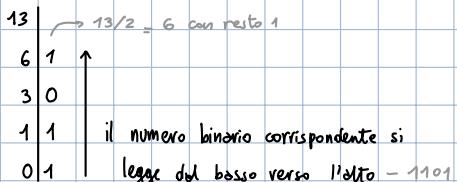


- con n cifre rappresento 2^n cifre: da 0 a $2^n - 1$
- quanti bit mi servono per rappresentare un numero? $\log_2 N$

CONVERSIONI

- da base 10 a base 2

dividere per 2 e segnare i resti



STESSA COSA

DA $b_{10} 4 + 16$

$\circ 8 \quad$ es. 378

$$378 : 16 = 23 + \text{RESTO } 10$$

$$23 : 16 = 1 + \text{RESTO } 7$$

$$1 : 16 = 0 + \text{RESTO } 1$$

$$16 \cdot 23 = 368$$

$$378 = 189 \quad 0$$

$$96 \quad 1$$

$$47 \quad 0$$

$$23 \quad 1$$

$$11 \quad 1$$

$$5 \quad 1$$

$$2 \quad 1$$

$$1 \quad 0$$

$$0 \quad 1$$

$$7 = 7$$

$$17A$$

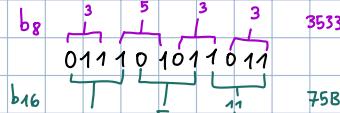
$$1 = 1$$

$$17A$$

- da base 2 a base 8/16

Raggruppare 3 cifre per b_8 e convertire
4 cifre per b_{16}

(partendo dalla destra - se le cifre non bastano,
aggiungere zeri \rightarrow sinistra)



3533

75B

VIRGOLA MOBILE

3 valori: ① Segno: 1 bit $\begin{cases} 0 \text{ pos} \\ 1 \text{ neg} \end{cases}$ ② esponente ③ mantissa - valori dopo la virgola
prima - sempre "1"!

IEEE 754 16 bit half-precision

bit: $s = 1$ $e = 5$ $m = 10$ bias = 15

Convertire in IEEE 754 half-precision: 13,45

① parte intera

si converte in binario: $13 = 1101$

② parte decimale \rightarrow si moltiplica per 2 la parte decimale fino ad arrivare a 1,0
o al numero necessario di bit

$m = 10$ e 13 sono 4 bit,

quindi ne ne bastano 6

$0,45 \times 2 = 0,90$ in ordine

$0,90 \times 2 = 1,80$ normale

$0,80 \times 2 = 1,60$

$0,60 \times 2 = 1,20$

$0,20 \times 2 = 0,40$

$0,40 \times 2 = 0,80$ v

oppure ne faccio 7 sapendo che
ci sarà il bit pre-virgola

quindi, 1101,011100

③ dobbiamo arrivare a 1,x, quindi sposto la virgola di quanto serve e "segno" l'esponente

$$1101,011100 = 1,10101100 \times 2^3$$

$$m = 10 \text{ quindi aggiungo uno } 0 \quad 1,101011000$$

④ Calcolo dell'esponente: $e = 3 + \text{bias} = 3 + 15 = 18 \rightarrow 10010$
segno = + = 0

⑤ metto tutto insieme $<0; 10010; 101011000>$

① Converti ogni lettera in binario

C	E	9	4
12	14	9	4
1100	1110	1001	0100

ovvero
 $\langle 1; 10011; 1010010100 \rangle$
 s e m

② $e = 10011 = 19$ sottraggo il bias $19 - 15 = 4$

$m = 1010010100$ considero l'1 implicato e moltiplico con l'esponente

$$1,10100101 \times 2^4$$

$$= 11010,0101$$

③ Converti in decimale

$$\begin{array}{r} 11010,0101 \\ \text{---} \\ 2^{-1} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4} \\ \frac{1}{2} \quad \frac{1}{4} \quad \frac{1}{8} \quad \frac{1}{16} \end{array}$$

quindi $\frac{1}{2} + \frac{1}{16} = 0,25 + 0,0625 = 0,3125$

$2^4 + 2^3 + 2^1$
 $= 26$
 $s = 1 = -$ quindi $-26,3125$

BCD rappresentazione di tutte le cifre decimali in binario
 ogni cifra - 4 bit (ma arriviamo fino al 9, 1001)

3 8 2 7
 0011 1000 0010 0111

CA1 E CA2

CA1 → Si invertono le cifre

CA2 → numero positivo: è il numero in binario, con uno 0 davanti per il segno

→ numero negativo:

si effettua il CA1, si aggiunge 1 bit a sx per il segno, e si somma 1

$$\text{es. } -3 \quad 11_2 \quad \overset{3 \text{ in}}{\text{CA1}} = 00 \quad \text{CA2} = 100 +$$

$\frac{1}{101}$ perché? il bit più significativo è un numero negativo e cui

si sommano gli altri bit (positivi)

$$\text{quindi } 101 = -2^2 + 2^0$$

$$-4 + 1 = -3$$

OPERAZIONI

- Somma $\rightarrow 1+1 = 10$

OVERFLOW • binario - se il numero di bit non basta

a rappresentare il numero

- CA2 - se sommando due negativi otteniamo un positivo o viceversa

- Sottrazione \rightarrow si usa il prestito

$$\begin{array}{r} 10110 \\ - 10011 \\ \hline 00011 \end{array}$$

← 0-1
con il prestito diventa 10-1
= 01

IN VIRGOLA MOBILE

addizioni/sottrazioni

- ① Confrontare gli esponenti e portarli

allo stesso valore (tipicamente il maggiore)

$$A < 0; 10011; 1010011010 > (26, 62)$$

$$B < 1; 10100; 0010110101 > (-37, 68)$$

$$\begin{array}{r} B \quad A \\ 10100 > 10011 \\ 20 \quad 19 \end{array}$$

quindi incrementiamo l'esp. di A di 1
e spostiamo a dx la mantissa
(ricordando l'1 IMPLICITO)

$$A = 0, 11010011010 \cdot 2^5$$

- ② Controllo i segni

→ sono discordi, è una differenza

quindi si sottrae quello minore in magnitudo a quello maggiore

Quale sarà il segno del risultato?

B > A e B è negativo, quindi sarà negativo

$$\begin{array}{r} 1, 0010110101 \\ 0, 1101001101 \\ \hline - 0, 0101101000 \end{array}$$

qui 100- per cui 001 → 0-1 diventa 10-1 ma 0-1 prende da 1 per diventare 10 e già serve tolto 1, quindi diventa 1 → 1-1=0

$$- 0, 0101101000 \cdot 2^5$$

③ normalizzazione - si riporta a 1, x

$$- 0, 0101101000 \cdot 2^5 = -1, 01101 \cdot 2^3$$

- ④ riscrittura in IEEE 754

$$e = 3 + bias = 3 + 15 = 18 = 10010$$

$$S = 1 \quad m = 0110100000$$

$$< 1; 10010; 0110100000 >$$

moltiplicazione

- ① segno { - operandi discordi
+ operandi concordi

- ② sommare gli esponenti (avendo sottratto il bias)

- ③ moltiplicare le mantisse

- ④ (se serve) normalizzazione

- ⑤ segno: 1

$$< 1; 10000; 0110000000 >$$

$$< 0; 10001; 0101000000 >$$

$$⑥ esp: 10000 = 16 - 15 = 1$$

$$10001 = 17 - 15 = 2$$

$$1+2 = 3 + 15 = 18 = 10010$$

$$\begin{array}{r} 1, 0101 \\ 1, 011 \\ \hline 10101 \\ 10101 \\ 00000 \\ 10101 \\ \hline 1, 1100111 \end{array} \cdot 2^3$$

$$< 1; 10010; 1100111000 >$$

se il sottralendo è maggiore del minuendo, posso sottrarre il minuendo dal sottralendo e riportare il risultato con il segno "-"

$$\begin{array}{r} 10011 \\ - 11110 \\ \hline 10011 \\ - 01011 \end{array}$$

ASSIOMI E PROPRIETÀ DELL'ALGEBRA DI BOOLE

(quelle non scontate)

- distributiva $\rightarrow x(y+z) = xy + xz$
 $x+yz = (x+y)(x+z)$

XOR $\rightarrow x \oplus y = \bar{x}y + x\bar{y}$

- complemento $\rightarrow x + \bar{x} = 1$
 $x \cdot \bar{x} = 0$
- el. neutro $\rightarrow x+0 = x$
 $x \cdot 1 = x$
- el. nullificatore $\rightarrow x+1 = 1$
 $x \cdot 0 = 0$
- idempotenza $\rightarrow x+x = x$
 $x \cdot x = x$
- assorbimento $\rightarrow x + xy = x$
 $x(xy) = x$
- II $\rightarrow x + \bar{x}y = x+y$
 $x(\bar{x}y) = xy$

De Morgan $\rightarrow \begin{matrix} \overline{x+y} \\ \overline{x \cdot y} \end{matrix} = \begin{matrix} \bar{x} \cdot \bar{y} \\ \bar{x} + \bar{y} \end{matrix}$

teorema del consenso $\rightarrow \bar{x}z + xy + yz = \bar{x}z + xy$

FORMA NORMALE - SOP / POS

① De Morgan ② Distributiva \rightarrow per SOP $x(y+z) = xy + xz$ ③ Eliminare termini ridondanti
 \hookrightarrow per POS $x+yz = (x+y)(x+z)$

$\bullet (\bar{x}y + \bar{x}\bar{y}) + (\bar{x} + z) + yz$ normale SOP

④ De Morgan $(\bar{x}y + \bar{x}\bar{y}) \cdot (\bar{x} + z) + yz$

$(\bar{x}y \cdot \bar{x}\bar{y}) \cdot (\bar{x} + z) + yz$

⑤ Assiomi/prop. $(\bar{x} + \bar{y}) \cdot (x + y) \cdot (\bar{x} + z) + yz$

$(\bar{x}y + \bar{x}\bar{y}) (\bar{x} + z) + yz$

$\bar{x}y + \bar{x}yz + x\bar{y}z + yz$

$\bar{x}y + z(\bar{x}y + x\bar{y} + y)$

$\bar{x}y + z(y + x\bar{y})$

2^a ass

$\bar{x}y + z(y + x)$

$\bar{x}y + yz + xz$

TEOR. CONSENSO

$= \bar{x}y + xz$

DA SOP A POS

$\bar{x}y + xz =$ faccio tutte le "combinazioni"

$(\bar{x}+x)(\bar{x}+z)(y+x)(y+z)$

$= (\bar{x}+z)(y+x)(y+z)$

FORMA DUALE

AND/OR e 0/1 sono scambiati

$xy + yz$ duale $\rightarrow (x+y)(y+z)$

- utile per la forma complementare di un'espressione \rightarrow si fa la duale e si complementa ogni variabile

MINTERMINI \rightarrow [SOP] termini prodotto in cui compaiono tutti i letterali $\overline{a}c + \overline{b} + \overline{abc}$ ("coordinate" degli 1 sulla tavola di verità)

MAXTERMINI \rightarrow [POS] termini somma in cui compaiono tutti i letterali $(\overline{a}+b) (\overline{a}+\overline{b}+c) (\overline{c})$ (" " degli 0 ")")

FORMA CANONICA

tutti i termini hanno tutti i letterali

SOP - tutti mintermini POS - tutti maxtermini

procedimento

① forma normale

$$\overline{a} + \overline{b}c$$

$$(\overline{a}+c) (\overline{b}+c)$$

② SOP

moltiplico i termini a cui manca

la variabile x per $(x + \bar{x})$

$$\overline{a} (\overline{b}+\overline{b}) (\overline{c}+\overline{c}) + \overline{b}c (\overline{a}+\overline{a})$$

$$(\overline{a}+c + b\overline{b}) (\overline{b}+c + a\overline{a})$$

③ proprietà distributiva

$$\overline{a}bc + \overline{a}b\overline{c} + \overline{a}\overline{b}c + \overline{a}\overline{b}\overline{c} + \overline{a}\overline{b}c + \overline{a}\overline{b}\overline{c}$$

$$(\overline{a}+b+c) (\overline{a}+\overline{b}+c) (\overline{a}+\overline{b}+c) (\overline{a}+\overline{b}+c)$$

④ elimino termini ridondanti

$$\overline{a}bc + \overline{a}b\overline{c} + \overline{a}\overline{b}c + \overline{a}\overline{b}\overline{c} + \overline{a}\overline{b}c$$

$$(\overline{a}+b+c) (\overline{a}+\overline{b}+c) (\overline{a}+\overline{b}+c)$$

ALL-NAND/ALL-NOR

$$\Rightarrow \cdot \text{NOT } \overline{a} = \overline{\overline{a}} \quad \cdot \text{AND } \overline{ab} = \overline{\overline{a}b} = \overline{\overline{a} \cdot b} = \overline{\overline{a}} \cdot \overline{b}$$

$$\Rightarrow \cdot \text{NOT } a = \overline{a+a} \quad \cdot \text{AND } ab = \overline{\overline{a}+\overline{b}} = \overline{\overline{a} \cdot \overline{b}} = \overline{a} + \overline{b}$$

portare a ALL-NAND/ALL-NOR

NAND - più SOP
NOR - più POS

① semplifico l'espressione come più mi conviene

$$\overline{z} + xy$$

$$(\overline{z}+x) (\overline{z}+y)$$

② doppia negazione (se ne svilupperà una sola)

$$\overline{\overline{z} + xy} =$$

$$(\overline{\overline{z}+x}) (\overline{\overline{z}+y})$$

$$= z \cdot \overline{xy} \underset{\text{NAND}}{\text{-NAND}} = z \text{ NAND } (x \text{ NAND } y)$$

$$= (\overline{\overline{z}+x}) + (\overline{\overline{z}+y}) = (\overline{z}+x) \text{ NOR } (\overline{z}+y) = (\overline{z} \text{ NOR } x) \text{ NOR } (\overline{z}+y) =$$

$$= ((\overline{z}+z) \text{ NOR } x) \text{ NOR } ((\overline{z}+z) \text{ NOR } y) = ((z \text{ NOR } z) \text{ NOR } x) \text{ NOR } ((z \text{ NOR } z) \text{ NOR } y)$$

↓ dipende, se il complemento è scelto si ferma alla riga sopra,
se no, si doppia

IMPORTANTE!!

ALL-NAND A DUE PORTE

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

① rendo NAND le due singole somme con la doppia negazione

$$\left(\overline{AB} \cdot \overline{\overline{A}\overline{B}} \right) \cdot \left(\overline{C\overline{D}} \cdot \overline{\overline{C}\overline{D}} \right)$$

② la moltiplicazione centrale si può rendere NAND con una doppia negazione e duplicazione

$$(\overline{AB} \cdot \overline{\overline{A}\overline{B}}) \cdot (\overline{C\overline{D}} \cdot \overline{\overline{C}\overline{D}}) \cdot (\overline{CD} \cdot \overline{\overline{C}\overline{D}})$$

SINTESI DI UN CIRCUITO COMBINATORIO

da descrizione verbale \rightarrow circuito

① (se necessario) ricavare un'espressione booleana

② tavolo di verità

③ mappe di Karnaugh per minimale

④ disegno

ANALISI DI UN CIRCUITO COMBINATORIO

da circuito \rightarrow descrizione verbale

① ricavare espressioni booleane delle uscite

② tavola di verità

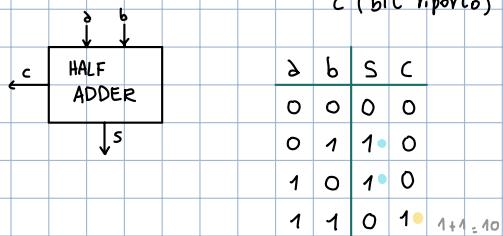
③ mappe di Karnaugh

④ verifica dell'ottimalità

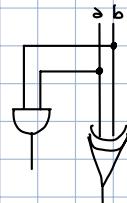
ADDITIONATORE

HALF ADDER

Somma 2 bit output: s (bit somma)
 c (bit riporto)

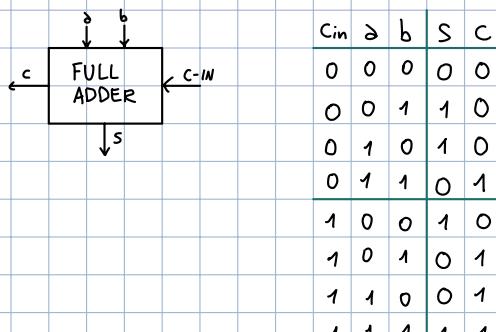


$$\begin{aligned} c &= ab \\ s &= a \oplus b \end{aligned}$$



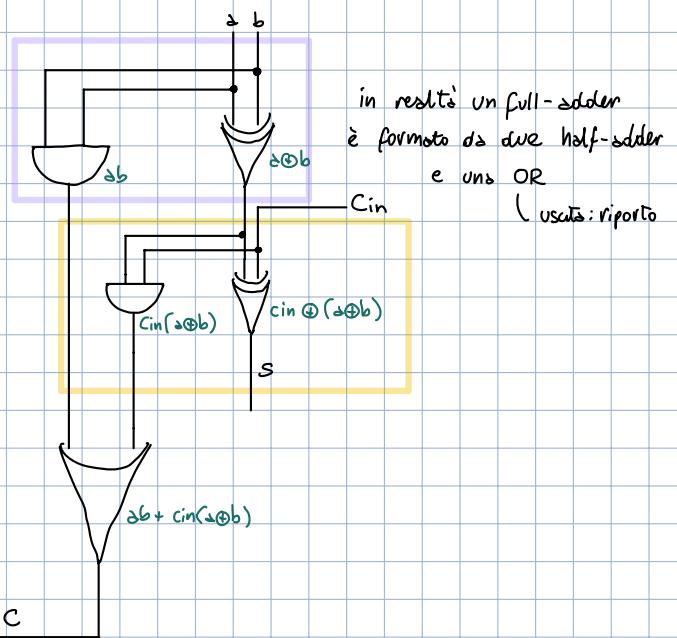
FULL ADDER

Somma 3 bit (anche un carry in entrata)



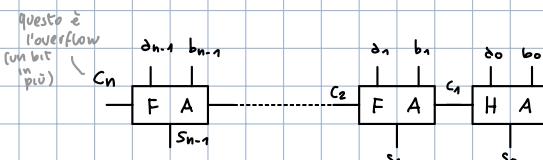
$$s = c_{in} \oplus (a \oplus b)$$

$$c = ab + c_{in}(a \oplus b)$$



RIPPLE-CARRY ADDER

un half-adder iniziale e vari full-adder

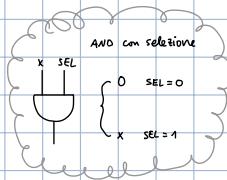


ADDER IN CA2

- in CA2 ci sono anche i numeri negativi

per ottenere $A - B$ (processo CA2)

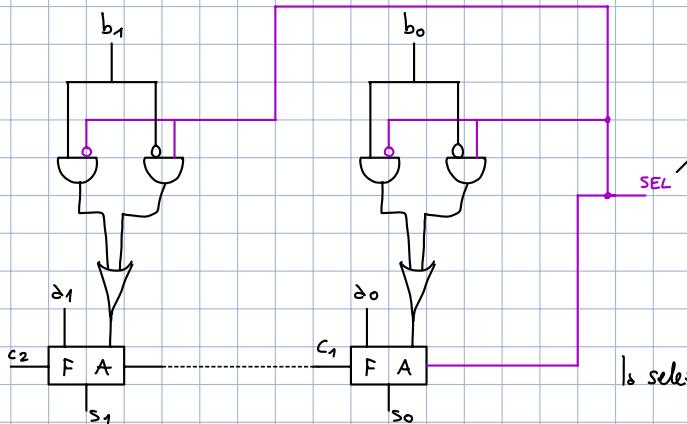
$$A - B = A + (-B) = A + (\bar{B} + 1)$$



quindi, si sostituisce il primo half-adder con un full-adder con un SEGNALE DI SELEZIONE

SERVONO, per la SOMMA $\rightarrow B$ e 0

DIFFERENZA $\rightarrow \bar{B}$ e 1

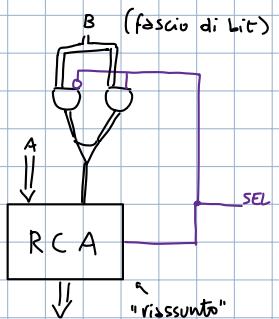


SEL = 1 passa \bar{B} (diff.)

SEL = 0 passa B (somma)

(perché SEL e B sono compensati
in due parti diverse, quindi: $SEL \equiv B$
e $SEL \equiv \bar{B}$)

La selezione di b è fatta da un MUX



ESERCIZIO

- Un circuito combinatorio produce la somma binaria di due numeri binari a 2 bits X_1X_0 e Y_1Y_0 . Outputs del circuito sono la somma S_1S_0 e il riporto C .

- 1 Fornire la tabella di verità del circuito
- 2 Progettare il circuito usando full adders

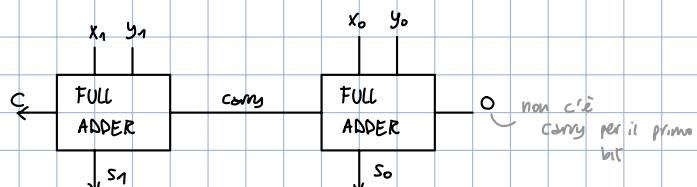
- ① Scrivo le combinazioni e calcolo somma e riporto

X_1	X_0	Y_1	Y_0	C	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	0	0	0
1	0	0	0	1	0	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

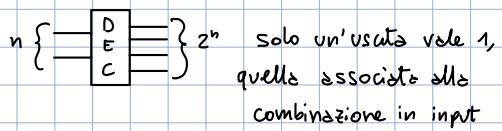
mi conviene
mettere primo il carry
perciò è come se
 $C S_1 S_0$ fosse
un numero a
3 bit

- ② Sono 2 bit \rightarrow 2 FA

con uscita somma tra i bit e con carry che si propaga



DECODIFICATORE e CODIFICATORE



DEC STANDAR

x_1	x_0	u_3	u_2	u_1	u_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

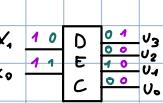
$$U_0 = \bar{x}_1 \bar{x}_0$$

$$U_1 = \bar{x}_1 x_0$$

$$U_2 = x_1 \bar{x}_0$$

$$U_3 = x_1 x_0$$

con matrice di AND



COD STANDAR

x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$y_1 = x_3 + x_2$$

$$y_0 = x_1 + x_3$$

Solo casi in cui un ingresso = 1

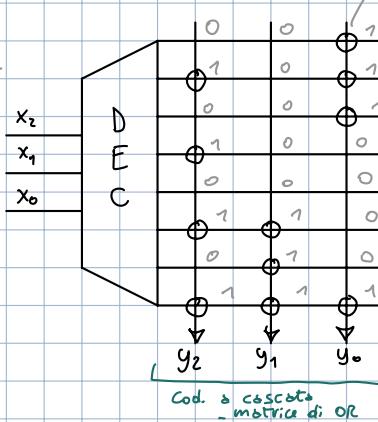
matrice di OR

ROM decodificatore + codificatore a cascata

le uscite del decoder sono MINTERMINI - utile per le canoniche

entrate

x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	1	0
1	1	1	1	1	1



$$y_0 = 1 \text{ se } x_2 x_1 x_0 = 000 (\bar{x}_2 \bar{x}_1 \bar{x}_0)$$

quindi, per esercizi con ROM

① tavola della verità

② il ROM ha come linee di ingresso gli n input, e 2^n linee di uscita (struttura della tavola di verità)

③ tiro giù tutte righe verticali quante sono le uscite che devo rappresentare

④ sugli "incroci", metto pallini vuoti dove ci sono gli 1 sulla tavola di verità. (+ porte OR per essere sicuri)

PLA realizza funzioni minimali (SOP)

- matrice di AND per i termini prodotto delle minimoli
- matrice di OR per sommarli

① tavola di verità

x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	1	0
1	1	1	1	1	1

② mappe di Karnaugh per minimoli

$$y_2 = x_0$$

$$\begin{array}{|c|c|c|c|} \hline x_2/x_1 & 00 & 01 & 11 & 10 \\ \hline x_2 \\ \hline \end{array}$$

$$0 \quad 0 \quad 0 \quad 0 \quad 0$$

$$1 \quad 0 \quad 1 \quad 1 \quad 1$$

$$y_1 = x_2 x_1 + x_2 x_0$$

$$x_2/x_1$$

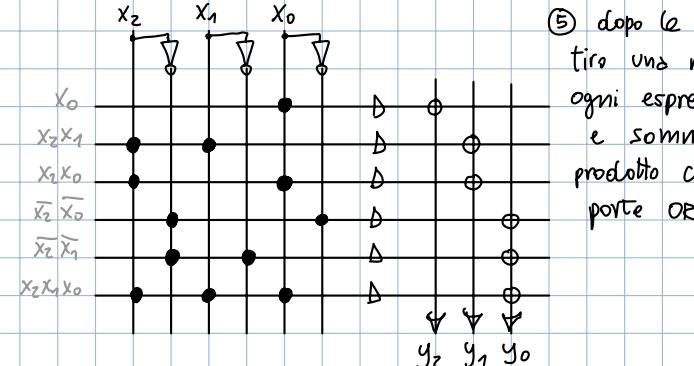
$$0 \quad 1 \quad 1 \quad 0 \quad 1$$

$$1 \quad 0 \quad 0 \quad 1 \quad 0$$

$$y_0 = \bar{x}_2 \bar{x}_0 + \bar{x}_2 \bar{x}_1 + x_2 x_0$$

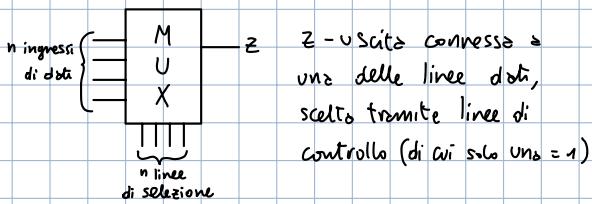
③ scrivo tutte le variabili che mi servono, complementate e non (righe verticali)

④ tiro righe orizzontali per tutti i termini prodotto con pallini pieni e porte AND

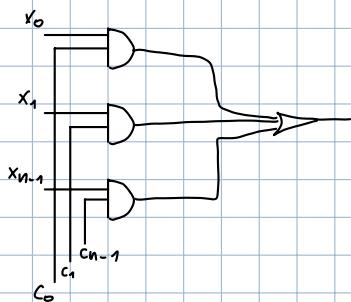


⑤ dopo le porte AND, tiro una riga verticale per ogni espressione e sommo i termini prodotto con pallini vuoti e porte OR

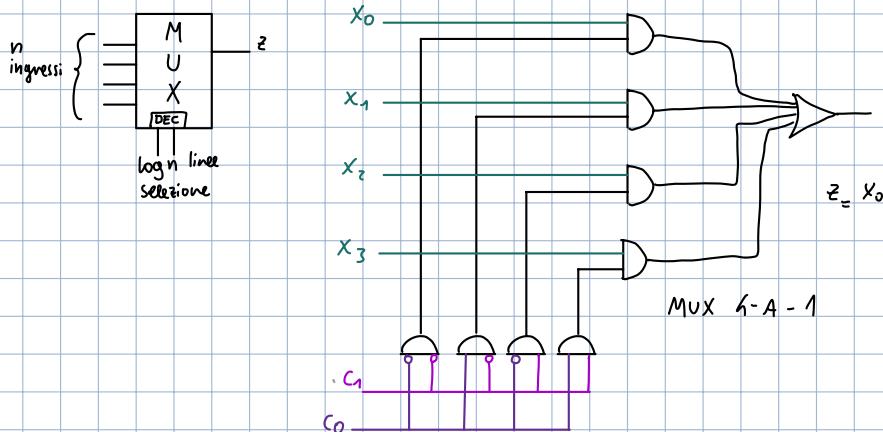
MULTIPLEXER



è un insieme di AND che confluiscono in una OR



in realtà! di solito si usa con un decodificatore



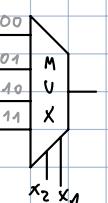
$$Z = \sum m(0, 1, 2, 3)$$

x_0	\bar{c}_1	\bar{c}_0	x_1	\bar{c}_1	c_0	x_2	c_1	\bar{c}_0	x_3	c_1	c_0
0	0	0	0	0	0	1	0	1	1	1	1

ESERCIZI CON MUX

- MUX "il più grande possibile"
 - input come linee di selezione
 - valori di f sulle linee dati

x_2	x_1	f
0	0	0
0	0	1
0	1	1
1	0	1
1	1	0



- con sottoinsieme di segnali di controllo (MUX più piccolo)

① tavola di verità

MUX 6-A-1

x_2	x_1	x_0	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

② scelgo le linee di selezione

(scego quelle più a sx)

il numero di linee è $\log n$

con n numero entrate

(dato dell'esercizio)

quindi MUX 6-A-1 → 2 linee

linee: x_2 , x_1

③ divido la tavola di verità in "sezioni"

(6 - dato dell'esercizio)
in base alle linee



④ guardo i valori di f in quelle sezioni

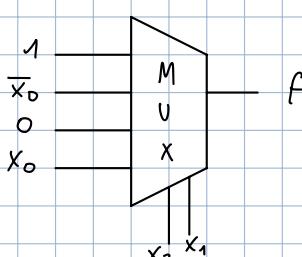
- Se è sempre 0 o 1, scelgo quello

- sempio, derivo la funzione dagli input (es. $x_2x_1 10 = \bar{x}_0$)

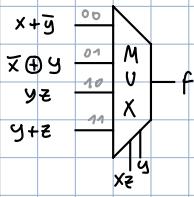
- al limite, trovo i mintermini

x_2	x_1	f
00		1
01		\bar{x}_0
10		0
11		x_0

⑤ Compongo il MUX



- da MUX a espressione



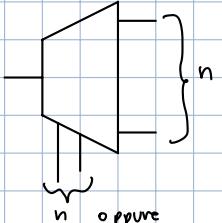
ricordiammo che il MUX è formato da varie AND tra linee di selezione ed espressione corrispondenti e una OR che le unisce, e che $X+Y$ corrisponde a 00, $\bar{X}+Y$ a 01 ecc.

quindi, compongo le combinazioni

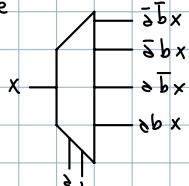
$$\bar{x}\bar{z}y(x+y) + \bar{x}\bar{z}y(\bar{x}+y) + x\bar{z}\bar{y}(yz) + x\bar{z}y(y+z) \quad (\text{questo andrà poi semplificato})$$

DEMULTIPLEXER

Sceglie l'uscita a cui collegare l'unico ingresso



USCITE



versione con dec: logn

CIRCUITI SEQUENZIALI

SET-RESET

set-reset (quindi 10 set 01 reset)

s	r	y	y	y	s	r
0	0	y(t-1)	0	0	0	δ
0	1	0	0	1	1	0
1	0	1	1	0	0	1
1	1	— non ammesso	1	1	δ	0

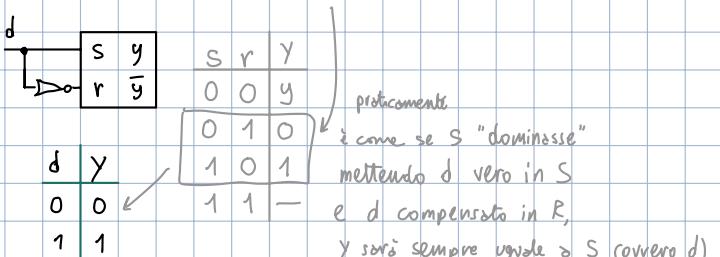
JK

una sorta di SR + toggle

j	k	y	y	y	j	k
0	0	y	0	0	0	δ
0	1	0	0	1	1	δ
1	0	1	1	0	δ	1
1	1	—	1	1	δ	0

DELAY

derivato dai valori discordi dell'SR



quindi, negli esercizi, per trovare Y

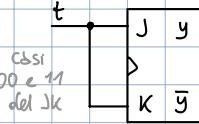
copiare il valore di d

(al contrario, per trovare d copiare
i valori di y(t+1))

TOGGLE

se è sul fronte di salita e l'input è 1, complemento
lo stato precedente

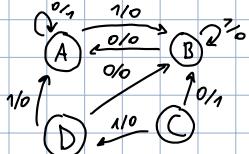
t	y	y	y	t
0	y	0	0	0
1	—	0	1	1
1	—	1	0	1
1	—	0	0	0



DIAGRAMMI DI STATO

MEALY

"palle" con stati,
archi con input/output



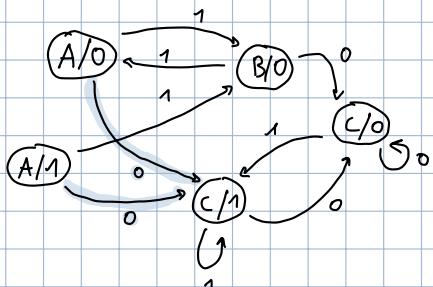
MOORE

le uscite sono associate agli stati · input sugli archi

- quindi ci sono più "palle" per stato - una per ogni

uscita che ha lo stesso stato con diverse uscite va agli stessi stati

(se A con input 1 va in B, allora sia A/1 che A/0 andranno in B con input 1)



ANALISI di un circuito sequenziale

① espressioni booleane delle funzioni di eccitazione (ingressi dei FF) e delle uscite

② tavola degli stati futuri:

③ stati presenti:

$\geq s_x$ input e bit di stato, $\geq d_x$ valori dei ff e uscite

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

④ ricavo gli stati futuri

(da stati presenti con valori FF)

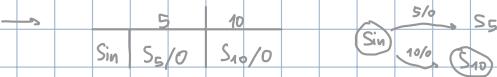
x	y ₁	y ₀	d ₁	d ₀	z	y ₁	y ₀
0	0	0	0	1	0	0	1
0	0	1	1	0	1	1	1

⑤ codifica degli stati e disegno dell'automa (tavella e/o Mealy/Moore)

SINTESI di un circuito sequenziale (dalla descrizione verbale)

① individuazione di input, output e stati

es: macchinetta distributrice: input 10c, 5c output: 1 se 20c stati: quanti cent



② codifiche: degli stati (guarda quanti stati ho e vedo quanti bit mi servono) e, se necessario, di input e output

③ tavola degli stati futuri:

④ input e stati presenti $\Rightarrow s_x$ stati futuri e output $\Rightarrow d_x$

dalla tabella/disegno
(dallo stato x, quando ha quelli input
va in y con uscite...)

⑤ scelta dei FF - valori derivati da confronto tra stati presenti e futuri secondo le tavole inverse

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

/ delay n = il valore
di y

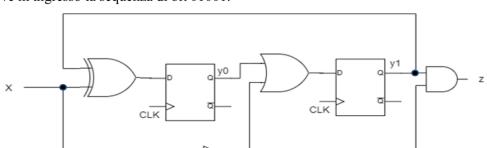
per inizializzazione $d_0 = 0 \Rightarrow 1$, toggle = 1

⑥ ricavo espressioni booleane delle funzioni di eccitazione (ingressi FF) e delle uscite
(basandosi su x, y_1, y_0) - con Karnaugh o deducendole

⑦ disegno del circuito

DIAGRAMMA TEMPORALE dato un circuito e un input

Esercizio 6 (2 punti) Si disegni il diagramma temporale del seguente circuito a partire dallo stato 01001 quando riceve in ingresso la sequenza di bit 01001.

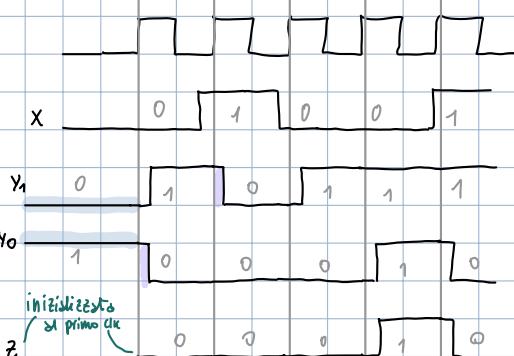


① ricavo le espressioni necessarie
(qui y_0, y_1 e z)

Sono FF delay, quindi mantengono i valori

$$y_0 = x \oplus y_1 \quad y_1 = y_0 + \bar{x} \quad z = y_1 \bar{x}$$

② preparo il diagramma temporale



5 input - 5 colpi di clock

scrivo l'input - deve prendere il valore necessario poco prima
del colpo di clock, così che, al clock, sia "utilizzabile" / registrato

o sul colpo stesso
dipende dalla persona a
cui chiedi

scrivo y_0, y_1 nello stato da cui partono (01)

ed ogni colpo di clock, vedo come cambiano i loro valori (dalle espressioni)
e scrivo i nuovi valori con un pochino di ritardo

Con un automa già progettato

- guardo la tavola di verità o la tabella dell'automa per vedere come si comportano i valori con un certo input
(stato presente + input = stato futuro con uscita)

MINIMIZZAZIONE DI UN AUTOMA

	0	1
A	G/00	C/01
B	G/00	D/01
C	D/10	A/01
D	C/10	B/01
E	G/00	F/01
F	F/10	E/01
G	A/01	F/11
H	D/10	A/01

① Vedo se ci sono stati equivalenti (stessi stati futuri e output) e li elimino

H è eq. a C, quindi lo elimino

② Costruzione della tabella triangolare

③ verticale - tutti gli stati tranne il primo

orizzontale - tutti gli stati tranne l'ultimo

B	CD
C	X X
D	X X CD AB
E	CF DF X X
F	X X DF CF AE BE X
G	X X X X X X
A	B C D E F

④ se due stati hanno output diversi, metto una X

se hanno lo stesso output: Scrivo > coppie i due stati futuri diversi

(es. B e A hanno G uguali, ma C e D diversi - scrivo CD)

⑤ se due stati si riferiscono a vicenda, non c'è bisogno di scrivere

• (es. C e D - sono uguali se C e D sono uguali - dnh?)

⑥ osservare le relazioni tra stati

$A=E$ se $C=F$

$C=F$ se $D=F$ e $A=E$

$B=E$ se $D=F$

$D=F$ se $C=F$ e $B=E$

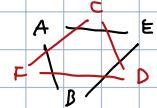
faccio uno schema con le coppie legate

$B=A$ se $C=D$

si richiamano tra loro

quindi $A=E=B$ e $C=D=F$

nuovi stati $A' = \{A, E, B\}$ $C' = \{C, D, F\}$, G



se ci sono tutti i rami

(rel. di eq. - transitiva, (ref), simmetrica)
allora sono equivalenti

⑦ nuova tabella (per gli stati futuri, sostituire i nomi dei nuovi stati)

	0	1
A'	G/00	$C'/01$
C'	$C'/10$	$A'/01$
G	$A'/01$	$C'/11$

REGISTRI

- insiemi di Flip-Flop che hanno due operazioni:
 - il segnale LOAD/ENABLE controlla le operazioni

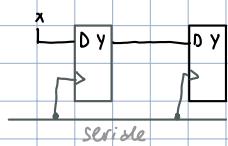
MEMORIZZAZIONE - mantenimento di un'informazione

CARICA - nuova informazione

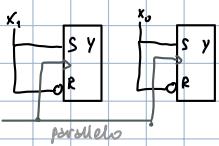
PARALLELO - tutti i FF insieme

SERIALE - da un FF all'altro

usiamo FF D

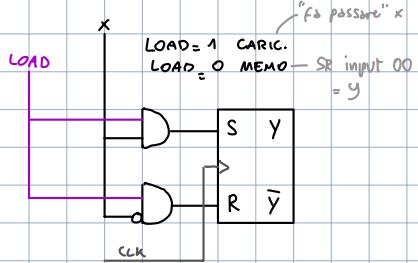


oppure "riciclo": Δ con gli SR

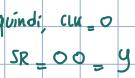
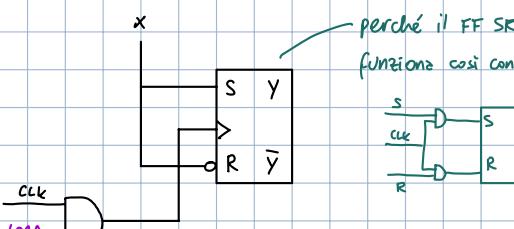


LA LINEA DI CONTROLLO LOAD PUÒ STARE IN DUE POSTI

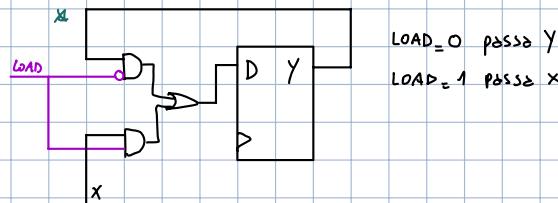
• SULL'INPUT



• SUL CLOCK



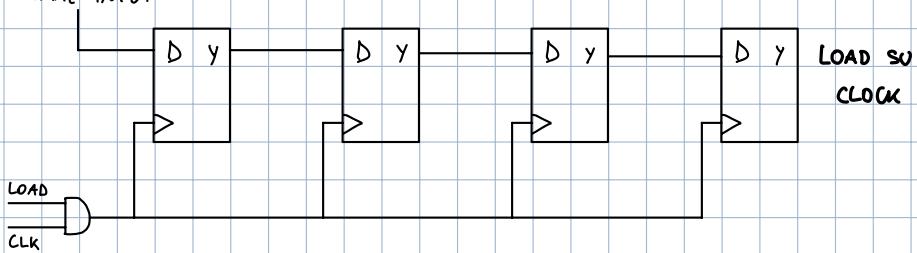
con FF D (load su CLK è uguale)



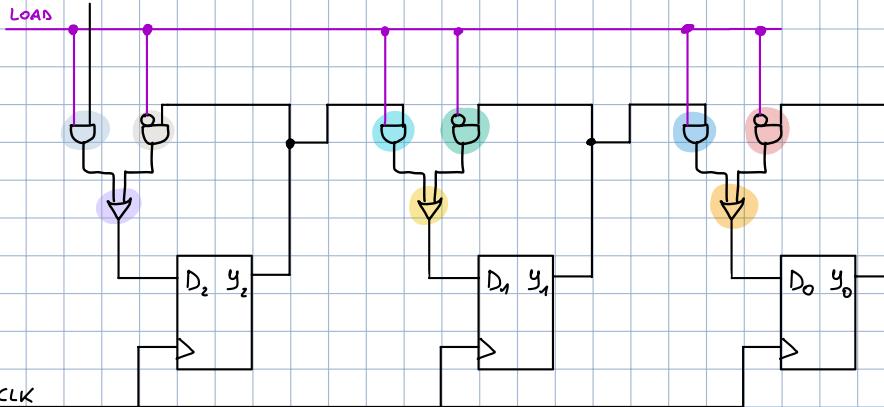
SHIFT REGISTERS



SERIAL INPUT



SERIAL INPUT



— Come visto su *

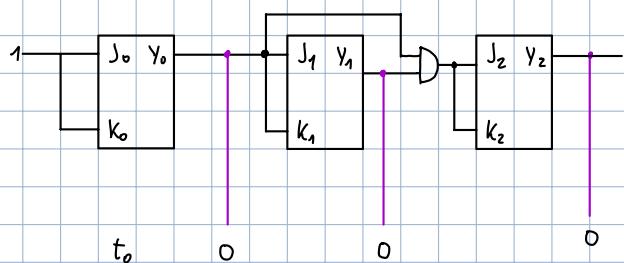
— per tutti: tranne che il primo, l'"input" (x) è lo stato di quello precedente —

CONTATORI

contatori mod 2^n contano fino a $2^n - 1 \rightarrow$ il numero di FF (e quindi di bit) è $\log_2 n$

CONTATORE MOD8 DI POS-EDGE DI CLOCK

Conta fino a 7



per aggiungere
Flip-Flop, mettere
una porta AND
con tutti i precedenti
(prendi quelli prima
dell'uscita della AND prima)

t_0	ingressi	ingressi	ingressi
	FF sono 11	FF sono 00	FF sono 00
- toggle -	- mantenimento -	- mantenimento -	
1	0	0	0
" "	ingressi: FF 11 - toggle	ingressi: 00 mant.	
0	1	0	

VERILOG™ (livello Massini, non Pontarelli)

OPERATORI

\sim not
 $\&$ and $\sim \&$ nand
 \wedge xor
 $|$ or \sim nor

ASSIGN - operatore di

ASSEGNAZIONE CONTINUO

- ricalcola l'uscita ogni volta che gli ingressi cambiano

? : - operatore di

ASSEGNAZIONE CONDIZIONALE

Seleziona, sulla base di una prima espressione, la seconda o la terza.

condizione = {
 1 → prima esp.
 0 → seconda esp}

MUX DA SOTTO
A SOPRA

SINTASSI BASE

inizializz. modulo
module nomemodulo (input logic a, b, c ,
output logic y);

assign $y = \sim a \& b \& c |$

$a \& b \& c;$

endmodule

bus multibit

input logic [3:0] a

come lo slicing, ma

l'inizio è compreso

$a[3:0] = a[3], a[2],$

$a[1], a[0]$

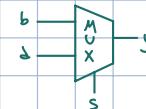
- ordinata arbitrariamente

ma ord. comuni: little-endian [0:N-1]

big-endian [N-1:0]

modulo per sommare (o operare) su tutti i bit

module add8 (input logic [0:7] a ,
output logic y);



assign $y = s ? a[3] :$

$a[2] \quad s = 0$

endmodule

assign $y = \sim a;$

→ più facile di fare

endmodule

assign $y = a[7] \& a[6] \& ...$

always @ (sensitivity list) istruzione

l'istruzione viene eseguita quando avviene ciò che è scritto nella sens. list

* con gli always si usano

= NON BLOCCANTE - sequenziale
= BLOCCANTE - combinatorico

↓
always_ff
always_latch
always_comb

FLIP-FLOP D sens. fronte salito

module flopD (input logic clk,
input logic [3:0] d,
output logic [3:0] q);

always_ff @ (posedge clk)
 $q \leftarrow d;$

endmodule

casez (dato)

supporta anche
? come un \exists

casez (a) \exists

$a'b1?? : y = b$

$a'b0?1 : y = c$

* sempre dentro un always

Variabili interne

se bisogna fare molte operazioni, si possono definire variabili interne che non sono né entrate né uscite

module grande (input logic a, b, c ,

output logic y);

define internamente

assign $p = a \& b;$

assign $j = b^1c;$

assign $y = p | j;$

endmodule

Module neg (input logic [3:0] a ,
output logic [3:0] y);

always_comb

$y = \sim a;$

endmodule

case (dato) :

Sceglie un'operazione in base al valore di dato
Si può usare default per definire le uscite nei casi non specificati
(es. se un valore compare 3 volte, invece di specificare 3 casi scelgo quello come default e specifico tutti gli altri)

* sempre dentro un always

module mux 4-3-1 (input logic a, b, c ,

input logic [0:7] x

output logic f);

always_comb

case (x)

$2'b01 : f = c|d;$

$2'b10 : f = 1'b1;$

default: $f = a \& b;$

endcase

endmodule

NUMERI

N'B valore

dimensione in bit

lettera che indica base

'b - binario 'd - decimale

'o - ottaglio 'h - esadecimale

• if, else → sempre dentro un ALWAYS

if (istruzione) conseguenza;

else if (istruzione) conseguenza;

else conseguenza;

begin/end

è necessario quando ci sono più istruzioni dentro un always

(MA non se c'è solo un if/else, perché corrisponde ad una singola istruzione)

in caso però meglio metterlo che non

REGISTRI

• RESET

SINCRONO

```
RESET
module flop (input logic clk,
             input logic reset,
             input logic [3:0] d,
             output logic [3:0] q);
endmodule
```

```
always_ff @ (posedge clk)
  if (reset) q <= 4'b0;
  else q <= d;
```

endmodule

RESET ASINCRONO

```
module flop (input logic clk,
             input logic reset,
             input logic [3:0] d,
             output logic [3:0] q);
```

```
always_ff @ (posedge clk, posedge reset)
  if (reset) q <= 4'b0;
  else q <= d;
```

endmodule

ABILITAZIONE

```
module flopnr (input logic clk,
                input logic reset,
                input logic en,
                input logic [3:0] d,
                output logic [3:0] q);
```

```
always_ff @ (posedge clk, posedge reset)
  if (reset) q <= 4'b0;
  else if (en) q <= d;
```

endmodule

MULTIPLY

```
module sinc (input logic clk,
             input logic d,
             output logic q);
```

logic m1;



```
always_ff @ (posedge clk)
begin
  contemporaneamente
  m1 <= d;  d va in m1 e m1 in d
  q <= m1;
end
```

endmodule

LATCH D

```
module latch (input logic clk,
               input logic d,
               output logic q);
```

```
always_latch
  if (clk) q <= d;
  else q <= d;
```

endmodule

LATCH SR

```
module latchsr (input logic s, r,
                 input logic clk,
                 output logic q, q_bar);
```

```
always_latch
  if (clk) begin
    case ({s, r})
      2'b00: {q, q_bar} <= {q, ~q};
      2'b01: {q, q_bar} <= 2'b01;
      2'b10: {q, q_bar} <= 2'b10;
      default: {q, q_bar} <= 2'bxx;
    end
  endcase
endmodule
```

DOMANDE ESAMI:

① FF T reset asincrono ✓

(2021 02 16 A)

② shift register 8 bit con reset sincrono ✓

(2021 02 16 B)

③ buffer tristate - NON L'ABBIAMO
FATTO

④ MUX 8-4-1 ✓

(2022 01 11 A e B)

⑤ Contatore mod₁₃ ✓

(2022 02 01 A mod₁₁ B)

MASSIMI:

① 2 FF delay (2023 01 18 A e B) ✓

② FF reset asincrono e segnale enable ✓

just in case

```
module buffer (input logic x,
               input logic en,
               output tri y);
```

assign y = en ? x : 1'bz

endmodule

① FLIP-FLOP T CON RESET ASINCRONO

```
module fftr (input logic clk,
              input logic res,
              input logic t,
              output logic q);
```

always_ff @ (posedge clk, posedge res)

```
if(res) q <= 1'b0;
else if(t) q <= ~q;
else q <= q;
```

alternativa

```
endmodule
```

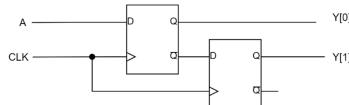
else q <= q ^ t

XOR, pendente

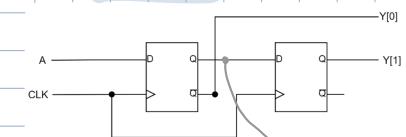
t	q	Q
0 0	0	0
0 1	1	1
1 0	1	0
1 1	0	0

② MASSINI 2021 18 GEN B

Esercizio 3 (4 punti) Descrivere in SystemVerilog il seguente circuito:



Versione A



input uguali

logic q;

always_ff @ (posedge clk)

begin

Y[0] <= d;

Y[1] <= ~Y[0]

end

endmodule

logic q;

always_ff @ (posedge clk)

begin

q <= d

y[0] <= ~q

y[1] <= q

end

endmodule

③ FF reset asincrono e segnale enable

```
module flop (input logic clk,
              input logic res,
              input logic en,
              input logic d,
              output logic q);
```

always_ff @ (posedge clk, posedge res)

if(res) q <= 1'b0;

else if(en) q <= d;

alternativa

q <= d & en

endmodule

④ shift register 8 bit con reset sincrono

```
module shift (input logic clk,
              input logic res,
              input logic d,
              output logic q);
```

logic [7:0] sh-reg;

always_ff @ (posedge clk)

begin

if(res) sh-reg <= 8'b0

else sh-reg <= {sh-reg[6:0], d};

end

assign q = sh-reg[7];

l'uscita è solo il primo bit (quello "shifted out")

endmodule

versione RS

{d, sh-reg[7:1]}

e uscita sh-reg[0]

concatenazione

7 bit del registro

+ 1 dell'input

y7 ← d
y0 ← d

shiftano e spingono via il msb

LEFT SHIFT

⑤ contatore Mod13

Conto fino a 12 → 1100

```
module count13 (input logic clk,
                  output logic [3:0] y = 4'b0);
```

always_ff @ (posedge clk)

begin

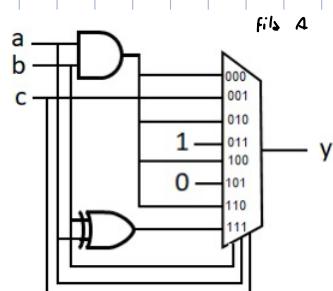
if(y == 4'b1100) y <= 4'b0

else y <= y+1

end

endmodule

④



module mux (input logic clk,

input logic a,b,c,

output logic y);

always_comb

case({a,b,c})

3'b001: c

3'b011: 1'b1

3'b101: 1'b0

3'b111: a'b1

default: a & b

endcase

endmodule

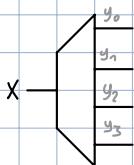
```
• module jk (input logic clk,
    input logic j,k,
    output logic q);
```

```
    always_ff @ (posedge clk)
    begin
        case ({j,k})
            2'b01: q <= 1'b0;
            2'b10: q <= 1'b1;
            2'b11: q <= ~q;
            default: q <= q;
        endcase
    end
```

endmodule

DEMUX 1-A-4

```
module demux (input logic x, a,b,
    output logic y3,y2,y1,y0);
```



```
assign y0 = ( {a,b} == 2'b0 ) ? x : 1'b0;
assign y1 = ( {a,b} == 2'b01 ) ? x : 1'b0;
assign y2 = ( {a,b} == 2'b10 ) ? x : 1'b0;
assign y3 = ( {a,b} == 2'b11 ) ? x : 1'b0;
```

endmodule

FULL-ADDER

```
module adder (input logic cin, a,b,
    output logic s, cout);
```

always_comb

```
begin
    s = (a ^ b) ^ cin;
    cout = a & b | ((a ^ b) & cin);
end
```

posso anche mettere
una variabile "di mezzo"

$a \wedge b$

endmodule

FSM Moore

```
Module moore (input logic x,
    input logic clk,
    output logic z);
```

reset)	
0	1
A	C/0 B/1
B	B/0 D/0
C	A/1 D/1
D	B/1 A/1

A1, B0/B1, C0, D0/D1	
001	010 100 110
	011 111

typedef enum logic {A1 = 3'b001, B0 = 3'b010, B1 = 3'b011, C0 = 3'b100, D0 = 3'b110, D1 = 3'b111} stato;

```
state [2:0] yp, [2:0] yf;
```

always_ff @ (posedge clk)

yp <= yf;

always_comb

```
casez (yp)
    A1 : yf = x ? B1.C0;
    3'b01? : yf = x ? D0.B0;
    C0 : yf = x ? D1.A1;
    3'b11? : yf = x ? A1.B1;
endcase
```

assign z = yp[0];

assign z = ((yp == D1) | (yp == B1));

	0	1	
S_{in}	$S_{in}/0$	$S_{1/0}$	$S_{in}0, S_{1/0}, S_{in}/0, S_{1/0}^3$
S_1	$S_{in}/0$	$S_{11}/0$	$S_{1/1}$
S_{11}	$S_{110}/0$	$S_{11}/0$	000
	$S_{in}/0$	$S_1/1$	010
S_{110}			011

module Moore [input logic x ,
output logic z];

typedef enum logic [2:0] { $S_{in} = 3'b000, S_{10} = 3'b010, S_{11} = 3'b011, S_A = 3'b100, S_B = 3'b110$ } state;
state $y_p, y_f;$

always_ff @ (posedge clk)
 $y_p \leftarrow y_f;$

always_comb
case z (y_p)

$S_{in} : y_f = x ? S_{in};$
 $3'b01? : y_f = x ? S_A : S_{in};$
 $S_A : y_f = x ? S_A : S_B;$
 $S_B : y_f = x ? S_{11} : S_{in};$
default: $y_f = 3'bX$

endcase

assign $z = y_p[0]$

endmodule

MEALY $\{S_{in}, S_1, S_{11}, S_{110}\}$

assign $y =$