codificati. In figura si nota una codifica di S_0 e S_1.



Multiplexer con porte three-state, a tre stati

Una delle due porte è aperta e l'altra è chiusa a seconda del valore del select.





Con un multiplexer tri-state ed un decoder possiamo realizzare diverse funzioni perché il decoder abiliterà gli ingressi di selezione del multiplexer ed il multiplexer realizzato con il three-state farà passare gli ingressi da I_0 a II_3 uno alla volta verso l'uscita Y, rappresentativo di implementazione di funzioni.

LEZIONE 12

CIRCUITI SEQUENZIALI SINCRONI I

Prof. Romeo Beccherelli
43'44"

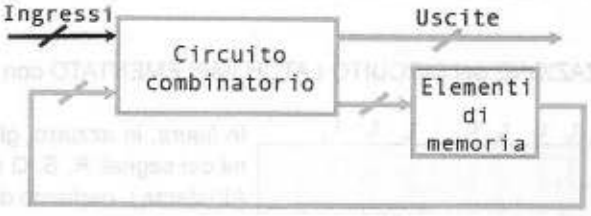
Circuiti Sequenziali Sincroni I
- Circuiti sequenziali
- Elementi di immagazzinamento dell'informazione: Latches e D-type Flip-Flop

Circuiti Sequenziali Sincroni II
- Elementi di immagazzinamento dell'informazione: JK-type e T-type Flip-Flop
- Analisi di circuiti sequenziali sincroni
- Assegnazione e riduzione degli stati
- Procedura di progetto

Circuiti Sequenziali Sincroni III
- Registri
- Registri a scorrimento
- Contatori "ripple" (ad ondulazione)
- Contatori sincroni
- Altri contatori

CIRCUITI SEQUENZIALI

A differenza di quelli combinatori, i circuiti sequenziali hanno una retroazione.



Sopra, un normale circuito combinatorio con una qualche funzione, vettoriale perché con uscite multiple, implementata.

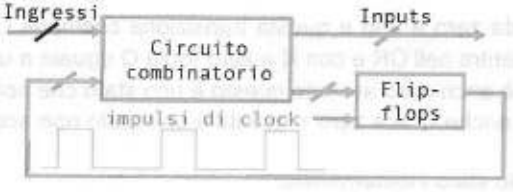
Dal circuito combinatorio escono anche altri segnali che vanno a finire in elementi di memoria i quali rientrano nel circuito combinatorio (da altre parti) e vengono trasformati andando ad influenzare le uscite intermedie e le uscite finali.

Gli elementi di memoria utilizzati sono i flip-flop.

Un flip-flop è un elemento di memoria binaria capace di immagazzinare un bit di informazione.

In uno stato stabile, l'output di un flip-flop è 0 o 1.

Circuiti sequenziali temporizzati



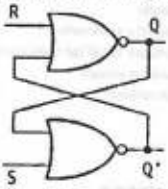
I circuiti sequenziali sono sincroni in presenza di un impulso di clock, che è un segnale di sincronismo periodico con impulsi equispaziati che, a livello o a fronte, controllano gli elementi di memoria. Si può dunque parlare di circuiti sequenziali temporizzati, come a lato.

Nei F/F la transizione da uno stato all'altro avviene solo ad intervalli dettati da impulsi di clock.

ELEMENTI di IMMAGAZZINAMENTO dell'INFORMAZIONE: LATCHES

Un elemento di immagazzinamento in un circuito digitale può mantenere indefinitamente uno stato binario (finché è alimentato il circuito). Il cambio di stato è dettato da un segnale di input.

SR LATCH implementato con NOR

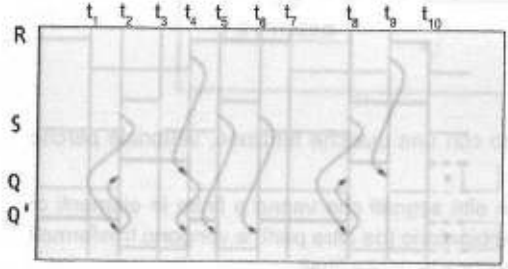


S	R	Q	Q'	
0	0	Q	Q'	nessun cambiamento
0	1	0	1	
1	0	1	0	
1	1	0	0	non determinato

IMPLEMENTAZIONE con NOR

Il Latch implementato con NOR sono due NOR in controreazione l'uno con l'altro.

TEMPORIZZAZIONE del CIRCUITO LATCH IMPLEMENTATO con NOR



In figura, in azzurro, gli oscillogrammi dei segnali R, S, Q e Q'.

All'istante t_1 partiamo da uno stato in cui Q e Q' sono a zero.

All'istante t_2 il segnale di set, S, va a 1 e forza Q' a zero. Quando Q' va a zero, il feedback torna indietro e forza anche Q a 1, in quanto R e Q' sono a 0.

All'istante t_3 non ci sono cose particolare interesse, il segnale va a zero e non cambia stato. Quando si arriva a T4 R transisce a uno e costringe Q ad andare a zero e quando Q va a zero forza Q' in sequenza a uno.

All'istante T5 il segnale S di set passa da zero a uno ed appena questo viene stiamo forzando Q' a zero.

Quindi, a questo punto, ci troviamo sia Q sia Q' a zero che è uno stato non gradito.

All'istante t_6 il segnale S passa da uno a zero e quindi forza la transizione di Q' e Q diventa zero.

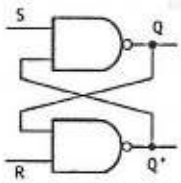
A t_8 vediamo che Set passa da zero a uno e questa transizione costringe Q' a diventare zero. Questo zero rientra nell'OR e con R a zero forza Q uguale a uno.

A t_9 Reset viene asserito, Set è anch'esso asserito questo è uno stato che non ci piace. Q è forzato a zero, ma anche Q' è a zero e questo è uno stato non accettabile.

A t_{10} la transizione ci porta nello stato indeterminato.

La situazione è dunque tale per cui non vogliamo in ingresso S = 1 e R = 1.

SR LATCH implementato con NAND



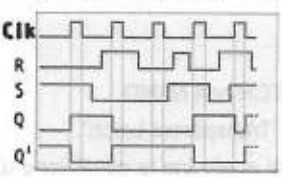
S	R	Q	Q'	
1	1	Q	Q'	nessun cambiamento
0	1	1	0	
1	0	0	1	
0	0	1	1	non determinato

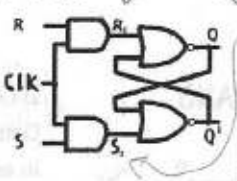
Gli ingressi S e R sono forzanti a 0.

Il problema dei latches è quello di avere stati non determinati, situazione che viene risolta nei D LATCH.

Di seguito i Gated Latch, in cui le porte (AND in un primo caso e NAND nel successivo), fanno da porte controllate da un segnale di clock.

Gated LATCH con NOR





Clk	S	R	S_n	R_n	Q(t+1)	Q'(t+1)	
0	0	0	0	0	Q	Q'	nessun cambiamento
1	0	0	0	0	Q	Q'	nessun cambiamento
1	0	1	0	1	0	1	
1	1	0	1	0	1	0	
1	1	1	1	1	?	?	non determinato

Gated LATCH con NOR

$R_l = S_l = 0$ se clock = 0.

Le transizioni di Q si verificano solo in presenza di un segnale di clock alto, altrimenti qualunque transizione di R e di S è del tutto irrilevante.

All'inizio della slide si nota che quando il clock è alto, il segnale Q transisce da 0 a 1 perché legge che S è alto e R è basso. Contemporaneamente, o quasi, transisce anche Q', da 1 a 0.

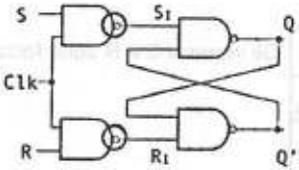
Se nel frattempo il segnale S è cambiato da 1 a 0, il segnale Q il segnale Q' non vengono più modificati perché sono in AND con lo 0 del clock, quindi la porta rappresentata dagli AND è chiusa.

Se la transizione è su R il discorso è analogo.

I segnali S e R riescono a passare attraverso la porta solo quando il segnale di clock è nuovamente abilitato, cioè è nuovamente alto.

La tabella di stato è visualizzata sopra nella figura relativa.

Gated LATCH con NAND

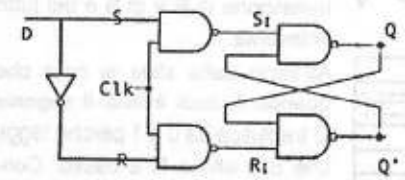


Clk	S	R	S1	R1	Q(t+1)	Q'(t+1)	
0	0	0	1	1	Q	Q'	nessun cambiamento
1	0	0	1	1	Q	Q'	nessun cambiamento
1	0	1	1	0	0	1	
1	1	0	0	1	1	0	
1	1	1	0	0	1	1	non determinato

Gated LATCH con NAND

Le porte NAND sono immediatamente disponibili.

D LATCH con NAND



Clk	D	S1	R1	Q(t+1)	Q'(t+1)	
0	0	1	1	Q	Q'	nessun cambiamento
1	0	1	0	0	1	
1	1	0	1	1	0	

D LATCH con NAND

Detti "Transparent Latch".

In essi si elimina la condizione di stato non gradito.

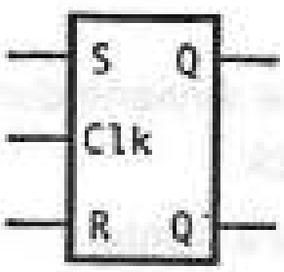
Si usa solo un segnale, D, che sta per dato. Il dato "entra" nel circuito, il dato setta o resetta lo stato Q.

La tabella è più semplice in quanto S e R non sono più indipendenti, ma c'è solo D.

il D Latch è implementato con 4 porte NAND e un invert, per un totale di 18 transistori.

Ogni porta NAND è implementata con 4 transistori (2 pMOS e 2 nMOS); l'invert con 1 pMOS e 1 nMOS.

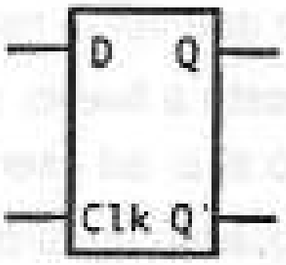
Simboli grafici dei latches



Gated SR latch

I 3 segnali

e le uscite

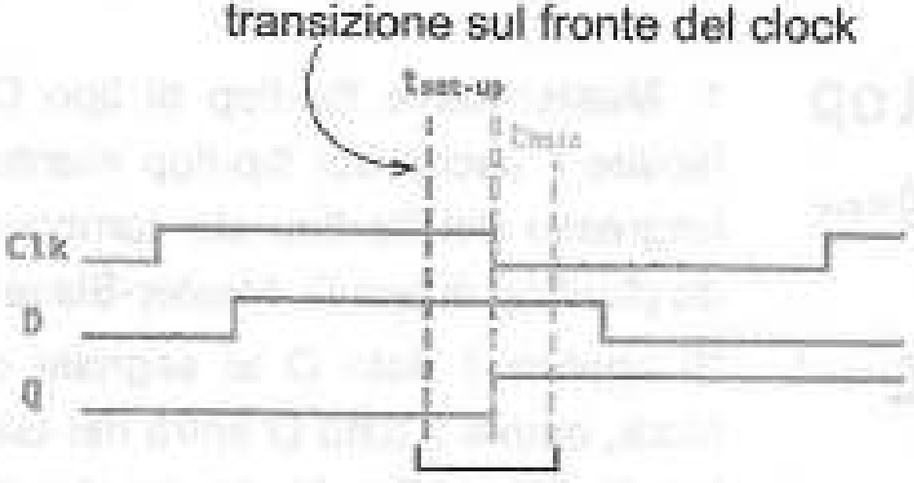


Gated D latch

I 2 ingressi (sx)

e le 2 uscite (dx)

Effetto del tempo di propagazione



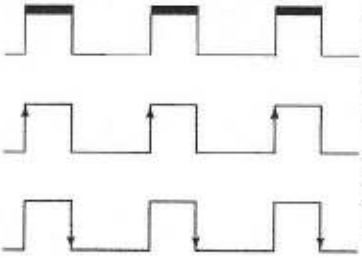
il dato non deve variare
all'interno di questa finestra
altrimenti lo stato è indefinito

Effetto del Tempo di Propagazione

Il tempo di set-up è il tempo in cui il segnale di dato deve rimanere costante senza variazione prima della transizione di clock.

C'è un secondo tempo che ci interessa: quanto tempo il segnale di dato deve rimanere senza una transizione dopo la transizione del clock e questo tempo è detto hold-time.

Concetto introduttivo di risposta a livello e fronte



Sono due tipi di risposta.

Risposta a livello: qualcosa che avviene durante il livello alto, ad esempio del clock.

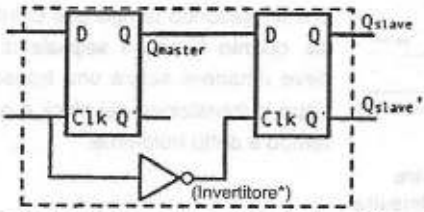
Per quanto riguarda la risposta a fronte, essa può essere a fronte positivo o a fronte negativo.

La risposta a fronte è la metodologia preferita.

• Nei Flip-Flop di tipo D occorre far commutare l'elemento di memoria solamente durante la transizione del segnale. Si deve cioè isolare l'ingresso dall'uscita per evitare transizioni indesiderate

• Sono possibili due implementazioni dei flip-flop di tipo D: Master-Slave o Edge-Triggered

Master slave D flip-flop



➡ Isolate l'uscita del flip-flop e impedisce che sia modificata quando il segnale di ingresso flip-flop varia

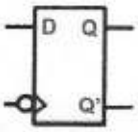
1. Master-Slave flip-flop di tipo D. Isolate l' uscita del flip-flop mentre l'ingresso del flip-flop sta cambiando (flip-flop di tipo D, Master-Slave).

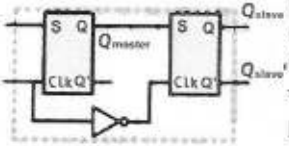
Si applica il dato D al segnale di clock, quindi il dato D entra nel Gated D Latch che fa da master e, dopo un certo tempo di propagazione, Q_{master} è settato, a 0 o a 1.

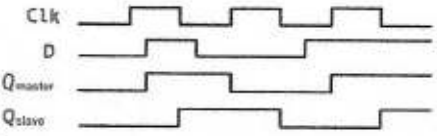
Nello slave non c'è nessuna transizione, essendo il clock negato.

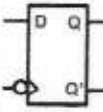
Quando si ha la seconda transizione e il clock che passa da 1 a 0, il master non si altera, lo slave fa transitare Q_{master} verso Q_{slave}. Master e slave lavorano nel mezzo periodo del clock.

L'invertitore (*) controlla il clock dello slave.





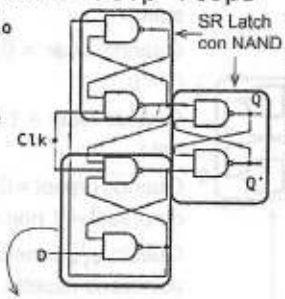




il segnale in uscita arriva quando il clock ha la transizione negativa

Edge-triggered D flip-flops

- Fa commutare solo durante una transizione del segnale di sincronizzazione (clock)
- disabilita il segnale d'ingresso durante la parte rimanente dell'impulso



2. Edge-triggered flip-flop di tipo D. Fate commutare solo durante la transizione (da O a 1 o da 1 a O) del segnale di sincronizzazione (clock) e disabilitate durante il resto dell'impulso di clock (flip-flop di tipo D, Edge-Triggered, ovvero con transizione a fronte)

Consideriamo una transizione partendo da una situazione originaria:

clk=0:

$P_1=P_3=1$,

Q nessun cambiamento

$P_4=D'$, $P_2=D$

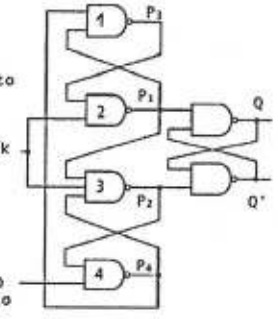
clk→1:

$P_3=D'$, $P_2=D$,

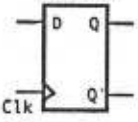
$Q=D, Q'=D'$

clk→0:

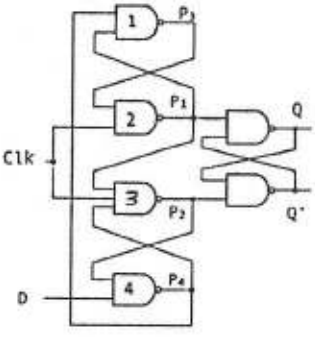
Q nessun cambiamento



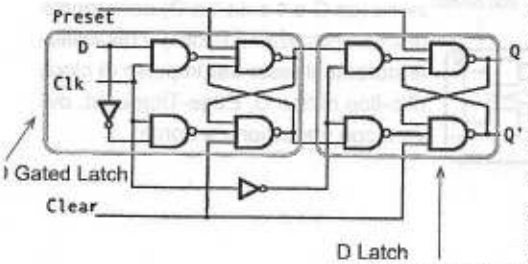
Simbolo:



Questo F/F è attivo sul fronte positivo



Master slave D flip-flops con Clear e Preset



Il clear ed il preset servono rispettivamente per cancellare e per settare lo stato.

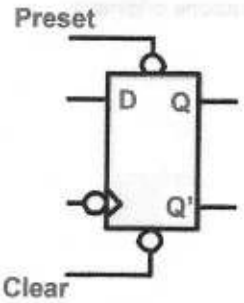
Quando clear = 0, esso setta Q' = 0 e Q = 0.

Quando clear = 1 lo stato rimane invariato.

Quando preset = 0 si forza Q = 1; quando preset = 1 non cambia nulla.

Questo ci permette di settare in modo asincrono rispetto al clock.

Simbolo



il F/F è attivo
sul fronte negativo

PDF al 74%

LEZIONE 13

CIRCUITI SEQUENZIALI SINCRONI II

Prof. Romeo Beccherelli

38'18"

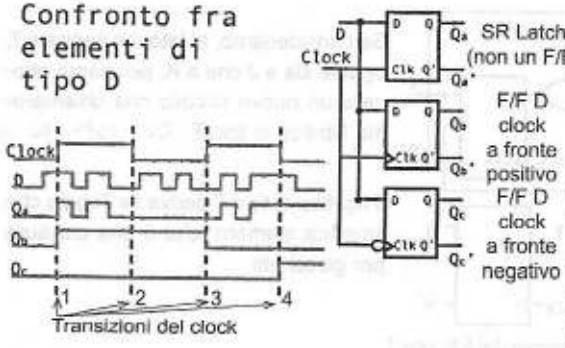
Circuiti Sequenziali Sincroni I
- Circuiti sequenziali
- Elementi di immagazzinamento dell'informazione: Latches e D-type Flip-Flop

Circuiti Sequenziali Sincroni II
- Elementi di immagazzinamento dell'informazione: JK-type e T-type Flip-Flop
- Analisi di circuiti sequenziali sincroni
- Assegnazione e riduzione degli stati
- Procedura di progetto

Circuiti Sequenziali Sincroni III
- Registri
- Registri a scorrimento
- Contatori "ripple" (ad ondulazione)
- Contatori sincroni
- Altri contatori

FLIP-FLOP di Tipo D

Di seguito tre tipi di flip-flop con analisi della sequenza temporale dei segnali in ingresso e in uscita. Il primo è in realtà un SR Latch, quindi non un flip-flop. Il secondo è un flip-flop di tipo D con segnale di clock a fronte positivo. Il terzo è un flip-flop di tipo D con segnale di clock a fronte negativo.



Abbiamo classico clock, con un duty cycle, cioè un fattore di riempimento del 50%. A tutti e tre gli elementi di immagazzinamento viene messo in ingresso un segnale D, che varia in modo arbitrario rispetto al clock.

Evidenziando le transizioni del clock si notano le varie uscite.

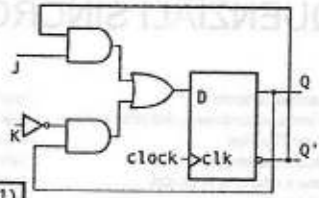
Q_a commuta più volte durante il livello alto del clock e non commuta durante il livello basso.

Per il flip-flop di tipo D, con segnale di clock a fronte positivo, il segnale di ingresso è memorizzato all'interno del flip-flop e permane all'uscita Q_b fino a che non c'è una nuova transizione.

Per il flip-flop di tipo D a fronte negativo, ci accorgiamo che se il flip-flop aveva memorizzato uno stato basso, alla transizione due viene memorizzato di nuovo lo stato basso, al fronte positivo non avviene nulla, mentre a fronte negativo viene memorizzato il valore di D.

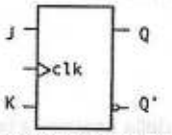
FLIP-FLOP di tipo JK e T

D=JQ'+K'Q



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q(t)'

→ Q + Q' = 1



Simbolo del F/F tipo JK
commuta sul fronte positivo

Abbiamo un flip-flop D a fronte positivo, con uscite Q e Q' che sono messe in contro reazione attraverso un circuito combinatorio; al flip-flop di tipo D c'è in ingresso anche un clock; alle porte AND definiamo due segnali, J e K, in funzione dei quali facciamo una tabella di stato.

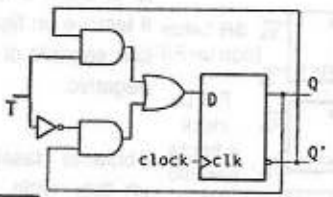
Si noti che Q(t+1) rappresenta lo stato successivo, Q(t) lo stato precedente.

D=JQ'+K'Q

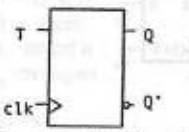
J=T

K=T

D=TQ'+T'Q



T	Q(t+1)
0	Q(t)
1	Q(t)'

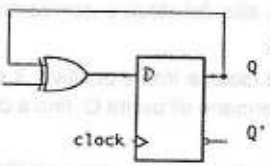


Simbolo del F/F tipo T
commuta sul fronte positivo

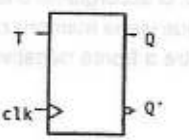
Se consideriamo, al lato, un segnale T, uguale sia a J che a K, possiamo ottenere un nuovo circuito che chiameremo flip-flop di tipo T. Derivato da quello JK.

Il flip-flop di tipo T deriva da Toggle che significa alamaro, che è una chiusura per giacchetti.

D=TQ'+T'Q =T⊕Q



T	Q(t+1)
0	Q(t)
1	Q(t)'



Simbolo del F/F tipo T
a commutazione sul fronte positivo

A lato la rappresentazione del flip-flop di tipo T, con uno XOR.

Questo flip-flop di tipo T è derivato dal flip-flop di tipo D.

Equazione caratteristica dei flip-flop

- D: $Q(t+1) = D$ (lo stato dell'istante successivo è pari all'ingresso D dello stato attuale)
- JK: $Q(t+1) = JQ' + K'Q$
- T: $Q(t+1) = TQ' + T'Q = T \oplus Q$

↑ stato successivo, next state

Tabella caratteristica dei flip-flop

➡ D: $Q(t+1)=D$

D	Q(t+1)
0	0
1	1

➡ JK: $Q(t+1)=JQ'+K'Q$

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q(t)'

➡ T: $Q(t+1)=TQ'+T'Q=T\oplus Q$

T	Q(t+1)
0	Q(t)
1	Q(t)'

Ci sono 3 operazioni che possono essere eseguite con un flip-flop: settarlo a 1, resettarlo a 0, o complementare il suo output. Con un solo ingresso, il flip-flop di tipo D può settare o resettare l'output, in funzione del valore dell'input di D subito prima la transizione del clock. Sincronizzato da un segnale di clock. Il flip-flop di tipo JK ha due input ed entrambi servono per le operazioni. Come da figura relativa, il flip-flop di tipo JK ha due input e può eseguire tutte e 3 le operazioni.
Come da figura relativa, l'input J setta il flip-flop a 1, l'input K resetta il flip-flop a 0, e quando entrambi gli input sono eccitati, l'output è complementato. Questo può essere verificato analizzando il circuito applicato all'input D, in riferimento alla figura in alto a sinistra della pagina accanto: $D = JQ' + K'Q$

3

ANALISI DEI CIRCUITI SEQUENZIALI SINCRONI

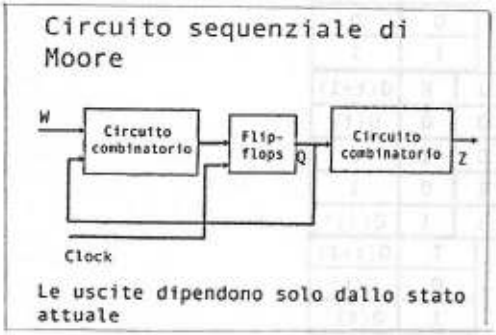
Macchina a stati finiti

→ I circuti sequenziali sono chiamati macchine a stati finiti "Finite State Machines (FSM)" poiché il loro funzionamento può essere rappresentato con un numero finito di stati.

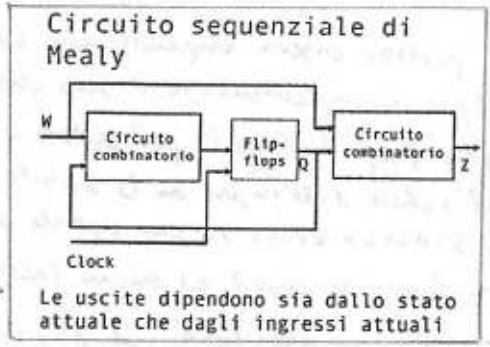
→ In un circuito sequenziale, i valori delle uscite dipendono sia dal comportamento passato del circuito sia dai valori attuali degli ingressi

→ Un circuito sequenziale contiene uno o più flip-flop e logica combinatoria (non SR Latch!)

→ I flip-flops vengono fatti commutare ("triggered") al fronte attivo del clock (positivo o negativo) 



Il Circuito Sequenziale di Moore è un insieme di F/F con un certo numero di uscite Q, che entrano in due circuiti combinatori, di cui uno eccita i F/F e l'altro che produce una uscita Z per il resto del mondo.



Il Circuito Sequenziale di Mealy si differenzia dal precedente in quanto le uscite dipendono sia dallo stato attuale che dagli ingressi attuali.

Il circuito combinatorio a destra accetta non solo gli stati ma anche gli ingressi attuali.

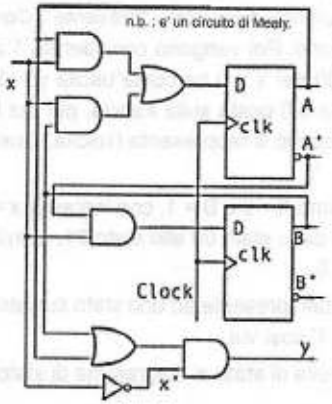
→ Derivare una espressione booleana che descriva il comportamento del circuito sequenziale

o

→ Derivare una tabella di stato

o

→ Derivare un diagramma di stato per la sequenza degli ingressi, uscite e stati interni



$A(t+1) = A(t)x(t) + B(t)x(t)$

$B(t+1) = A'(t)x(t)$

$y(t) = [A(t)+B(t)]x'(t)$

Tabella di stato

present			prossimo		
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

$A(t+1) = A(t)x(t) + B(t)x(t)$

$B(t+1) = A'(t)x(t)$

$y(t) = [A(t)+B(t)]x'(t)$

n ingressi e m stati:

➡ 2^{n+m} righe

Tabella di stato

presente		prossimo				uscite	
		x=0		x=1		x=0	x=1
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

Tabella di stato

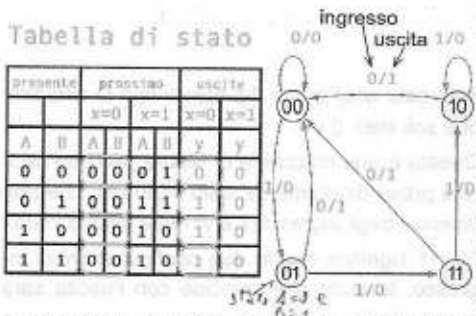
presente		prossimo				uscite	
		x=0		x=1		x=0	x=1
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

presente			prossimo		
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

E' prassi rappresentare la tabella di stato sopra nel modo a lato.

Sono rappresentati in due blocchi separati lo stato all'istante successivo (prossimo) e l'uscite nell'ingresso attuale. Si scrive lo stato A, ad es., all'istante t+1 nel caso in cui x=0 e nel caso in cui x=1.

Si separano gli stati attuali (A e B presente) da gli ingressi attuali (prossimo, x=0 e x=1). Le due tabelle sono corrispondenti ma riportano i dati in modo differente. Sarà la tabella di stato a sinistra che verrà presa in considerazione, per determinare il cosiddetto diagramma di stato.



A lato la tabella di stato con relativo diagramma di stato ("pallogramma").

In esso, nei cerchi, c'è il valore booleano dello stato, quindi 00 vuol dire A=0 e B=0. Con le frecce sono rappresentate le transizioni che avvengono.

Ad esempio lo stato presente A = 0 e B = 0 è rappresentato dal cerchio in alto a sinistra; esso passa allo stato A = 0 e B = 0, vd. prossimo, per x=0 e allo stato A=0 e B=1 per x=1: il passaggio è indicato con una freccia, a ridosso della freccia è riportato il valore dell'ingresso x a sinistra ed il valore dell'uscita y a destra, separati da /.

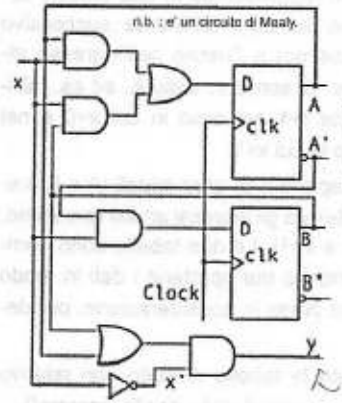
Quindi nel cerchio c'è il valore booleano dello stato, per cui lo stato 00 corrisponde ad A = 0 e B = 0, la prima riga della tabella di stato, colonna dello stato "presente". Con le frecce si rappresentano le transizioni che avvengono. Poi vengono considerate 1 a 1 tutte le possibilità nello stato "prossimo": lo stato 00 per x = 0 ha come uscita y = 0, questa informazione viene scritta come una etichetta 0/0 posta sulla freccia, per cui il primo codice 0 rappresenta lo stato prossimo e il secondo 0 rappresenta l'uscita. Questa situazione ha un senso nella macchina di Mealy.

Nel caso successivo di A = 0 e B = 0, per stato prossimo A = 0 e B = 1, con ingresso x = 1 , l'uscita è y = 0. La macchina a stati finita transita dallo stato 00 allo stato 01, quindi in presenza di un ingresso x = 1 si ha una uscita y = 0.

La freccia rappresenta il passaggio di stato, da uno stato presente ad uno stato successivo, a fronte di un ingresso che produce una uscita. E così via...

Quindi quanto visto dimostra come passare dalla tabella di stato al diagramma di stato.

Equazioni per gli ingressi dei Flip-Flop



$D_A = Ax + Bx$

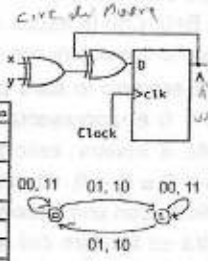
$DB = A'x$

$y = (A + B)x'$

Queste sono le equazioni di eccitazione dei Flip-Flop di tipo D.

Analisi con flip-flops D

$D_A = A \oplus x \oplus y$



presente	ingressi		prossimo
A	x	y	B
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

In questa analisi con flip-flop di tipo D abbiamo due soli stati, 0 o 1.

Questa è una macchina di Moore perché l'uscita è presa direttamente sullo stato (A), che non dipende dagli ingressi (x e y) ma è solo lo stato.

00, 11 significa che in due configurazioni di ingresso, lo stato che coincide con l'uscita sarà sempre lo 0.

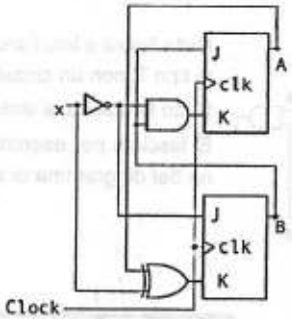
Analisi con flip-flops JK

$J_A=B$

$K_A=Bx'$

$J_B=x'$

$K_B=A'x+Ax'=A\oplus x$



Nella figura a lato l'analisi con flip-flops di tipo JK e le relative 4 equazioni caratteristiche, 2 per flip-flop.

Sotto la relativa tabella di stato e successivamente una ulteriore tabella di stato, nella forma per determinare il diagramma di stato. L'ingresso è x.

presente			prossimo		ingressi flip-flop			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

Analisi con flip-flops JK

$J_A=B$

$K_A=Bx'$

$J_B=x'$

$K_B=A\oplus x\ Ax'$

$A(t+1)=JA'+K'A$

$B(t+1)=JB'+K'B$

presente			prossimo		ingressi flip-flop			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

$J_A=B \qquad K_A=Bx' \qquad J_B=x' \qquad K_B=A\oplus x$

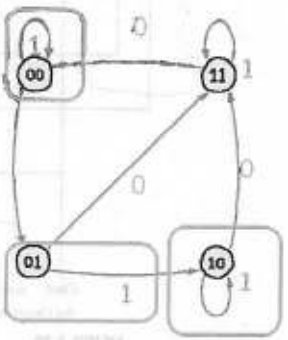
$A(t+1)=JA'+K'A=BA'+(Bx')'A=A'B+AB'+Ax$

$B(t+1)=JB'+K'B=$

$=x'B'+(A\oplus x)'B=B'x'+ABx+A'Bx'$

Di seguito il riepilogo di tutte le equazioni e lo stato successivo.

Tutto questo ci permette di realizzare il diagramma di stato.



Analisi con flip-flops T

$Q(T+1)=TQ'+T'Q=T\oplus Q$

$T_A=Bx$

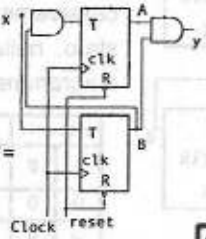
$T_B=x$

$y=AB$

$A(t+1)=(Bx)'A+(Bx)A'=$

$=AB'+Ax'+A'BX$

$B(t+1)=x\oplus B$



Nella figura a lato l'analisi con flip-flops di tipo T, con un circuito di Moore.

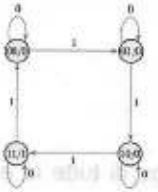
Sotto la tabella di stato.

E' lasciato per esercizio la realizzazione del diagramma di stato.

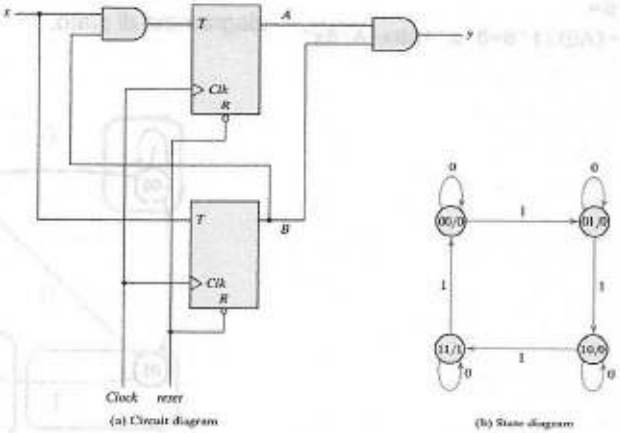
presente		in	prossimo		out
A	B	X	A	B	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

E' un contatore binario.

Il circuito è sequenziale, modello di Moore, infatti l'output dipende solo dai valori dei flip-flop e questo fa si che l'output sia funzione solo dello stato presente.



A lato il relativo diagramma di stato



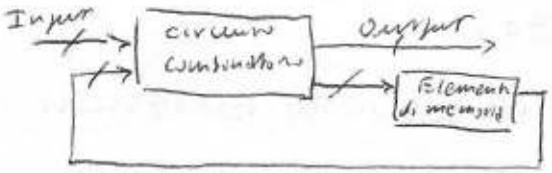
(a) Circuit diagram

(b) State diagram

FIGURE 5.20
Sequential circuit with T flip-flops (Binary Counter)

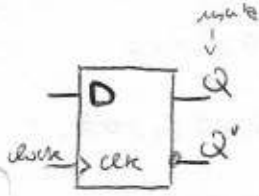
CIRCUITI SEQUENZIALI SINCRONI

I CIRCUITI SEQUENZIALI HANNO UNA RETROAZIONE DAGLI ELEMENTI DI MEMORIA (FLIP-FLOP, che determinano lo stato) VERSO IL CIRCUITO COMBINATORIO



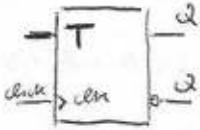
Gli elementi di memoria sono Flip-Flop.
Ogni Flip-Flop rappresenta uno stato, la transizione da uno stato all'altro avviene solo ad intervalli definiti da impulsi di clock (concetto di circuito sincrono).

LO STATO DEL FLIP-FLOP È LA SUA USCITA, ASSERITA O COMPLEMENTATA.

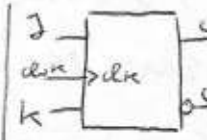


$Q(t+1) = D$

lo stato all'istante successivo è pari all'ingresso D nello stato attuale



$Q(t+1) = TQ' + T'Q =$
$= T \oplus Q$



$Q(t+1) = JQ' + K'Q$

L'ANALISI DI CIRCUITI SEQUENZIALI SINCRONI CONSISTE NELL'OTTENERE UNA TABELLA O UN DIAGRAMMA PER LA SEQUENZA TEMPORALE DEGLI INPUT, E DEGLI STATI INTERNI

È ANCHE POSSIBILE SCRIVERE ESPRESSIONI BOOLEANE CHE DESCRIVONO IL COMPORTAMENTO DEL CIRCUITO SEQUENZIALE (n.s.: detto schematico).

LA TABELLA DI STATO ENUMERA LA SEQUENZA TEMPORALE DEGLI INPUT, DEGLI OUTPUT E DEGLI STATI DEI FLIP FLOP.

ESSA CONSISTE IN 4 SEZIONI ETICHETTATE COME "STATO PRESENTE", "INPUT", "STATO SUCCESSIVO" E "OUTPUT".

LO "STATO PRESENTE" E "INPUT" VENGONO ENUMERATI PER OGNI POSSIBILE COMBINAZIONE.

LO "STATO SUCCESSIVO" È DETERMINATO IN FUNZIONE DELLO STATO DEI FLIP-FLOP AL SUCCESSIVO

CLOCK E TALE "SUCCESSIVO STATO" È DETERMINATO DALLA EQUAZIONE DI STATO.

L'OUTPUT È DETERMINATO DALLA RELATIVA EQUAZIONE DI OUTPUT.

QUALCHE VOLTA È CONVENIENTE ESPRIMERE LA TABELLA DI STATO IN UNA FORMA LEGGERMENTE DIVERSA, CON SOLE TRE SEZIONI: STATO PRESENTE, STATO PROSSIMO E OUTPUT. LE CONDIZIONI DI INPUT SONO RIVERSATE SOTTO LE SEZIONI STATO PROSSIMO E OUTPUT.

QUINDI LE DUE FORME DI TABELLA DI STATO SONO

Presente		Input	Prossimo		Output
A	B	x	A	B	Y
⋮	⋮	⋮	⋮	⋮	

oppure

Presente		Prossimo		Output	
		x=0	x=1	x=0	x=1
A	B	A B	A B	Y	Y
⋮	⋮	⋮ ⋮	⋮ ⋮	⋮	⋮

IL DIAGRAMMA DI STATO È LA RAPPRESENTAZIONE GRAFICA DELL'INFORMAZIONE DISPONIBILE NELLA TABELLA DI STATO.

UNO STATO È RAPPRESENTATO DA UN CERCHIO E LE TRANSIZIONI (AL CLOCK) FRA STATI SONO INDICATE DA LINEE ORIENTATE CHE CONNETTONO I CERCHI. IL DIAGRAMMA DI STATO FORNISCE LE STESSE INFORMAZIONI DELLA TABELLA DI STATO, DA CUI È OTTENUTO. NEL DIAGRAMMA DI STATO DEVE ESSERE INDICATO L'INPUT, L'OUTPUT E, TRAMITE LA FRECCIA, LO STATO SUCCESSIVO.

DUNQUE I PASSI LOGICI VISTI SONO:

SCHEMATICO DEL CIRCUITO SEQUENZIALE SINCRONO

↓

EQUAZIONI (Esp Booleane) oppure TABELLA di STATO

↓

DIAGRAMMA DI STATO

[Rif. to pag. 55 Mano - Ciletti pagg. 204 ÷ 209]

Segue "Analisi con Flip-Flop"

Circuiti Sequenziali Sincroni III

Prof. Romeo Beccherelli
39'50"

- Assegnazione e riduzione degli stati
- Procedura di progetto

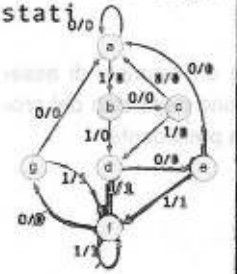
ASSEGNAZIONE E RIDUZIONE DEGLI STATI

→ Derivare un diagramma di stato per la sequenza temporale di ingressi, uscite e stati

→ Derivare una tabella di stato

→ Derivare uno schematico

Riduzione degli stati



e è equivalente a g

Stato presente	Stato prossimo		Uscita	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Da uno stato presente si va in uno stato prossimo al variare dell'ingresso, con un certo valore di uscita.

Il diagramma di stato è un grafo riporta graficamente questo riportando tutte le informazioni: una freccia è un passaggio di stato, a fianco di essa viene riportato l'ingresso e l'uscita nella forma i/o.

Si noti che per i due stati presenti "e" e "g" i due stati prossimi relativi e le uscite relative sono uguali, in corrispondenza degli stessi ingressi: lo stato e è detto equivalente allo stato g, quindi uno dei due può essere rimosso. Questo, in linea di principio, può portare a ridurre il numero di Flip-Flop nel circuito e quindi il costo del circuito.

Dunque, scegliendo di rimuovere g, l'ultima riga della tabella di stato viene cancellata e il pallotto g nel diagramma di stato viene etichettato con e, per cui abbiamo due stati "e" nel diagramma di stato. Nella tabella di stato si sostituiscono gli stati g con gli stati "e".

Fatto questo ci accorgiamo che lo stato d ed f sono equivalenti e si fa la stessa operazione, scegliendo di togliere f per cui tutte le occorrenze di f nelle righe rimanenti della tabella di stato vengono sostituite con d come pure nel diagramma di stato.

Il diagramma di stato risultante può essere semplificato, a partire dalla "nuova" tabella di stato.

Stati equivalenti

→ Due stati sono equivalenti se producono la stessa uscita per uguale eccitazione di ingresso e se portano il circuito nello stesso stato successivo

→ Uno dei due stati può essere rimosso

Assegnazione degli stati

→ Per m stati, i codici devono contenere n bit, con $2^n < m$

→ assegnazione semplice: codice binario

→ assegnazione più robusta: codice di Gray

→ decodifica più semplice: codice "one-hot"

→ Il codice one-hot comporta aumento del numero di flip-flop ma consente una logica di decodifica più semplice

→ Non sempre la minimizzazione degli stati determina il circuito con il costo minore

Assegnazione degli stati

Stato	Binario	Gray	One-hot
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	110	10000

A lato un esempio di assegnazione degli stati del problema precedente.

PROCEDURA DI PROGETTO

1. Ottenere le specifiche del circuito desiderato (in termini di costo, dissipazione di potenza, ecc.)
2. Derivare gli stati per la macchina scegliendo uno stato iniziale (es: quello successivo al reset, che porta ad uno stato ben noto, lo stato di reset. Il reset è un impulso che resetta tutti i flip-flop. E' asincrono rispetto al clock)
   2.1. Considerare tutte le valutazioni degli ingressi al circuito
   2.2. Creare i nuovi stati necessari affinché la macchina possa rispondere agli ingressi
3. Partendo dal diagramma di stato, creare una tabella di stato
4. Decidere quante variabili di stato sono necessarie per rappresentare tutti gli stati
   4.1. Assegnate gli stati

4.2. Minimizzare il numero degli stati

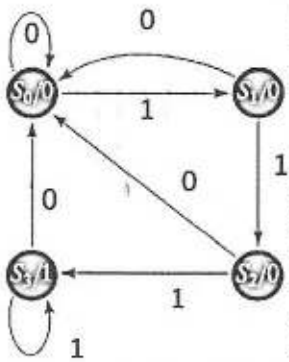
5. Scegliere il tipo di flip-flops da usarsi nel circuito

5.1. Derivare l'equazione di ingresso dei flip-flops che produca gli stati desiderati

5.2. Derivare una espressione logica per le uscite del circuito (in funzione degli stati, se si tratta di una macchina di Moore, in funzione degli stati e degli ingressi, se si tratta di una macchina di Mealy)

6. Disegnare il circuito schematico come indicato nell'espressione logica

7. Designare una forma d'onda di eccitazione per la verifica e collaudo del comportamento del circuito. Forma d'onda per ognuno degli ingressi e per ogni combinazione degli ingressi per verificare il corretto funzionamento del circuito. Si tratta di realizzare dei test bench in cui le forme d'onda immesse nel circuito possono anche essere simulate.



Identificatore di sequenza

→ Disegnare un circuito che identifichi la presenza di tre o più 1 da un flusso seriale di bit (specifica del problema)

A lato il diagramma di stato, con uno stato S_0 pensabile come uno stato iniziale che si verifica dopo il reset.

Se l'input è 0 si ritorna in questo stato e l'uscita è 0, ovvero non c'è stata una sequenza di 3 bit a 1 (o più di tre).

Se l'input è 1 si passa allo stato S_1 in cui l'uscita è 0, perché l'input non è quello della specifica. Il comportamento è chiaro.

In S_3 siamo nello stato finale in cui l'uscita è 1 perché ci sono stati 3 o più bit 1 in sequenza. Se in S_3 si ha input 1 si rimane in S_3 con uscita 1, se si ha input 0 si va in S_0 con uscita 0.

Si noti come nell'etichetta siano stati messi il nome dello stato e l'uscita, mentre sulla linea di transizione è riportato l'ingresso, in maniera diversa vista precedentemente: tale situazione è equivalente alla precedente ed in letteratura si trovano entrambi le metodologie.

Spunti di riflessione

Assegnate una sequenza binaria per gli stati dell'identificatore di sequenza.

Sintetizzate un circuito prima con flip-flop di tipo D e poi di tipo T.

Ora usate un codice one-hot e ripetete l'esercizio.

Identificate quale delle tre implementazione ha il minor costo.

PROCEDURA di PROGETTO Circuiti Sequenziali Sincroni e ANALISI

1. Ricavare un diagramma di stato del circuito (palloncini)
2. Ridurre eventualmente gli stati
3. Assegnare valori binari agli stati
4. Ricavare la tabella di stato
5. Scegliere il tipo di flip-flop da usare
6. Ricavare le equazioni semplificate di input dei flip-flop e di output (ovvero il next state)
7. Disegnare lo schema logico.

- La tabella di stato, si presenta nella forma seguente, per tutti i casi, in cui ogni flip-flop scelto

stato presente				Input		stato successivo			
A	B	C	....	x	y	....	A	B	C	....
0	0	0		0	0					
0	0	0		0	1					
0										
0										
...										
1	1	1		1	1					

tutte le combinazioni possibili.

I valori in funzione dei valori dello stato presente e degli input

parte che cambia, a seconda del tipo di f.f. scelto

Il n. dei f.f. è dato dal n. degli stati:

2 f.f. => 2^2 stati

3 f.f. => 2^3 stati

Questa situazione è comune a tutti i casi, ovvero qualunque sia il tipo di f.f. scelto

A seconda del tipo di flip-flop scelto, per realizzare la "memoria" del circuito, considera la restante parte della tabella di stato: per flip-flop di tipo D si ha una parte di output, con una o più variabili, w, z..., con valori in funzione dello stato presente e degli input. Per flip-flop di tipo JK oppure T, si ha una parte di input dei flip-flop. Per il JK si hanno due input (J e K) per ogni flip-flop necessario; per il T ne ha un input per ogni flip-flop necessario. I valori assunti da queste colonne sono funzione dello stato presente e dello stato futuro in relazione alla tabella di eccitazione del flip-flop usato.

Quindi

Output

w	z
0	...
0	...
1	...
...	...

per f.f. di tipo D

e Input F.F.

$J_{A,B,...}$	$K_{A,B,...}$
0,1,X	0,1,X

per f.f. di tipo JK

e Input F.F.

$T_{A,B,...}$
0,1

per f.f. di tipo T.

Il passo successivo è di ottenere le equazioni booleane delle uscite e degli ingressi dei flip-flop per flip-flop di tipo D e dei soli ingressi dei flip-flop per flip-flop di tipo JK e T.

Si vogliono ottenere dunque equazioni del tipo

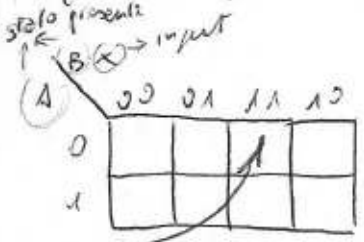
$D_A = Ax + Bx$ $\quad D_B = Ax + B'x$ $\quad J = AB$

↓ Ingresso del F.F. di nome A, di tipo D

↓ Output

Questo per flip-flop di tipo D.

Si ottengono con l'uso delle mappe di Karnaugh, dove sugli assi si mettono A, B ecc. dello stato presente e gli input, e nelle celle si mettono gli 1 delle colonne A, B, ecc. dello stato successivo e gli output.



→ Tante mappe quante sono i F.F. e le uscite. Si devono ricavare D_A, D_B, ..., J ecc.

l'1 di A, B stato succ., oppure Y, rispettivamente per D_A, D_B e Y

A questo punto, data la equazione, è possibile disegnare lo schema, il che conclude l'analisi

Per flip-flop di tipo JK o T vanno ugualmente determinate le equazioni booleane, ma dei J e dei K in un caso, e dei T nell'altro, ovvero equazioni del tipo

$J_A = B x'$ $\quad K_A = B x$
$J_B = x$ $\quad K_B = (A \oplus x)'$

F.F. JK

oppure

$T_A = A_1 A_0$
$T_B = A_0$
$T_C = 1$

F.F. T

Ottenute queste, tramite le mappe di Karnaugh, con le variabili dello stato presente ed eventualmente gli input sugli assi, e con i valori degli input dei flip-flop nelle celle, si può disegnare lo schema, concludendo l'esercizio.

LEZIONE 15

Registri e Contatori

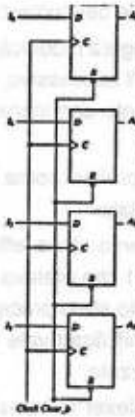
circuiti sequenziali sincroni

Prof. Romeo Beccherelli
42'00"

- Registri
- Registri a scorrimento
- Contatori "ripple" (ad ondulazione)
- Contatori sincroni
- Altri contatori

REGISTRI

Semplice registro a quattro bit

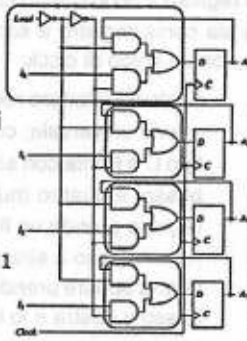


A lato un semplice registro a quattro bit, formato da 4 flip-flop di tipo D commutati a fronte positivo, ogni flip-flop prende un input I, c'è un reset asincrono attivo basso e ci sono delle uscite.

Questo registro è dunque un insieme di flip-flop di tipo D pilotati con uno stesso clock ed uno stesso segnale di reset = clear.

Registro a quattro bit con comando di caricamento parallelo

Multiplexer 2 a 1 quadruplo



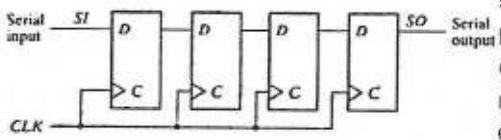
Sotto un ulteriore registro a quattro bit, con comando di caricamento parallelo. In esso si noti come il dato in uscita dal flip-flop, lo stato Q, esce e viene rifatto ricircolare attraverso un AND.

Osserviamo che il blocco evidenziato si comporta sostanzialmente da selettore in cui quando il segnale load è 1 fa passare l'ingresso, quando è 0 fa ricircolare l'uscita.

Dunque tale parte combinatoria del circuito decide se viene caricato l'ingresso o se ricicla lo stato in uscita. Quindi il comportamento è quello di un multiplexer dove il segnale di selezione è il segnale load.

Questo vale per gli altri blocchi (evidenziati) per cui si ha un multiplexer 2 a 1 quadruplo.

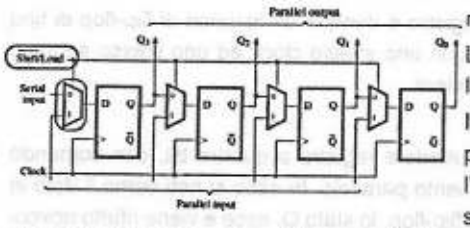
Registro a scorrimento elementare



Un semplice registro a scorrimento è costituito da un ingresso seriale (SI), un segnale di clock (CLK) che carica flip-flop di tipo D commutati a fronte positivo, carica l'uscita del flip-flop che lo precede.

Qualunque dato al primo clock si trova nel primo flip-flop, al secondo nel secondo flip-flop ecc. Dopo quattro clock si trova alla fine. Il flusso seriale dei bit procede sincrono con il clock. I flip-flop sono considerabili come elementi di ritardi. Il ritardo è contabile dal numero di clock.

Registro a scorrimento con ingresso ed uscita paralleli



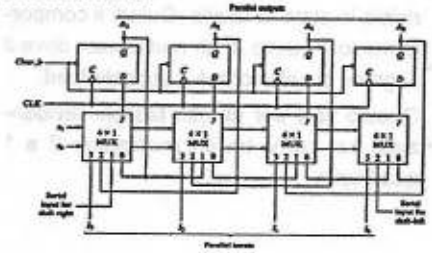
Da una leggera modifica di questo circuito otteniamo il successivo, che è un registro a scorrimento con ingresso ed uscita paralleli.

L'uscita si preleva come è evidenziato dalla figura a lato.

Il caricamento viene effettuato con multiplexer 2 a 1 che preleva l'ingresso oppure l'uscita dello stato precedente. Quando il segnale shift'/load vale 0 (shift = 0) il segnale viaggia come nel registro precedentemente analizzato.

Quando il segnale shift'/load vale 1 (load = 1) il multiplexer fa transitare il segnale di ingresso che viene caricato. Quindi questo registro a scorrimento si può comportare sia come registro a scorrimento verso l'uscita sia come registro a scorrimento con uscita parallela. Il dato può cioè scorrere o uscire con un colpo di clock.

Registro a scorrimento universale



A lato un ulteriore registro, detto a scorrimento universale, con quattro flip-flop di tipo D a fronte con segnale di clock attivo basso. e quattro multiplexer 4 in 1. Tale registro prende un flusso seriale dalla linea in basso a sinistra e lo fa scorrere a destra oppure prende un flusso seriale in basso a destra e lo trasla verso sinistra.

Se la codifica è 1 facciamo lo shift right, se la codifica è 2 facciamo lo shift left. Se la codifica è 0 ricicliamo il dato dall'uscita e lo rimettiamo nuovamente in ingresso.

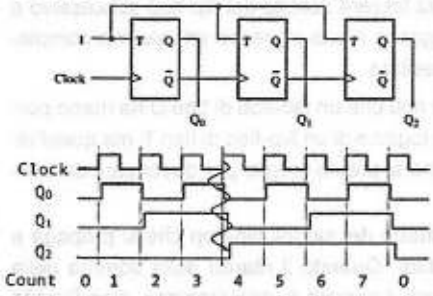
è un convertitore serie-parallelo bidirezionale

Se $S_1 = 1$ e $S_0 = 1$, cioè la codifica è il codice 3, saranno caricati i dati provenienti dalle

quattro linee in basso, che sono dati in parallelo. In sostanza si tratta di un convertitore serie-parallelo bidirezionale.

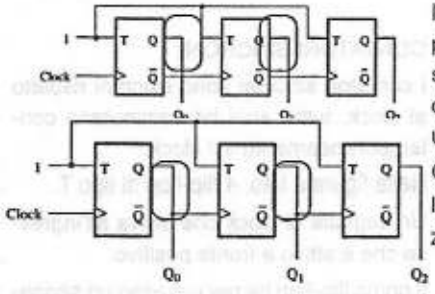
CONTATORI "RIPPLE" (AD ONDULAZIONE)

Contatore ad incremento (up-counter) con flip-flops T



In figura a lato 3 flip-flop di tipo T messi in cascata. All'ingresso del primo flip-flop c'è un 1 per cui esso commuterà ad ogni transizione positiva del clock e tutti e 3 i flip-flop commuteranno ad ogni transizione positiva del clock (up -counter).

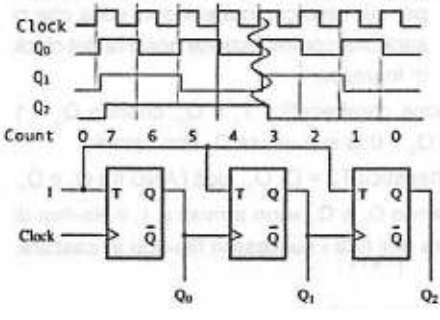
Si noti lo schema temporale, dal quale si nota come questo dispositivo conta da 0 a 7 ed è di norma definito contatore modulo 8.

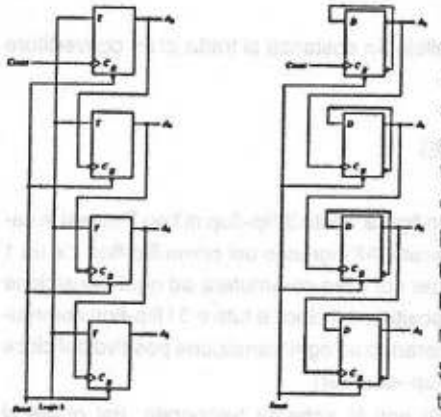


Nella figura a lato, partendo dal contatore sopra, ne analizziamo uno leggermente diverso., che è quello che permette di contare per decrementi. Si noti come sono collegate le uscite.

Questo contatore è un down counter, si noti lo schema temporale che dimostra il suo funzionamento.

Down counter con flip-flops T





Nella figura a lato un contatore che oltre a flip-flop di tipo T (a sinistra, ripple counter, con transizione a ogni fronte negativo del suo ingresso e forzerà il successivo flip-flop a commutare, ovvero a contare) anche con flip-flop di tipo D (a destra, che lavora anch'esso sul fronte negativo del clock).

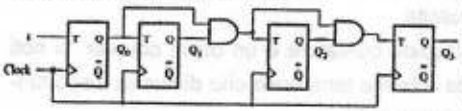
Nei flip-flop di tipo D ogni transizione negativa forzerà l'uscita del flip-flop successivo a leggere il suo ingresso attuale ma complementato.

Si noti che un flip-flop di tipo D ha meno porte logiche di un flip-flop di tipo T, ma quest'ultimo si presta meglio per descrivere un contatore ripple, ad ondulazione.

I contatori ad ondulazione sono limitati dal ritardo dei singoli flip-flop che si propaga e si somma all'aumentare del numero degli stadi. Quando il ritardo della somma della propagazione è maggiore del periodo del clock il circuito si disincronizza, cioè l'uscita è sfasata di un periodo rispetto all'ingresso. Questo non è tollerabile in un circuito sequenziale. In molte situazione non è tollerabile anche che sia sfasato di mezzo periodo.

Per ovviare a questo problema si introducono i contatori sincroni.

Contatore sincrono ad incremento con 4 bit



- $T_0=1$
- $T_1=Q_0$
- $T_2=Q_0Q_1$
- ....
- $T_n=Q_0Q_1...Q_{n-1}$

CONTATORI SINCRONI

I contatori sincroni sono sincroni rispetto al clock, tutti i suoi bit commutano contemporaneamente sul clock.

Nella figura a lato, 4 flip-flop di tipo T.

Un segnale di clock che arriva all'ingresso che è attivo a fronte positivo.

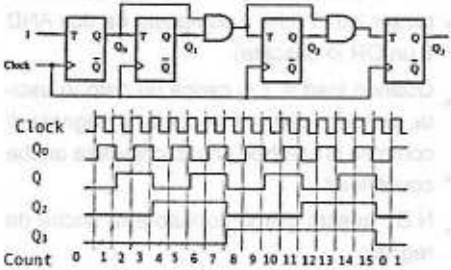
Il primo flip-flop ha per ingresso un segnale di controllo che è una costante pari a 1 per cui esso commuterà ogni volta che ci sarà una commutazione positiva del clock in ingresso.

Il secondo flip-flop, T_1, che ha come equazione caratteristica $T_1 = Q_0$, quando $Q_0 = 1$ succede che Q_1 commuterà, mentre quando $Q_0 = 0$ la sua uscita Q_1 non cambia.

Il terzo flip-flop, T2, ha come equazione caratteristica $T2 = Q_0 Q_1$, cioè l'AND fra Q_0 e Q_1.

Il contatore conta quando c'è un clock e, quando Q_0 e Q_1 sono arrivati a 1, il flip-flop di ordine 3, ovvero T_2, commuterà; questo varrà per tutti i successivi flip-flop in cascata.

Contatore sincrono ad incremento con 4 bit



che dovranno commutare quando i precedenti sono a 1.

Quindi Q_3 diventerà 1 quando sia Q_0, Q_1 e Q_2 sono 1.

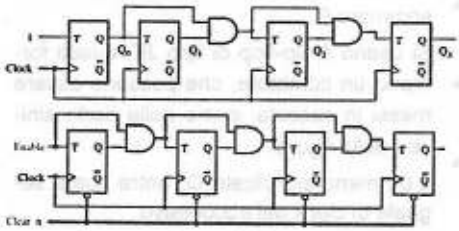
Nella figura a lato il comportamento delle uscite Q_i in funzione del clock e relativo valore di conteggio.

Si noti che in questo caso, rispetto al contatore ripple, non c'è uno sfasamento in quanto tutti i flip-flop hanno lo stesso identico segnale di sincronizzazione. L'unico vincolo temporale è quello per cui non deve essere troppo lungo il ritardo introdotto dalle porte AND; comunque sia, l'uscita avviene sempre in modo sincrono col clock.

Il ritardo delle uscite Q_i è lo stesso, esse sono isocrone al clock a meno del tempo di propagazione di ognuno dei flip-flop, che si assume essere sostanzialmente uguale fra tutti.

I ritardi non si sommano come avveniva per il ripple counter.

Contatore sincrono ad incremento con 4 bit con clear ed enable

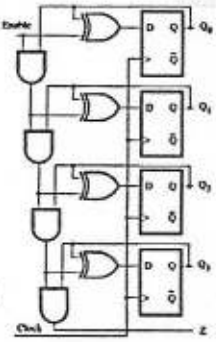


Nella figura a lato una variante del contatore precedente, con segnale di clear, che fa ottenere il risultato di reset dei flip-flop. Inoltre, invece di avere un segnale fisso a 1 si introduce un segnale di enable, cioè di abilitazione all'ingresso.

Quindi se enable = 1 il contatore conta, se enable = 0 il contatore non conta e rimane nello stato in cui è.

Contatore sincrono ad incremento con 4 bit con flip-flop D

- $D_0=Q_0 \oplus 1=Q_0'$
- $D_1=Q_1 \oplus Q_0$
- $D_2=Q_2 \oplus Q_1Q_0$
- $D_3=Q_3 \oplus Q_2Q_1Q_0$
- ...
- $D_i=Q_i \oplus Q_{i-1}Q_{i-2}..{}_2Q_1Q_0$
- ....



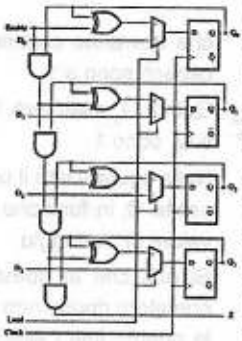
Nella figura a lato una ulteriore variante, in cui il contatore è realizzato con flip-flop di tipo D, realizzabili fisicamente in modo più facile.

Abbiamo quattro flip-flop attivi a fronte positivo con uno XOR per ogni flip-flop.

Il comportamento è ricavabile dalle equazioni di stato riportate in figura.

Costruendo il grafico temporale, si realizza visivamente il fatto che questo circuito è un contatore sincrono ad incremento.

Contatore sincrono ad incremento con 4 bit con flip-flop D e parallelo e caricamento parallelo



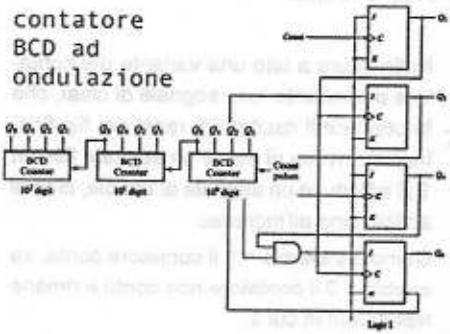
In figura a lato una variante del precedente contatore; questo contatore ha un caricamento parallelo ed è realizzato con multiplexer 2 in 1 (che è composto da due AND e un OR in cascata).

Quando load = 1 si carica un dato in uscita; quando load = 1 si conta. Il segnale di controllo si sarebbe potuto chiamare anche count'/load.

N.B.: questo ultimo contatore fa anche da registro.

ALTRI TIPI DI CONTATORE

contatore BCD ad ondulazione



Il contatore BCD ad ondulazione: per contare in BCD occorrono 4 bit.

Facendo la somma in decimale occorre sommare 6.

Si usano 4 flip-flop di tipo JK, questi formano un contatore, che possono essere messi in cascata, come nella parte sinistra della figura.

Il bit meno significato Q_0 entra come segnale di clock del successivo.

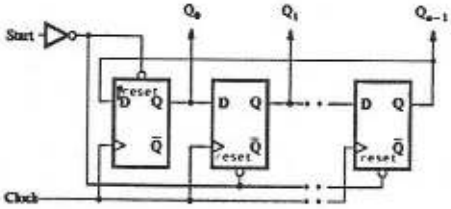
Per esercizio è richiesto:

. ricavare l'espressione booleana del contatore.

. ricavare la tabella degli stati.

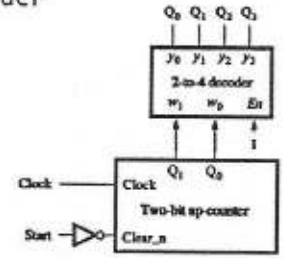
. ricavare il diagramma di stato.

Contatore ad anello



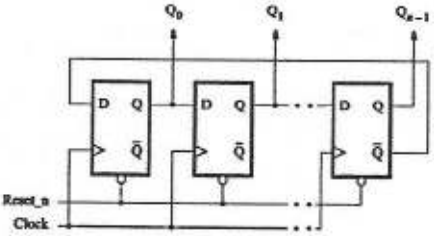
A lato il contatore ad anello, di semplice realizzazione, con un comando di start. Reset e preset sono attivi bassi.

Contatore ad anello con decoder



A lato un contatore ad anello con due soli bit, invece di quattro, ed un decoder 2 in 4.

Contatore di Johnson



A lato un contatore di Johnson, che è un contatore ad anello visto inizialmente in cui facciamo riciclare l'uscita complementata per cui il contatore ha un numero di bit pari al doppio del numero di flip-flop.

Capitolo 7

Memorie e logica programmabile

7.1 Introduzione

Una unità di memoria è un dispositivo sul quale l'informazione binaria è trasferita per la memorizzazione e dal quale l'informazione è ricavata quando c'è la necessità di elaborarla.

Quando l'elaborazione dei dati comincia, l'informazione è trasferita dalla memoria a dei registi selezionati nella unità di elaborazione.

I risultati intermedi e finali ottenuti nell'unità di elaborazione sono ritrasferiti per essere memorizzati nella memoria. L'informazione binaria ricevuta da un dispositivo di input è memorizzata nella memoria, e l'informazione trasferita a un dispositivo di output è presa dalla memoria. Una unità di memoria è una collezione di celle capaci di memorizzare una grande quantità di informazioni binarie.

Ci sono due tipi di memorie che sono usati nei sistemi digitali: random access memory (RAM) e read only memory (ROM). La RAM memorizza le nuove informazioni per usarle in un secondo tempo. Il processo di memorizzare nuove informazioni nella memoria è indicato come una operazione di scrittura nella memoria. Il processo di trasferimento dell'informazione memorizzata fuori dalla memoria è indicato come una operazione di lettura della memoria. La RAM può eseguire sia operazioni di lettura sia operazioni di scrittura. La ROM può eseguire solo l'operazione di lettura. Questo vuol dire che l'informazione binaria adatta è già memorizzata all'interno della memoria e può essere acquisita o letta in ogni momento. Comunque quella informazione non può essere alterata da una scrittura. La ROM è un dispositivo logico programmabile (PLD). L'informazione binaria che è memorizzata in tale dispositivo è specificata in qualche modo e incorporata nell'hardware in un processo definito come programmazione del dispositivo. La parola "programmazione" si riferisce in questo caso alla procedura hardware che specifica che i bit sono inseriti in una configurazione hardware del dispositivo.

La ROM è un esempio di un PLD. Altri tipi di dispositivi come questi sono gli array logici programmabili (PLA, programmable logic array), gli array a porte a campo programmabile (FPGA, field-programmable gate array). Un PLD è un circuito integrato con porte logiche interne connesse attraverso percorsi elettronici che si comportano in maniera simile ai fusibili. Nello stato iniziale del dispositivo, tutti i fusibili sono intatti. La programmazione del dispositivo coinvolge operazione di bruciare quei fusibili che si trovano lungo i percorsi che devono essere rimossi per ottenere una particolare configurazione della funzione logica desiderata.

7.2 Random-Access Memory - RAM, Memoria ad Accesso Casuale

Una unità di memoria è una collezione di celle di memorizzazione, insieme a circuiti che servono a trasferire informazioni verso il dispositivo e fuori del dispositivo. L'architettura della memoria è tale per cui l'informazione può essere selettivamente ricavata da ognuna delle sue locazioni interne. Il tempo che ci vuole per trasferire le informazioni verso o da ogni desiderata locazione casuale è sempre lo stesso, e da questo ne deriva il nome del tipo di memoria. Viceversa, il tempo necessario per ricavare un'informazione memorizzata, ad esempio, su un nastro magnetico dipende dalla posizione del dato.

Una unità di memoria memorizza informazioni binarie in gruppi di bits chiamati parole (words). Una parola in memoria è una entità di bit che si muovono da e verso il dispositivo come una unità. Una parola in memoria è un gruppo di 1 o 0 e può rappresentare un numero, una istruzione, uno o più caratteri alfanumerici, oppure ogni altra informazione codificata in binario. Un gruppo di 8 bit è chiamato 1 byte. La maggior parte delle memoria di computer usano parole che sono multipli di 8 bit, pertanto una parola a 16 bit contiene 2 byte, e una parola a 32 bit è fatta di 4 byte la capacità di una unità di memoria è normalmente riportata come il numero totale di byte che l'unità può memorizzare.

La comunicazione tra la memoria e il suo ambiente avviene attraverso linee di dati di input e di output, linee di selezione degli indirizzi, e linee di controllo che specificano la direzione del trasferimento. Un diagramma a blocchi di una unità di memoria è mostrata in figura. Le n linee di dati in input forniscono all'informazione da memorizzare in memoria, e le n linee di dati in output forniscono l'informazione che esce dalla memoria. Le k linee di indirizzo specificano la particolare parola scelta fra quelle disponibili. I due input di controllo specificano la direzione desiderata per il trasferimento: l'input di tipo Write fa sì che il dato binario sia trasferito nella memoria, e l'input di tipo Read fa si che il dato binario sia trasferito dalla memoria.

L'unità di memoria è specificata dal numero di parole che contiene e dal numero di bit in ogni parola. Le linee di indirizzo selezionano una particolare parola. Ogni parola in memoria a un numero di identificazione assegnato, chiamato indirizzo, che fa parte a zero fino a 2^k - 1, dove k è il numero di linee di indirizzo. La selezione di una parola specifica nella memoria è fatta applicando l'indirizzo a k bit alle linee di indirizzo. Un decodificatore interno accetta questo indirizzo e apre i percorsi necessari a selezionare la parola specificata.

Operazioni di scrittura e lettura

Le due operazioni che una RAM può fare sono di scrittura e lettura. Come detto prima il segnale di scrittura specifica una operazione trasferimento verso il dispositivo e il segnale di lettura specifica operazione di trasferimento dal dispositivo verso l'esterno. Nell'accettare uno di questi segnali di controllo, i circuiti interni alla memoria forniscono l'operazione desiderata.
I passi che devono essere fatti per trasferire una nuova parola da memorizzare nella memoria sono:

1. Applicare l'indirizzo binario della parola desiderata alle linee di indirizzo.
2. Applicare i bit di dati che devono essere memorizzati in memoria alle linee dei dati in input.
3. Attivare l'input è di tipo write.

L'unità di memoria prenderà i bit dalle linee di input dei dati e le memorizzerà nella parola specificata dalle linee di indirizzo.
I passi da eseguire per trasferire una parola fuori dalla memoria sono:

1. Applicare l'indirizzo binario alla parola desiderata alle linee di indirizzo.
2. Attivare l'input è di tipo read.

L'unità di memoria prenderà i bit dalla parola è stata selezionata dall'indirizzo e li applicherà alle linee di output dei dati. Il contenuto della parola selezionata non cambierà dopo l'operazione di lettura.

Memorie

Prof. Romeo Beccherelli
38'20"

- Memorie a semiconduttore ad accesso casuale (Semiconductor Random-Access memories - RAMs)
- RAM statiche (SRAM)
- RAM dinamiche (DRAM)
- DRAM sincrone (SDRAMs)
- Double data rate SDRAMs (DDR), variante delle SDRAMs

MEMORIE AD ACCESSO CASUALE (RAM)

Un insieme organizzato di celle di immagazzinamento indirizzate da linee di indirizzo cui si accede in lettura o lettura/scrittura su linee di dati

La definizione di memorie ad accesso casuale (RAM) è:

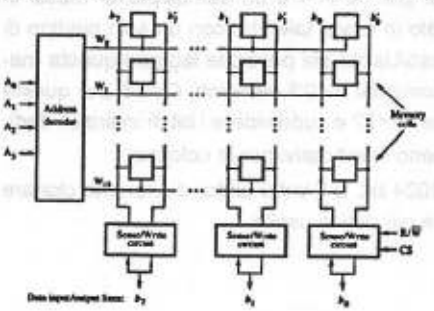
un insieme organizzato di celle di immagazzinamento indirizzate da linee di indirizzo cui si accede in lettura o lettura/scrittura su linee di dati.

A lato una immagine rappresentativa, sotto una immagine di come può essere organizzata una memoria, in cui ci sono delle linee di indirizzi (4, a sinistra) e c'è un decodificatore degli indirizzi che è un decodificatore 4 in 16. Quindi sono prodotti 16 segnali che corrispondono alla decodifica dei 4 bit, da 0 a 15. Questi identificano le parole (w_0 ... w_{15}), ogni parola è costituita da 8 bit (b_0 ... b7) e questi 8 bit sono gli elementi di immagazzinamento. Notare che per ogni bit ci sono due linee, una asserita ed una negata.

Quindi si hanno 16x8 bit.

Queste celle sono celle di bit singoli, organizzate in matrici; le righe individuano e selezionano le parole (word); le colonne sono connesse a linee di bit, in questo caso 2 linee.

Organizzazione della memoria



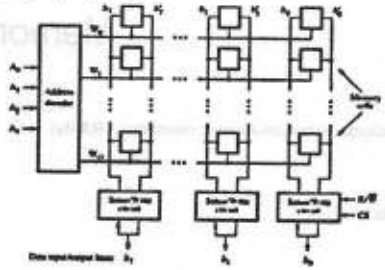
Il circuito di lettura/scrittura (Sense/Write) sono interfacce fra linee di bit interne e i piedini di I/O. Notare che Sense da la misura per leggere, "sentire", il contenuto delle celle.

I controlli Read/Write e chip select (CS) sono comuni ai circuiti di lettura/scrittura.

Indicativamente una lunghezza di parola di una RAM può andare da 1 a tipicamente 16 bit.

In questo caso notare che il decoder abilita una delle celle. Infatti un decoder seleziona

16 x 8 bit



le linee di word da un indirizzo a 4 bit (in questo caso).

Ci sono, in questo caso, due linee di bit complementari fra loro per ogni bit di bit.

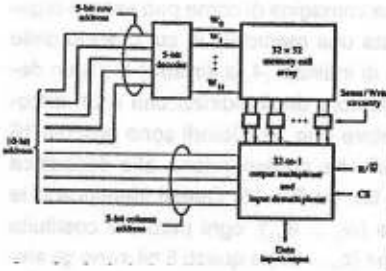
I dati provengono da una sorgente esterna che asserisce anche Chip Select e Read/Write.

Per operazioni di lettura, i circuiti Sense/Write trasferiscono i dati dai piedini di I/O alle celle selezionate.

Quindi, in questo esempio di un chip 16x8, esso ha 128 celle e richiede 14 piedini (4 per gli indirizzi, 8 per i dati, 2 per i controlli e 1 per Vdd e 1 per GND).

Se volessimo una organizzazione di 1024x1 bit, ovvero 1024 celle, quindi 128 parole, esse sarebbero organizzate come 128×8 (7+8+2+2=19 piedini) o 1024×1 (10+1+2+2=15 piedini).

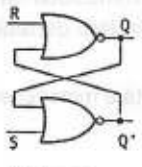
Il numero dei piedini aumenta velocità e costi, per bassi costi considerate 1024×1.



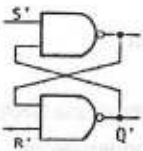
Di seguito un metodo per semplificare la struttura interna nella figura a lato, in cui ci sono 10 bit di indirizzo, essi vengono separati in 5 bit di righe e 5 bit di colonna. In questo modo abbiamo una matrice 32x32. Dai circuiti di Sense si usa un multiplexer per 32 in 1 e un demultiplexer messi in parallelo in modo tale che con un solo piedino di Ingresso/Uscita sia possibile leggere questa matrice costituita da 2^{10} elementi. Quindi per questa soluzione è stato deciso di usare una matrice 32×32 e suddividere i bit di indirizzo dedicando i 5 più significativi per le righe e i 5 meno significativi per le colonne.

Su questa base, se volessimo passare da 1024 bit, a 8 volte tanto, dovremmo clonare questo sistema e quindi avere più memoria e più dati in uscita.

I LATCH



S	R	Q	Q'	
0	0	Q	Q'	nessun cambiamento
0	1	0	1	
1	0	1	0	
1	1	0	0	non determinato



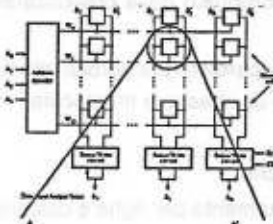
S'	R'	Q	Q'	
1	1	Q	Q'	nessun cambiamento
0	1	1	0	
1	0	0	1	
0	0	1	1	non determinato

I Latch

Si ricordano i latch implementati con dei NOR, in cui l'1 è forzante e una implementazione in NAND in cui è forzante lo 0.

Si noti che nella implementazione con il NAND il controllo è attivo basso (S' e R'), questo per una rappresentazione simmetrica rispetto all'implementazione con il NOR.

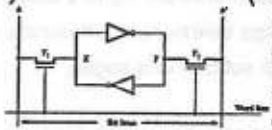
Una cella di RAM statica



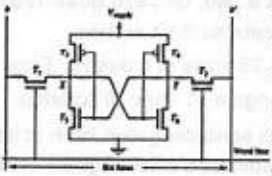
Gli elementi della matrice di memoria vista in precedenza possono benissimo essere degli SR Latch. Gli sR LAtch sono realizzati con 8 transistori, 4 di tipo p e 4 di tipo n, indipendentemente dalla loro realizzazione in NAND o NOR, in cui cambia la configurazione.

La rappresentazione può essere semplificata, come nell'esploso in figura, in cui, invece di realizzare 2 NAND e 2 NOR, sono stati realizzati due invertitori. Ogni invertitore ha bisogno di due soli transistori, uno p e uno n. Inoltre ci sono 2 pass transistors, T_1 e T_2 che abilitano la scrittura. Arbitrariamente viene definito il valore logico 1 come x = 1 AND y = 0.

Definiamo il valore logico 1 come x=1 AND Y=0

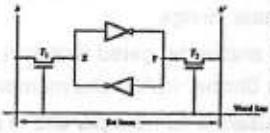


Una cella di RAM statica implementata in CMOS



Questa tecnologia è realizzata in CMOS, come in figura a lato, in cui sono evidenti i 6 transistori ottenuti dalla semplificazione per cui ne sono stati risparmiati 2, pari al 25%.

Definiamo il valore logico 1 come x 1 AND Y=0



Il costo deriva dal fatto che dobbiamo pilotare entrambi i segnali e non uno solo come nel caso dei NOR o dei NAND. Quindi servono due piste, ma queste piste che portano i segnali di bit e di bit negato sono comuni per lo stesso ordine di bit di tutte le parole.

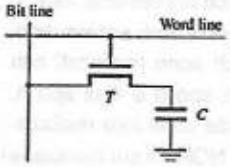
RAM DINAMICHE (DRAM)

→ Le RAM statiche hanno tempi di accesso corti, perché il dato è immediatamente disponibile, ma necessitano di numerosi transistori per cella, quindi la loro densità è ridotta

→ Le RAM dinamiche sono più semplici e consentono maggiore densità e minor costo, ma hanno tempi di accesso più lunghi

→ I vantaggi per densità e costo spesso compensano la lentezza

→ Le RAM dinamiche sono ampiamente usate nei calcolatori elettronici



Nella figura a lato la rappresentazione di una cella di una DRAM.

→ La cella è costituita da un transistore e un condensatore;

→ Lo stato è individuato dalla presenza/assenza di carica sul condensatore;

→ La carica si disperde (o per perdite interne o per perdite attraverso la porta) e deve essere rinfrescata.

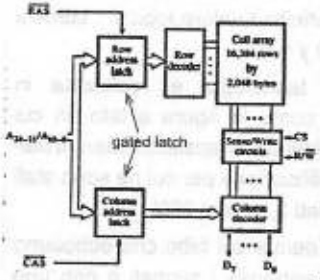
Il funzionamento di chip di RAM dinamiche

→ Basato sui principi generali di indirizzamento per righe e colonne;

→ In lettura, la carica della cella sulla riga selezionata è misurata dai circuiti di "sense";

→ 1 o 0 se la carica è al di sopra o al di sotto di una soglia;

32M × 8 DRAM



In figura a lato, un caso quasi reale, una DRAM da 32M organizzata su 8 bit di dato.

Ci sono 25 linee di indirizzo. Esse sono suddivise in 14 linee di riga e 11 linee di colonna.

A valle ci sono dei gated latch utilizzati per memorizzare questi indirizzi c'è un segnale di "Raw Address Strobe" (RAS) che definisce sostanzialmente il campionamento del segnale di riga.

Ci sono anche dei gated lach con un segnale "Column Address Strobe" (CAS) che memorizza il dato.

Questo significa che è stato deciso di usare il numero più alto di piedini (15) per multiplare i segnali di indirizzamento di righe e colonna e memorizzarli. Quindi, in due istanti successivi verranno scritti gli indirizzi di riga e gli indirizzi di colonna.

C'è poi un decoder di riga (Row decoder) e l'insieme delle celle.

C'è un circuito di Sense e Write su cui intervengono dei Chip Select che abilitano questo particolare circuito integrato e dei Read/Write.

Infine c'è un decodificatore di colonna (stessa funzione svolta dal multiplexer).
Abbiamo quindi

→ 16384 righe organizzate come 2048 byte;

→ 14 bit per selezionare le righe, 11 bit nella riga;

(14 bit = 16384 righe; 11 bit = 2048 byte per riga)

→ Per risparmiare pin, si multiplano righe e colonne sugli stessi piedini (si utilizzano 14 pin per le righe e soltanto 11 per le colone, con un risparmio di 3 piedini non utilizzati dalle colonne);

→ I (gated) latch di riga/colonna memorizzano i bit di indirizzo; cioè memorizzano al comando di un segnale RAS e di un segnale CAS rispettivamente le righe e le colonne e quindi gli indirizzi di riga vengono messi per primi, quelli di colonna per secondi; il segnale RAS memorizza gli indirizzi di riga, poi arrivano gli indirizzi di colonna che, con il segnale CAS, vengono memorizzati nei latch. Poi viene effettuata la decodifica di righe e di colonne;

→ Segnali di memorizzazione degli indirizzi di riga/colonna RAS e CAS asseriti bassi;

→ DRAM asincrone: hanno un ritardo di accesso; infatti dobbiamo asserire per ogni byte RAS e CAS;

→ E' necessario un controllore esterno che legge e rinfresca periodicamente il contenuto delle celle in quanto le DRAM, per loro stessa natura, sono dinamiche;

→ Nell'esempio precedente tutte le 16384 celle su una riga vengono accedute contemporaneamente (e rinfrescate);

→ Ma solo 8 bit di dati vengono trasferiti per ogni ciclo di indirizzamento riga/colonna;

Per avere un accesso più efficiente si utilizza la soluzione come sotto descritta, cioè il fast page mode.

Fast Page Mode

→ Per un accesso più efficiente dei dati sulla stessa riga, dei latch a valle nei circuiti di "sense" che possano immagazzinare il contenuto;

→ Per leggere dati consecutivi, basta asserire CAS ed incrementare l'indirizzo di colonna sulla stessa riga; quindi per leggere bit consecutivi sulla stessa riga, individuati dagli 11 bit di indirizzo di colonna, con un solo segnale di RAS mandando solo il segnale di CAS;

→ Questa modalità di "pagina veloce" è utile per trasferimento a blocchi;

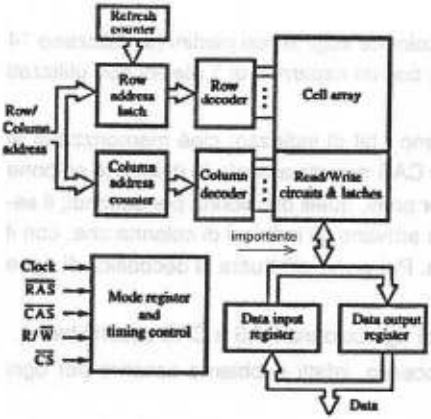
Nel fast page mode ci risparmiamo il dover asserire ogni volta il Raw Address Strobe, ma asseriamo solamente il Column Address Strobe che rilegge i nuovi indirizzi come emessi.

RAM SINCRONE (SDRAM)

→ Nei primi anni '90, la tecnologia delle DRAM migliorò con l'introduzione di un segnale di clock;

→ Fu aggiunta ulteriore circuiteria;

→ Circuiti integrati siffatti sono chiamati synchronous DRAMs (SDRAM);



Nella figura a lato una rappresentazione, con gli indirizzi di riga e di colonna multiplexati sulle stesse linee.

Idealmente gli indirizzi di riga e di colonna sono della stessa dimensione, ma possono essere anche leggermente asimmetrici.

Gli indirizzi di riga vengono memorizzati in un latch, gli indirizzi di colonna sono memorizzati in un contatore (e non un latch come nelle memorie asincrone); c'è un contatore di rinfresco (refresh counter) che controlla i latch. Ci sono decodificatori di riga e di colonna. I primi agiscono selezionando l'insieme di celle nella matrice, i secondi abilitano la lettura e la scrittura della circuiteria e ci sono dei latch che bloccano questi dati.

In basso a sinistra c'è una parte di controllo che prende diversi segnali e, in base ad altri segnali di configurazione (non mostrati in figura), definisce le modalità di lettura della memoria. Cosa importante è che a valle dei latch, quindi dei circuiti asincroni di memorizzazione dei dati, ci sono dei registri di lettura e dei registri di scrittura dei dati; in uscita i dati sono multiplati verso il mondo esterno. Ci sono dunque dei contatori degli indirizzi di colonna, un contatore che controlla il rinfresco e quindi i latch degli indirizzi di riga, ma soprattutto ci sono dei registri che memorizzano in ingresso e in uscita i dati.

→ I circuiti di Sense hanno funzionalità di latch;

→ Ulteriori vantaggi da immagazzinamento interno e disponibilità di un clock di sincronizzazione;

→ Un contatore di riga interno consente il rinfresco senza controllore esterno;

→ Le SDRAM includono registri per i dati oltre latch per gli indirizzi;

→ Nuove operazioni di accesso possono essere iniziate quando le precedenti sono ancora in transito;

→ La configurazione dei circuiti di controllo delle SDRAM richiede una configurazione all'accensione;

→ Il controllore della memoria inizializza la memoria;

→ Il controllore della memori specifica la lunghezza per trasferimento a blocchi e i ritardi necessari per la sincronizzazione;

Trasferimento a blocchi efficiente

→ Le DRAM asincrone incorrono in ritardi dovuti all'asserzione di CAS per ogni indirizzo di colonna;

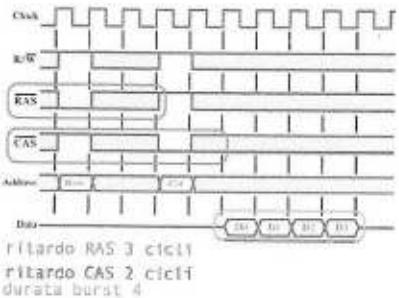
→ Nelle SDRAM si riducono i ritardi asserendo CAS solo per l'indirizzo iniziale di colonna

→ La circuiteria delle SDRAM incrementa il contatore di colonna e trasferisce dati consecutivi automaticamente;

→ La lunghezza del "burst" determina la durata del trasferimento;

→ Considerate l'esempio di un burst di lunghezza 4, ritardo nel RAS di 3 cicli e di CAS di 2 cicli; il burst è uno "scoppio", in questo caso può essere tradotto come pacchetto;

Lettura burst nelle SDRAM



Nella figura a lato un esempio di diagramma temporale di una lettura di un burst di lunghezza 4, ritardo nel RAS di 3 cicli e di CAS di 2 cicli.

Il segnale RAS (Raw Adrress Strobe) è attivo basso, quindi è utile quando il segnale è 0. Nel simbolismo in cui la riga è sia alta che bassa il dato non è significativo. Si noti che contemporaneamente a RAS è stato asserito Read per cui c'è la lettura di un burst, per due periodi dopo tale evento, quello che succede ai dati è del tutto irrilevante. Lo è anche quello che sta avvenendo a RAS durante questi tre cicli.

Dopo questi due cicli in cui RAS ha permesso la corretta memorizzazione degli indirizzi di riga nel latch di riga il segnale CAS è asserito basso e dopo due cicli gli indirizzi sono validi e viene letto un burst di 4.

DOUBLE DATA RATE SYNCHRONOUS RAMS (DDR)

→ Le prime SDRAM trasferivano dati solo sul fronte di salita del clock;

→ Nei miglioramenti successivi si trasferiscono dati sia sul fronte di salita che di discesa

→ Si raddoppia il flusso di dati, dopo asserzione di RAS/CAS;

→ Circuiteria di clock e controllo più complessa;

→ Accesso alla matrice interna e primo accesso non significativamente più veloci, cioè il primo RAS ed il primo CAS non sono cambiati significativamente;

Al crescere della lunghezza del burst (pacchetto) l'overhead, ovvero il sovraccarico, per il RAS ed il CAS e quindi la memorizzazione degli indirizzi di riga e degli indirizzi di colonna diventa sempre meno influente.

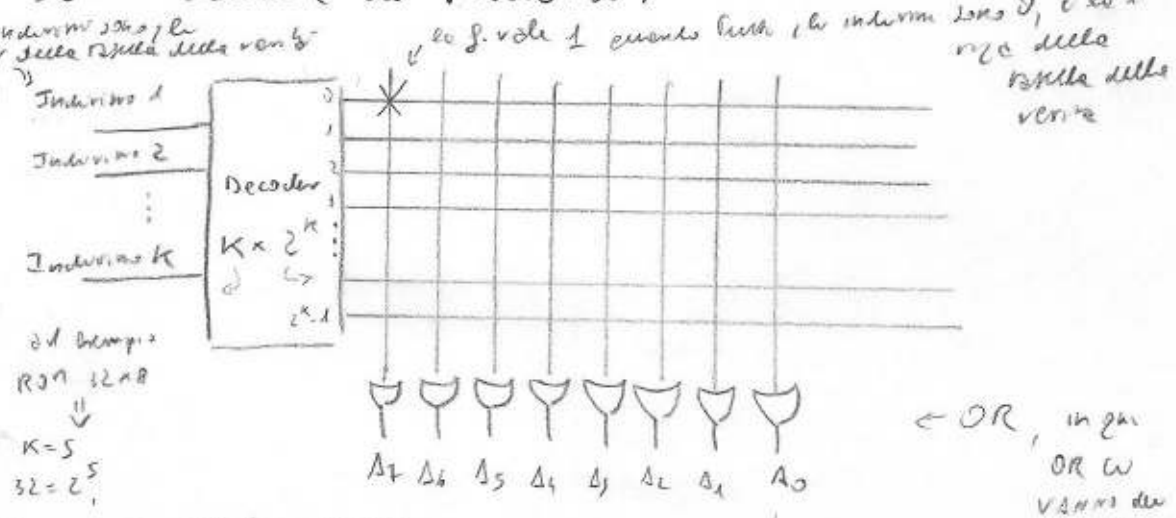
Nell'esempio precedente avevamo un burst di 4, 3 cicli di clock di latenza per RAS e 2 per CAS, quindi avevamo un overhead di 5 su 4 dati. Passando ad un double data rate sullo stesso esempio didattico, avremmo 8 dati e quindi non più un overhead del 125% ma un overhead di 5 su 8 che risulta sensibilmente ridotto.

ROM - Read Only Memory

È una scatola caratterizzata da $2^K \cdot n$ locazioni - Ad esempio una ROM 32×8, ovvero $2^5 \times 8$ locazioni, quindi $K = 5$ ed è il numero delle uscite (numero delle indirizzi), si avranno $n = 8$ uscite (dati). Le uscite sono OR di 1 quindi sono permanenti.

Le vantaggi di uscite sono ~~K~~, pari al n. di indirizzi. Entrano in un decoder $2^K \times n$ locazioni, quindi il decoder ha un input a K bit e 2^n output, da 0 a $2^n - 1$.

Vengono prodotti i minterm e le uscite sono Somme di Prodotti.



LEZIONE 17

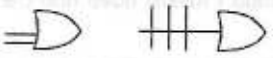
Dispositivi Logico-Programmabili

Prof. Romeo Becchereli
43'11"

- Memorie a sola lettura (Read-Only memories - ROM)
- Dispositivi logici combinatori
- Dispositivi logico-programmabili complessi (Complex Programmable Logic Devices - CPLDs)
- Celle standard e matrici di porte
- Matrici di porte programmabili sul campo (Field-Programmable Gate Arrays - FPGA)

MEMORIE A SOLA LETTURA (READ ONLY MEMORIES -ROM)

Verrà usata una convenzione grafica nell'uso di matrici.

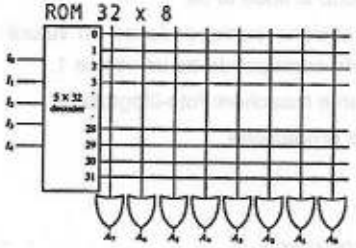


→ Invece di linee multiple in ingresso alla porta, disegniamo una singola connessa alla porta

→ Le linee di ingresso sono perpendicolari a questa singola linea e sono connesse attraverso fusibili

→ Analogamente per porte NOR, AND e NAND

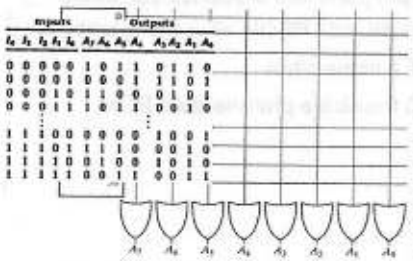
Fig 7.11 pag. 317



La ROM è una "scatola caratterizzata da 2^k x n locazioni.

Questo numero di locazioni sono individuate da k ingressi (indirizzi) e si avranno n uscite (dati).

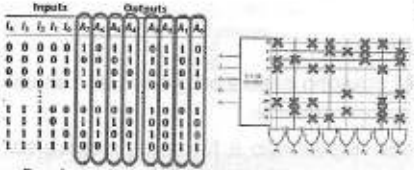
In figura a lato una ROM 32 x 8, ovvero 2^5 locazioni individuate da 5 linee di indirizzo e caratterizzate da una parola di 8 bit; in uscita ci sono OR da A_0 a A_7.



A lato un insieme di funzioni di uscita per le 8 variabili di uscita A_0 A_7 in funzione degli ingressi I_0 ... I_4.

Abbiamo una serie di combinazioni di 0 e 1 e andiamo a vedere come si programma la ROM.

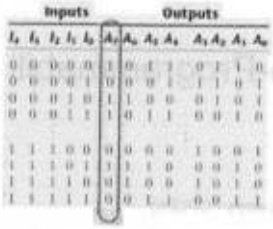
Consideriamo la funzione A_0: dove ci sono 0 non mettiamo un incrocio, dove c'è 1 si mette un incrocio mettendo un contatto tra le linee incrociate che entrano nell'OR. Questo vuol dire che in fase di produzione del sistema alcune connessioni vengono mantenute ed altre vengono bruciate.

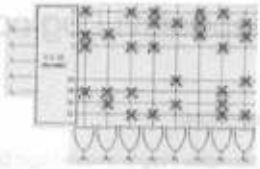


⇒Produce tutti i minterms

⇒Funzioni ad uscite multiple come SoP

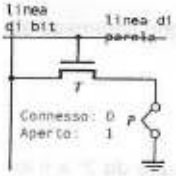
Le altre funzioni vengono realizzate analogamente. Avremo dunque 8 funzioni di 5 variabili.

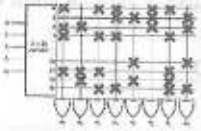




Queste funzioni di 5 variabili sono espresse come somma di prodotti. Questo vuol dire che il metodo produce tutti i minterm e produce funzioni ad uscite multiple come somme di prodotti.

La programmazione della ROM avviene in fabbrica, bruciando i fusibili dove non c'è connessione.





In figura a lato si mostra come sono realizzate le connessioni.

Sulle linee ci sono le linee di parola; sulle colonne ci sono le linee di bit.

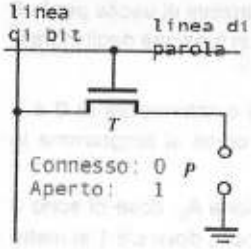
Una connessione corrisponde ad un valore 0, un aperto corrisponde ad un valore 1.

→ Le ROM vengono programmate in fonderia mediante maschere foto-litografiche

→ Le ROM risultano convenienti per grandi volumi di produzione

Programmable Read-Only Memory (PROM)

Sono ROM programmabili "sul campo", per volumi non grandi.



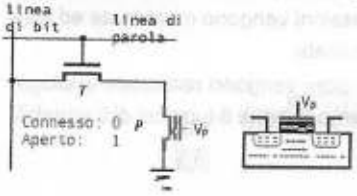
→ Le PROM vengono programmate bruciando dei fusibili

→ Tutti i fusibili sono intatti sulle PROM all'uscita di fabbrica

→ La bruciatura fusibili è irreversibile

→ Le PROMs sono più flessibili e pratiche delle ROM

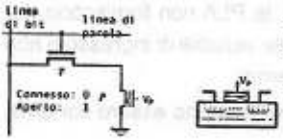
Erasable Programmable Read-only memory (EPROM)



→ Le PROM possono essere programmate con un transistore a porta flottante

→ Cancellabili esponendo a fotoni UV attraverso finestra trasparente

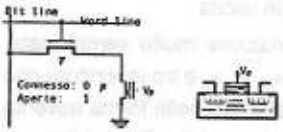
Electrically Erasable Programmable Read-only Memory (EEPROM, E²PROM)



→ Le PROM vengono programmate con un transistore a porta flottante

→ Selettivamente cancellabili mediante combinazione di tensioni

Flash Memory



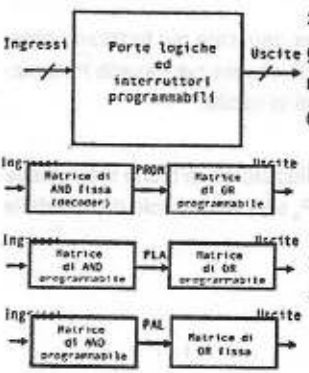
Ulteriore evoluzione, per avere costi sempre più bassi.

Sono le memorie utilizzate in lettori MP3, telefonini, nelle penne USB e in sostituzione di dischi magnetici.

→ Le PROM vengono programmate con un transistore a porta flottante

→ Simili alle EEPROM, ma è possibile scriverle (e cancellarle) solo per blocchi interi

DISPOSITIVI LOGICI COMBINATORI



Si tratta di un insieme di porte logiche e interruttori programmabili in cui ci sono degli ingressi (multipli); analogamente ci saranno uscite multiple.

Ce ne sono di diversi tipi:

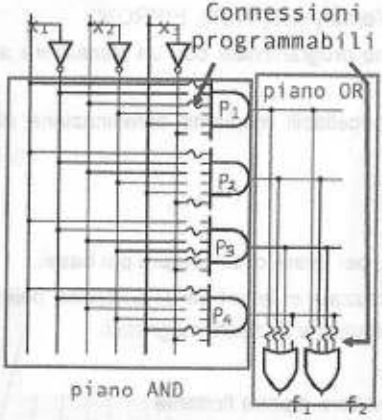
. PROM

. PLA (Programmable Logic Array)

. PAL (Programmable Array Logic)

Segue una rappresentazione circuitale di quanto esposto.

17.3



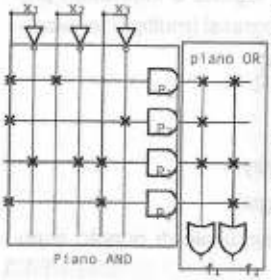
Programmable logic array (PLA)

→ Simili alle PROM, le PLA non forniscono decodifica completa delle variabili di ingresso e non generano tutti i minterms

→ Nelle PLA, i termini possono essere condivisi fra più OR

→ Può essere necessaria minimizzazione congiunta delle funzioni in uscita

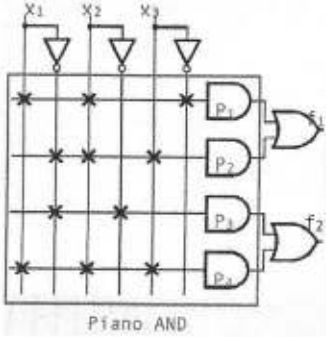
In questa rappresentazione molto semplificata, abbiamo 3 ingressi, x_1, x_2, x_3 e tre invertitore che producono tutti i letterali sia nella forma asserita che nella forma complementata. E' possibile selezionare questi letterali, in una forma o nell'altra, in ingresso a degli AND. Questi AND, nella rappresentazione sono 4, sono programmabili. Le connessioni programmabili saranno aperte o chiuse e saranno quindi stabiliti dei percorsi fra le variabili o i loro complementi verso la porta AND. Si costruiranno dunque dei termini prodotto (di 4 AND) per produrre due funzioni in uscita, f_1 e f_2.



Nella figura a lato il fatto che per generare più funzioni occorre condividere più OR e che può essere necessaria minimizzazione congiunta delle funzioni in uscita.

Si noti come il termine P_1 sia utilizzato sia in f_1 che in f_2, stessa cosa per il termine P_3. Il P_3 e il P_4 sono usate solo da una delle due funzioni.

Programmable array logic (PAL)



→ Nelle PAL, solo il piano degli AND è programmabile

→ Minimizzazione congiunta non è in genere necessaria perchè non ci sono condivisioni di termini prodotto fra le funzioni.

In figura a lato una rappresentazione semplificata, con tre variabili (x_i) e i loro complementi. Abbiamo 4 termini prodotto e due funzioni in uscita. Il piano AND continua ad esistere ed è programmabile, ma le funzioni prendono solo combinazioni di prodotti.

Il piano dell'OR non è programmabile, il che rende tutto

più semplice, ma, per contro, è meno versatile.

Nelle PROM, che sono più versatili, ci saranno porte in più, che però non saranno utilizzate.

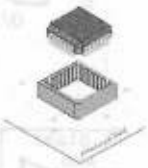
Programmazione di PLA e PAL

→ Programmazione degli interruttori in apposita unità

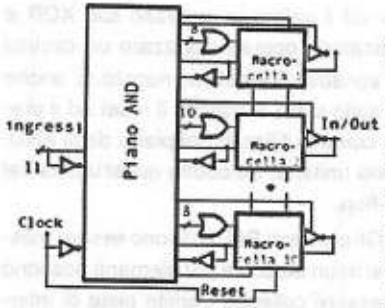
→ I circuiti integrati sono montati su zoccolo

→ Spesso usati i Plastic-leaded chip carrier (PLCC) con <100 piedini

(n.b.: questo tipo di realizzazione è valido anche per le PROM)



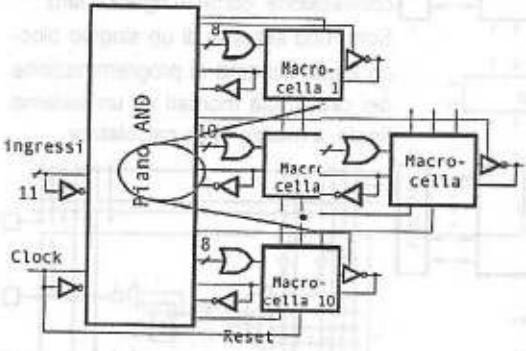
DISPOSITIVI LOGICI PROGRAMMABILI COMPLESSI (CPLDs)



A lato la semplificazione di un dispositivo , di tipo PAL, con un certo numero di ingressi (11), sia asseriti che complementati. All'uscita del piano degli AND ci sono OR multipli e delle parti che sono definite macrocelle.

Agli ingressi degli OR non sempre c'è lo stesso numero di ingressi e questo consente di avere un certo livello di flessibilità.

Notare inoltre i circuito di reazione.

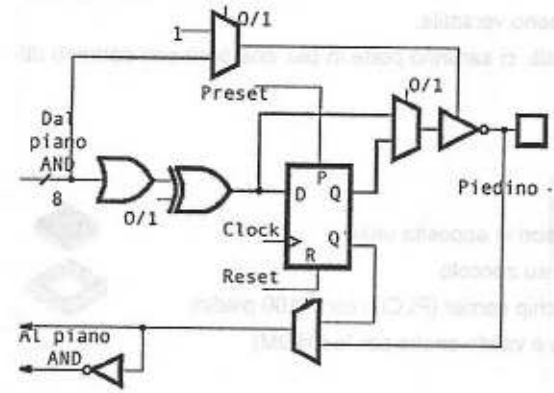


Per quanto riguarda la macrocella abbiamo degli OR, notiamo che dei segnali possono tornare indietro e notiamo che all'uscita della macrocella c'è un invertitore ed un percorso in reazione.

Nella figura successiva un modo di realizzare una macrocella.

Notiamo che dal piano degli AND abbiamo 8 linee; queste linee entrano in un OR e quindi avere una somma di prodotti, fino ad un massimo di 8 prodotti; le linee sono lanciate in uno XOR con 1 o con 0. Questa operazione fa si che la variabile che entra nello XOR ne esca vera o complementata.

Abbiamo poi un flip-flop di tipo D presettabile e resettabile per cui è possibile controllare

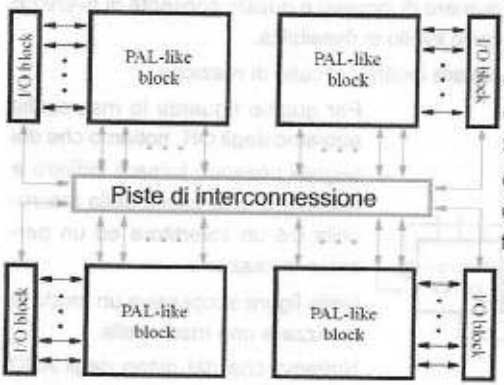


lo stato iniziale del flip-flop con delle linee di pilotaggio.

Il piano degli AND può essere sparato in un muliplexer 2 in 1 (in alto) che ci consente di bypassare l'elemento di memoria e di controllare la porta ad alta impedenza tri-state.

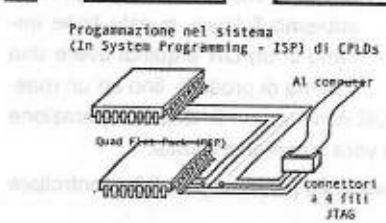
E' possibile decidere di prendere il segnale prima del flip-flop o dopo e questo determina avere un circuito combinatorio piuttosto che un circuito sequenziale; poi, l'uscita Q (se il transmission gate è aperto) o l'uscita Q' possono essere selezionate in reazione e ributtate nel piano degli AND attraverso un multiplexer 2 in 1 (in basso).

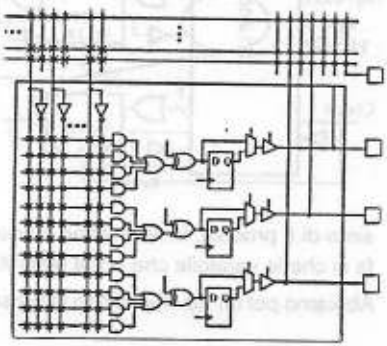
Quindi, controllando il segnale di tre multiplexer ed il valore in ingresso allo XOR è possibile realizzare un circuito puramente combinatorio oppure realizzare un circuito sequenziale in cui torna in reazione il valore della variabile Q e del suo negato. E' anche possibile partire in ogni momento dallo stato 0 o dallo stato 1, tramite il reset ed il preset. L'uscita può essere presa da un utente così come riutilizzati nel piano degli AND, si tratta di un sistema a stati finiti con una memoria unitaria, introdotta quest'ultima dal flip-flop oppure non unitaria se si mettono più flip-flop.



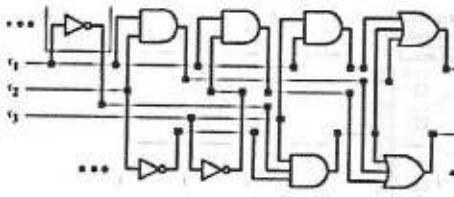
Gli elementi PAL possono essere messi in un blocco e più elementi possono essere collegati tramite piste di interconnessione, come in figura a lato.

Sotto uno schema di un singolo blocco e uno schema di programmazione dei circuiti già montati su un sistema finale, a mezzo di un calcolatore.





CELLE STANDARD E MATRICI DI PORTE



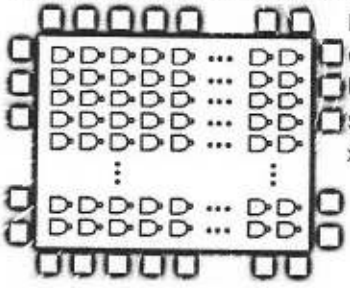
A lato una sezione di due righe di un circuito integrato con celle standard.

→ Le porte sono tutte simili fra loro e sono le stesse per tutti i possibili progetti

→ Lo specifico progetto definisce le connessioni con le ultime maschere fotolitografiche

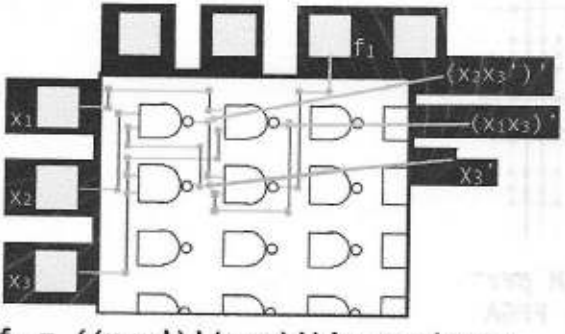
Si ottengono in uscita le funzioni per le variabile in ingresso grazie alle connessioni.

Matrici di porte (Sea-of-gates arrays)



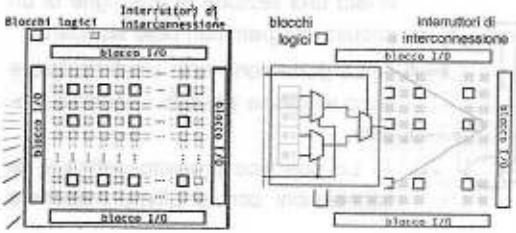
E' un sistema più generico delle porte standard precedenti, composto da un numero molto elevato di porte NAND.

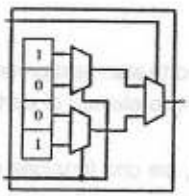
Si noti, nella figura sotto, come f_1 sia una funzione di x_1, x_2, x_3, puramente combinatoria.



$f_1 = ((x_2x_3')'(x_1x_3)')' = x_2x_3' + x_1x_3$

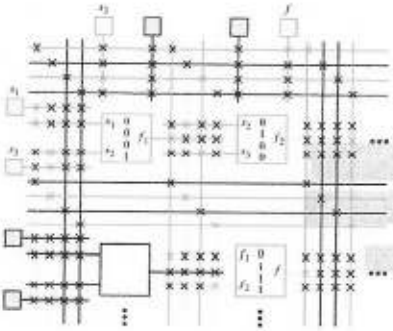
FIELD PROGRAMMABLE GATE ARRAYS (FPGAs)



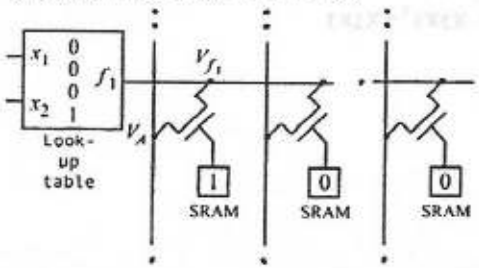


x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

funzione XOR



Transistori e SRAM per programmazione di FPGA



4.8

LEZIONE 18

Progetto a livello di trasferimento tra registri RTL

Prof. Romeo Beccherelli
40'07"

- Partizione di un sistema sequenziale in unità di controllo e unità di elaborazione
- Livello del trasferimento tra registri (RTL, Registry Transfer Layer)
- Macchina algoritmica a stati (ASM, Algorithmic State Machine)

Partizione di un sistema sequenziale in unità di controllo e unità di elaborazione

→ Il comportamento della maggior parte dei sistemi digitali dipende dalla storia dei loro ingressi e "memoria" del passato (sistemi sequenziali)

→ I flip-flops dei sistemi sequenziali immagazzinano tale memoria, i flip-flop sono clockati

→ Certe condizioni definiscono il contenuto dei flip-flops e determinano le azioni future

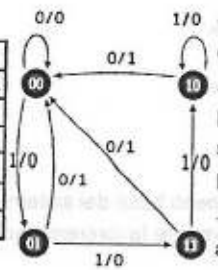
→ Il comportamento dei circuiti sequenziali può essere descritto da equazioni Booleane, tabelle di stato o diagrammi di stato

→ Queste rappresentazioni considerano ingressi e stati dei flip-flop

Tabella di stato e diagramma di stato

Tabella di stato e diagramma di stato

presente		prossimo				uscite	
		x=0		x=1		x=0	x=1
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0



Riprendiamo una tabella di stato e diagramma di stato e vediamo come procedere.

→ E' difficile specificare un sistema digitale di grandi dimensioni con una tabella di stato o diagramma di stato a causa dell'elevato numero di stati

→ I sistemi digitali sono progettati con un approccio modulare

→ Il sistema è partizionato in sottosistemi, ognuno con una sua funzione

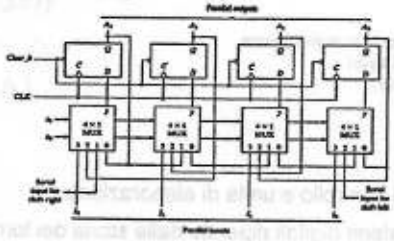
→ I flip-flops sono raggruppati in registri e contatori

→ I moduli sono costruiti con registri, decodificatori, multiplatori, elementi aritmetici e logica di controllo

→ Le operazioni sui registri sono shift left, shift right, count, clear, preset (setta il contenuto a 1) e load

Registro a scorrimento universale

Registro a scorrimento universale

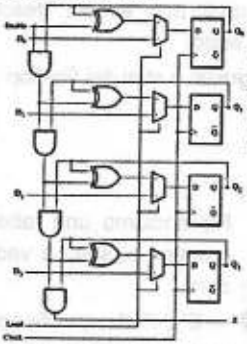


è un convertitore serie-parallelo bidirezionale

Nella figura a lato riprendiamo lo schema di un registro a scorrimento universale, un registro che consente, attraverso quattro multiplexer 4 in 1 di effettuare tutte le operazioni che ci interessano, siano esse quelle di scorrimento verso sinistra, di scorrimento verso destra, le operazioni di caricamento parallelo dall'esterno, l'operazione di memorizzazione del dato che cicla. I multiplexer selezionano attraverso due segnali (S_0 e S_1) gli ingressi del multiplexer e quindi il dato che viene caricato nell'ingresso D del flip-flop di tipo D.

Contatore sincrono ad incremento

Contatore sincrono ad incremento con 4 bit con flip-flop D e parallelo e caricamento parallelo



A lato un contatore sincrono ad incremento, molto utile anch'esso, in una rappresentazione a quattro bit, a caricamento parallelo (ripple counter). Questo contatore ci permette ad esempio di contare gli stati.

Quindi, visto questo, a questo punto:

→ Assumiamo che i registri siano i componenti base dei sistemi digitali (building block), con multiplexer e decoder, ma prevalentemente lavoreremo su un livello di astrazione più alto di quello delle singole porte

→ Il flusso di informazione e le elaborazioni dei dati immagazzinati nei registri sono chiamati operazioni di trasferimento di registro (*carica registro, scorri a destra o a sinistra, incrementa o decrementa un contatore*)

Si tratta dunque di un livello di astrazione più alto, più basso di quello di un linguaggio di programmazione, ma comunque strettamente correlato alla struttura hardware del sistema.

LIVELLO DEL TRASFERIMENTO TRA REGISTRI (RTL)

Rappresentazione al livello del trasferimento tra registri (RTL) comprende

1. Un insieme di registri nel sistema
2. Le operazioni svolte sui dati immagazzinati nei registri (i contatori saranno considerati registri che possono essere incrementati)
3. Un sistema di controllo che supervisiona la sequenza delle operazioni nel sistema digitale

Esempi di operazioni su registri

→ Trasferimento:	R2 ← R1	(move R1 into R2)
→ Aritmetica:	R1 ← R1 + R2	
→ Aritmetica:	R3 ← R3 + 1	
→ Scorrimento dx:	R4 ← shift_r R4	
→ Logico:	R5 ← R6 OR R7	
→ Reset:	R5 ← 0	

→ Operazioni in un sistema digitale sono controllate da segnali che impongono una sequenza prestabilita alle operazioni

→ Certe condizioni che dipendono dai risultati di precedenti operazioni possono determinare la sequenza di sequenze di operazioni future

Esempi di condizioni

→ If (T1 = 1) then (R2 ← R1)

→ If (T2 = 1) then (R2 ← R1, R1 ← R2) scambio di contenuto, avviene in un ciclo di clock

→ If (T3 = 1) then (R2 ← R1 + R2, R1 ← 0) allo stesso colpo di clock

→ Le uscite della logica di controllo di un sistema digitale sono variabili binarie che avviano le operazioni nei registri del sistema

MACCHINA ALGORITMICA A STATI - ALGORITHMIC STATE MACHINES (ASMs)

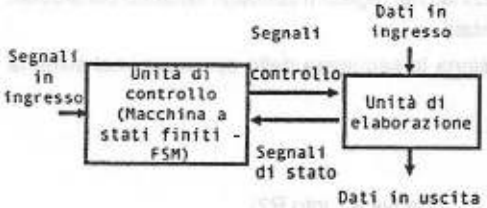
→ I moduli vengono definiti da un insieme di registri e di operazioni svolte su informazione binaria immagazzinata negli stessi registri

→ L'informazione binaria può essere dati o controlli

→ I moduli sono interconnessi in unità di elaborazione ("datapath") a segnali di controllo (control unit) per formare un sistema digitale

→ I segnali di controllo coordinano le operazioni nell'unità di elaborazione del sistema

Interazione fra controllo ed elaborazione



Nell'immagine a lato una rappresentazione di interazione tra controllo ed elaborazione. C'è una unità di controllo, che è una macchina a stati finiti, con segnali in ingresso, ad esempio di reset o di start. Vengono prodotti dalla macchina dei segnali di controllo verso una unità di elaborazione che riceve dei dati in ingresso, ad esempio legge una memoria. L'unità di controllo genera dei segnali di stato che entrano nella macchina a stati finiti. I dati in uscita dall'unità di elaborazione sono quelli che vogliamo produrre.

Per fare questo occorre:

→ Dati organizzati in parole e immagazzinati in registri

→ Dati manipolati con operazioni aritmetiche, logiche, di scorrimento, ...

→ Operazioni fisicamente implementate con sommatori, moltiplicatori, decodificatori, multiplatori, contatori, registri a scorrimento ...

→ Un diagramma di macchina algoritmica a stati (ASM) è un diagramma di flusso che definisce l' algoritmo da eseguirsi nel sistema digitale

Il diagramma ASM

→ Il diagramma ASM descrive la sequenza degli eventi nel tempo e la temporizzazione delle relazioni fra gli stati di un controllore sequenziale e gli eventi che si manifestano passando da uno stato al successivo

Un diagramma ASM è sostanzialmente diverso da un grafo di flusso perché la componente tempo all'interno di un diagramma ASM è fondamentale in quanto le operazioni che avvengono sui registri sono sempre sincronizzate dal clock.

I tre elementi del diagramma ASM

→ La casella di stato

→ La casella di decisione

→ La casella condizionale

→ Le caselle sono connesse in ordine di precedenza

Diagramma ASM per casella di stato

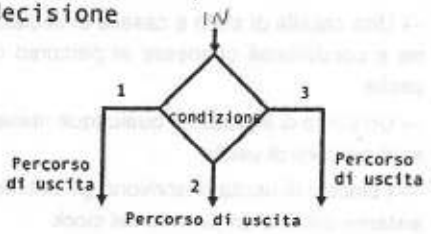
Diagramma ASM per casella di stato

E' una scatolina rettangolare in cui in alto a sinistra è indicato il nome dello stato. A volte è riportato un codice binario che identifica l'associazione numerica dello stato.

All'interno delle casella vengono elencate le operazioni effettuate sui registri e i segnali di uscita della casella di stato che sono segnali di uscita di una macchina sequenziale di tipo Moore. Ogni casella ha un ingresso e una uscita.

Diagramma ASM per casella di decisione

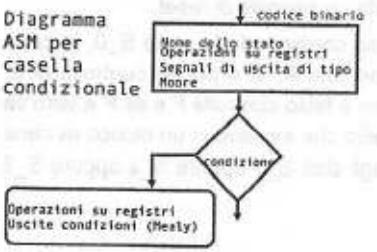
Diagramma ASM per casella di decisione



E' un rombo, detto diamond in inglese.

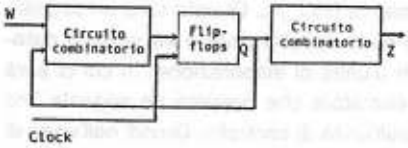
La decisione è tipicamente, ma non necessariamente, di tipo binario, con un ingresso e due uscite, una per vero ed una per falso. Più percorsi di uscita indicano una condizione di tipo case. Con due uscite è del tipo if... then... else. C'è un solo ingresso

Diagramma ASM per casella condizionale



Circuito sequenziale di Moore

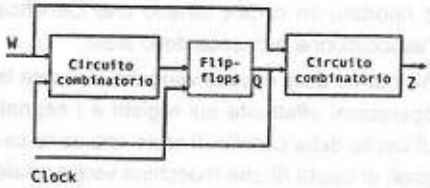
Circuito sequenziale di Moore



Le uscite dipendono solo dallo stato attuale

Circuito sequenziale di Mealy

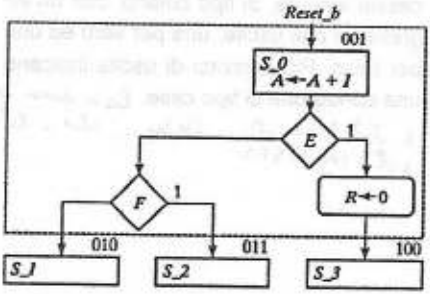
Circuito sequenziale di Mealy



Le uscite dipendono sia dallo stato attuale che dagli ingressi attuali

Blocco ASM (ASM block)

Blocco ASM



→ Una casella di stato e caselle di decisione e condizionali connesse al percorso di uscita

→ Un punto di ingresso e qualunque numero di percorsi di uscita

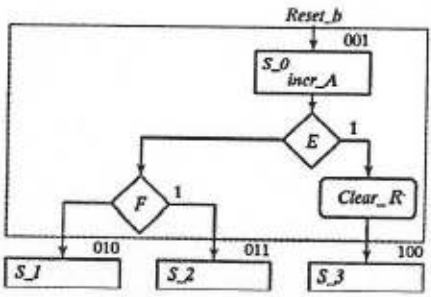
→ I blocchi di uscita descrivono gli stati del sistema durante un periodo del clock

Si noti in figura a lato, come il blocco ASM inizia da un segnale di reset.

Ad esso corrisponde lo stato S_0, in cui è incrementato A. Vi è poi un controllo di E, che, se vero pone A zero R e va nello stato S_3; se è falso controlla F e se F è vero va nello stato S_2, se è falso va nello stato S_1. Quello che avviene in un blocco avviene nello stesso istante di clock, quindi il passaggio agli stati S_1 oppure S_2 oppure S_3 avviene in un successivo tempo di clock.

Blocco ASM: il controllore

Blocco ASM: il controllore

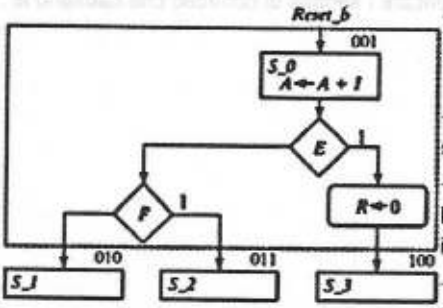


Se vogliamo separare il blocco ASM dalla unità di controllo, il controllore, allora invece dell'operazione (A ← A + 1) si indica il comanda (incr_A). Questo sarà un segnale che esce da una pista e va verso il datapath (l'unità di elaborazione) in cui ci sarà un contatore che riceverà un segnale "inc A" dall'unità di controllo. Quindi nell'unità di controllo è bef definito che si deve generare

un solo bit di linea per controllare l'unità di elaborazione.

Nell'unità di controllo non c'è il registro ma c'è l'elettronica necessaria a produrre il segnale, ad esempio quello di clear più sotto.

Temporizzazione diagramma ASM



→ Il clock controlla i registri dell'unità di elaborazione e tutti i flip-flops nella macchina a stati che implementa l'unità di controllo

→ Assumiamo che anche gli ingressi siano sincronizzati al clock

→ In un diagramma di flusso convenzionale le operazioni seguirebbero una dopo l'altra in sequenza

→ Prima, viene incrementato il registro A e solo dopo viene valutato E

→ Se E = 1, allora il registro R è azzerato e il controllo passa allo stato S_3

→ In un diagramma ASM l'intero blocco è una sola unità

→ Tutte le operazioni su registri all'interno del blocco si realizzano in sincronismo sullo stesso fronte di transizione dello stesso impulso mentre il sistema transisce da S_O allo stato successivo

1. Il registro A viene incrementato
2. Se E = 1, il registro R è azzerato
3. Il controllo fa transire allo stato successivo

Algorithmic state machine and datapath (ASMD)

E' una variante dell'ASM.

1.Un diagramma ASMD non elenca le operazioni sui registri nelle caselle di stato

2.I bordi di un diagramma ASMD sono annotati con le operazioni sui registri che sono simultanee alle transizioni di stato indicate sui bordi

3.Un diagramma ASMD include caselle condizionali che identificano i segnali che controllano le operazioni sui registri annotate sul bordo del diagramma

Dai diagrammi ASM a ASMD

1. Nei diagrammi ASM, sono rappresentati solo gli stati del controllore e i segnali di ingresso che determinano le transizioni di stato

2. Si converte il diagramma ASM in ASMD annotando i bordi del diagramma ASM per indicare le operazioni concorrenti sui registri dell'unità di elaborazione

3. Si modifica il diagramma ASMD per identificare i segnali di controllo che causano le operazioni nell'unità di controllo

LEZIONE 19

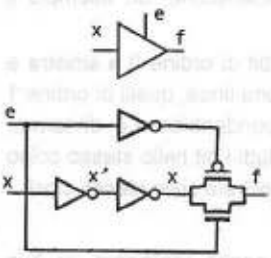
Progetto a livello di trasferimento tra registri, RTL
Esempi 1/2

Prof. Romeo Beccherelli
39'50"

- Struttura del bus
- Progetto dell'unità logica di controllo
- Progetto di un moltiplicatore binario sequenziale
- Progetto di un divisore fra interi

STRUTTURA DEL BUS

Tri-state buffer



e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1

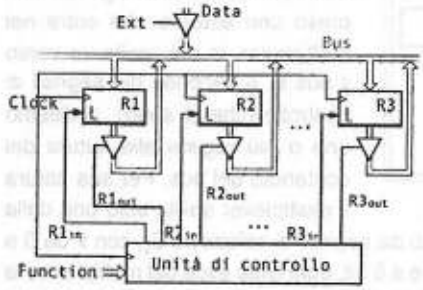
La funzione f di uscita è in funzione degli ingressi x ed e. Si costruisce la tabella di stato.

Quando il segnale e (enable) è zero l'uscita è in alta impedenza e non è connessa all'ingresso.

Quando e = 1 l'uscita è uguale al suo ingresso.

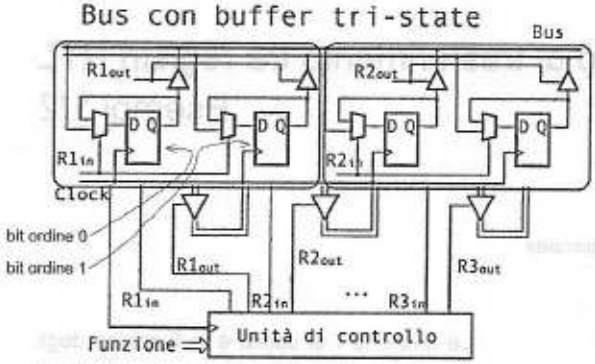
La realizzazione è fatta di due transistori, uno a canale p (sopra) ed uno a canale n (sotto), con i source e i drain accoppiati; essi vengono pilotati, dal segnale di abilitazione il canale n, dal negato il canale p. Il segnale di ingresso x passa attraverso una cascata di due invertitori per tirare su sostanzialmente la quantità di corrente per cui l'uscita passa a seconda del segnale di abilitazione. Il nodo di uscita può essere connesso oppure completamente isolato dal segnale di ingresso. L'utilità fondamentale del tri-state buffer è quella di poter interconnettere uscite di porte logiche che, se connesse direttamente tra loro sostanzialmente comporterebbe la loro bruciatura.

Bus con buffer tri-state



Cioè le porte logiche non possono essere accoppiate uscita su uscita, possono essere accoppiati ingressi, un ingresso e una uscita, ma non possono essere accoppiate uscite in controserie.

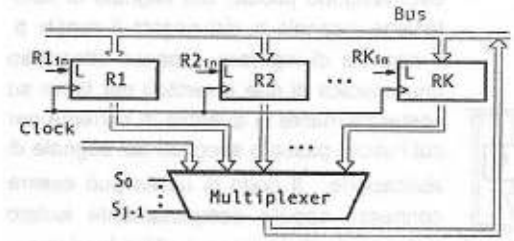
Nella realizzazione di un bus con buffer tri-state abbiamo una serie di registri (R_1, R_2 e R_3), una unità di controllo che controlla il segnale di load dei registri (R_{1in}, R_{2in}, R_{3in}), un clock in quanto l'unità è sincrona, un segnale Ext dal mondo esterno che dice di cari-



care sul bus, una funzione in ingresso che dirà ad esempio "carica Rx". Tale funzione dirà anche di far transitare attraverso il bus il contenuto di un registro per essere visto agli ingressi di tutti i registri; solo uno potrà caricare sul bus, ma tutti potranno leggere.

Sul bus uno parla e tanti possono ascoltare contemporaneamente nei limiti di alcune caratteristiche, ad esempio il fan-out delle porte che insistono sul bus in uscita.

Nella figura sopra un bus con due registri a due bit, con il bit di ordine 0 a sinistra e quello di ordine 1 a destra. I bit di ordine 0 si affacciano su una linea, quelli di ordine 1 su un'altra linea. Quindi i bit di pari peso stanno sulle corrispondenti linee e, di norma, quando il bus ha più linee, tutti i registri scrivono o leggono tutti i bit nello stesso colpo di clock perché ognuno dei flip-flop all'interno dei registri è clockato sullo stesso clock.



Si può anche realizzare un bus con multiplexers, come da figura a lato, con un numero R_k di registri. Si notano i flussi dei segnali, compreso uno esterno che entra nel multiplexer, la cui uscita va verso il bus e, a seconda dei segnali di controllo (che ci sono), abilitiamo uno o più registri alla lettura del contenuto del bus. Per sua natura il multiplexer abilita solo una della uscite verso il bus. Il multiplexer è selezionato da segnali di selezione S_x, con x da 0 a j - 1 e con la condizione $k <= 2^{j-1}$. Se il registro è a 8 bit, ogni linea esce dal multiplexer, la linea 0 viene multiplexata verso la linea 0 del bus e così per le altre linee, la 1, la 2 ecc. Quindi in realtà il multiplexer è una molteplicità di multiplexer pari al numero di bit di ogni registro.

Si noti che nella realizzazione di un bus il numero di buffer tri-state è limitato rispetto a quello di multiplexer.

PROGETTO DELL'UNITÀ LOGICA DI CONTROLLO

L'unità logica di controllo (controllore o controller) è separata dalla unità di elaborazione (datapath).

→ L'unità logica di controllo è una FSM, final state machine, ovvero una macchina a stati finiti, ovvero una macchina sequenziale che può essere di tipo Mealy o di tipo Moore

→ Le uscite di tipo Mealy e Moore controllano le operazioni dell'unità di elaborazione (datapath)

→ Gli ingressi dell'unità di controllo sono segnali esterni (tipo "inizia le operazioni" oppure "reset") e segnali di stato prodotti dall'unità di elaborazione

→ Il sistema viene sintetizzato da una descrizione RTL ottenuta dal diagramma ASMD (la versione commentata dell'ASM)

→ Alternativamente, possiamo derivare manualmente la logica che governa gli ingressi dei flip-flops che immagazzinano lo stato del controllore

→ L'informazione necessaria per formare il diagramma di stato del controllore è contenuto nella casella di stato del diagramma ASMD

→ Le caselle di decisione determinano le condizioni logiche per la transizione allo stato successivo nel diagramma di stato e determinano le asserzioni delle uscite condizionali

→ I metodi manuali sono poco pratici se il numero di stati e ingressi è elevato

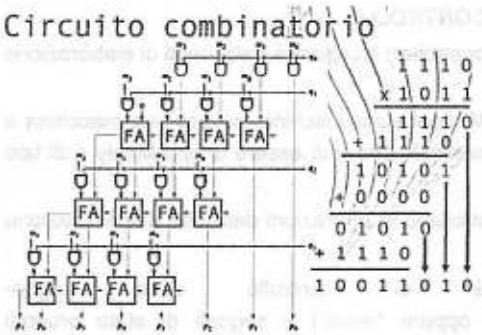
→ Sistemi dalle dimensioni moderate possono comunque essere progettati seguendo una metodologia di progetto strutturata

Progetto di un moltiplicatore binario sequenziale

Moltiplicando senza segno

```
Moltiplicando M  (14)          1 1 1 0
Moltiplicatore Q (11)        x 1 0 1 1
Prodotto parziale 0            1 1 1 0
                           + 1 1 1 0
Prodotto parziale 1          1 0 1 0 1
                         + 0 0 0 0
Prodotto parziale 2        0 1 0 1 0
                       + 1 1 1 0
Prodotto P       (154)   1 0 0 1 1 0 1 0
```

Circuito combinatorio



A lato il circuito combinatorio che ricalca le operazioni svolte sui numeri da moltiplicare. Notare che FA sono Full Adder.

Il problema è che per parole con un grande numero di bit (>4) il numero di porte in un circuito combinatorio può diventare proibitivo.

Per numeri a 32 bit avremmo 31x32 Full Adders e un numero considerevole di porte AND. Questo occupa molta superficie del chip e quindi costa.

Dunque, per la stessa soluzione adottiamo un progetto sequenziale piuttosto che combinatorio, come riportato sotto, usando uno pseudo-codice e, nella immagine successiva il relativo diagramma ASM. In quella successiva ancora il diagramma ASM della unità di controllo.

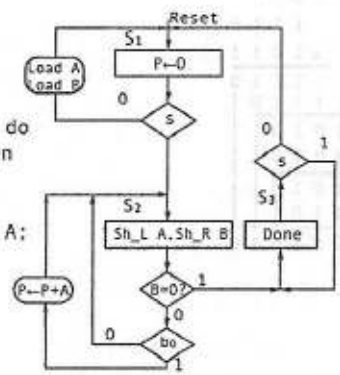
Circuito sequenziale

```
P = 0;
for i=0 to n-1 do
    if bi=1 then
      P=P+A;
    end if;
    Left-shift A;
end for;
Per parole con grande
numero di bit,
il numero di porte in un
circuito combinatorio può
divenire proibitivo
```

```
      1 1 1 0
    x 1 0 1 1
      1 1 1 0
  + 1 1 1 0
    1 0 1 0 1
  + 0 0 0 0
    0 1 0 1 0
  + 1 1 1 0
1 0 0 1 1 0 1 0
```

Diagramma ASM

```
P = 0;
for i=0 to n-1 do
    if bi=1 then
      P=P+A;
    end if;
    Left-shift A;
end for;
```

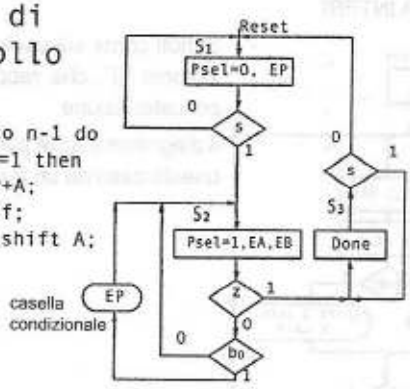


Nel diagramma ASM partiamo da uno stato noto (reset).

Poi individuiamo tre stati => tre caselle di stato, che sono rettangoli.

Unità di controllo

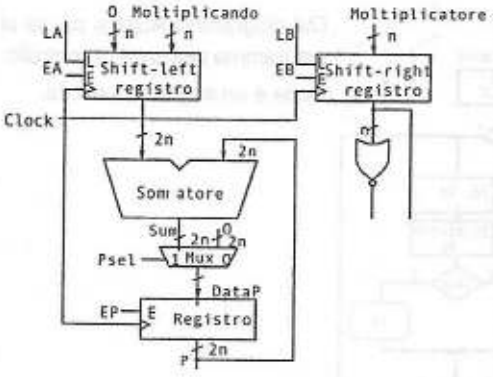
```
P = 0;
for i=0 to n-1 do
    if bi=1 then
      P=P+A;
    end if;
    Left-shift A;
end for;
```



Nel diagramma ASM dell'unità di controllo i segnali che l'unità di controllo deve produrre sono il segnale di selezione del registro P (vedi in figura, dopo il reset e sotto) ed il segnale di abilitazione alla scrittura nel registro P (vedi EP).

Poi i segnali di abilitazione dei registri A e B per fare gli shift.

Circuito Datapath



Nella figura a lato il circuito datapath, in cui notiamo come sia realizzabile l'unità di controllo sopra descritta.

Abbiamo due registri per gli shift, un sommatore, un multiplexer che carica secondo il segnale P_{sel} di selezione, e carica 1 o 0.

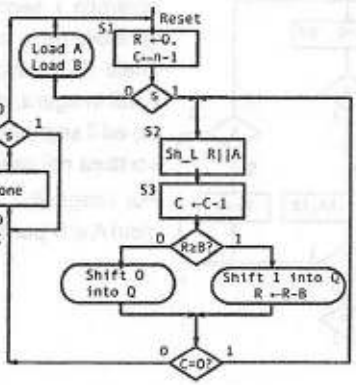
Notare che sia il moltiplicando che il moltiplicatore hanno n bit. Il moltiplicando viene messo in un registro di 2n bit dove i 2n bit di peso più significativo sono posti a zero. Questo perché semplifica le operazioni di registro. Il registro P come il multiplexer hanno 2n bit.

PROGETTO DI UN DIVISORE FRA INTERI

Diagramma ASM

A: dividendo
B: divisore
Q: quoziente
R: resto

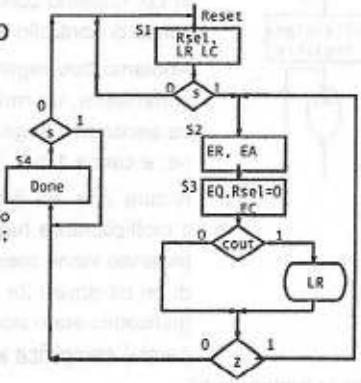
```
R=0;
for i=1 to  n-1 do
  Left-shift R||A;
  if R>B then
    q0=1;
    R=R-B
  else
    q0=0;
  end if
end for
```



Si noti come sia usato un nuovo simbolo "||", che rappresenta la concatenazione.

Il diagramma ASM parte anche in questo caso da un reset.

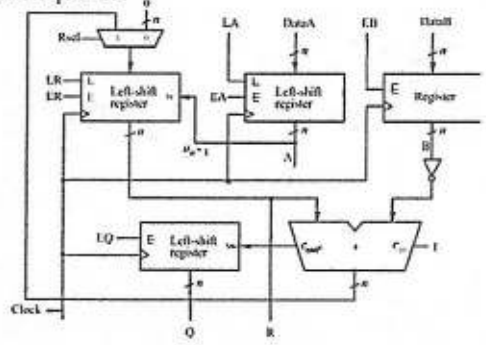
Unità di controllo unit



Dal diagramma ASM si passa al diagramma dell'unità di controllo.

Done è un segnale di uscita.

Datapath



L'unità datapath.

LEZIONE 20

Progetto a livello di trasferimento tra registri, RTL
Esempi 2/2

Prof. Romeo Beccherelli
43'19"

- Progetto con un decoder
- Progetto con multiplexer
- Progetto di un semplice processore

PROGETTO CON UN DECODER

→ Registro e decoder (cioè mantenere stati codificati in un registro e utilizzare un decoder, che permette di utilizzare un numero di stati pari a 2^n, per n bit il numero dei registri utilizzati)

→ Occorre determinare una tabella di eccitazione dei flip flop

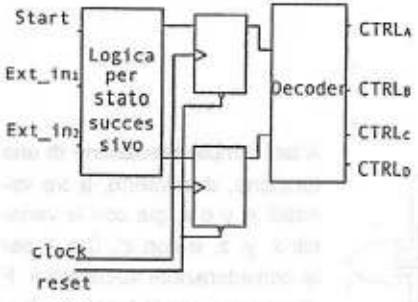
→ Registro con assegnazione "one-hot state" (assegnazione una alla volta, uno solo dei bit è 1, questo implica che il decoder non è necessario)

→ Decoder non necessario

→ Registro di maggiori dimensioni

→ Multiplexer seguito da registro e decoder

Progetto con decoder: Moore

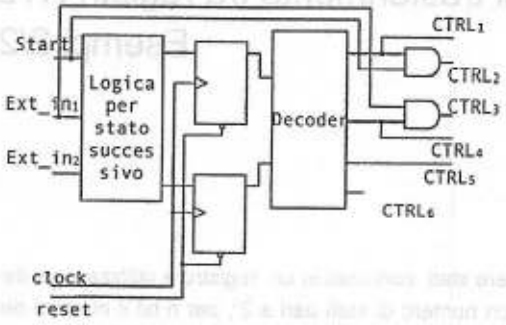


Nella figura a lato un circuito sequenziale di un decoder con due flip-flop, una logica di eccitazione per lo stato successivo dei flip-flop, un decoder a valle.

La logica prende tre segnali, uno di start e due segnali esterni generici ed essa consente di definire gli stati dei due flip-flop, che rappresentano un registro a due bit. I due stati vengono decodificati producendo quattro segnali di controllo, di tipo generico. I quattro segnali ci consentiranno di andare a controllare il datapath, cioè l'unità di elaborazione.

Il circuito è di tipo Moore, ma può essere anche realizzato un circuito di tipo Mealy, come nella figura successiva, che ci permette di realizzare uscite di tipo Mealy ($CTRL_2$ e $CTRL_3$), mentre le altre sono uscite di tipo Moore. Il fatto che ci sia almeno una uscita di tipo Mealy rende il circuito di tipo Mealy. Per inciso, "la macchina di Mealy è un automa a stati finiti che genera un'uscita a partire dagli stati d'ingresso e dallo stato corrente, a dif-

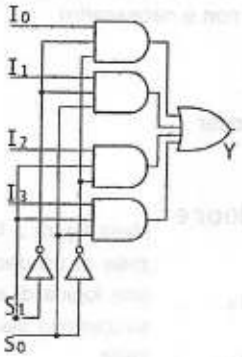
Progetto con decoder: Mealy



ferenza della macchina di Moore, che invece lavora solo in funzione dello stato corrente ".

PROGETTO CON MULTIPLEXER

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



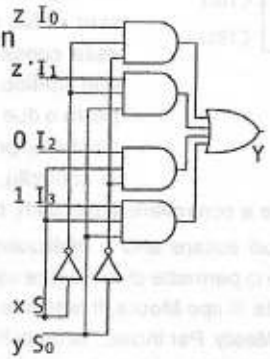
Il multiplexer può implementare funzioni, sotto in figura ne è riportata una che è espressa in termini di minterm, somma di prodotti.

A lato un multiplexer 2 in 1, due segnali di selezione S_1 e S_2, 4 dati in ingresso I_x, 1 uscita Y.

Funzione implementata con multiplexer

$\Sigma(1,2,6,7)$

x	y	z	F	
0	0	0	0	F=z
0	0	1	1	
0	1	0	1	F=z'
0	1	1	0	
1	0	0	0	F=0
1	0	1	0	
1	1	0	1	F=1
1	1	1	1	



A lato l'implementazione di una funzione, di minterm, a tre variabili x, y e z, già con le variabili x, y, z, e con z', 0 e 1 per le considerazioni successive: F viene espressa in funzione di z, nell'ultima colonna.

F = z e F = z', in quanto F prende tali valori; poi F = 0 e F = 1 nei casi in cui F vale 0 o 1 a prescindere dai valori di z.

Quindi l'attenzione è stata posta

solo sull'ultima variabile.

Abbiamo dunque un multiplexer che con 4 ingressi (2^2) va in 1, quindi con due ingressi di selezione.

Dunque, alla luce di quanto visto, possiamo generalizzare il ragionamento nel seguente modo.

→ Per una funzione con n variabili, scegliere un multiplexer 2^{n-1} in 1

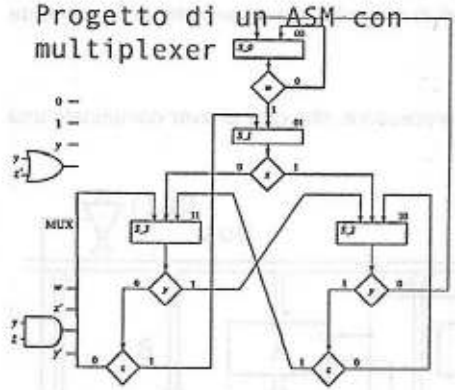
→ Riempire la tabella della verità

→ Applicare le prime n-1 variabili agli ingressi di selezioni del MUX (variabili x e y nell'esempio)

→ Per ogni combinazione di variabili di selezione, valutare le uscite come una funzione dell'ultima

→ Questa funzione puó essere 0, 1, la variabile o il suo complemento

→ Questi sono applicati agli ingressi dati dei MUX



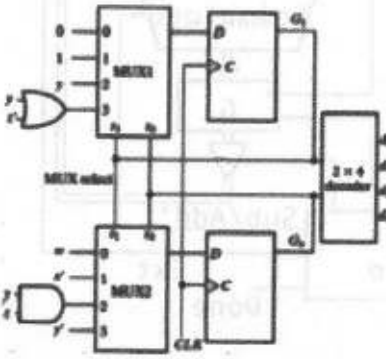
A lato il diagramma ASM con cui realizziamo l'unità di controllo con un multiplexer.

E' un diagramma ASM a 4 variabili, w, x, y, z, che determinano delle decisioni.

Abbiamo 4 stati (rettangoli), S_0, S_1, S_2, S_3.

Il diagramma non è ASMD perché non è etichettato dalle variabile prodotte in uscita dall'unità di controllo; cioè non vi è feedback dall'unità di elaborazione verso l'unità di controllo.

L'unità di controllo riceve le 4 variabili di ingresso e basta.



A lato l'immagine della mappatura del diagramma ASM su due multiplexer e due flip-flop.

I 4 stati sono mappati da 2 flip-flop e decodificati da un decoder 2 in 4. Quindi una volta che abbiamo codificato le stringhe 00, 01, 11, 10, definiamo gli stati dall'uscita del decoder, per cui ogni stato è determinato da una uscita del decoder.

Per determinare gli stati ci accorgiamo che, ad esempio, nella sequenza dei primi due loop, se

w = 0 cicliamo; se x = 0 passiamo in uno stato successivo. Usiamo dunque gli stati 00 e 01 per abilitare il multiplexer. Quindi le variabili di controllo del multiplexer saranno esclusivamente gli stati in uscita: lo stato 00 seleziona 0 in MUX1. Se abbiamo G0 G1 uguale a 0 1 sarà selezionato 1 in MUX1 e transiteremo dallo stato w verso lo stato 01. Idem per le altre due condizioni.

PROGETTO DI UN SEMPLICE PROCESSORE

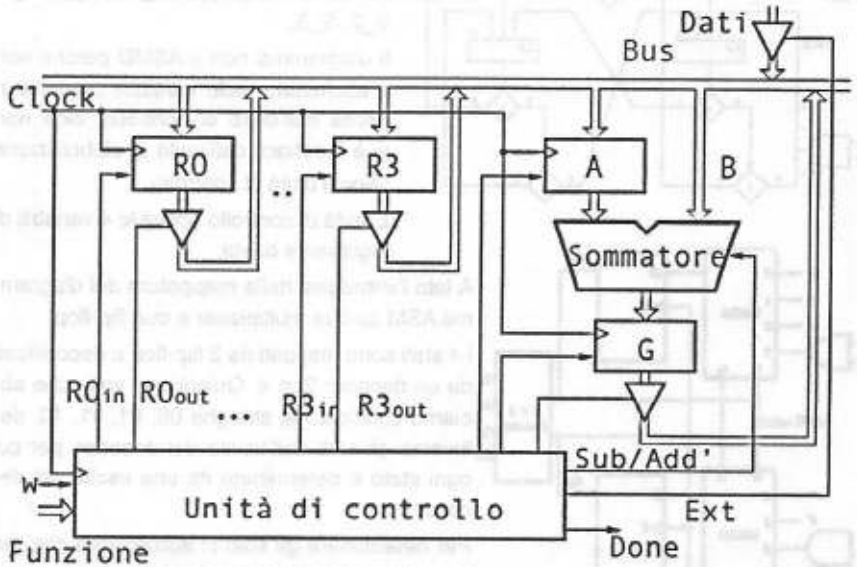
Il processore è un sistema di elaborazione e controllo che svolge delle funzioni, sostanzialmente di natura algebrica, di movimentazione dei dati.

Esso è in grado di ricevere un codice di una funzione da svolgere, produrre i segnali di controllo per l'unità di elaborazione, che è essa stessa contenuta nel sistema.

Dovranno essere controllati i segnali di abilitazione dei registri e far si che, usando un bus, ci sia uno ed uno solo dei registri affacciati sul bus che parli. Sul bus uno parla e tanti possono ascoltare.

In in processore che svolge operazioni avremo quindi un blocco, ad esempio un blocco "moltiplicatore" che verrà attivato da segnali di controllo che dicano che la funzione da svolgere è una moltiplicazione.

Di seguito la realizzazione di un semplice processore, che dice di aver completato una operazione, non dicendone il risultato.



Dunque quello che consideriamo è un processore così fatto: c'è una unità di controllo che riceve in ingresso una funzione e un segnale di attivazione ("fai l'operazione"); il sistema è sincrono, c'è un clock; ci sono dei registri da R0 a R3; i registri devono essere abilitati all'ingresso (R_{xin}) in modo che i registri possano leggere quello che c'è sul bus; ci sono segnali di controllo che abilitano i three-state buffer e quindi uno ed uno solo di essi possa scrivere sul bus.

Altri elementi del controllore sono: un registro A affacciato su un sommatore/sottrattore in complemento a 2, a seconda del segnale (a destra nel Sommatore); c'è un accumulatore G, dove il risultato del registro A, che non è indicizzato ma è un registro un pò particolare in questo esempio. L'uscita di uno dei registri general purpose da R_0 a R_3, a differenza del registro A, un pò più specializzato, o il segnale dall'esterno è letto e utilizzato come uno dei due addendi nel sommatore. Il risultato dell'elaborazione è rimessa sul bus per poter essere riletta da uno dei registri Rx affacciati sul bus. Quando l'operazione è terminata si manda un segnale esterno Done per dire "abbiamo finito". Il segnale Ext abilita il caricamento dei dati dall'esterno.

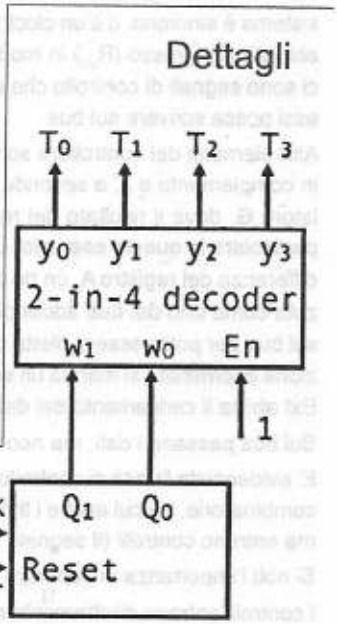
Sul bus passano i dati, ma non c'è nulla che legga sul bus.

E' evidenziata l'unità di controllo, tutto il resto è l'unità di elaborazione, costituita da parti combinatorie, tra cui anche i three-state buffer, e da parti sequenziali. In ingreso al sistema entrano controlli (il segnale w e il segnale funzione che è a più bit) ed entrano dati.

Si noti l'importanza di separare i controlli dai dati.

I controlli entrano direttamente nell'unità di controllo e qui vengono elaborati per produrre i segnali di controllo ad uso dell'unità di elaborazione. I dati non vengono normalmente letti dall'unità di controllo, cioè il dato è il dato dell'unità di elaborazione così come i controlli in ingresso sono i dati dell'unità di controllo.

Operazione	Funzione	f_1	f_0
Load Rx, Data	Rx←Data	0	0
Move Rx, Ry	Rx←[Ry]	0	1
Add Rx, Ry	Rx←[Rx]+[Ry]	1	0
Sub Rx, Ry	Rx←[Rx]+[Ry]	1	1



- Load: 1 ciclo
- Move: 1 ciclo
- Add: 3 cicli
- Sub: 3 cicli

La metodologia prevede di evidenziare le funzione che vogliamo far svolgere al sistema con la relativa funzione (vedi immagine sopra).

Le operazioni sono:

Load R_x, ovvero carica dall'esterno R_x, con x da 1 a 3

Move R_x, R_y, ovvero copia il contenuto di R_y e copialo in R_x; R_y rimane immutato

Add R_x, R_y, ovvero R_x contiene la somma di R_x e R_y

Sub R_x, R_y, ovvero R_x coniene la differenza di R_x e R_y

Di queste possiamo determinarne i cicli di clock.

Per l'operazione load occorre un solo ciclo di clock in quanto si abilita il registro R_x interessato ed alla prima transizione del clock (positiva, in questo caso) il registro è caricato. Quello che è importante è che il dato sia valido sul bus e quindi che l'unità di controllo abbia asserito il segnale di enable per il multiplexer dall'esterno in modo tale che il dato sia valido quando arriva il clock.

Anche l'operazione di move è una operazione ad un unico ciclo di clock. Quando arriva il segnale di clock il segnale out di un registro sorgente deve essere abilitato e quindi il dato deve essere visibile sul bus; deve essere già asserito il segnale in del registro destinazione che abilita tale registro. Quindi se il dato è stabile da una quantità di tem-

po stabilita dalle famiglie logiche e dai datasheet dei componenti allora all'istante in cui arriva il clock il dato viene caricato nel registro destinazione.

L'operazione di add richiede 3 cicli. Questo perché come prima cosa deve essere caricato in A il contenuto di uno dei quattro registri R_x, che richiede un ciclo di clock. Poi al successivo ciclo di clock il dato è caricato nel registro G, al terzo ciclo di clock il dato è riportato in uno dei tre registri R_x.

Per la sub occorrono, analogamente all'add, tre cicli di clock. La differenza sta nel controllo di carry in ingresso e il complemento del sottraendo, nel sommatore.

Nella stessa immagine ci sono alcuni dettagli, ad esempio come si possa sintetizzare l'unità di controllo con dei decoder e dei flip-flop. In questo caso dobbiamo generare un numero di stati corrispondenti al numero dei cicli necessari.

Abbiamo 3 cicli al massimo; è saggio avere anche un ciclo iniziale in cui non si fa nulla.

Abbiamo quindi bisogno di un flip-flop con due bit, e un decoder 2 in 4. Nessuno ci vietava di avere un "one hot assignment" e quindi una assegnazione a simbolo 1 ed avere quindi un registro a 4 bit.

Il decoder lo abilitiamo (enable = 1) in quanto non è possibile non avere una delle fasi del sistema.

Quello che produciamo sono dunque quattro istanti, T_0, T_1, T_2 e T_3 del sistema.

Definiamo inoltre che per una operazione da svolgere, descritta da una funzione relativa, la funzione sia codificata con due bit. Avendo avuto più bit avremmo potuto implementare altre funzioni. Ad esempio una multiply e un division.

Avendo assegnato i codici alle funzioni da implementare e avendo definito 4 istanti di clock (T0, T1, T2 e T3), grazie al registro a due bit e dal decoder 2 in 4, possiamo definire quali sono i segnali di controllo che dobbiamo generare.

Si prende ogni operazione e si determinano azioni sui segnali da fare negli istanti temporali.

Quindi per la load e la move abbiamo stabilito che serve un solo istante temporale ; per le operazioni di add e sub servono tre istanti temporali.

Per la load nel primo istante temporale dobbiamo produrre il segnale Ext (External) che abilita la scrittura attraverso il tri-state buffer del bus da parte del mondo esterno. Dobbiamo abilitare il registro X, con R_{in} = X. Una volta fatto questo, tutto in sincronia, possiamo asserire il segnale Done per dire che abbiamo fatto la load.

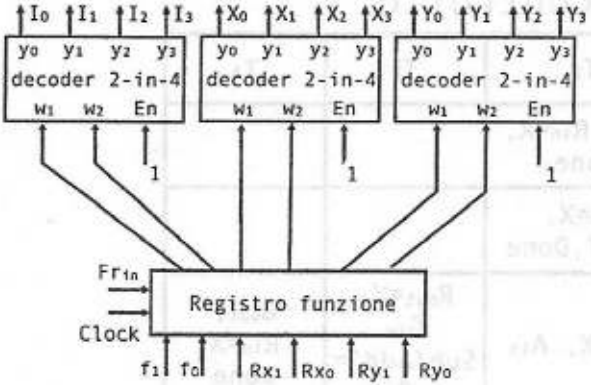
Per la move invece di abilitare il segnale Ext che ci fa scrivere dal mondo sterno verso il processore, dobbiamo abilitare in uscita la variabile Y, con R_{out} = Y; l'operazione sarà quella di portare Y in X, quindi dobbiamo abilitare X con R_{in} = X.

Per l'operazione add, nel primo passo bisogna abilitare il tri-state buffer del registro X con R_{in} =X

Segnali di controllo

Operazione	T_1	T_2	T_3
Load: I_0	Ext, R_{in}=X, Done		
Move: I_1	R_{in}=X, R_{out}=Y, Done		
Add: I_2	R_{out}=X, A_{in}	R_{out}=Y, G_{in} Sub/Add'= 0	G_{out}, R_{in}=X Done
Sub: I_3	R_{out}=X, A_{in}	R_{out}=Y, G_{in} Sub/Add'= 1	G_{out}, R_{in}=X Done

produzione dei segnali



$Clear=w'T_0+Done$

$FR_{in}=wT_0$

$Ext=I_0T_1$

$Done=(I_0+I_1)T_1+(I_2+I_3)I_3$

$A_{in}=(I_2+I_3)T_1$

$G_{in}=(I_2+I_3)T_2$

$G_{out}=(I_2+I_3)T_3$

$Sub/Add'=I_3$

$RO_{in}=(I_0+I_1)T_1X_0+(I_2+I_3)T_3X_0$

$RO_{out}=I_1T_1Y_0+(I_2+I_3)(T_1X_0+T_2Y_0)$