Corso Luigi Einaudi, 55/B - Torino

Appunti universitari
Tesi di laurea
Cartoleria e cancelleria
Stampa file e fotocopie
Print on demand
Rilegature

NUMERO: 2083A - ANNO: 2018

A P P U N T I

STUDENTE: Massara Andrea

MATERIA: Elettronica dei sistemi digitali - prof. Zambon

Il presente lavoro nasce dall'impegno dell'autore ed è distribuito in accordo con il Centro Appunti.
Tutti i diritti sono riservati. È vietata qualsiasi riproduzione, copia totale o parziale, dei contenuti inseriti nel presente volume, ivi inclusa la memorizzazione, rielaborazione, diffusione o distribuzione dei contenuti stessi mediante qualunque supporto magnetico o cartaceo, piattaforma tecnologica o rete telematica, senza previa autorizzazione scritta dell'autore.

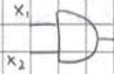
ATTENZIONE: QUESTI APPUNTI SONO FATTI DA STUDENTI E NON SONO STATI VISIONATI DAL DOCENTE.
IL NOME DEL PROFESSORE, SERVE SOLO PER IDENTIFICARE IL CORSO.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 106

CAPITOLO 1 - DIGITAL DESIGN

VARIABILI E FUNZIONI

- AND



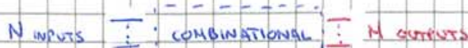
$X_1 \cdot X_2$

- OR

$X_1 + X_2$

- NOT

- Circuiti combinatori vs sequenziali



Queste sono forme generali di circuiti logici combinatori e sequenziali!

Le mie funzioni circuitali possono essere espresse mediante tavole di verità e vedere dove ho gli '1': infatti posso esprimere $f(\ldots)$ come sommatoria dei minterm che creano '1' nella tavola di verità (copertura degli '1'). In alternativa copro gli '0' con i maxterm

$$f(x_1, x_2, x_3) = \sum m(1, 4, 5, 6)$$

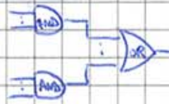
$$f'(x_1, x_2, x_3) = \sum m(0, 2, 3, 7)$$

$$f(x_1, x_2, x_3) = M_0 \cdot M_2 \cdot M_3 \cdot M_7$$

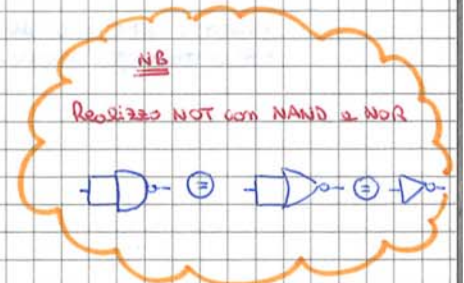
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Ho pertanto 2 modi equivalenti di progettare:

- SOMMA DI PRODOTTI

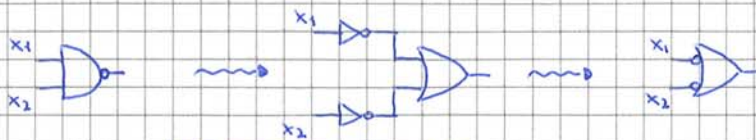


- PRODOTTO DI SOMME



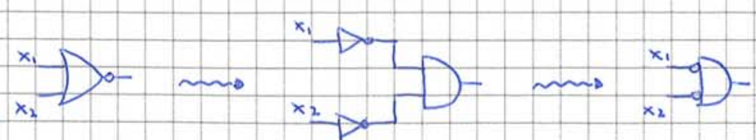
Corre in aiuto il teorema di De Morgan secondo cui:

1°)



$$\overline{x_1 \cdot x_2} = \overline{x_1} + \overline{x_2}$$

2°)



$$\overline{x_1 + x_2} = \overline{x_1} \cdot \overline{x_2}$$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 106

CAPITOLO 2 - VHDL: BASIC PRINCIPLES (COMBINATIONAL)

- IEEE 1164 - STD_LOGIC data type ⇝
  - 0 tensione LOW
  - 1 " HIGH
  - Z alta impedenza
  - \- don't care
  - L '0' debole } se c'è qualcosa che gli fa cambiare idea, quello
  - H '1' debole } vince
  - U caso di uninitialized (FF in cui non ho scritto)
  - X caso di conflitto forte
  - W " " " debole

  Non ho + solo 0 e 1!!!

- Regole:
  - il VHDL è case insensitive
  - "--" indica un commento
  - i nomi devono essere composti da codici alfanumerici e con underscore. Devono iniziare con una lettera e non terminare con underscore.

- Dichiarazione SIGNAL:

  ```
  SIGNAL signal_name: signal_type;
  ```
  ⇝ devo mettere le due fusi di ritornie!

  esempi:
  - INTEGER

    ```
    signal w : integer;
    ```
  - SIGNED e UNSIGNED (CA2) ⇝ usare use ieee.numeric_std.all

    ```
    signal y : signed (7 downto 0);
    ```
  - ENUMERATED (usate nelle macchine a stati)

    ```
    TYPE STATE_TYPE IS (A, B, C);
    SIGNAL Y : STATE_TYPE;
    ```

- Dichiarazione CONSTANT:

  ```
  CONSTANT nome: type := value
  ```

- Memorie e file:

  le memorie le compongo come combinazione di array:

  ```
  (1) TYPE word is STD_LOGIC_VECTOR (15 downto 0);
  (2) TYPE arr_type is ARRAY (0 to 31) of word;
  (3) SIGNAL MEM_A : arr_type;
  ```

- Modalità segnali:
  - IN: ingresso
  - OUT: uscita (userà <=)
  - INOUT: segnale usato come in e come out da altri
  - BUFFER: lo piloto io ma il valore di uscita lo uso internamente nella stessa entity!

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 7 di 106

Come fatto prima devo compattare la scrittura:

```
Y <= CONV_STD_LOGIC_VECTOR(2**(TO_INTEGER(S)), 8);
```

\# da convertire — # bit su cui scrivere l'uscita convertita

Prendo S, lo converto in INT. Lo elevo a potenza ottenendo il numero da riconvertire in STD-LOGIC-VECTOR su 8 bit

NOTA: quando scrivo VHDL per mandarlo a un sintetizzatore è inutile definire ritardi temporali perché il CAD funziona con i blocchi base che ha. L'espressione "after gate-delay" non è letta dal sintetizzatore che non costruirà il circuito con porte aventi quel delay.

NOTA: non conta + l'ordine come in C! Le operazioni sono eseguite tutte in concomitanza.

L'approccio che si usa è GERARCHICO, non FLAT:

il succo è creare i COMPONENT ⇝

```
COMPONENT MUX4TO1
    PORT ( w0, w1, w2, w3 : IN STD_LOGIC;
           S : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
           f : OUT STD_LOGIC );
END COMPONENT
```

} parte dichiarativa

```
BEGIN
Mux1: MUX4TO1 PORT MAP
      ( w(0), w(1), w(2), w(3) , S(1 DOWNTO 0), m(0));
      :
Mux4: MUX4TO1 PORT MAP
      ( w(12), w(13), w(14), w(15) , S(1 DOWNTO 0), m(3));
Mux5: MUX4TO1 PORT MAP
      (m(0), m(1), m(2), m(3), S(3 DOWNTO 2), f);
```

} parte esecutiva di collegamento

Il port map fa un'associazione posizionale, ossia metto i segnali nell'ordine della dichiarazione!! L'alternativa è farla per nom. e cioè, nell'esempio del MUX4TO1:

```
MUX4TO1 PORT MAP
( w0 => w(0), w1 => w(1), w2 => w(2) ...
```

Ne guadagno in leggibilità e nel fatto che l'ordine d'inserimento non conta +!!

Ma di fatto come lavora il simulatore VHDL dopo che ho scritto tutto ciò?

La modalità è detta EVENT DRIVEN: lavora evento per evento e ricalcola le uscite ogni qual volta che qualcuno tocca i suoi ingressi in modo che le uscite cambino. Dopodiché si rimette a dormire. Se poi il time delay è alto a lui non frega perché il tempo di simulazione non dipende dal tempo assoluto, bensì dal solo # di eventi analizzati dal simulatore.

↳ Va avanti evento per evento!!! È evidente che in assenza di eventi lui dorme!!

- Ritardi nelle porte

INERZIALE: se arriva un impulso + piccolo del tempo di propagazione, tale impulso viene cancellato

TRANSPORT: serve per modellizzare linee che trasportano senza gate

DELTA: OUTPUT <= INPUT after K ns

tutti gli impulsi di INPUT che hanno durata < di K ns vengono filtrati. Altrimenti li trovo in uscita ritardati di K ns.

Se nella scrittura VHDL non scrivo lo statement per il delay NON È VERO CHE il ritardo è nullo. Il sintetizzatore ne mette uno infinitesimo K con $K \to 0^+$. Impone dunque il + piccolo ritardo possibile in modo da non mettere le variazioni degli ingressi in contemporanea. Il simulatore sposta del minimo tempo il tutto per garantire la consecutio temporale degli eventi.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 7 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 106

Finora si sono visti metodi di sintesi che partono dalla tabella di verità.
È possibile sintetizzare un circuito partendo dall'equazione booleana e non passando dalla tavola? sì e ci viene in aiuto il teorema di espansione di Shannon:

∀ funzione booleana f:

$$f(w_1, w_2, \dots, w_n) = w_1' \cdot f(0, w_2, \dots, w_n) + w_1 \cdot f(1, w_2, \dots, w_n)$$
$$= w_1' \cdot w_2' \cdot f(0,0,\dots,w_n) + w_1 w_2 f(1,1,\dots,w_n) + w_1' w_2 f(0,1,\dots,w_n) + w_1 w_2' f(1,0,\dots,w_n)$$

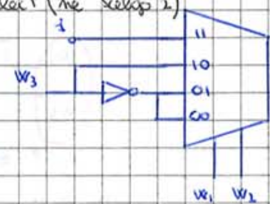
Riduco la funzione di un grado di complessità ogni volta che lo applico e procedendo così arrivo a ottenere un'espressione di f descrivibile con sole somme di prodotti ⇝ MUX

ES: $f = \overline{w_1}\,\overline{w_3} + w_1 w_2 + w_1 w_3$ usando solo MUX 4TO1 ⇝ 4 BIT ⇒ 2 di select (ne scelgo 2)

$$f = \overline{w_1}\,\overline{w_2}\, f_{\overline{w_1}\overline{w_2}} + \overline{w_1} w_2\, f_{\overline{w_1} w_2} + w_1 \overline{w_2}\, f_{w_1\overline{w_2}} + w_1 w_2\, f_{w_1 w_2}$$

(con $f_{\overline{w_1}\overline{w_2}} = \overline{w_3}$, $f_{\overline{w_1} w_2} = \overline{w_3}$, $f_{w_1\overline{w_2}} = w_3$, $f_{w_1 w_2} = 1 + w_3 = 1$)

$$= \overline{w_1}\,\overline{w_2}\,\overline{w_3} + \overline{w_1} w_2 \overline{w_3} + w_1 \overline{w_2} w_3 + w_1 w_2$$



• LUTs

Abbiamo detto che nella sintesi con il MUX metto a selezione gli ingressi (non tutti per forza) e dopodiché scrivo '1' e '0' in base al valore della funzione su quella riga. Se immagino di non sapere che sia un MUX, bensì una scatola nera all'interno della quale posso completare la tabella che sintetizza la funzione.
Risultato: sintetizzo qualsiasi funzione f con gli ingressi che ho.

ES: $f = w_1 f(1, w_2, w_3, w_4) + \overline{w_1} f(0, w_2, w_3, w_4)$

le realizzazioni dei cofattori le fa la LUT che in tal caso sarà su 3 bit.

ES: $f = \overline{s_1}\,\overline{s_0} w_0 + \overline{s_1} s_0 w_1 + s_1 \overline{s_0} w_2 + s_1 s_0 w_3$ uso solo LUT a 4 vie.

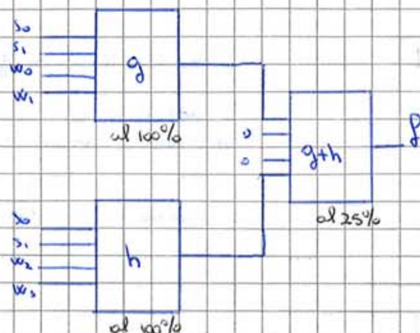
avendo solo 4 input a disposizione parto dall'inizio e riempio i 4 spazi ⇝ s_1, s_0, w_0 e passando al secondo, che ha i medesimi meno w_0, aggiungo w_1. Gli altri termini hanno altri input ma non avendo + spazio devo fermarmi.
Faccio la prima LUT che chiamo g e che ha s_0, s_1, w_0, w_1 come IN:

$$g = \overline{s_1}\,\overline{s_0} w_0 + \overline{s_1} s_0 w_1$$

Analogamente creo h:

$$h = s_1 \overline{s_0} w_2 + s_1 s_0 w_3$$

Ne userò una terza per mettere insieme g e h!! Sarà sottodimensionata e due ingressi li metto come voglio perché non mi interessano.



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 11 di 106

Ma da cosa nasce?

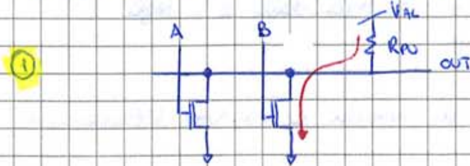
nasce dalla configurazione "altra" in fase di commutazione.
La copertura con Karnaugh crea due implicanti slegati e la commutazione fa sì che ci si sposti da un implicante all'altro. Se nel mezzo nascono altre configurazioni nasce l'alea.

Come la tolgo?

aggiungo ridondanza, ossia un terzo implicante che garantisce che durante la commutazione un implicante su 3 resti fisso a 1. Assicurazione

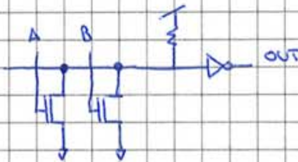
CAPITOLO 4: PROGRAMMABLE CIRCUIT LOGIC

Mattone base è il WIRED OR: realizzo la funzione OR a partire da una linea condivisa

① 

Forzo un '1' sul gate di un MOS, questo si chiude e la RPU si scarica forzando OUT a ∅.
Basta che uno tra A e B faccia ciò.
Ho creato la funzione NOR → $OUT = \overline{A+B}$

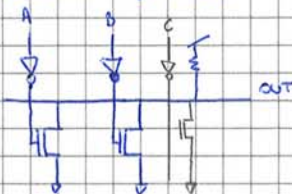
② e se complemento l'uscita? creo la OR



Funzione OR:

$OUT = A+B$

③ e se complemento gli ingressi?



L'uscita sta a 1 quando entrambi gli ingressi sono alti in modo da non chiudere i mos.

Ho creato la funzione AND

$OUT = A \cdot B$

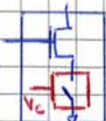
Se ad una NOR (che è è una AND con ingressi negati per De Morgan) nego gli ingressi, ecco che ho una AND!

Dato che il mos di C non è collegato alla linea di collegamento di C, ecco che C non compare come variabile della funzione logica

Attraverso la condivisione di una linea realizzo le funzioni OR e AND (e complementi)

Idea: se potessi decidere quali ingressi mandare, quali collegare e quali complementare io decido quale funzione logica creare.

Le matrici di connessione fanno questo: mettono ogni ingresso diritto e complementato all'ingresso, poi un blocchetto (spiegato dopo) fa sì che il source del MOS va a massa o se il MOS si interdice



l'interruttore, opportunamente collegato ad una tensione di controllo, assicura o no la partecipazione di quell'ingresso nella funzione.

Ma come realizzo l'interruttore? 5 STEP verso le PLA!

METODO ARCAICO) Martello e scalpello sul silicio. Il costruttore programma ma è evidente che i costi sono alti perché lui deve fare una maschera apposita. Se ha una maschera comune i costi calano, ma sono alti se mi serve una sola scheda. 1.000.000 di schede ha senso

METODO ANNI 80) L'interruttore è fatto dal fusibile che ha un metallo tale che quando metto sulla linea una tensione tale da garantire corrente che bruci il fusibile io ottengo l'isolamento del transistor. Altrimenti no.
La macchina di programmazione con operazioni a tappeto decide chi vive e chi muore.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 11 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 106

Nella PAL il costruttore fissa le uscite e non c'è modo di cambiare a valle il sistema. Pagando in velocità posso realizzare blocchi + complessi riportando indietro le uscite

Dato che così usi blocchi combinatori non faccio nulla posso pensare di mandare le uscite a dei FF in modo da memorizzare dei bit ottenuti programmando!

Volendo ulteriormente rendere + flessibile la PLA posso pensare di aggiungere uno stadio di EXOR a fondo del piano OR e così ottengo un ulteriore migliorìa: programmo le inversioni in uscita dunque faccio sì che creo uscite come somme di prodotti e uscite come prodotti di somme.

ES SLIDES: $\overline{AB+\bar{A}\bar{B}} = \overline{\overline{A\oplus B}} = A\oplus B$ ho creato un XOR!!

Come realizzo fisicamente i piani AND e OR del PLA?

Generalmente si costituiscono i piani a partire dal WIRED OR che garantiva la funzione NOR ~▷ PLA NOR-NOR
Infatti, io posso creare ∀ funzione logiche con sole NOR a patto che questa sia esprimibile come POS.
(II) Se si è + canonici e si è + affini alla modalità SOP allora uso le OR e dunque in uscita al piano NOR metto gli inverter. Vedi slides agli schemi ∓.

Di fatto nella pratica creo il piano NOR come matrice del WIRED OR, cioè condivido + linee collegate alla R_{pu} e tutte alla Vdd. Le linee del piano NOR delle AND sono le uscite del primo piano NOR ma ingressi per il secondo, dunque coloro che alimentano i gate dei MOS del II piano NOR, fatto identicamente.

Facciamo ora un'evoluzione della tecnologia: CPLD, complex programmable logic device

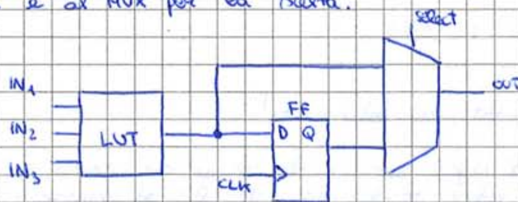
Metto insieme tanti blocchi nella logica PLA, rendendo sempre + complesso e flessibile la capacità di sintesi della circuiteria

Ma, unico problema è che rendendo + complessa la tecnologia i fili devono fisicamente essere lunghi, dunque tempi di propagazione lunghi!! Il cocchiere riceve l'avviso di fermata 2 fermate dopo quella richiesta! INACCETTABILE.

Ammettiamo di buttare via la struttura WIRED OR e cerchiamo una logica di programmazione + efficiente a parità di complessità!

Si usano MUX e LUT: scrivo in una piccola memoria ∅ e 1 e programmando opportunamente i bit di selezione creo tutte le funzioni che voglio. Il vantaggio è che i tempi di ritardo calano drasticamente.

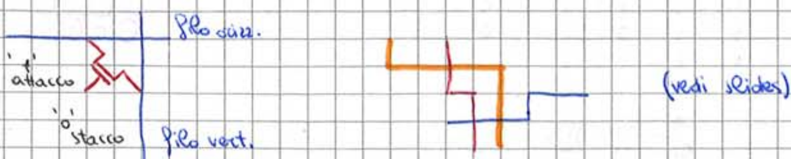
Il blocco logico fondamentale diventa una LUT che fa ∀ funzione (mi frego di NOR, AND, ecc), vai al FF se vuoi memoria e al MUX per la scelta.



Metto tanti di questi oggetti per creare una matrice che realizzi ∀ funzione combinatoria/sequenziale.
Ecco che la programmazione è fatta come PGA (programmable gate array) e dato che programmo sul campo (scrivo in memoria senza programmatore) nascono le FPGA
↓
Field programmable gate array

Il VHDL fa tutto questo senza dare turbe.

Come collego i vari blocchi nella FPGA? devo programmare le interconnessioni:



(vedi slides)

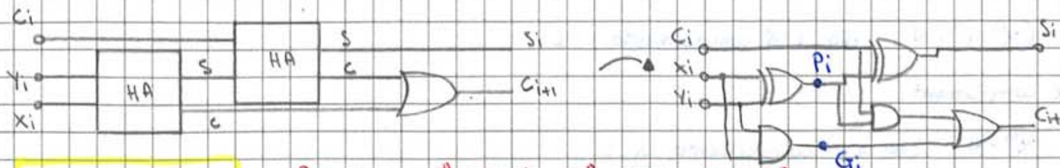
Posso collegare tutti con tutti a meno che non ci siano + canali disponibili. A quel punto non posso passare da lì e devo scegliere altre vie (sono utili le switch box)
Le strutture di oggi mirano pertanto ad aumentare le connessioni, non i gate!! Quello è l'obiettivo

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 106

7

Un modo alternativo di fare il FA è quello di combinare degli half-adder per creare i segnali di PROPAGATE e GENERATE e poi il FA:

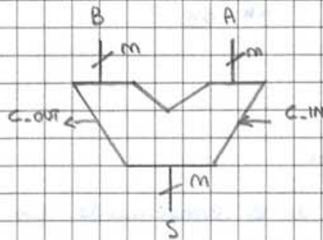


$C_{i+1} = C_i \cdot P_i + G_i$

P_i = propago il carry in sul carry out perché almeno un ingresso è a '1'

G_i = genero il carry out a prescindere dal carry in perché entrambi gli ingressi sono a '1'

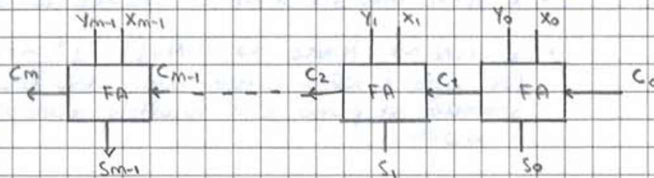
Posso dunque creare sommatori binari:



! STESSO PARALLELISMO FRA A, B ed S

RIPPLE CARRY ADDER

Rimbalzo il carry out dello stadio precedente come carry in dello stadio successivo



esempio di A·3 come A+A+A (vedi parallelismo):

- posso usare 2 FA e fare A+A+A ⇝ 2 FA
- posso fare (A·2)+A dove A·2 è uno shift verso sx di una posizione e metto uno ∅ nella posizione meno significativa. Poi sommo ancora A ⇝ 1 FA

Ⓑ Sottrazione

- 1001 − 0111 = 0010 → 9 − 7 = 2 **OK**
- 0100 − 0111 = 1101 → 4 − 7 = 13 **NO**

Cosa c'è che non va?

4−7=−3 !! il fatto che c'è un prestito di un 1 che arriva da chissà dove me lo segno come "−" e poi sommo 2^m al # che ottengo.

Quindi se A−B<0 io devo mettere un − e poi dopo scrivere $A-B+2^m$

0100 − 0111 = − 1101

10000 − 1101 = − 0011

Operare in hardware implementando ciò con i FA è una cagata pazzesca!

Si opera + facilmente usando i complementi!

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 106

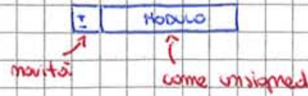
B • INTERI SIGNED

Per evitare casini con i segni nella sottrazione unsigned, introduco una notazione che contempli già i segni:

$$(S)\ a_{n-2} \dots a_2 a_1 a_0$$

S=0 numero +
S=1 " -

• Codifica modulo e segno:



• Codifica in CA2

il primo bit dà il segno e pesa con peso 2^{n-1}

Nascono indeterminazioni di codifica:

M&S: 0000 +ϕ
1000 -ϕ

vado da 2^{n-1} a -2^{n-1} (SIMMETRIA)

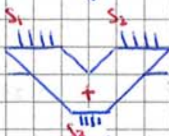
CA2: 0000 +ϕ
1000 -8

vado da 2^{n-1} a -2^{n} ⇝ Talvolta è un limite (ASIMMETRIA)

Si usa nel mondo somma e sottrazione in CA2 e non in M&S ⟶ il motivo è che sia per somme che per sottrazioni si usa lo stesso sommatore già visto.

(A) Somma:

faccio la somma come nulla fosse semplicemente includendo i bit di segno



se il risultato ha segno diverso dal segno dei due addendi (che avevano segno uguale) allora c'è stato OVF

REGOLA PER VEDERE SE C'È OVF IN ADDIZIONI SIGNED

(B) Sottrazione:

complemento il numero che sottraggo e seguo le regole dell'addizione

esempi di somma:

5 + 2 = 7

0101 +
0010 =
0111 ok

-5 + 2 = -3

1011 +
0010 =
1101 ok

5 - 2 = 3

0101 -
0010 =

0101 +
1110 =
(1) 0011 lo ignoro perché 5>2

-5 - 2 = -7

1011 -
0010 =

1011 +
1110 =
(1) 1001 ok — lo ignoro perché 5>2

quando segno di A = segno di B ⇝ segno del risultato = ⇝ NO OVF

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 106

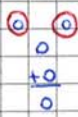
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 19 di 106

OVERFLOW DETECTION

Può esserci se:

- Sommo numeri con stesso segno
- Sottraggo " con ≠ segno (è una somma con stesso segno)

Analizziamo casi corretti:

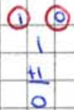

0 +0 = 0

1 +1 = 1

C_N e C_{N-1} sono uguali.

Analizziamo casi scorretti:

0 +0 = 1


1 +1 = 0

C_N e C_{N-1} sono ≠.

In definitiva vedo l'overflow se:

$$OVF = C_N \oplus C_{N-1}$$

In alternativa posso lavorare direttamente sui bit di segno perché vale:

$$segno_SUM = C_{N-1} \oplus segno_A \oplus segno_B$$

dunque:

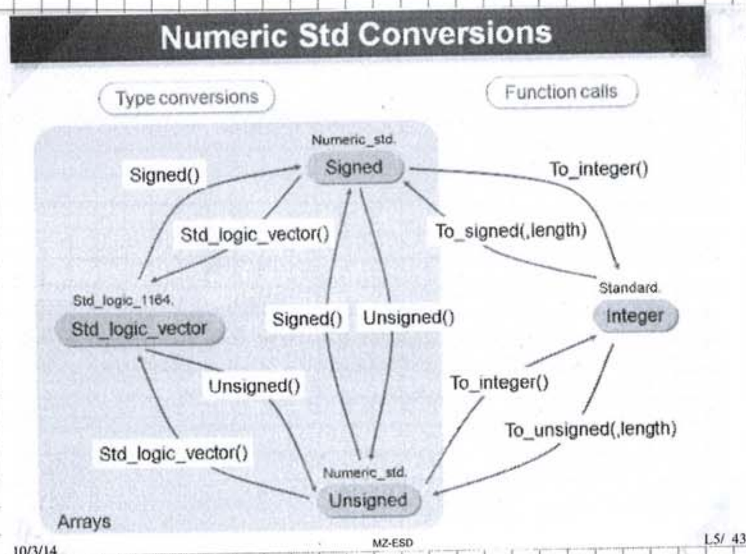
$$C_{N-1} = segno_SUM \oplus segno_A \oplus segno_B$$

↓

$$OVF = segno_SUM \oplus segno_A \oplus segno_B \oplus C_OUT$$

($C_OUT = C_N$)

PACKAGE NUMERIC-STD



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 19 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 21 di 106

⑤ LOOKAHEAD ADDER

L'idea è che posso esprimere il carry di ogni punto della mia catena di somme come funzione Booleana (SOP) di propagate e generate dei passi precedenti.

$$c_{i+1} = g_i + p_i c_i$$
$$= g_i + p_i (g_{i-1} + p_{i-1} c_{i-1})$$
$$= \dots$$
$$= g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1} + \dots + p_i p_{i-1} \cdots p_2 p_1 g_0 + p_i p_{i-1} \cdots p_2 p_1 c_0$$

In definitiva esprimo il carry-out di ogni porzione di adder con 2 livelli di logica (OR e AND).

Si ok, ma maggiori la OR ha 37 ingressi ⇝ è lenta

Per ovviare a ciò si divide in blocchi e l'obiettivo del look ahead è quello di calcolare i carry con il minimo # possibile di passaggi da porte logiche evitando porte con troppi ingressi!!!

ALTRE OPERAZIONI MATEMATICHE

Si parte dal design by contraction ⇝ tronco via ciò che di fatto nel circuito non serve avere. Nel specifico controllo se le porte presenti effettivamente servono o no. Riduco la complessità e semplifico la leggibilità del circuito.

Ecco che nel sottocaso di fare un incrementatore (A+1) parto da un FA su 3 bit (A è su 3 bit) e arrivo ad avere 1 INVERTER + 1 AND + 2 XOR ⇒ RIDOTTO

- SOMMA/SOTTRAZIONE PER NUMERO FISSO

Uso design by contraction ⇝ A+1 A-4 B-1 B+4

- MOLTIPLICAZIONE/DIVISIONE PER NUMERO POTENZA DI DUE (2^m)

opero uno shift

UNSIGNED
- MOLTIPLICAZIONE ⇝ sposto di m posizioni verso sx e accodo '0' a valle
- DIVISIONE ⇝ sposto di m posizioni verso dx e metto davanti '0' a monte

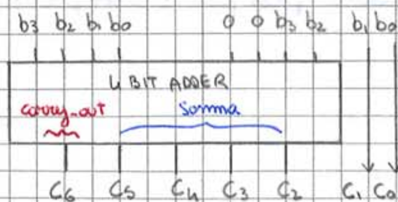
- MOLTIPLICAZIONE PER COSTANTE

$(b_3 b_2 b_1 b_0) \cdot (101)$

Parto da dx ⇝
$B \cdot (--1)$ è come fare B
$B \cdot (-0-)$ è come fare 0 ⇒
$B \cdot (1--)$ è come fare $B \cdot 4$ (shift 2 verso sx)

Sommo in riga e trovo

Basta un sommatore!



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 21 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 23 di 106

- **FLOATING-POINT NUMBERS**

Consistono di una mantissa e di un esponente

SINGOLA PRECISIONE → 32 bit totali

S	E	M
0/1	8 bit di esp.	23 bit di mantissa

DOPPIA PRECISIONE → 64 bit totali

S	E	M
0/1	11 bit di esp.	52 bit di mantissa

Il calcolatore come opera il numero?

$$\text{valore} = \pm 1.M \cdot 2^{E-127}$$ per la singola precisione

$$\text{valore} = \pm 1.M \cdot 2^{E-1023}$$ per la doppia precisione

I vantaggi rispetto alla logica fixed point sono molti a partire dalla dinamica del segnale che viene allargata molto;
Gli svantaggi è che le operazioni in floating point sono + lunghe perché devo far coincidere le normalizzazioni.

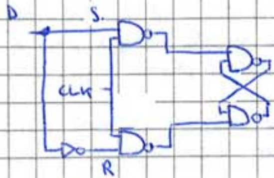
Se ho tempo uso la floating e guadagno in dinamica
Se voglio andare 10 volte + veloce di Matlab, C, C++ uso la fixed perché devo vincere in velocità

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 23 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 25 di 106

2°) Dal momento che non ho eliminato la condizione proibita (CLK=1, S=1, R=1) noto che Q è forzato ad un valore opposto a $\overline{Q}$ solo quando $S=\overline{R}$!

Ecco che allora unisco i due ingressi e metto un inverter. Ora posso solo avere S=1 e R=0 o viceversa



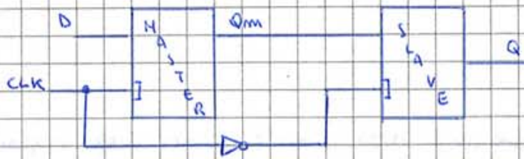
CLK	D	Q(t+1)
0	X	Q(t)
1	1	1
1	0	0

Ho creato il FF di tipo D.

SR LATCH ⇝ GATED SR LATCH ⇝ FLIP FLOP di tipo D!

Questo è un elemento così detto LEVEL SENSITIVE perchè opera in base ad un segnale di livello (CLK) (Si indica con un rettangolo).

Lo sviluppo del FFD è il FLIP FLOP MASTER SLAVE EDGE TRIGGERED



CLK = '1' ⇝ sistema in memoria perchè il master è trasparente e lo slave in memoria; dunque si mantiene il dato che c'era nel ciclo precedente

CLK = '0' ⇝ master in memoria e slave trasparente. Sono in grado di avere in uscita l'ultimo dato prima della chiusura dei battenti (Q=Qm)

Ho creato un oggetto che fotografa l'ingresso un istante prima del fronte di discesa del CLK e lo memorizza per un intero ciclo di clock, finchè non scatto nuovamente la foto.

Siamo arrivati al FF che usiamo:

POSITIVE EDGE TRIGGERED NEGATIVE EDGE TRIGGERED

! { EDGE TRIGGERED ⇝ lavora sulla sola commutazione
LEVEL SENSITIVE ⇝ " lungo tutto il tempo per cui un segnale è alto/basso!

Nel fare la foto vanno rispettati i tempi di "messa a fuoco":

devo garantire che l'ingresso non cambi in concomitanza con il clock, ma che si rispettino certi tempi

In modo asincrono posso poi mettere due segnali di PRESET e CLEAR senza passare dal CLK!!

Posso farlo anche in modo sincrono mettendo in AND il clear e il dato D.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 25 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 27 di 106

6) Timing diagram

Guardo l'evoluzione temporale della macchina. Ogni segnale subisce variazioni che si mantengono tali per un tempo determinato e le uscite devono assolutamente mantenersi per multipli dei colpi di clock. Questo perchè la macchina è di Moore, dunque le uscite sono dipendenti dallo stato e lo stato cambia in corrispondenza del clock, perciò le uscite non possono cambiare + volte in un clock o a metà clock.

Nella macchina di Mealy, da usare solo se alla canna del gas, le uscite dipendono anche dagli ingressi che sto ricevendo → le uscite sono + reattive (unico vantaggio di Mealy) perchè risparmio colpi di clock.
La sintesi è identica a quella di Moore.
Oltre al vantaggio della velocità quali sono gli svantaggi? Certo, a partire dal fatto che ogni glitch dell'ingresso me lo posso trovare in uscita! Inoltre una grossa differenza tra Moore e Mealy è che per Moore gli ingressi sono fisicamente scollegati dalle uscite perchè c'è una barriera di FF; per Mealy invece c'è un bypass degli ingressi che finiscono a pilotare le uscite con reti combinatorie.

Entrando + nel dettaglio:

cosa cambia se metto come elemento con memoria un FF piuttosto che un latch?

il latch per un certo tempo resta trasparente e se la macchina deve fare la foto nell'istante in cui lui è trasparente, perchè c'è ad esempio un percorso veloce + di quanto previsto, il rischio è quello di finire in stati assolutamente non consentiti e magari la macchina si incastra } esempio corridori

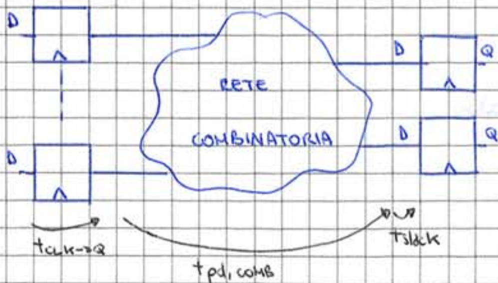
con il flip-flop invece questo non succede perchè fino al colpo di clock successivo la foto non viene fatta. Quindi finchè il corridore + lento non passa il traguardo la macchina non cambia stato.

⇒ Nelle FSM usiamo solo FF e non LATCH per memorizzare le variabili di stato!

Diventano dunque fondamentali i parametri temporali dei nostri oggetti!!

- t_{setup}
- t_{hold}
- T_W : tempo dell'impulso di clock → oltre un certo tot il FF non vede l'impulso del clock
- t_{px} : propagation delay → tempo dopo cui cambia l'uscita dall'arrivo del clock

Dai tempi di azione del sistema è derivabile la massima frequenza di funzionamento del circuito. Dunque io, se voglio calcolarla devo sommare i tempi del circuito:



- $t_{pd,COMB}$ (≥ 0) : tempo della logica combinatoria. Prendo il + lento
- t_{slack} (≥ 0) : quando arriva alla foto può succedere che arrivo esattamente con un t_{setup} di anticipo (max ottimizzazione) oppure che arrivo con + anticipo e sto fermo. Quello è lo slack

$$t_{P_{TOT}} = t_{pd,FF} + \max(t_{pd,COMB}) + t_{slack} + t_{setup}$$

⇝ Questo tempo limita f_{max}

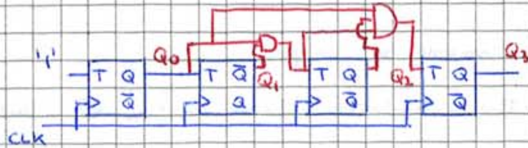
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 27 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 29 di 106

Quello però non è un circuito sincrono ⇝ i clock non sono gli stessi e le uscite cambiano una in catena all'altra ⇒ possono insorgere errori di conteggio causa la propagazione dei dati e maggiori avere dei missing code in uscita! Ho limitazioni sulla f_{MAX} perché il Δt_{pd} la fa da padrone!

Non posso sincronizzare le uscite?

SYNCHRONOUS COUNTER



0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
⋮

Q_0 cambia sempre ⇝ $T_0 = '1'$

Q_1 cambia quando Q_0 è 1 ⇝ $T_1 = Q_0$

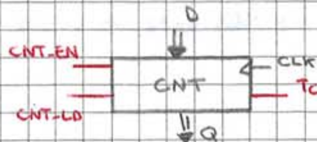
Q_2 " " Q_0 è 1 e Q_1 è 1 ⇝ $T_2 = Q_0 \cdot Q_1$

e così via

Problemi: costi hardware
tempi di propagazione crescenti al crescere del numero dei bit di conta ⇝ porte AND a n ingressi!!

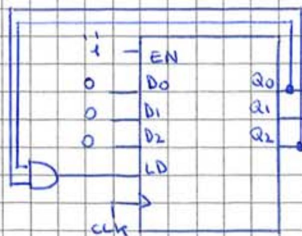
Al mio contatore posso aggiungere funzionalità quali:

- CNT_EN → segnale che abilita il conteggio altrimenti mantiene il conto che aveva
- TC → segnale che va a '1' quando è stato raggiunto il numero massimo di conta; dopodiché torna a zero
- CNT_LD → abilita la conta da un numero che voglio io



Ma è possibile lavorare con un contatore non modulo potenza di 2? Sì, con un trucco!!

esempio: MODULO 6



TRUCCO: carico con il LOAD la configurazione di partenza andando in ingresso le uscite con rete combinatoria → quando c'è 101 il LD si attiva e lui carica il dato 000
Ho una sorta di reset sincrono!

Volendo mettere il reset asincrono non conto modulo 6, bensì 5 perché se metto una rete combinatoria appena arrivo a 5 questa immediatamente manda a ϕ. Ecco che a cinque ci arrivo per pochissimo, poi riparto → NON BENE

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 29 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 31 di 106

CAPITOLO 7: VHDL SEQUENTIAL ASSIGNEMENT STATEMENTS


```
[process_label:]
PROCESS [(signal_name)]
   [VARIABLE declarations]
BEGIN
   [simple signal assignement statements]
   [variable assignement statement]
   [IF statements]
   [CASE   "   ]
   [LOOP   "   ]
END PROCESS
```

→ vediamo ora uno alla volta cosa posso mettere dentro il process

! il process esegue tutte le istruzioni nell'ordine in cui sono messe!!!

IF STATEMENT

```
IF ... THEN
   statement
ELSIF ...
   statement
   :
ELSE
   statement
END IF
```

CASE STATEMENT

```
CASE ... IS
   WHEN constant_value =>
        statement
   WHEN constant_value =>
        statement
   WHEN OTHERS =>
        statement
END CASE
```

importante la freccia al contrario (=>)

LOOP STATEMENT

```
FOR variable_name IN range LOOP
   statement
END LOOP
```

oppure

```
WHILE boolean_expression LOOP
   statement
END LOOP
```

NB: ogni segnale dovrebbe essere pilotato da un solo processo! RICORDARE PER ESAME
Non fare due processi che pilotano lo stesso segnale d'uscita: vuol dire mettere in corto le uscite dei due processi

NB: se voglio descrivere processi combinatori puri i segnali della sensitivity devono presentare tutti i segnali nella parte dx degli assignements e anche nelle condizioni (IF, CASE..)

NB: inoltre, per i combinatori, nel caso di utilizzo di statement IF, ELSEIF e ELSE devo sempre avere la condizione sull'uscita su tutti i rami della condizione.

NB: il processo fa le assegnazioni procedendo a tempo fisso! Ecco che l'ordine di scrittura è fondamentale perchè se faccio + assegnazioni all'uscita x verranno buttate via tutte meno l'ultima che sarà la sovrascrittura delle precedenti.

PROCESSI CLOCKATI

```
IF CLK'event and CLK = '1' THEN
      :
END IF
```

→ è il modo con cui il sintetizzatore capisce che deve operare su un fronte del clock che dunque sarà un segnale della sensitivity list

Ogni segnale uscito sotto il controllo di processo clockato che lavora sul fronte del clock causa l'inserzione di un flip-flop

NB: i segnali esterni, come ad esempi set/reset asincroni in un counter, devono comparire nella sensitivity list
Nel caso di avere che dello essere sincroni invece nella sensitivity list non compaiono ovviamente perchè il processo è svegliato dal clock, non da loro. Saranno dopo la dicitura CLK'EVENT ecc. ecc.

NB: sulla riga del CLK'EVENT NON DEVO METTERE ALTRO sennò rischio che il fronte di salita/discesa non venga riconosciuto.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 31 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 33 di 106

BUFFER TRISTATE

Se En-Buffer è attivo il dato in ingresso è copiato in uscita, altrimenti si va in alta impedenza che nel package std-logic è indicato con 'Z'

VHDL:

```
Sig-out <= Local-Data WHEN En-buffer = '1' ELSE
           (others => 'Z');
```

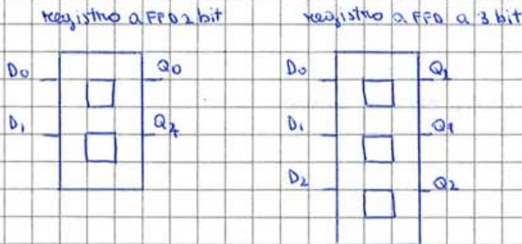
Se avessi + buffer tristate che pilotano la stessa uscita:

```
Out-bus <= In1bus when En1 = '1' ELSE
           (others => 'Z');

Out-bus <= In2bus when En2 = '1' ELSE
           (others => 'Z');
```

} unico caso possibile di 2 processi ≠ che pilotano la stessa uscita. Non voglio conflitti ovviamente

Se ci sono però 2 driver sulla stessa linea di fatto un conflitto c'è sempre; il VHDL risolve comunque questi conflitti con il package STD-LOGIC (≠ dal STD-ULOGIC che non risolve i conflitti). Quindi avere STD-LOGIC mi permette di risolvere i conflitti; → unresolved
I puristi del VHDL usano STD-ULOGIC perché si accorgono di aver messo + driver sulla stessa linea, però nella pratica si usano sempre gli STD-LOGIC!!!



Resolution Function: std_logic & resolved()

- Resolving values for std_logic types:

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X

- Pairwise resolution of signal values from multiple drivers
- Resolution operation must be associative

IMPLIED MEMORY

Se nella descrizione di un processo nel quale uso anche dei condizionali (IF) non metto degli else ∀ condizione possibile dell'ingresso, il sintetizzatore reagisce inferendo un elemento di memoria, tipo latch.
Questa è la condizione di implied memory.

GENERICS

registro a FFD 2 bit: D_0 → Q_0, D_1 → Q_1

registro a FFD a 3 bit: D_0 → Q_0, D_1 → Q_1, D_2 → Q_2

⇝ Teoricamente 2 oggetti ≠ hanno 2 entity diverse. Se qualcuno vuole il registro a 32 bit lo devo rifare.

Il generic mi aiuta proprio a non dover rifare tutto, ma attraverso l'uso di un parametro personalizzabile, rende tutta la descrizione uguale, per fare il 32 bit anziché il 5 basta cambiare il valore di N.
Sono dunque elementi che parametrizzano il sistema.

```
ENTITY XOR2 IS
GENERIC (gate-delay : time := 2ns);
PORT (
   :
   :
)
END ENTITY XOR2;

ARCHITECTURE Behaviour OF XOR2 IS
BEGIN
  Z <= (IN1 XOR IN2) AFTER GATE-DELAY;
END ARCHITECTURE Behaviour;
```

(gate-delay = nome; time = tipo; 2ns = valore)

```
ARCHITECTURE generic-delay of ha IS
  COMPONENT XOR2
    GENERIC (gate-delay : time);
    PORT (
    )
  END COMPONENT;
  COMPONENT AND2
    GENERIC (gate-delay : time);
    PORT (
  END COMPONENT;
BEGIN
  EX1 : XOR2 GENERIC MAP (gate-delay => 6 ns)
             PORT MAP (- - - -);
  AND1 : AND2 GENERIC MAP (gate-delay => 4 ns)
             PORT MAP (- - -);
```

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 33 di 106

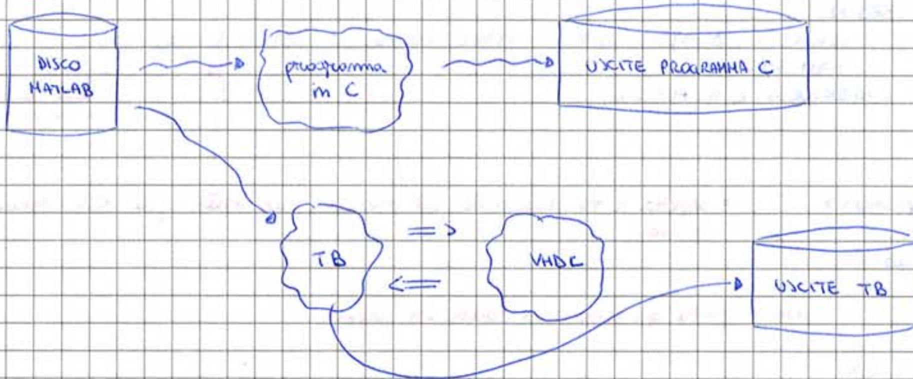
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 106

PACKAGE

Metto insieme procedure, funzioni, valori di costanti note ⇒ un package descritto è molto utile dunque se devo personalizzarmi pezzi per me ⇝ ogni modifica si propaga su tutta la logica!!

Infine i package li metto nelle librerie che richiamo con i comandi "USE"

BASIC INPUT/OUTPUT



* il confronto tra le due uscite mi dice se la realizzazione in HD combacia con quella in SOFTWARE!

Come posso però a livello VHDL leggere e scrivere sul e dal file?

Tipi di file: * TYPE text is FILE of string; ⇜ consigliato perché guadagno in leggibilità

* TYPE IntegerFileType is FILE of integer;

Con il package TEXTIO accedo alle modalità di lettura e scrittura del file ⇝ accedo al file attraverso una lettura linea per linea (non tutto insieme). Lettura e scrittura avvengono perciò in 2 tempi:

- read e write preparano la linea
  (agiscono come preparazione!)
- readline e writeline effettuano il comando di lettura e scrittura di quanto preparato!

EXAMPLE: USE OF THE TEXTIO PACKAGE

```
use STD.Textio.all;
entity formatted_io is -- this entity is empty
end formatted_io;

architecture behavioral of formatted_io is
begin

process is
file outfile :text; -- declare the file to be a text file
variable fstatus :File_open_status;
variable count: integer := 5;
variable value : bit_vector(3 downto 0):= X"6";
variable buf: line; -- buffer to file
begin
file_open(fstatus, outfile,"myfile.txt",
write_mode); -- open the file for writing

L1: write(buf, "This is an example of
formatted I/O");
L2: writeline(outfile, buf); -- write buffer to file
L3: write(buf, "The First Parameter is =");
L4: write(buf, count);
L5: write(buf, ' ');
L6: write(buf, "The Second Parameter is = ");
L7: write(buf, value);
L8: writeline(outfile, buf);
L9: write(buf, "...and so on");
L10: writeline(outfile, buf);
L11: file_close(outfile); -- flush the buffer to the file
wait;
end process;

end architecture behavioral;
```

```
This is an example of formatted IO
The First Parameter is = 5 The Second Parameter is = 0110
...and so on
```

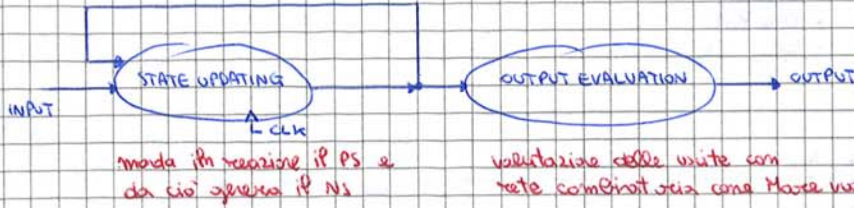
07/9/2012 M2-ESD L7/ 66

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 106

Allora come progetto la macchina a stati finiti, che realizza la control unit, in VHDL?

- Dal pallogramma arrivo sino alle porte che sintetizzo (da sapere ma lungo e poco usato)
- uso la potenza del VHDL!



1) Definizione degli stati che la macchina può enumerare con il tipo dato ENUMERATE!
   è come se dicessi i nomi dei pallogrammi
   il tipo che definisco può enumerare solo quei valori, uno dei quali deve essere sempre uno stato di RESET.
   Può essere utile anche un IDLE

   Così facendo guadagno che nel momento in cui aggiungo uno stato non devo rifare la codifica ma basta dire che c'è anche D oltre ad A, B e C.

2) Un processo che faccia lo state updating e quindi dica in funzione di dove sono dove vado a finire (generalmente sequenziale)

3) Un processo che calcola le uscite in funzione del PS (generalmente combinatorio)

State updating:

generalmente è implementato con un case perché da valore a un'uscita in base ai valori di altri segnali.
Faccio un WHEN ∀ stato con gli eventuali IF annidati

NON DIMENTICARE { Anche se ho messo tutte le condizioni su tutti gli stati è buona norma mettere un WHEN OTHERS per mandare la macchina in stati noti e sicuri in caso ci si finisca!

Golden rules:
- usare un reset asincrono e dargli uno stato dedicato
- usare gli statement case/when
- usare un when others finale

output evaluation:

devo creare una rete combinatoria in cui dico stato per stato quanto valgono le uscite!
Per evitare l'inferenza di latch devo mettere l'uscita ∀ STATO enumerabile
→ NON + COMBINATORIO
Aggiungere anche qui il WHEN OTHERS evita molti problemi!!

default output assignment { Una cosa utile per evitare mille righe di codice è andare a mettere i valori di default delle uscite prima del case. Tale valore di default è il valore per cui l'uscita non deve fare nulla. Nel case metto solo quando essa cambia, ecco che guadagno molte righe!!

Riassumendo, cosa serve nella sintesi di una FSM?

- una variabile per il NS
- " " " " PS
- un clock
- un segnale di reset
- la conoscenza degli stati e delle loro evoluzioni sulle uscite

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 106

CAPITOLO 8: COMPLEX FSM

STATE ASSIGNEMENT PROBLEM

il sintetizzatore come codifica gli stati?

① Vai avanti in sequenza ← + lenta

la fa direttamente il sintetizzatore secondo il suo criterio

Svantaggio: cambio locazione / tolgo stati / aggiungo stati : tutto da rifare!

② Codifica di Gray: ← velocità media

cambia un bit alla volta dando così meno complessità al circuito a parità di FF

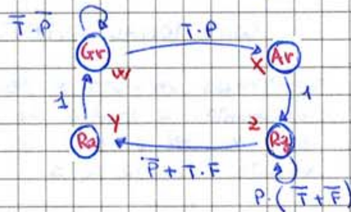
③ Codifica one-hot: ← la + veloce

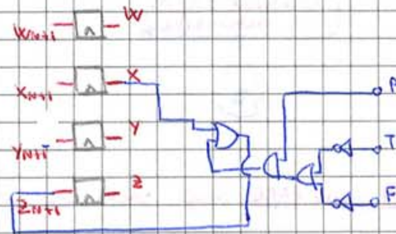
è ridondante ⇒ ∀ STATO ho un FF che è attivo solo se sono in quello stato!

dunque solo un FF è a '1', gli altri a 'ø' e quell'unico uno si sposta fra i vari FF che rappresentano i vari stati!

È evidentemente + costosa (unico elemento negativo)

i collegamenti tra i vari FF però come li faccio? guardo direttamente il pallogramma





e così gli altri

per le uscite il discorso è analogo:

$R = Y + Z$

$a = Y$

⋮

$A = X$

! Ho fatto tutto senza conoscere Karnaugh!!!

Non è evidentemente una soluzione ottimale a livello di FF ma va molto veloce!
Ci guadagno in semplicità nel fare i collegamenti e che abbiamo complessità lineale (solo porte AND, OR, XOR).

Oggigiorno quale tecnica di codifica scegliere è richiesto dal sintetizzatore direttamente con un menù a tendina solitamente!

STATE MINIMIZATION

Lo lasciamo ai matematici!!!

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 41 di 106

Passi operativi:

1) Devo avere chiaro cosa fare

Moltiplicazione di bit $n \times n$, devo shiftare a sx $n-1$ volte e poi sommare gli n codici ottenuti ad ogni passo, ottenendo così un risultato su $2n$ bit

2) Scrivo uno pseudo-codice

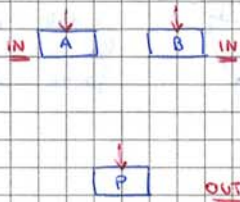
Rappresentazione (no rules) in una forma informatica di quell'algoritmo a parole

```
P = 0;                    -- inizializzo P
for i = 0 to n-1 do       -- itero n volte
   if b(i) = 1 then       -- solo se b(i) = 1 fai qualcosa
      P = P + A;
   end if
   left-shift A;          -- shifto A
end for;
```

3) Definizione delle variabili da usare

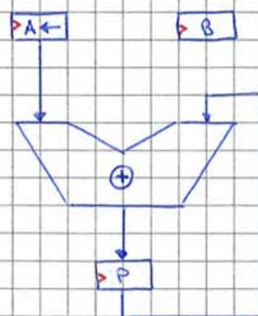
Devo memorizzare A
Devo memorizzare B
" " P

4) Quali sono input e output del mio circuito?



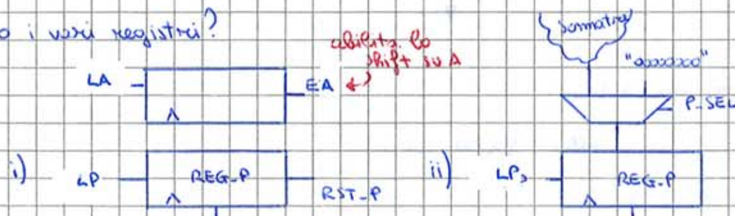
5) Definizione degli operatori da utilizzare

- Guardando lo pseudo codice ho sicuramente bisogno di un sommatore
- Serve uno shift register per spostare A



5.1) Quali funzionalità hanno i vari registri?

- A ⇝ LOAD
  SHIFT A SX
- P ⇝ LOAD
  RESET



Quale meglio per P? è uguale! La i) per i tempi, la ii) per la complessità del registro P

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 41 di 106

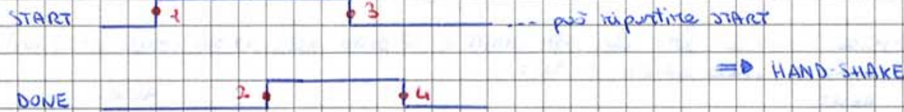
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 43 di 106

Gli elementi cerchiati sono quelli che la FSM (control unit) deve fornirmi e deve pilotare
Quelli neri quelli che la CU riceve per decidere

(7)

Dunque passiamo a fare la control unit ASM:

La macchina parte quando ho i dati (attendo un segnale di START). Arrivato, evolve. Attendo dunque un segnale di fine (attendo un DONE). Dopodiché dovrei essere pronto per l'elaborazione successiva che comincia con un nuovo start!



Mondo esterno ⇝ 1 ⇝ parti

Dispositivo ⇝ 2 ⇝ ho finito

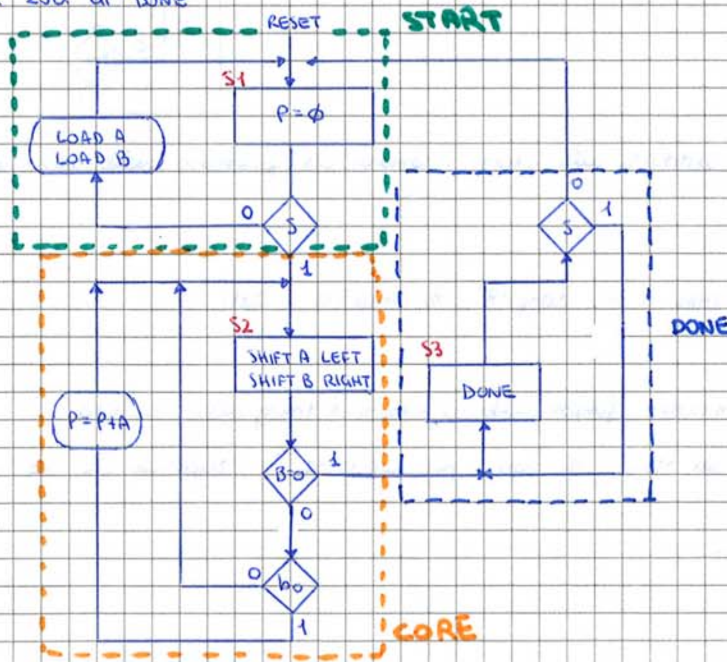
Mondo esterno ⇝ 3 ⇝ ok, ho capito che hai finito, tolgo i dati

Dispositivo ⇝ 4 ⇝ mi fermo, quando vuoi ci sono

} Questi passi vanno rispettati per avere una sincronizzazione tra mondo esterno e dispositivo.

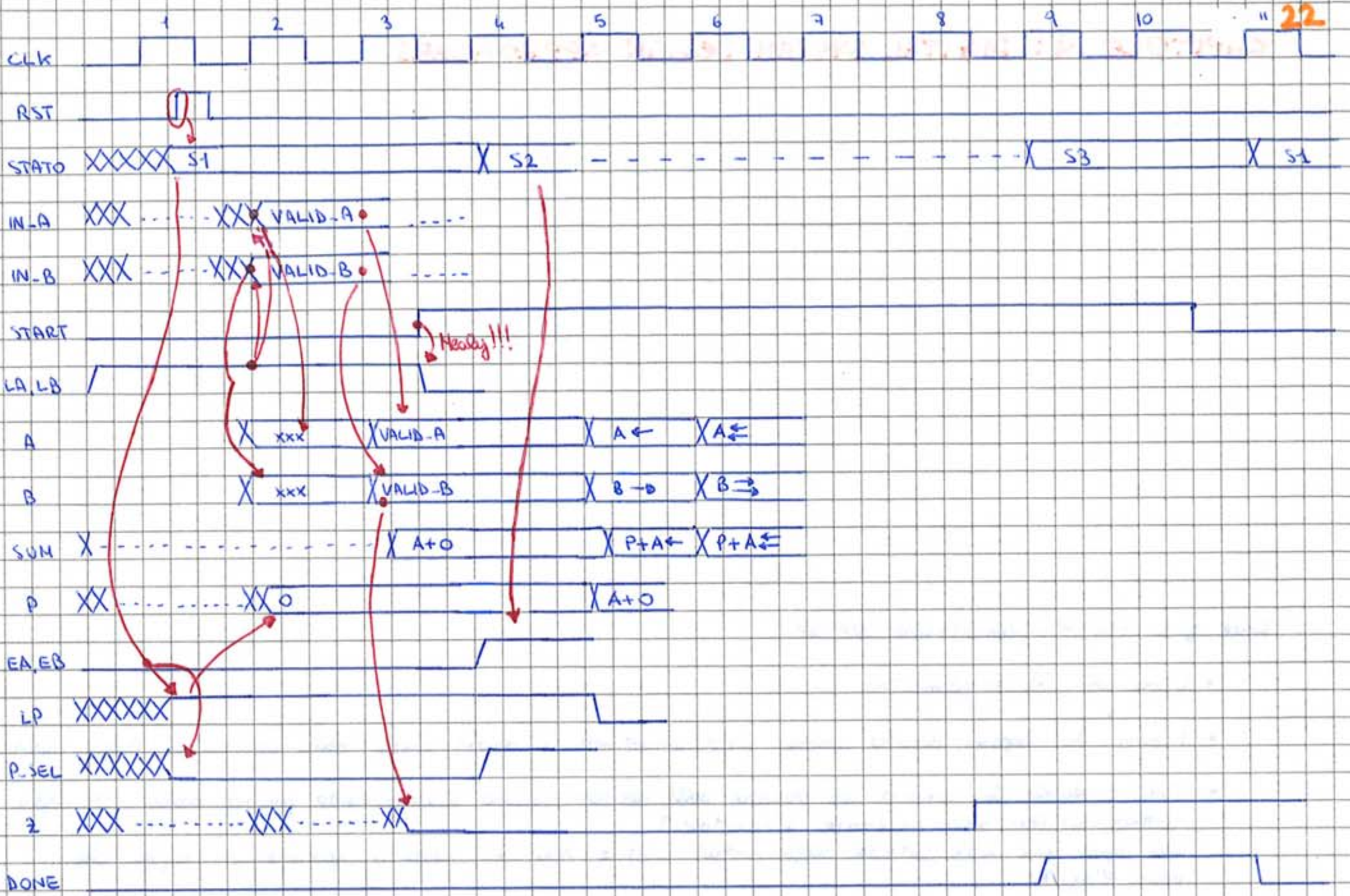
La mia ASM chart avrà pertanto:

- una zona di START
- un core
- una zona di DONE



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 43 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 106



© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 106

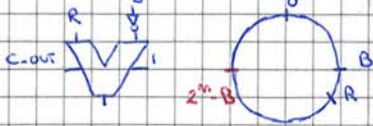
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 47 di 106

Passi operativi:

① Aver chiaro cosa fare

Per ogni bit di A, partendo dal + significativo, lo attacco al resto e confronto il numero uscente con B. se B ≥ RESTO salvo uno ∅, altrimenti se B ≤ resto salvo 1.
Non avendo un blocco base che confronta numeri faccio R-B e vedo se esce positivo (salvo 1) o negativo (salvo ∅).
Se la differenza è positiva faccio la differenza e poi riscrivo tutto dentro R, operazione che ho già fatto per verificare che l'uscita sia positiva.

Come vedo però se R-B è ≥ 0?



devo fare R-B ⇝ complemento B e lo sommo a R:

$$R+2^m-B = 2^m+(R-B) = R+(2^m-B)$$

se passo dal via genero un carry-out = 1, altrimenti no.

↓

$R-B>0 \rightsquigarrow R>B \rightsquigarrow 2^m+(R-B)>2^m \rightsquigarrow$ carry-out = '1'
$R-B<0 \rightsquigarrow R<B \rightsquigarrow 2^m+(R-B)<2^m \rightsquigarrow$ carry-out = '0'

È proprio carry-out il segnale che cerco ⇝ in base al suo valore la FSM saprà se mettere ∅ o 1 nel quoziente.

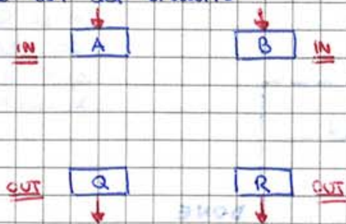
② Pseudo codice

```
R = ∅;
for i = ∅ to m-1 do
  left shift (R||A);   -- vuol dire shifta il MSB di A e attaccalo al LSB di R (dunque attacco a dx)
  if R≥B then
    R = R-B;
    qi = 1;
  else
    qi = 0;
  endif;
endo for;
```

③ Variabili da usare

Memorizzo A
" B
" R
" Q

④ IN e OUT del circuito



⑤ Operatori da usare

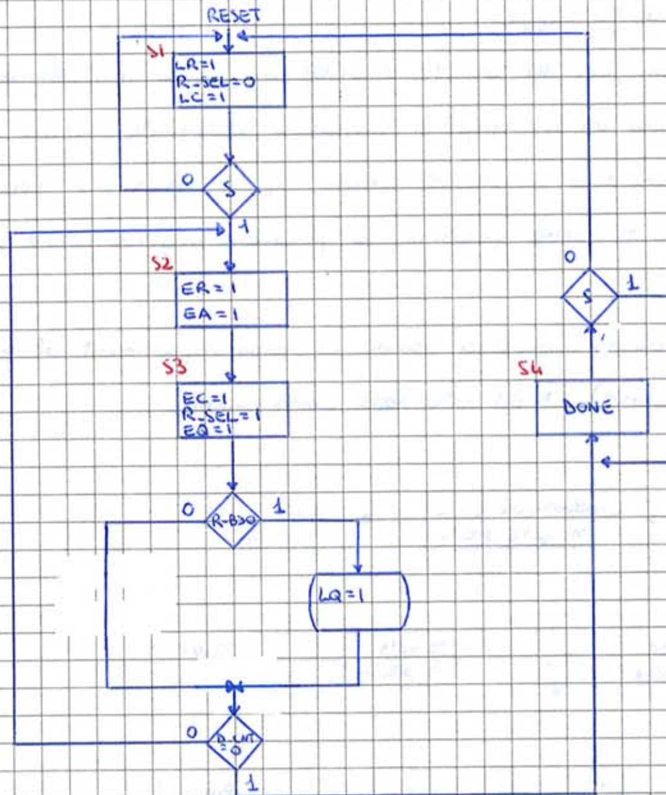
- Sottrattore (parto dal sommatore)
- Registri: Q è un left-shift
  B è un registro normale
  A è un left-shift
  R " " " "

- Contatore!

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 47 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 106

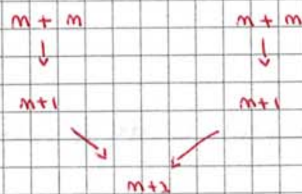
(8) Asm control



(9) VHDL

(10) TIMING

! Ho 4 numeri su n bit. La somma su quanti bit viene?



Volendo fare i pari

$\log_2(\text{termini_sommati})$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 51 di 106

CAPITOLO 10 : MEMORY

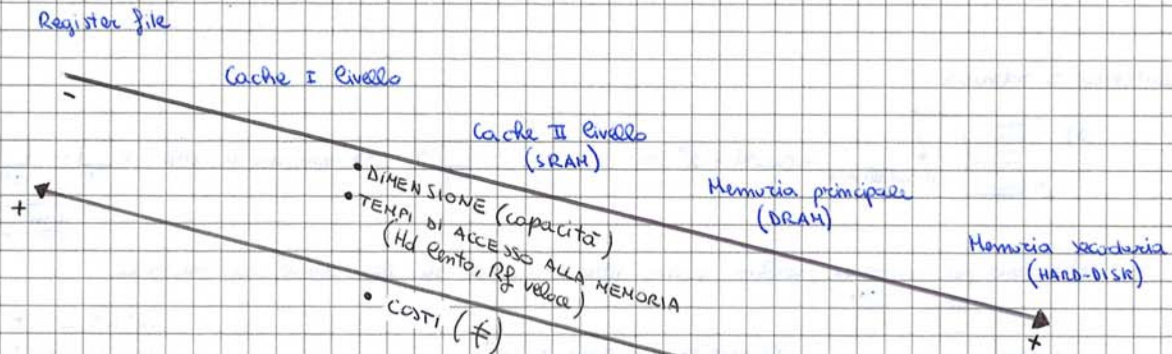
DEFINIZIONI

Memoria: collezione di celle di memoria tenute insieme da circuiti che prelevano e portano ad esse le info

- Com è organizzata?

  A vettore; dove ogni elemento del vettore racchiude uno scaffale. Il parallelismo della memoria è questo è quanto questo scaffale (bit + locazione).
  Quale locazione è scelta dall'indirizzo

- Random Access Memory (RAM): posso accedere in qualunque momento a qualunque locazione della memoria
- Elementi: bit
  byte (8 bit)
  word (collezione di byte)
- Operazioni: scrittura
  lettura
- Gerarchia delle memorie:



Perchè non posso avere un processore che abbia la totalità della memoria di macchina a disposizione? I motivi sono 2:

- limite fisico ⇝ non ci sta! Area!
- limite temporale ⇝ i tempi di accesso dell' hard disk sono ordini di grandezza maggiori ai tempi richiesti dal datapath

Ecco l'obiettivo della gerarchia di memoria:

far vedere al processore i Tb (ENORMI) con i tempi accesso del datapath (MINUSCOLI)

Il principio che regola tutto ciò è IL PRINCIPIO DI LOCALITÀ secondo cui non serve avere sempre tutti i dati disponibili ma poiché l'utente è abitudinario + o - usa sempre gli stessi.

<u>ESEMPIO BIRRA</u>: prendo una birra dal frigo e quando vado penso ci sia perchè qualcuno l'ha messa lì. Non posso avere ∞ birre in frigo quindi magari lì ne ho 2 fresche e il resto della cassa nella dispensa. Statisticamente nella dispensa metto ciò che bevo; se odio la gazzosa è inutile che la riempio di gazzosa.
Fin qui il register file è il frigo e la cache di I livello è la dispensa che rimpingua il frigo.
Dopodichè a furia di prendere la dispensa si svuota e cosa faccio? Vado al mini market e le prendo. Il minimarket non serve solo me, ma anche altri; quindi deve avere una capacità e una versatilità maggiore della cache I livello. Il mini è la cache di II livello che si rifornisce dalla DRAM (ipermercato) e sua volta dall'hard disk (grossista).
Man mano che esco dal register file le memorie aumentano di dimensione e di fornitura soprattutto perchè devono essere i bacini da cui + utenti attingono dati ₣.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 51 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 53 di 106

3D) Serie di matrici una sull'altra; dunque ho 3 indirizzi:

- BLOCCO INDIRIZZO
- RIGA INDIRIZZO
- COLONNA "

} decodifica a 3 livelli!

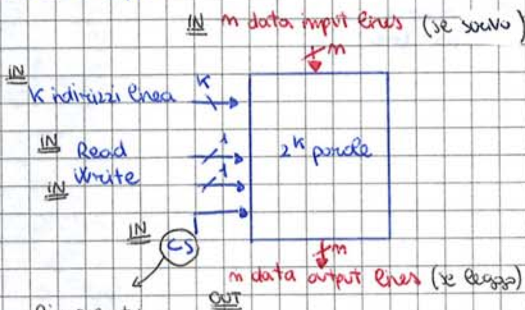
Lavora su blocchi non è tridimensionale!
- □ □ □ □ sì
- ▣ NO

Vantaggio:
- sono tante le matrici ~> singolarmente + piccole ~> + veloci
- quando ne uso una le altre possono dormire ~> consumo meno

LE MEMORIE ATTUALI SONO ORGANIZZATE COSÌ!

Il blocco è suddiviso ~> i percorsi sono suddivisi ~> tempi molto minori rispetto 2D

- Block diagram



BIT INDIRIZZO = $\lceil \log_2 (\text{locazioni}) \rceil$ ~> memoria da 1kbyte -> 10bit
1Mbyte -> 20bit
1Gbyte -> 30bit

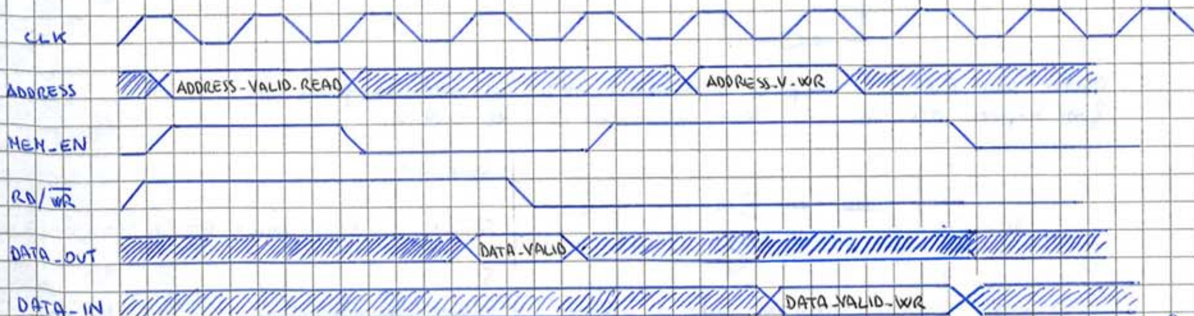
Chip select:
se nello schema a blocchi di sopra è evidente che devo sapere quale memoria lavori! Il segnale che sceglie quella che serve è il chip select

- Operazioni di memoria:

esempio guardando là!!

READ: operazione che legge un valore memorizzato ~> - piazzo un indirizzo valido sulla linea indirizzi
- aspetto che il dato letto arrivi stabile

WRITE: operazione che scrive un dato nella memoria ~> - piazzo un indirizzo valido sulla linea indirizzi e fornisco il dato da inserirci
- attendo l'ok del sistema



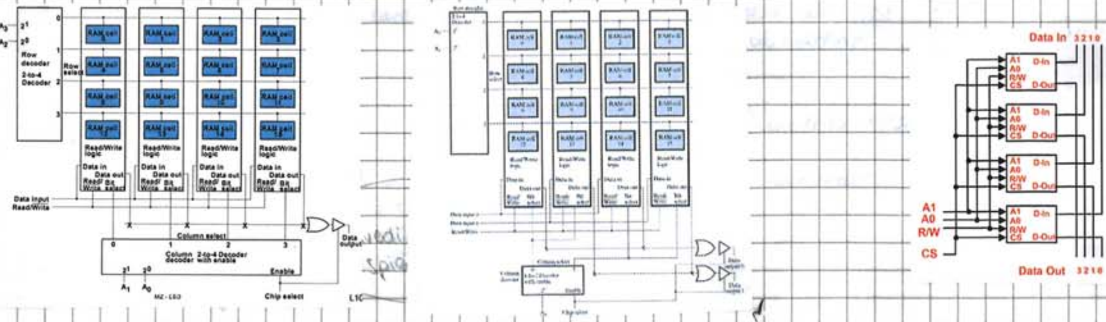
! Dato il tempo di accesso alla memoria che non è nullo, io devo mantenere attivo CS e il dato finché la memoria necessita di finire l'esecuzione. Questi tempi sono assoluti per memorie asincrone e sono contati in colpi di clock per quelle sincrone.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 53 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 55 di 106

Moltiplico le bit slices e creo una struttura matriciale: con due decoder poi seleziono word line e bit line e raggiungo la cella desiderata.

Da notare le ≠ tra una memoria RAM 16x1 e 8x2 e 4x4:



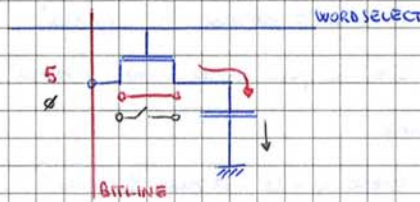
Il funzionamento è parallelo dove ogni slice legge o scrive un bit per volta!

Allargando la struttura creo memorie + grosse!

② DRAM

La SRAM ha comunque bisogno di 6 transistor. Non posso farla con meno così da aumentare la capacità?

Certo, uso un condensatore per memorizzare e un transistor come interruttore per caricarlo e scaricarlo



⤳ 1T + 1C ! Sfrutto l'area e massimizzo la capacità!

Ecco che dunque il MOS fa da rubinetto e il condensatore da bacinella che può o no, a seconda del rubinetto ricevere l'acqua dall'acquedotto (sistema)

- SCRITTURA: piloto la bit line e giro il rubinetto
  - LETTURA: non piloto la bit line ma sto a sentire cosa succede (vedo se le tensioni nel contenitore scendono o se salgono nell'acquedotto) usando il sense amplifier

    Ma in fase di lettura la situazione è critica perché quando travaso cariche dal contenitore all'acquedotto io ho completamente dimenticato se prima ero pieno o vuoto. Ho perso il dato ⇒ STRUTTURA DISTRUTTIVA

    Per evitare ciò vado a pilotare il sistema in modo che ripristini lo stato pre-lettura ⇒ LETTURA NON DISTRUTTIVA

Idealmente ci siamo ma: il condensatore un po' perde, il rubinetto si apre/chiude a volte } Non mi piace una memoria che nel tempo perde informazioni!

I tempi di perdita attuali dei condensatori è della decina dei ms!

Come si fa? Come coi i bambini!! Dimentichi le cose? Te le ridico e ti rinfresco:

REFRESH: da fare prima che la cella perda l'info. Se quando leggo io ripristino il dato per non creare letture distruttive, di fatto l'operazione di lettura sta già agendo da rinfresco.
TUTTO OK ma 2 domande:

① Quanto tempo per tutto il rinfresco?

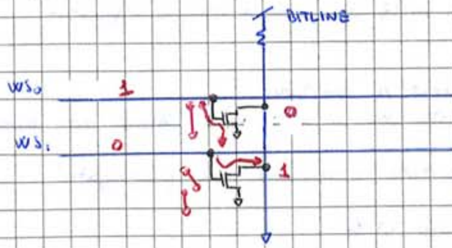
Se ho una memoria a matrice che lavora sulla word line moltiplico il rinfresco ∀cella per il numero di celle: 1Mbit ⤳ 10^6 celle. 60 ns = 60 μs che è 1/1000 del tempo di smemoria della memoria (~60 ms))!! ⤳ una volta su mille la memoria è inaccessibile xk chiusa per lavori!!

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 55 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 106

ROM

Devo solo mantenere la memoria e il dato immagazzinato!



Vantaggio: il dato è scritto nel silicio. La memoria NON È VOLATILE. Lo scrive la silicon foundry ~> mo cioè a 5000 $

Dal punto di vista del costo non è accessibile per modeste quantità.

E se non ho quei soldi?

- PROM: programmable read only memory

la struttura è simile alle PLA dal momento che oltre ai transistor, il collegamento a massa è effettuato con un fusibile che decido se bruciare (alta corrente e apro il transistor) o no (lo chiudo)

il vantaggio è che programmo a casa mia, pagando però oltre ai transistor i fusibili e, nel caso di tecniche duali, gli antifusibili

Sono comunque nel campo delle memorie OTP

Andando avanti con le evoluzioni, dato che quei fusibili poi fondono/diffondono ho modo di creare qualche alternativa?

- EPROM: erasable programmable ROM

Prendiamo un mos avente floating gate ~> G, D, S (FAMOS = floating avalanche mos)

in condizioni normali quel gate è isolato (carica neutra). E se le condizioni portano gate e drain a 20 volt? gli e⁻ iniziano a migrare verso il drain ma essendo attratti anche dal gate girano e alla fine si schiantano a muro caricando negativamente il gate, dunque diminuendo l'effettiva tensione che vi è su esso.
Il gate NON È + NEUTRO ~> non formo comunque il canale perchè il voltaggio con il gate negativo il transistor non si forma.

↳ Ho traslato la trans caratteristica in modo che se ci sono cariche sta OFF, altrimenti se non ci sono va ON.
<u>Dispositivo che se caricato non è + accedibile</u>

Ma gli e⁻ li posso convincere ad andare via? Li energizzo a tal punto che gli e⁻ se ne vanno nel canale. L'energia è fornita per illuminazione e quando si usa il dispositivo perde la memoria (utilizzo di scotch opachi). Ho cancellato tutto, ossia ecco perchè erasable.

La riscrittura non è fattibile ∞ volte: ~100!!

Può essere utile cancellare queste memorie senza doverle togliere dalla scheda, ma solo lavorando sul campo

- EEPROM: electrically erasable programmable ROM

rimando gli e⁻ sul canale per effetto tunnel semplicemente facendo sì che la distanza tra il gate floating e il substrato sia piccola (decina di nm).
Con tale distanza basta la tensione che applico normalmente per rispedire indietro gli e⁻

Altro vantaggio è che nelle EPROM cancellavo tutta la memoria illuminando!
Qui posso cancellare celle mirate.

- FLASH EEPROM:

Le riscrivo ~10000 volte, mentre le E²PROM ~100!

La lettura e la scrittura sono uguali a come avvengono nelle altre due tipologie ma l'unica ≠ è che la distanza gate-substrato è sempre + piccola e permette + scritture
NB: L'operazione di scrittura è molto + lunga della lettura.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 59 di 106

ESEMPIO: in una FSM cambio il segnale d'ingresso esattamente con il clock (fronte). I FF della macchina possono interpretare ognuno a modo loro tale cambio. uno dice c'è stato, uno no, gli altri BOH → Hanno tutti ragione ma l'uscita è errata

Entro in metastabilità

- Non prevedo + le uscite
- Non garantisco + i tempi di commutazione

} la macchina si incasta! In una one hot ho ϕ stati attivi o + di 1!!! MALE!!

Come risolvo il problema?

Fra il mondo asincrono e quello sincrono interpongo uno stadio di sincronizzazione (SYNCHRONIZER): tipicamente un FF basta e in tal modo $t_{CLK-> Q} + t_{COMB} + t_{SU}$ magari rientra nei tempi macchina dei FF della FSM.

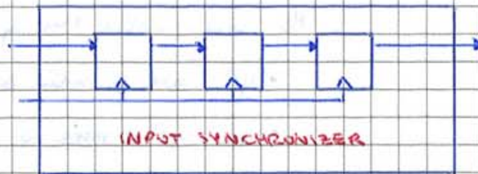
Inconvenienti?

- Ritardo di un colpo di clock
- il FF del sincronizzatore è ora affetto dagli stessi problemi che avevano gli altri FF prima perché il problema è stato solo spostato ~> l'ingresso del sincronizzatore può cambiare durante il t_{SU} o il t_h dell'FF del sincronizzatore. Possono aumentare i tempi CLK->Q ma il rischio si ripercuote sul resto della macchina perché tale ritardo può aumentare a tal punto da essere così sfigato da farlo arrivare in coincidenza del fronte dei FF della FSM.

Se però mettere il sincronizzatore non toglierà il problema, ma sicuramente abbassa drasticamente la probabilità che dato e clock cambino insieme.

Il FF del sincronizzatore magari riesce a stabilizzare il dato prima del fronte successivo, magari no ma se non lo fa la probabilità è < che se non ci fosse.

Allora io metto un secondo stadio di sincronizzazione e se non basta ne metto un terzo.
Sto pagando in colpi di clock che perdo perché rallento la fmax però, sicuro a ϕ di probabilità non vado mai, ma se da 1 passo a 10^{-15} non è male. ∀ applicazione vanno visti quanti stadi di sincronizzazione servono: generalmente metto il tempo di rischio crash paragonabile al tempo di vita del componente



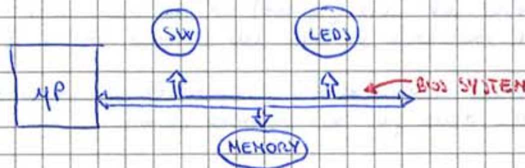
Il sistema perde in reattività ~> il sistema è narcotizzato però almeno ho abbassato tanto la probabilità di crash.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 59 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 106

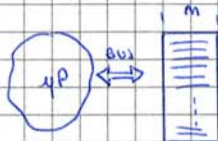
2) Dispositivi input/output:

LED, pulsanti, switch ~> il μP scambia dati con tutti questi sistemi attraverso la comunicazione su di un unico collegamento che è il BUS SYSTEM



Dunque a parità di area ho un μP + potente se metto solo quello ma se voglio un μC lo rimpicciolisco per aggiungere memorie, I/O, bus, applicazioni potenza ecc. ecc.

3) Memoria



- se ho un μP a 8 bit e la memoria è a 8 bit vuol dire che leggo una riga per volta.
- se il μP ha bisogno di dati su 16 bit sono necessari 2 cicli di bus, uno per il primo byte, uno per il secondo. In base a come scelgo di agire la memoria metto prima il byte alto e poi il basso o viceversa.

Prima il byte + significativo (BIG ENDIAN) è come lavora Freescale che dunque non comunicherà con Intel che lavora al contrario, ossia mettendo prima il - significativo (LITTLE ENDIAN)

4) Bus:

mezzo di comunicazione che si suddivide in:

- BUS ADDRESS: per gli indirizzi
- BUS DATA: " i dati
- BUS CONTROL: " i controlli (leggi oppure scrivi)

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 63 di 106

In cosa mi devo portare un # di istruzioni ragionevole:

non le prendo a caso: né troppe né troppo poche ~▸ se ho delle fetch che poi vengono un jump quelle dopo le butto tutte

└▸ Nella 9S12 ho una prefetch che riempe una QUEUE (FIFO) di 3 valori su 16 bit!

oltre i 4 blocchi fondamentali, cos'altro c'è?

- CLOCK AND RESET GENERATION BLOCK (CRG):

il clock di sistema può essere generato in vari modi

1) Quarzo esterno sinusoidale che poi viene squadrato e usato
2) onda quadra che arriva direttamente dall'esterno
3) Oscillatorino interno che uso direttamente (soprattutto come routine di scorta, se gli altri 2 si rompono) ~▸ EMERGENZA

Più in generale può essere utile lavorare ad una frequenza che sia scorrelata dalla frequenza dell'oscillatore interno

└▸ A livello software io istanzio due numeri (m ed n) e poi il blocco (Phase Locked Loop) PLL aumenta con frazione $\frac{m}{n}$ la frequenza che arriva da una sorgente esterna.

⟹ TUTTO È PROGRAMMABILE

Dopodiché il sistema lavora con il SYSTEM CLOCK, che lavora a f doppia del BUS CLOCK e che resta dentro il processore, e il BUS CLOCK, appunto, che serve per lavorare con le periferiche

Conclusioni:

parto da dove voglio, con o senza PLL, e derivo BUS e CONTROL CLOCK.

- WAIT E STOP MODE:

serve perché se non c'è nulla da fare e il µP deve stare fermo è inutile fare lavorare il sistema, quindi congelo la macchina e lavoro LOW POWER. Clock fermo e macchina ibernata.

- PLL:

come descritto prima aumento in flessibilità aumentando/diminuendo la frequenza di lavoro moltiplicando quella della sorgente per una quantità $\frac{m}{n}$ (m e n salvati in 2 registri)

Ma dove li scrivo i dati del PLL?
Appunto in registri che sono SYNR e REFDV

$$PLLCLK = 2 \cdot OSCCLK \cdot \frac{SYNR+1}{REFDV+1}$$

PLL CLOCK FREQUENCY = SYSTEM CLOCK FREQUENCY

Per poi farlo funzionare davvero devo abilitarlo (PLLSEL) e decidere da dove prendere il clock sorgente (OSCCLK oppure PLLCLK)

∀ registro abilito o disabilito funzioni scrivendo 0 e 1 nei bit appositi e preposti alla funzione.

└▸ CONSULTO MANUALE !!!

NOTA: durante il funzionamento della macchina, se PLL_MODE = 1 non posso improvvisamente decidere di mettere PLL_MODE = 0.
D'altro canto posso cambiare m e n dunque cambiare PLLCLOCK ma non è detto che sia istantanea la modifica perché ci posso volere fino a tot colpi di clock in cui il sistema deve ovviamente stare fermo xk non ha un clock fisso.

- REGISTRI MACCHINA:

* A e B ~▸ registri dati da 8 bit ⟹ diventano D da 16 bit
* IX e IY ~▸ " puntatori da 16 bit
* SP: gestisce lo stack pointer
* PC: program counter

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 63 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 106

Torniamo ora un passo indietro e dopo aver analizzato alcuni particolari registri cerchiamo di capire come lavora la macchina.

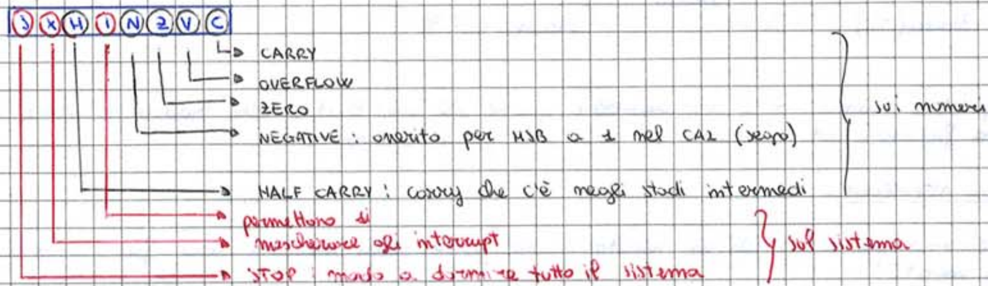
- Accumulatori

7	A	0	7	B	0
15			D		0

ldaa (#) $50 ⤳ carica nell'accumulatore A il dato (#) esadecimale ($) 50!
↳ immediate.

ld.d # $0150 ⤳ carica nel registro D il dato esadecimale 0150

- Condition code registers (CCR):



- Come lavora la macchina:

Parte e dunque parte da un reset ⤳ va alle istruzioni ultima e penultima (FFFF e FFFE) e legge da quali istruzioni deve partire. Questa è la prima cosa che la macchina fa. Dunque il processo è istruzione → ne prendo l'indirizzo e vado in memoria → ne ricevo il dato dalla memoria → eseguo l'istruzione in sequenza e vado avanti finché c'è da fare decodificando volta per volta l'istruzione (IR) puntata dal PC nella memoria.

(IR: contiene il codice operativo dell'istruzione $xxxx)

NB: Ci saranno istruzioni che durano 1 clock e istruzioni che ne durano di +.

Flow:

Reset ⤳ istruzioni dopo reset ⤳ vado alla memoria e prendo il dato ⤳ scrivo quel dato nel PC ⤳ quel dato punterà alla memoria ⤳ prendo il dato che è il codice dell'istruzione da eseguire, dunque lo metto nell'instruction register ⤳ ecc. ecc.

In definitiva:

ESEGUO LE ISTRUZIONI PUNTATE DAL PC. LA MEMORIA MI DA, ALL'INDIRIZZO PUNTATO, IL DATO CHE MESSO NELL'IR, CONTENENTE IL CODICE OPERATIVO, PERMETTE DI ESEGUIRE QUANTO DECODIFICATO!

↳ Ecco da dove escono fetch, decode e execute.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 67 di 106

CAPITOLO 15: C e ASSEMBLER

- Operatori

|	OR
&	AND
^	XOR
~	COMPLEMENTO
%	MOD

- Variabili:

1) GLOBALI e STATICHE:

sono inizializzate a zero e allocate nella RAM nello stesso ordine della loro dichiarazione

2) LOCALI:

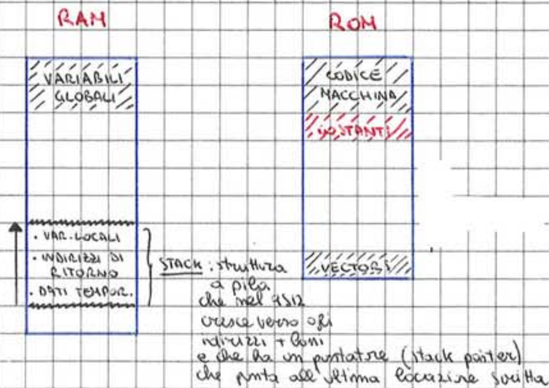
Sono inizializzate e allocate nello stack. Vivono solo laddove dichiarate e al di fuori non esistono.
Se voglio che una variabile locale sia dichiarata come "statica" allego che questa diventa globale (va nella RAM) ma la sua visibilità resta limitata

Un altro parametro che può essere dato alle variabili è VOLATILE:

sono variabili modificabili anche dal programma che non è quello che sto eseguendo!
Cosa succede? 2 cose:
- il compilatore non mette quella variabile in registri ma solo nella memoria
- quando quella variabile compare in un'espressione il compilatore ricalcola pedestremente quell'espressione senza fare alcuna ottimizzazione.

ORGANIZZAZIONE DEI DATI IN MEMORIA

Tutti i dati che gestisco nell'assembler che governa il mio embedded dove vengono messi in memoria?



NOTA: se lo stack pointer decresce fino ad arrivare alla locazione 0000 il passo successivo è andare a FFFF. Ho avuto uno stack overflow e da quel momento il sistema va in crash.

↓

Lo SP va sempre bilanciato oltre che per evitare lo stack overflow anche per evitare di andare a puntare in celle già scritte, dunque sovrascriverle.

PROGRAMMAZIONE STRUTTURATA TOP-DOWN

Il programma tutto FLAT non regge. Lavoro invece TOPDOWN lavorando a gerarchia e dunque a livelli diversi di complessità semplicemente richiamando queste routine.

↳ A livello assembler questa gerarchia è realizzabile con le SUBROUTINES!

Operazioni di base con subroutine:

- bsr ⇝ Branch to subroutine
- rts ⇝ return from subroutine
- jsr → Jump to subroutine

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 67 di 106

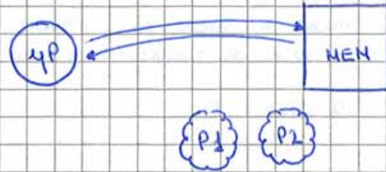
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 106

CAPITOLO 16: PARALLEL I/O

Il μC deve prelevare ingressi da porte d'ingresso e fornire uscite su porte di uscita. È questo il suo modo di comunicare con l'esterno.

! Ogni periferica interagisce attraverso opportune porte con il μP e le porte I/O devono essere precedentemente configurate (DEVICE INITIALIZATION) e cioè dire cosa deve fare ogni porta quando utilizzata.

I modi di gestire le periferiche sono due:



① MEMORY MAPPING:

Faccio convivere memoria e registri periferiche usando gli stessi indirizzi della memoria anche per accedere alle periferiche

- SV: parte della memoria viene mangiata dai registri delle periferiche
- V: tutto + compatto

② I/O MAPPED:

Distinguo gli spazi di indirizzamento di memorie e di periferiche. In questo modo non devo occupare parte della memoria effettiva (ho 64K effettiva) ma dovrò avere un segnale in + che mi dice "voglio parlare con l'indirizzo 315 delle periferiche, non con il 315 della memoria".

In passato era molto usato l'I/O MAPPED perché avevo memorie piccole. Oggi si usano le strutture memory mapping xk quell'ostacolo è superato e il 9S12 usa proprio questo.

Effettivamente come si interfaccia la singola porta con il nostro μP?

Ogni porta usa dei pin fisici del μP. Utilizza dei registri che ne forniscono controlli, dati e stati e cioè:

- DATA REGISTERS (R/W): permettono al μP di passare dati con il periferico sia in lettura che in scrittura
- CONTROL REGISTERS (W): abbiamo detto che il periferico va configurato e inizializzato. Per farlo sono usati questi registri che dal punto di vista del μP son a sola scrittura
- STATUS REGISTERS (R): registri che il periferico usa per informare il μP dello stato di certe operazioni, dunque per il μP a sola lettura

E quanti ne ho? Dipende dalla complessità di ogni periferico!

INTERFACE (PERIPHERAL) CHIP

Ho il μP da una parte e il periferico dall'altra. Come si parlano i due? Grazie all'interface chip

Da un lato comunica con il μP, dall'altro pilota e acquisisce dall'I/O. Esso è di fatto piazzato nel mio μC ed è appunto quell'insieme di porte che caratterizzano e personalizzano gli I/O.

Nel μC pertanto ci sarà tutta la struttura che permette di dialogare con il dispositivo periferico effettivo, quindi tutta la configurazione dell'interface chip viene fatta nel territorio del μC.

! Sarà dunque un cuscinetto tra μP e I/O DEVICE; è comunque un blocco hardware che con il μP sarà interfacciato esattamente come con la memoria dato che per il μP il peripheral chip è una serie di celle. Ciò che cambia è come esso parla con il device I/O e ci sono ≠ modi.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 71 di 106

CAPITOLO 17: EXTERNAL OUTPUTS

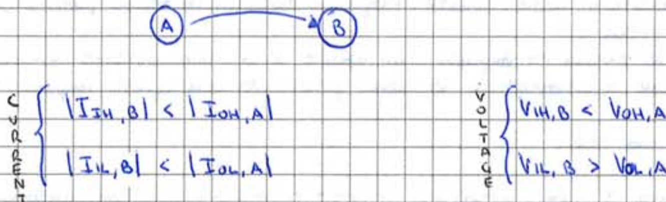
Che attacco all'esterno?

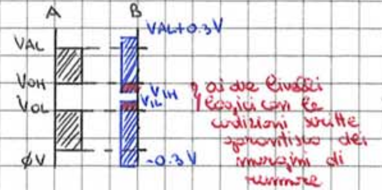
(possono essere sia analogici e digitali)

- INPUT DEVICES: sensori, user-input, ecc. ecc. (ogni cosa che da fuori viene usata come input per il µP)
- OUTPUT DEVICES: attuatori, display, ecc. ecc. (" " " voglio fornire come info fuori)

A livello di compatibilità tutte le periferiche devono essere pilotabili:

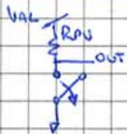
devo rispettare condizioni sulle tensioni e sulle correnti in modo da garantire la corretta comprensione da ambo i lati dei due livelli logici (H e L)

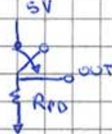




Facciamo ora una carrellata di dispositivi:

① INPUT SWITCH:





A prescindere dalla configurazione ho un terminale a tensione fissa e uno collegato a una resistenza. L'uscita in base a come è messo l'interruttore assume 5V o ØV dunque 1 o Ø logico.

② DIP SWITCH:

Nel caso in cui servono tanti interruttori si usano questi circuiti che sono integrati e per mezzo di una levetta di selezione esterna permettono di inviare dalla scheda al µP Ø o 1 semplicemente creando cortocircuiti o circuiti aperti. Erano quelli che c'erano sulla DE2 per esempio.
Sono usati per modificare agilmente dei segnali, dunque per la configurazione ad esempio.

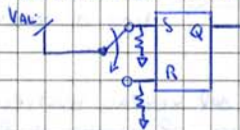
③ PUSH BOTTON KEYS:

A ≠ degli switch hanno una sola posizione di riposo, dunque lo schiaccio e lui si riapre poi.

Cosa succede? La lamina rimbalza, ritorna, oscilla, rimbalza, ecc. ~> Anzichè fare un contatto ne fa 40!
Questo meccanismo si chiama BOUNCING (⊓∿∿), dunque il regime arriva dopo una serie di oscillazioni

Dobbiamo attuare un meccanismo di debouncing che elimini i rimbalzi:

- SOLUZIONE HARDWARE: A) deviatore con latch SR



Trasformo il bottone in deviatore che attaccato a un latch può dare 3 cose: SET, RESET o SCONNESSO che vuol dire R=0, S=0 dunque condizione di memoria

∀ INTERRUTTORE = 1FF + 2 RESISTENZE

B) Buffer digitale

memorizza il valore quando lo switch è staccato
(costa meno di 1FF e 2 RES)

C) Circuito RC

Ho una τ lunga che non cambia il valore (Schmitt-trigger) del C (costa ancora meno) in modo che non sento le oscillazioni.

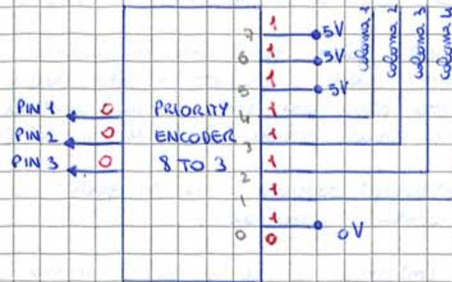
→ la soglia va superata dopo un tempo paragonabile alle oscillazioni

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 71 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 73 di 106

Pare dunque che ho ridotto i pin di lavoro ma non ho un sistema funzionante.

La soluzione consiste nell'usare, anziché un encoder 4:2, un encoder 8:3, sempre priority ma il bit in più consente di evidenziare la condizione nessun tasto premuto



Il mio priority encoder riconosce e tira fuori in binario la posizione dello ϕ a valore + alto!
Se non ho tasti premuti i bus delle colonne portano tutte 1 ed essendo il + basso imposto a zero esce "000" che io denoto come NO TASTI PREMUTI

SCANNING VS MULTIPLEXER: Pago il non avere la possibilità di vedere la pressione di tasti multipli come nello SCANNING ma ci guadagno l'uso di 5 PIN (2 righe + 3 colonne) contro 8 (4+4)

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 73 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 75 di 106

- Interrupt non mascherabili: * XIRQ (li mando dall'esterno ⇝ es: stampante)
  * OPCODE TRAP (interrupt di codice)
  * SWI:

  software interrupt istruction ⇝ causa un interrupt in assenza di un segnale richiesto. Esso è generato dall'istruzione SWI, ma perché nasce?
  * se il programmatore scrive in C lui esegue e poi gli torna il controllo. Il controllo gli torna grazie alla SWI che a termine esecuzione fa cose che restituiscono il controllo
  * quando debuggo e metto dei breakpoint io sto bloccando il sistema. Quando arriva al breakpoint il debugger sostituisce l'istruzione in cui si blocca con la SWI. Alla fine c'è lo scambio inverso e tutto riprende

  * RESET sia come Power on reset che come manual reset

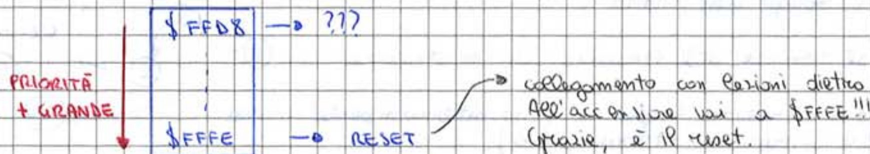
⇒ SE ARRIVANO DUE INTERRUPT NON MASCHERABILI LI ESEGUO SECONDO PRIORITÀ

- Interrupt mascherabili:

Se i non mascherabili sono pochi ma anche lì vige la PRIORITÀ, nei mascherabili che sono molti la PRIORITÀ vige a maggior ragione.

Sono tipicamente eventi legati ai periferici che posso dunque sentire immediatamente o posticiparli semplicemente settando a '1' il bit I della CCR. Quando li vorrò sentire lo forzerò a zero con opportune istruzioni quali CLI (clear I) oppure SEI (select I)

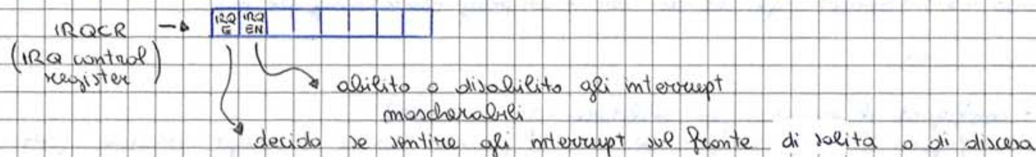
La priorità degli interrupt è generalmente + elevata in ordine discendente:



NOTA: se ho un interrupt (ADC) che trova sempre chi ha priorità + alta rischio di non sentirlo mai. Posso io decidere a quale interrupt dare priorità?
Sì, lo faccio scrivendo il vettore dell'interrupt nel registro (HPRIO) high priority rendo così quello il + privilegiato fra gli interrupt mascherabili → I NON MASCHERABILI NON LI SUPERO!!!

* $\overline{IRQ}$:

è un interrupt che può arrivare dall'esterno su qualsiasi porta collegata. Nella fase di configurazione devo settare anche se voglio sentire questo tipo di interrupt oppure no (IRQCR)



Ma conviene lavorare level sensitive o edge sensitive?

- FRONTE: se + dispositivi sono collegati non sono in grado di vedere 2 fronti insieme
- LIVELLO: a livello invece la linea viene messa a zero e a zero resta, dunque è come un meccanismo di OR ⇝ finché il µP campiona ∅ (ci sono interrupt) lui se ne accorge.
  Ma dato che ci sono campionamenti continui non devo fare avviare il sistema perché il dispositivo che servo mantiene ad interrupt la linea. Ciò che devo fare è dunque non campionare + fino alla fine del servizio dell'interrupt iniziato e inoltre terminare la ISR rimettendo a 1 (no interrupt) la linea del dispositivo appena servito in modo che si levi dalle scatole!

Ne vengono analizzati altri dietro + nel dettaglio!

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 75 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 77 di 106

INTERFACCIAMENTO TRA µP e PERIFERICI

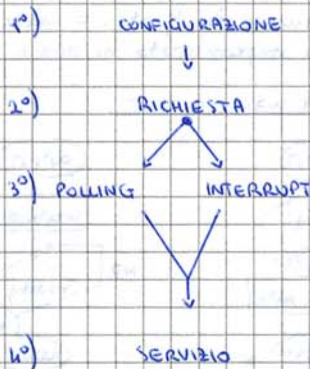
Oltre a tutto l'hardware visto (circuiti di interfaccia, peripheral chip) è necessaria una parte di software/firmware che gestisce il protocollo di comunicazione, ossia l'insieme delle istruzioni che devono essere svolte. Parliamo del DRIVER!!!

* Come misuriamo le prestazioni di interfacciamento?

  - LATENZA: Δt che c'è da quando il periferico chiede servizio a quando il µP finisce di servirlo! (non voglio che l'emergenza in auto si accenda dopo 10s)
  - BANDA (o TROUGHPUT): numero massimo di dati che possono essere processati. Può essere limitato da chi manda o da chi riceve che non riesce a smaltire tutti i dati.
  - PRIORITÀ: determina l'ordine di servizio quando ci sono + richieste. Impatta sulla frequenza del sistema perché la scelta di dare priorità a un periferico piuttosto che a un'altro incide.

* Sistemi real-time:
  - HARD REAL TIME: garantisco una latenza max ∀ periferico
  - SOFT REAL TIME: garantisco la presenza di una priorità che se mal gestita può creare latenza infinita per i meno prioritari!!

* Come avviene alla fin fine sta interfaccia?



Si nota come filosoficamente le due modalità agiscano nello stesso modo; cambia solo la modalità con cui l'interfaccia vera e propria arriva.

La differenza dov'è?

1. POLLING: il µP manda un dato al periferico ma l'interrupt del periferico è mascherato. Solo il µP comunica con il periferico e continuamente gli domanda se vuole servizio. (µP → P.)
2. INTERRUPT: il µP manda lo stesso dato al periferico ma l'interrupt questa volta parte dal periferico e va al µP. Ecco che sta la comunicazione funziona in ambo le direzioni e solo ad interrupt arrivato il µP interroga il periferico per chiedergli se è stato lui a mandarlo. (µP ⇄ P)

I tempi di comunicazione iniziano dunque a giocare ruoli importanti! Se il periferico è lentissimo il µP molto spesso si troverà in situazione di idle perché è fermo (I/O BOUND); se il periferico è una scheggia e il µP è lento ho la situazione opposta xk attende il periferico (CPU BOUND)

↳ Il mio obiettivo è che nessuno dei due aspetti!

La soluzione per non fare attendere nessuno qual è? Un buffer che salvi i dati in modo che chi ha bisogno va lì e se li prende senza attendere. Anzi, se il buffer è sufficientemente grande entrambi gli attori operano al max della velocità.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 77 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 79 di 106

③ INTERRUPT:

il µP non interroga + continuamente il periferico, ma solo quando arriva l'interrupt. Dunque il µP può fare altro nel mentre, oppure anche andare in qualche modalità low-power.

Cosa avviene all'arrivo dell'interrupt lo si era già analizzato! Dal salvataggio dei dati nello stack fino all'istruzione RTI.

I → Configurazione: devo abilitare gli interrupt dei periferici; esiste poi un secondo bit che permette di "armare" il periferico, ovvero dargli la possibilità di scatenare un interrupt quando ha un dato disponibile.
È come se caricassi la pistola e quando voglio sparo.
Posso inoltre decidere se l'interrupt va generato a livello (H o L) o a fronte (salita o discesa)

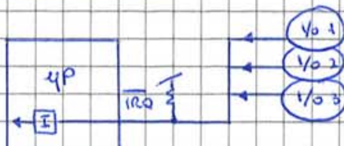
II → Richiesta: ho abilitato tutto, ho armato tutto ⇝ vado e faccio altro ⇝ quando l'evento arriva il periferico viene servito

ARM VS DISABLE ! { L'armamento deve essere fatto ogni volta che voglio che venga generato un evento successivo! Infatti, senza l'armamento quell'evento non si scatena dato che la pistola è scarica e non può sparare.
Con il bit "I" abilito a sentire tutti gli interrupt mascherabili che in quel momento sono armati oppure li disabilito tutti.

Una tra le cose che si fanno in una ISR è appunto disabilitare gli altri interrupt in modo da non essere interrotti (tocco il bit "I" della CCR). Con la RTI poi tutte le CCR sono ripristinate.

Dato che posso operare con + periferici come opero?

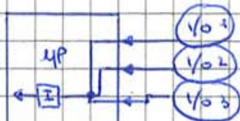
* SHARED REQUEST



La linea è condivisa e per le ragioni già viste devo lavorare solo con $\overline{IRQ}$ a livello e non a fronte

\+ ⇝ è facile aggiungere un nuovo periferico (+ FLEX)
⇝ semplice per l'hardware
− ⇝ + complicata per il firmware

* MULTIPLE DEDICATED



Ogni richiesta ha il suo bus, dunque posso, volendo, anche lavorare a fronte e non solo a livello!

− ⇝ aggiungere un periferico non è così semplice (− FLEX)
⇝ + complicata per l'hardware
\+ ⇝ + semplice per il firmware

III → Interrupt:
- salvo registri
- setto il bit "I" nella CCR
- vedo da dove partire e lo scrivo nel PC

IV → Servizio:
- esecuzione delle istruzioni
- riarmo se necessario
- RTI e il µP riprende da dove aveva interrotto

* Vantaggio di interrupt vs polling: il µP può fare altro

* Svantaggio di " vs " : ho un hardware + complicato

④ PERIODIC POLLING:

Se non posso gestire gli interrupt che faccio? Voglio che il µP lavori dunque questo modo è di compromesso fra polling e interrupt. Anziché fare un polling continuo eseguo un polling periodico dove il tempo è determinato dal blocchetto RTI.
È chiaro che perdo di efficienza rispetto all'interrupt ma almeno non passo la vita in polling!!

L'unico problema può essere che l'ennesimo periferico può stare in attesa per molto.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 79 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 81 di 106

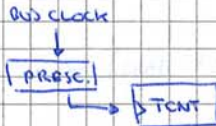
CAPITOLO 19: TIMERS

(esce una ogni due appelli)

Ho bisogno di avere funzionalità temporali per tutta la gestione del µP!!

Il cuore del timer è un contatore (TCNT nel 9S12) che lavora in modalità free running! Il clock del contatore è il bus clock (la conta avviene in base ai colpi di clock che passano) che può essere mantenuto così com'è oppure modulato attraverso un blocchetto (PRESCALER) che divide il clock proveniente dal bus e creando il nuovo clock su cui si basa il TCNT.
Il prescaler è programmato dalla scrittura di bit in un registro opportuno (tipicamente 3 bit, quindi il timer clock oscilla fra Bus-clock (000) e Bus-clock/128 (111) dove ogni configurazione è una potenza di due per il 9S12)

Negli altri µC io scalo il Bus-clock di quanto voglio (7, 15, 33) scrivendo in un registro opportuno!

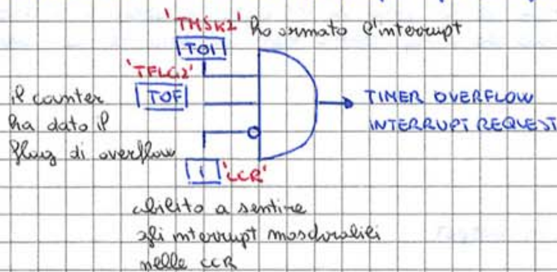


NOTA: il valore del prescaler può cambiare durante l'algoritmo, ma non è garantito che la modifica abbia effetto immediato ⤳ talvolta funziona solo dopo che il contatore è ritornato alla configurazione 0000.

Nel 9S12 il contatore è su 16 bit e avrò chiaramente un bit che abilita la conta (TE) e una volta abilitato NON LO FERMO PIÙ perché è free running!
Il valore normale di partenza al reset è ϕ (su 16 bit) e in certi casi + particolari posso dirgli di partire da un numero che decido.

(unica lettura) Il µP legge il valore del COUNTER grazie al registro TCNT, che ne contiene il valore. Il registro TCNT non è su 16 bit ma ha un byte HIGH e uno LOW. La lettura deve essere dunque fatta tutta insieme e non prima l'HIGH e poi il LOW o viceversa!

Quando il COUNTER arriva a tutti uni, lui riparte da tutti zeri e contemporaneamente setta un segnale di overflow che indica il passaggio dal via. Parliamo del TOF (time overflow flag) che è il MSB nel registro TFLG2. Capito che c'è stato un overflow io vado a scrivere in quel registro e l'operazione di scrittura resetta automaticamente quel bit TOF, riportandolo a ϕ.
Volendo posso decidere di mandare un interrupt ogni overflow:



NB: questo è un modo per scrivere un periodic polling se vogliamo.

ES: se voglio aspettare 1s lavorando con il counter a 16 bit e a 8 MHz

$T = \frac{1}{8MHz} = 125\ ns$ ⤳ $T_{CNT} = 125\ ns \cdot 2^{16+(n)} \sim 8\ ms$ ((n) → fattore di prescaler = 0) ⤳ $OVF = \frac{1\ s}{8\ ms} \sim 122$ overflow

In software posso eseguire sia in polling che in interrupt tutto ciò:

Polling:

- Resetto il bit di overflow (TOF)
- abilito il contatore a contare (TE)
- inizializzo il contatore a 122
- loop continuo di polling vedendo TOF è settato
- quando arriva il set su TOF, decremento il counter e riparto finché il counter non è a zero.

↓ se ci sono interrupt la macchina va avanti contando correttamente il tempo

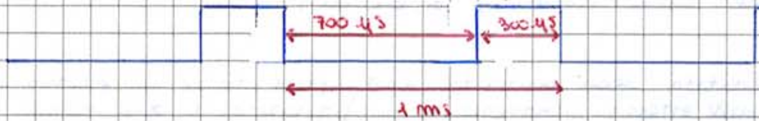
Interrupt:

- Configuro l'armamento dell'interrupt (TOI)
- " il non mascheramento (I)
- Configuro da quale indirizzo deve partire la ISR scrivendo l'indirizzo del PC
- Resetto il counter facendolo partire da zero
- Quando arriva un interrupt incremento il counter, rimetto TOF a ϕ e ritorno a dormire
- quando arriva il CNT a 122 sono arrivato a 1 secondo; questo è fatto comparando il valore del contatore con 122.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 81 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 83 di 106

esempio: generare un'onda quadra alla frequenza di 1KHz con DC = 30% e di farlo sul canale ϕ. Ipotizzare di lavorare in polling supponendo di avere una frequenza del bus clock di 24 MHz.

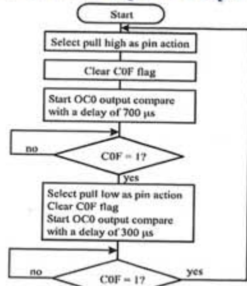


setto il PRESCALER a PR = 011 (vale 8) ⟶ Timer. clock = 3 MHz

Ogni intervallo del contatore vale $\frac{1}{3MHz} = \frac{1}{3}$ µs

Devo programmare il sistema perché conti 900 (per la fase alta) e 2100 (per quella bassa)

Operation of the Output-Compare Function



The program logic flow for digital waveform generation

C Program for Generating 1 KHz Digital Waveform

```
#include "9s12.h"
#define hi_time 900
#define lo_time 2100
void main (void)
{
    TSCR1    = 0x90;             /* enable TCNT and fast timer flag clear */
    TIOS  |= OC0;                /* enable OC0 function */
    TSCR2    = 0x03;             /* disable TCNT interrupt, set prescaler to 8 */
    TCTL2 = 0x03;                /* set OC0 action to be pull high */
    TC0      = TCNT + lo_time;   /* start an OC0 operation */
    while(1) {
        while(!(TFLG1 & C0F));   /* wait for PT0 to go high */
            TCTL2 = 0x02;        /* set OC0 pin action to pull low */
            TC0   += hi_time;    /* start a new OC0 operation */
        while(!(TFLG1 & C0F));   /* wait for PT0 pin to go low */
        TCTL2   = 0x03;          /* set OC0 pin action to pull high */
        TC0  += lo_time;         /* start a new OC0 operation */
    }
}
```

esempio: generare un ritardo multiplo di 1 ms dove quanti devono essere è un parametro che pongo io. Bus clock lavora a 24 MHz e setto il prescaler a 64. Usare la output compare in polling

f Timer. clock = $\frac{24 \text{ MHz}}{64} = 375$ KHz ⟶ tempo di ogni incremento = 2.6667 µs

dovendo aspettare 1 ms ⟶ $\frac{1 \text{ ms}}{2.6667 \text{ µs}} = 375$ ⟹ incrementi del free running counter per fare passare 1 ms

```
void delayby1ms(int k)
{
    int ix;
    TSCR1 = 0x90;          /* enable TCNT and fast timer flag clear */
    TSCR2 = 0x06;          /* disable timer interrupt, set prescaler to 64 */
    TIOS  |= OC0;          /* enable OC0 */
    TC0   = TCNT + 375;
    for (ix = 0; ix < k; ix++) {
            while(!(TFLG1 & C0F));
            TC0 += 375;
    }
    TIOS &= ~OC0;          /* disable OC0 */
}
```

Fast timer flag clear: è una modalità che alcuni µC offrono e serve per resettare automaticamente un flag la prima volta che accedo al registro in cui esso è contenuto; evito in questo modo di dover fare un reset puntuale del flag ogni volta.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 83 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 85 di 106

esempio: misurare il periodo di un segnale con la IC sul canale ϕ. Il periodo si sa che è meno di 128 ms. Assumere la $f_{bus\ clock} = 24\ MHz$. Lo faccio in polling.

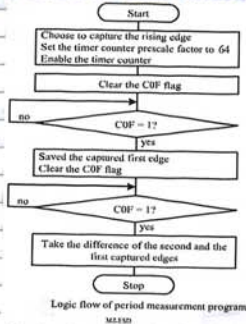
Il contatore fa un giro completo dopo $2^{16} \cdot \left(\frac{1}{24\ MHz}\right) = 2.73\ ms$ senza prescaler.
Dovrei così fare una considerazione valutando tutti gli overflow che arrivano.

Di quanto posso dividere la frequenza del bus per avere un periodo maggiore di 128 ms che mi garantisce così che non ho avuto overflow?

$$\frac{128\ ms}{2.73\ ms} = 46.88 \rightsquigarrow \text{metto } 64!$$

Sono entrambe due strade possibili e implementabili con accortezze opportune.

Ipotizziamo di aver scelto la strada che divide il bus-clock per 64.



Logic flow of period measurement program

C Program for Period Measurement

```c
#include "9s12.h"
void main(void)
{
    unsigned int edge1, period;
    TSCR1  = 0x90;   /* enable timer counter, enable fast flag clear*/
    TIOS  &= ~IOS0;  /* enable input-capture 0 /
    TSCR2  = 0x06;   /* disable TCNT overflow interrupt, set prescaler to 64 */
    TCTL4  = 0x01;   /* capture the rising edge of the PT0 pin */
    TFLG1  = C0F;           /* clear the C0F flag */
    while (!(TFLG1 & C0F));      /* wait for the arrival of the first rising edge */
    edge1 = TC0;          /* save the first captured edge and clear C0F flag */
    while (!(TFLG1 & C0F));      /* wait for the arrival of the second rising edge */
    period = TC0 - edge1;
    asm ("swi");
}
```

esempio: misurare la lunghezza di un impulso con IC sul canale ϕ. Il bus clock frequency è 24 MHz e il prescaler è a 32.
Ipotizziamo che la durata dell'impulso può essere > di 2^{16} clock cycles. Dobbiamo tener conto del # di overflow.

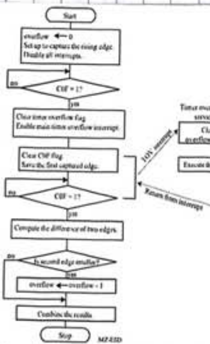
$\rightsquigarrow T = 2^{16} \cdot m + \underbrace{STOP - START}_{\text{con segno}}$

Se invece volessi fare STOP - START senza considerare il segno?

OVCNT = # di OVF
DIFF = differenza tra 2 fronti consecutivi
EDGE1 = primo fronte catturato
EDGE2 = secondo " "

CASO 1: EDGE2 > EDGE1 $\rightsquigarrow T = OVCNT \cdot 2^{16} + DIFF$

CASO 2: EDGE2 < EDGE1 $\rightsquigarrow T = (OVCNT - 1) \cdot 2^{16} + DIFF$



C Program for Pulse Width Measurement

```c
#include   <9s12.h>
#include   <vectors12.h>
unsigned  diff, edge1, overflow;
unsigned  long pulse_width;
void INTERRUPT tovisr(void);
void main(void)
{
    UserTimerOvf = (unsigned short)&tovisr;
    overflow     = 0;
    TSCR1        = 0x90;       /* enable timer and fast flag clear */
    TSCR2        = 0x05;   /* set prescaler to 32, no timer overflow interrupt */
    TIOS        &= ~IOS0;      /* select input-capture 0 */
    TCTL4        = 0x01;       /* prepare to capture the rising edge */
    TFLG1        = C0F;        /* clear C0F flag */
    while (!(TFLG1 & C0F));    /* wait for the arrival of the rising edge */
    TFLG2        = TOF;        π/* clear TOF flag */
    TSCR2       |= 0x80;       /* enable TCNT overflow interrupt */
    asm("cli");
    edge1        = TC0;        /* save the first edge */
    TCTL4        = 0x02;       /* prepare to capture the falling edge */
    while (!(TFLG1 & C0F));    /* wait for the arrival of the falling edge */
    diff         = TC0 - edge1;
    if (TC0 < edge1)
        overflow  -= 1;
    pulse_width = overflow * 65536 + diff;
    asm ("swi");
}
void INTERRUPT tovisr(void)
{
    TFLG2         = TOF;        /* clear TOF flag */
    overflow = overflow + 1;
}
```

Nella misura di una forma d'onda ci imbattiamo in 3 parametri:

1. RISOLUZIONE: minimo intervallo di tempo misurabile di cui possiamo apprezzarne l'esistenza
2. PRECISIONE: numero di misure separate e distinguibili che puoi fare
3. RANGE: intervallo di tempo in cui effettuo la misura

Laddove gli interrupt vadano misurati + in fretta di quanto l'hardware mi consenta, una buona idea è quella di sdoppiare i canali di input capture

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 85 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 87 di 106

CAPITOLO 21: EXTERNAL OUTPUTS

Dopo avere visto tutti i dispositivi che possono agire come external inputs (ne mancano ancora alcuni) vediamo che cosa il μC può pilotare:

DISPLAYS	ATTUATORI
· LCD	· MOTORI
· CRT	· SOLENOIDI
· LED	· RELAYS
· ⋮	· ⋮

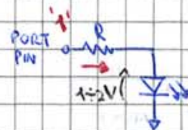
Agirò sempre su porte del μP, dunque pilotaggio vorrà dire scrivere sulle porte di I/O programmate opportunamente come porte di uscita.

Laddove io debba pilotare uscite su 32 bit (4 porte da 8 bit) ma ho solo 2 porte da 8 bit disponibili, come faccio? O cambio μP oppure espando quelle porte con le comunicazioni seriali di cui si parlerà poi.

Come posso gestire l'interfaccia del μP con il dispositivo I/O esterno?

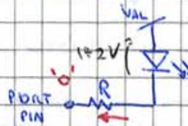
① LED:

• Catodo comune:



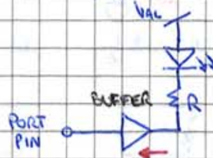
Il catodo è collegato a massa. Per accendere il LED la porta deve generare un '1' logico sull'anodo forzando una tensione Voh che fa passare, grazie ad R, una corrente nel LED che si illumina. In questo caso la porta EROGA corrente (sourcing)

• Anodo comune:

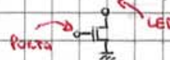


L'anodo è collegato all'alimentazione. Per accendere il LED la porta deve generare uno '0' logico sul catodo forzando una tensione Vol che fa passare, grazie ad R, una corrente nel LED che si illumina. In questo caso la porta BEVE corrente (sinking)

• Open drain / open collector:



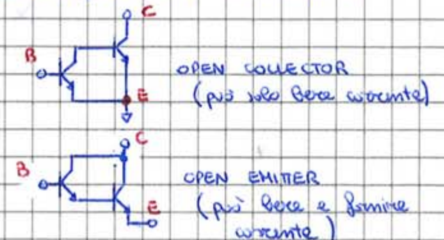
Se la porta in questione non è in grado di pilotare il LED, bisogna metterci in mezzo un buffer che non è altro che un dispositivo open drain (MOS) o open collector (BJT) aventi un terminale a massa e uno floating su cui attacco il LED



In questo caso la porta saprà solo BERE la corrente

Dal momento che i periferici vengono poi pilotati su + bit e non solo uno, esistono dei circuiti integrati che sappiano fornire correnti relativamente ingenti su + bit. Alcuni di questi sono i cosiddetti LED DRIVERS:

Realizzati con la tecnica del Darlington:



In definitiva il mio μP non è spesso in grado di pilotare direttamente il dispositivo. Si interpongono dei dispositivi (DRIVER) come quelli sopra mostrati che gli consentono di superare i suoi limiti e due tipologie sono quelle mostrate.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 87 di 106

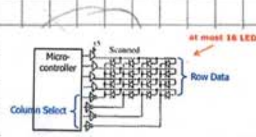
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 89 di 106

→ fino a 16/porta

Soluzione 3) PILOTAGGIO SCANNIG:

È + o meno la soluzione esaminata per la tastiera.

Ovvero: i led sono inseriti in una matrice definita per righe e colonne. Ogni led è inserito all'incrocio di ogni riga colonna. Per accendere ogni led io dovrò forzare un '1' sul suo anodo e uno '0' sul suo catodo → ecco che passa corrente nel led e ne provoco l'accensione.



Per le righe effettuo un collegamento con un driver open emitter in modo che eroghi corrente a partire dalla tensione di alimentazione e una resistenza (→ impone val. riga da pilotare)

Per le colonne il meccanismo è il medesimo ma devo forzare uno ϕ per porre a ϕ i catodi, dunque metto un driver di tipo open collector.

LED ACCESO = RIGA '1' + COLONNA '0'

NOTA: questo meccanismo mi permette di accendere contemporaneamente solo i led di una stessa colonna. Non posso accendere insieme due led di due colonne distinte

Ma se voglio accenderne di colonne diverse come faccio?

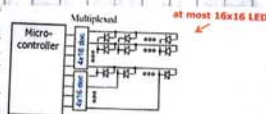
Fisicamente non posso, il trucco è quello di usare le scarse capacità dell'occhio che memorizza l'info per un massimo di 20 ms. Se io switcho tra 2 led ad una frequenza tale da non permettere all'occhio umano di stare dietro a tutto, l'occhio media ciò che vede e all'apparenza li vede tutti e due accesi.
Con questo modo posso simulare di avere tutti i 16 led accesi contemporaneamente. Basta solo switchare una colonna per volta e in modo continuo.
L'occhio umano medierà anche la luminosità di ogni led.

Al crescere dei bus riga e colonna la matrice quadrata aumenta e posso pilotare + led.
Dato che su ogni porta ho 8 bit però, con una porta più che 16 led non piloto (4x4).
L'alternativa all'uso di + porte da 8 bit affiancate è lavorare con un'altra modalità:

→ fino a 256/porta

Soluzione 4) PILOTAGGIO MULTIPLEXED:

Il pilotaggio di riga e colonna della matrice subisce una piccola modifica perché il μP questa volta non va direttamente sulla matrice di led ma i suoi bus passano uno stadio di decoder per le righe e uno per le colonne. La riga e la colonna che il μP scrive non saranno altro che la codifica binaria della riga e della colonna da pilotare.



↓

8 bit / porta ⇒ 4 riga e 4 colonna

↓

il decoder sarà DEC4TO16 e quindi posso pilotare 16x16 LED

↓

Ogni porta pilota 256 LED !!!

Che cosa pago?

Ogni combinazione riga-colonna ora accende un solo led per volta, quindi se li voglio tutti accesi come prima il μP non deve avere altro da fare per fare sì che la frequenza di switch che l'occhio non vede sia garantita. Ogni LED è acceso per 1/256 del tempo e quindi la corrente che ci passa deve essere tale da mostrarlo acceso alla vista (occhio a non bruciare una δ che lo bruci!)

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 89 di 106

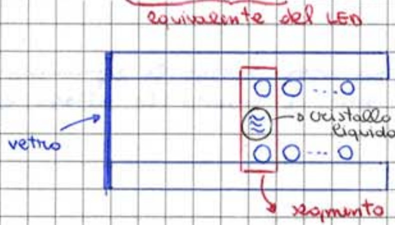
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 91 di 106

se poi il mio μC non si può permettere di pilotare dei display sette segmenti, uso dei dispositivi esterni preconfezionati che sono gli SPI

③ LCD:

Liquid Cristal display hanno sostituito i display 7 segmenti per ovvie ragioni!

Come funziona un segmento LCD? equivalente del LED



Ho due grandi famiglie di LCD:

RIFLESSIVI: uno dei due terminali funziona da riflettore di luce esterna. se il segmento fa passare la luce vedo quel segmento illuminato, altrimento è opacizzato. Il vantaggio è che consuma pochissimo, quasi nulla xk la visione si deve alla luce esterna

TRASMISSIVI: deve esserci una sorgente luminosa che accende i segmenti.

Come piloto un segmento LCD?

il segmento si attiva se applico ai suoi terminali un segnale alternato a frequenze dai 30 Hz al chiloHz; tale segnale cambia il liquido di mezzo che cambia così la sua luminosità.
Se metto un segnale in DC quel liquido viene irreparabilmente danneggiato!

! NO DC, SÌ AC

Come genero quel segnale in AC allora:

l'oscillatore a 60 Hz va da una parte diretto al segmento, dall'altro all'XOR.

se CNTRL='0' il segnale passa tale e quale la XOR fino esattamente uguale sui 2 terminali del segmento che vedrà così tensione nulla ⇝ OPACIZZA

se CNTRL='1' il segnale viene commutato dalla XOR e va sul terminale speculare a quello diretto con il risultato che ai capi del segmento ho $2V_p$ o $-2V_p$! ⇝ ILLUMINA

Anche in questo caso uso poi strutture matriciali in modo da pilotare + segmenti !!!

- Applicando tensioni in fase con una di riferimento ⇝ OPACIZZO
- " " " controfase " " " ⇝ ILLUMINO

Nelle matrici con almeno 4 segmenti il trucco è usare una tensione che non accenda il segmento in condizioni normali, e cioè:

genero un'onda quadra che oscilla fra V_{AL} e $-V_{AL}$ ma che abbia anche fasi a ϕ ($V_{AL}/2$, $-V_{AL}/2$) inoltre $V_{AL}/2$ in modulo non basta per accendere il segmento.
Se io voglio accenderne uno devo shiftare l'onda quadra apposita all'altro terminale una uguale e opposta in modo che ai capi si generi V_{AL} almeno. Per quelli che non voglio accendere shifto opportunamente per avere ai capi del segmento una tensione $\leq \frac{V_{AL}}{2}$!!!

↳ il trucco è fare shift di $\frac{1}{4}$ di periodo ogni onda (4 LCD)
" " " " " " $\frac{1}{8}$ " " " " (8 LCD)
⋮
e così via!!

ALLA FINE... { il μP non pilota direttamente l'LCD! Ho in commercio controllori appositi che si interfacciano con il μP e fanno tutto loro!

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 91 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 93 di 106

Il driver, nella realtà, è un transistor dunque io devo scontrarmi anche, nel progetto, con i parametri del dispositivo:

(A) se è un BJT ~▷ · V_{be}
· h_{fe}
· $V_{ce_{SAT}}$

⇒ I parametri e i loro valori cambiano significativamente a seconda che io lavori in piccolo segnale piuttosto che nell'elettronica di potenza.

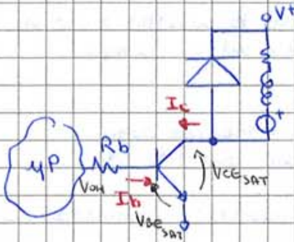
Lavora dunque in due zone:

- LINEARE:

$I_C = h_{fe} \cdot I_b$ se $V_{BE} \leq V_{BE_{SAT}}$

- SATURAZIONE:

$I_C < h_{fe} \cdot I_b$ se $V_{BE} > V_{BE_{SAT}}$ e $V_{CE} = V_{CE_{SAT}}$



⇒ Io sono in una situazione uguale!
Quando il transistor satura (o—o), la tensione sull'avvolgimento è fissa, dunque lo è anche la corrente I_C, che è quella dell'avvolgimento!
I_C è un parametro di progetto e per garantire la saturazione basta progettare il circuito d'ingresso garantendo che $I_b > I_C / h_{fe}$

Progetto:

- V_{AL}: note le caratteristiche elettriche dell'avvolgimento (4A - 12V), a seconda della $V_{CE_{SAT}}$ che cambia a seconda delle correnti in gioco io ho subito chiaro il valore di V_{AL}:

$$V_{AL} = V_{AVV} + V_{CE_{SAT}}$$

NB
Fatto prima di V_{AL} in modo da sapere $V_{CE_{SAT}}$ e poter calcolare V_{AL}

- Scelta del transistor:

devo scegliere uno che mi garantisca la corrente di cui ho bisogno

- Progetto la parte d'ingresso:

garantire che il transistor saturi ~▷ $I_b > I_C / h_{fe}$

dunque scelgo R_b ~▷ $R_b < (V_{OH} - V_{BE_{SAT}}) \frac{1}{I_b} = h_{fe} \cdot (V_{OH} - V_{BE_{SAT}}) \frac{1}{I_C}$

! Da quel valore è buona norma prenderne la metà per avere un problema risolto in maniera ingegneristica!!!

(B) se è un MOS è ancora più semplice il tutto perché tecnicamente non serve progettare la resistenza d'ingresso.

Di fatto poi la si introduce (~1KΩ) e così evito che nelle commutazioni il μP debba dare picchi di corrente molto grandi perché i MOS usati sono di potenza, dunque le capacità interne sono grosse.

L'avvolgimento è composto da resistenza + induttanza ~▷ quando il driver satura e si comporta da corto circuito la corrente cresce con andamento esponenziale:

$$\tau = L / R$$

La costante di tempo è un parametro essenziale nel progetto per certe applicazioni! Rischio che la corrente ci metta tanto ad arrivare al regime

$$V_C(t) = (V_{C0} - V_{\infty}) e^{-t/\tau} + V_{\infty}$$
$$I_L(t) = (I_{C0} - I_{\infty}) e^{-t/\tau} + I_{\infty}$$

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 93 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 95 di 106

② SOLENOIDE:

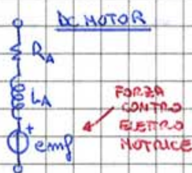
Il solenoide contiene all'interno dell'avvolgimento un nucleo ferromagnetico. Su scala diversa rispetto al relay, il suo obiettivo è lo stesso: spostare tale bacchetta che provoca una variazione su determinate e svariate cose.

La grandezza del solenoide è ovvio che dipende dalla grandezza di ciò che deve spostare come grandezza fisica! Il principio di funzionamento è lo stesso dei relays dato che le posizioni di un solenoide sono ON e OFF.
Se vogliamo oggetti che abbiano + posizioni bisogna spostarsi ai motori

③ DC MOTORS:

Costituiti da uno statore (parte fissa) che è composta da magneti che generano il campo; all'interno dello statore vi è un rotore con degli avvolgimenti che, se percorsi da corrente, fanno ruotare il rotore ~> se ruotato ci sarà una forza contro elettro motrice che tenderà a riportarlo alla condizione di partenza.
Ciò che si fa è creare una serie di avvolgimenti che, scelto uno con un collettore a spazzole rotanti, posso continuare la rotazione applicando corrente dove necessario. L'inversione di corrente provoca l'inversione del senso di rotazione.

Per il resto il motore ha uno schema già visto

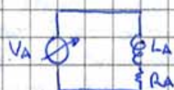


La fem dipende dalla velocità di rotazione ma anche dalla coppia applicata nella rotazione
Segue che la corrente nell'avvolgimento dipende da tale forza e poiché la tensione ai capi dell'avvolgimento è costante, tanto + grande è la emf e + è piccola la corrente; essa cambia inoltre anche in funzione del carico applicato!!!

Il pilotaggio di questi dispositivi avviene fissando una corrente I_A nell'avvolgimento e guardando quale tensione si localizza ai capi dello stesso. Per quanto detto la corrente non ha andamento lineare perché cambia con la velocità di rotazione e con il carico perciò questi motori hanno senso se in qualche modo riesco a controllarla (CATENA CHIUSA CON SENSORI).

Pilotaggi:

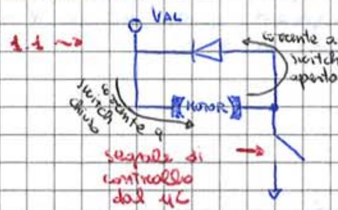
• TENSIONE VARIABILE:



Segnale variabile = corrente variabile ~> non va bene come soluzione però perché oltre a dissipare a riposo se il motore è grande le correnti lo sono pure

• DIGITAL OUTPUTS:

1.1 ~>



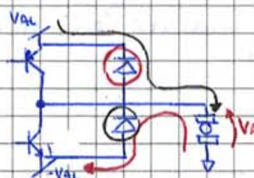
Con un'interfaccia digitale così il motore lavora in 2 condizioni:

Ciò che faccio però è applicare una tensione VAL a frequenze di switch dell'interruttore superiori alla frequenza del polo meccanico dello stesso che quindi assume comportamento da LPF ~> vede solo il valore medio!

Se poi con la tecnica del PWM vario il duty cicle dell'onda che controlla l'interruttore vario di conseguenza il valor medio del segnale applicato ai capi del motore. Posso così variare la tensione sull'avvolgimento e pilotare il motore in modo continuo anche con un pilota che di base lavora ON-OFF.
Il vantaggio è anche energetico!!!

Raddoppiando l'hardware posso usare il motore in due quadranti (↷ e ↶): basta che si giri la corrente

1.2 ~>



CONFIGURAZIONE HALF-H-BRIDGE

Ha l'inconveniente di dover avere 2 alimentazioni

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 95 di 106

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 97 di 106

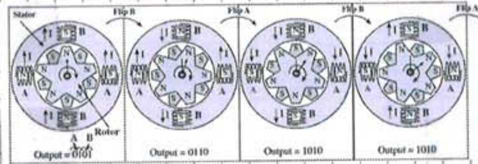
→ Esistono anche modalità di applicazione "HALF-STEPPER" in cui raddoppio la risoluzione del motore semplicemente pilotando insieme le due fasi φ_A e φ_B. Se in un 4 poli la risoluzione FULL-STEPPER era di 90° (), nello stesso motore ma HALF-STEPPER trovo le posizioni intermedie, dunque risoluzione di 45° (). Il pilotaggio half stepper si ottiene alternando l'accensione di una fase (raggiungo i quarti) e due fasi (raggiungo gli ottavi).

! HALF-STEP nelle fasi in cui pilota φ_A e φ_B insieme crea una coppia doppia rispetto a quando c'è solo φ_A (o φ_B). La coppia è variabile e in certi casi mi va bene, in altri no.

La tecnica + evoluta è quella del "μSTEPPING" che in base alla quantità di corrente che scorre negli avvolgimenti consente di far ottenere risoluzioni maggiori!

→ Se ora però vogliamo avere sempre 2 avvolgimenti, senza lavorare in μSTEPPING, posso ottenere risoluzioni + fini di 45° eseguendo + di 8 passi per fare un giro completo? Sì, si usa la tecnica del "BIPOLAR STEPPER":

Il rotore è composto da una stella a 5 punte (NORD) applicata ad un'altra stella a 5 punte (SUD). Per la minima riluttanza ho un nord e un sud messi davanti allo statore e poi gli altri NORD e SUD sono a metà strada!

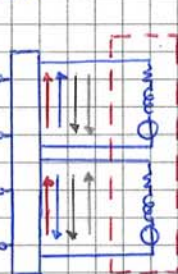


Ogni 4 step (corrente dritta su A, dritta su B, inversa su A, inversa su B) il movimento è uguale a:

$360° : 5$ poli $: 4$ step $= 18°$.

⇒ se aumento i poli aumento la risoluzione!!

Nel bipolare stepper (TWO-PHASE-ON) dei due avvolgimenti attivi solo una alla volta viene invertito!



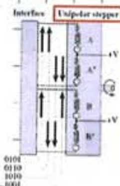
BIPOLAR STEPPER

Cambiare l'ordine con cui piloto le correnti fa cambiare giro al rotore!

Ho un problema però: per farlo funzionare la corrente deve andare prima in un verso e poi nell'altro! Questo vuol dire tornare alla situazione in cui ho di nuovo due alimentazioni

→ Utilizzando la tecnica degli "UNIPOLAR STEPPER" la corrente non cambia verso!

L'unico modo per generare campi opposti partendo dalla stessa struttura è mettere due avvolgimenti avvolti in sensi opposti in modo che in base a quale attivo, avendoci un nodo comune, ottengo una rotazione da una parte o dall'altra. Faccio lo stesso sul secondo avvolgimento e ho risolto!



! L'unipolare lo riconosco xk ha 5 morsetti che escono (4 + i comuni), il bipolare solo i 4.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 97 di 106