NUMERO: 1054 DATA: 02/09/2014

A P P U N T I

STUDENTE: Rossi

MATERIA: Controllo Digitale di Convertitore e Azionamenti

Prof. Pellegrino

CONTROLLO DIGITALE DI CONVERTITORI E AZIONAMENTI.

prof. PELLEGRINO

INDICE

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 164

CONTROLLO

Con il termine SISTEMA EMBEDDED si identificano generalmente TUTTI quei sistemi elettronici di elaborazione e microprocessore progettati per una particolare applicazione ossia non riprogrammabili dall'utente per altri scopi. Un sistema embedded ha dei compiti noti già durante lo sviluppo, che eseguirà dunque grazie ad una combinazione hardware/software specificatamente studiata per tale applicazione. Grazie a ciò l'hardware può essere ridotto ai minimi termini per ridurre lo spazio occupato riducendo così anche i consumi, i tempi elaborazione, e costo di fabbricazione. Inoltre l'esecuzione del software è spesso in real Time per permettere un controllo deterministico dei tempi di esecuzione.

La centralina è una scheda digitale che svolge azioni di controllo su una parte di sistema. Ha un coordinamento gerarchico e gerarchico di queste centraline. EMBADDED indica che la scheda è autonoma/dedicata per svolgere particolari funzioni. Le schede ha una buona dote di flessibilità

I componenti principali sono:

* CPU (processore)
* periferiche (comandi, misure, e comunicazioni con altre parti di sistema).

le comunicazioni vanno su linea seriale (digitale)

1

Campiono in certi istanti e dopo finestre che mi dicono cosa fa la variabile (I) per resto del tempo non so nulla.

Quantizzare vuol dire che il numero che devo convertire lo faccio su un certo valore di n° di bit.

La quantizzazione è un valore di troncamento.

l'errore che si commette nel troncamento è trascurabile in un processore a 64 bit

Me ho da considerare l'errore di troncamento dovuto a conversione analogico-digitale; me volta che sono in digitale non problemi con la quantizzazione

(es)

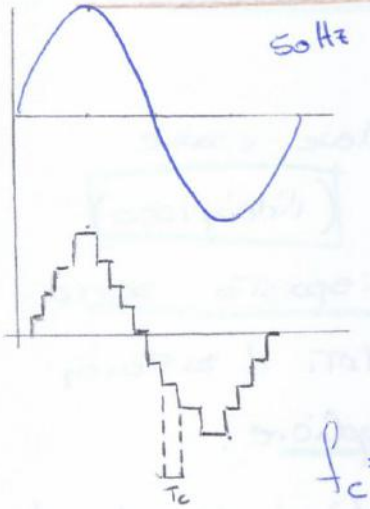
Considero una sinusoide ±10A su 256 livelli (8 bit)

la risoluzione $= \frac{20A}{256}$

→ RITARDO DI ESECUZIONE occorre aspettare intervallo successivo (interrupt successivo).

CAMPIONAMENTO

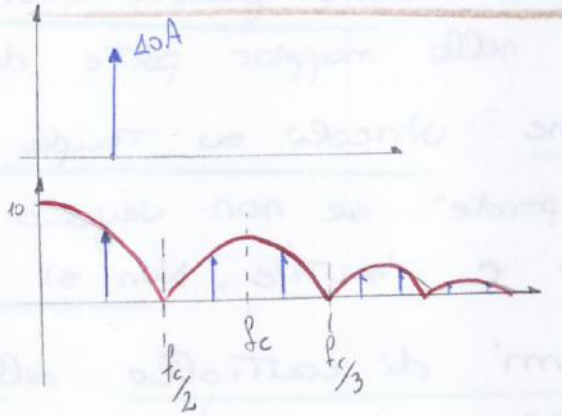
Si commette errore in quanto la grandezza analogica la conosco solo in alcuni intervalli di Tempo.



$f_c = \frac{1}{T_c}$

Il segnale a 50Hz è campionato

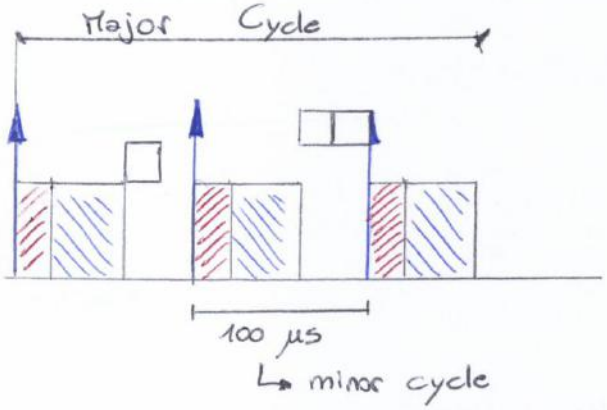
possiamo analizzare lo spettro di f



TEMPORIZZAZIONE ALGORITMO DI CONTROLLO

Se abbiamo algoritmi di controllo di tipo discreto la struttura tipica dell'algoritmo è un ciclo infinito che altera la lettura dei sensori, il calcolo delle risposte e del prossimo stato, il comando attuatori. Molti sistemi embedded trovano però applicazione in sostituzione di sistemi di controllo analogici preesistenti, dove il controllo viene invece effettuato in modo continuo: le grandezze fornite dai sensori vengono elaborate in modo analogico per fornire comandi agli attuatori in ogni istante, spesso in quello che si chiama anello di controreazione. Nel controllo digitale di questi sistemi vale a dire le grandezze misurate dai sensori possono essere rilevate solo ad istanti di tempo discreti. Il Teorema del campionamento di Nyquist-Shannon garantisce che se un segnale viene campionato ad una sufficiente frequenza, l'insieme dei campioni basta a ricostruire il segnale senza perdita di informazione. In particolare il teorema dice che dato un segnale di cui si conosce la frequenza massima f, la frequenza minima di campionamento F per evitare perdita di informazione è $2 \cdot f$.
Si può perciò compiere una discretizzazione delle misure continue ottenute da sensori analogici.
La frequenza di campionamento F dei sensori dovrà quindi essere maggiore del doppio di essa. Tale F definisce il periodo $T = 1/F$ dell'algoritmo di controllo. In un sistema Real Time si può definire il periodo tipico dell'algoritmo di controllo:

Devo avere un Tempo sufficiente.
Posso avere un MULTITASKING STATICO.
Ossia il Tempo in più lo uso per altri processi che non sono necessari ogni ciclo poiché se lo fare! ogni ciclo non ci starei nei Tempi.



Devo comunicare con l'esterno e calcolare la velocità ogni µs.

Faccio quello che è detto "sequenzer" cioè fare il TASK in sequenza in modo da non fare casini
Es: faccio ogni 2 la comunicazione e ogni 4 il calcolo velocità.

Ci occupiamo di audizzare ST 62xx. Ci occupiamo di audizzarlo e non di programmarlo.

E' un microcontrollore a 8 bit con A/D converter, con 2 Timer e quando lo si resetta da solo si controlla

Il singolo cip contiene

* processore
* memoria
* periferide

Di queste famiglie di cip ne abbiamo uno de ha, una finestrella. Questa serve per cancellare

Abbiamo una "program memory" (dove scrivo il programma) ve de 1K o 2K.

Una RAM da 64 byte. Una alimentazione operativa de va da 3 a 6 V. Una frequenza di clock di 8MHz max cle vuol dire de la fclock non è fissata ne scelta dal progettista, poiché i dispositivi digitali consumano quando funzionano altrimenti sono passivi. Le elettroniche abbiamo 2 tecnologie

TTL
hanno bisogno di essere sempre alimentati (polarizzati)

CMOS
consumano in virtù delle operazioni de fanno

I microcontrollori della famiglia ST62xx a 8 bit sono usati in applicazioni di media complessità basati su un approccio modulare: un nucleo comune è circondato da una serie di periferiche on-chip

ST62E01C è la versione programmabile EPROM cancellabile. Le parole abbiamo 3 versioni

- OTP
- EPROM
- ROM

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 164

Il dispositivo è di 16 pin.
Ho un bus principale dei dati e degli indirizzi.
La memoria è divisa in PROGRAM e in DATA.

DESCRIZIONE PIN

V_{DD} e V_{SS} è l'alimentazione

OSC_{in} e OSC_{out}: sono i pin a cui è collegato il circuito oscillatore interno, perché funzioni correttamente occorre applicargli un quarzo e due condensatori.

RESET lo attivo basso e viene usato per resettare il microcontrollore.

NMI (interrupt) Durante il funzionamento normale deve rimanere a livello logico 1, applicando a questo piedino un impulso negativo, il programma che sta eseguendo si interrompe e passa ad una subroutine.

PORTA / PORT B permette la comunicazione con l'ext ci sono due canali digitali: I/O due canali analogici, in questo caso 4.

TIMER applicando un livello logico 1 ....

MEMORIA

La MCU opera in 3 spazi di memoria separati:

Lo spazio program
" " dati
" " stack

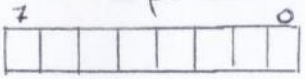
Nello spazio program è contenuto il codice del programma utente in OTP e vettori degli utenti.
Lo spazio dati contiene i dati utente in RAM e in OTP
Lo spazio per stack ospita sei livelli per sottoprogramma e interrompe il servizio di ridirezione di routine.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 164

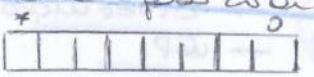
REGISTRI CPU

Al core della CPU dispone di 6 registri e 3 paia di flag e dispositivo del programmatore.

ACCUMULATOR (A): è un registro general purpose a 8 bit utilizzato in tutti i calcoli aritmetici, operazioni logiche, e manipolazioni di dati. L'accumulatore può essere considerato come qualsiasi registro.

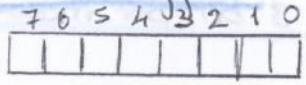


REGISTER X-Y utilizzati in modalità di indirizzamento indiretto come puntatori alle posizioni di memoria nello spazio dato / dato spazio.

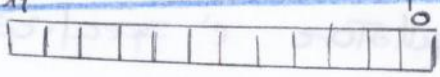


(X e Y)

REGISTER U-W usati in modalità di rapido indirizzamento. Ciò significa che i dati memorizzati in tali registri possono essere raggiunti con una istruzione da un byte



PC è un registro a 12 bit che contiene l'indirizzo della prossima istruzione da eseguire.

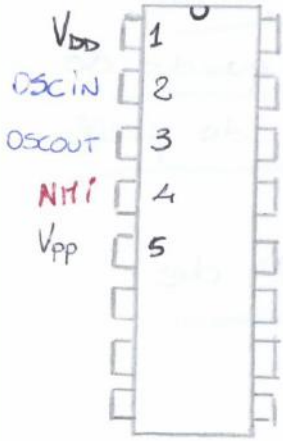


La CPU dello ST6 usa una tecnologia LIFO ossia l'ultimo che metto in pila è il primo che prendo.
Lo stack consiste di 6 posizioni separate.
Quando si verifica una chiamata a subroutine, il contenuto di ogni livello viene separati nel livello successivo verso il basso, mentre il contenuto del PC viene spostato nel primo livello. Quando ritorno da un sottoprogramma il registro di primo livello viene spostato nuovamente nel PC e il valore di ogni livello è spostato con il livello precedente.

Un puntatore è un Tipo di dato, una variabile che contiene l'indirizzo in memoria di un'altra variabile

Continuiamo a parlare del ST 6200 C con 16 pin.

Partiamo dei pin-out presenti nel cap. 21 circuito integrato ha i pin numerati da 1 a 16



2 pin out V_{DD} e V_{SS} sono l'alimentazione

5V o 3V3 — 0V (GRAUND GND)

Alimentazione che viene stabilizzata con l'inserimento di un condensatore.

2 pin 2 e 3 sono "OSC IN" e "OSCOUT" Servono per inserire un osallatore esterno o prendere il clock dell'osallatore interno.

al numero 4 è l'NMI (non mestrable interrupt)

Il pin 5 è il V_{PP} è e pin di proprommabile

→ pin 5 → $V_{PP} = 5V$ è detto NORMAL OPERATION

→ $V_{PP} = 12V$ è quello che è chiamato Eprom Programming mode. Ossia posso riprogrammare il dispositivo.

Normalmente posso riprogrammare il microcontrollore in 3 modalità

1) OTP (one time programming) la programmo una sola volta. → dispositivi industriali prodotti in grandi serie.
2) EPROM erasable programmable ROM
3) EEPROM (E^2PROM) electrical erasable —.

le memorie EPROM sono memorie che possono essere cancellate e riscritte (alcuni si cancellano con i raggi ultravioletti. Per programmarlo sul pin 5 devo dare 12V).

Le memorie E^2PROM ha un più facile utilizzo. Lo posso cancellare e riscrivere elettricamente da USB o con un interfaccia con il PC da cui programmo.

CONVERTITORE ANALOGICO - DIGITALE

Il convertitore A/D è un dispositivo a 8 bit, è del tipo ad approssimazioni successive e ha la possibilità di scrivere su 4 canali multiplexati.

Multiplexati vuol dire che i segnali sono 4 e possono essere serviti solo uno ad un dal convertitore. Il Multplex li mette in coda. Non li esegue assieme perché altrimenti dovrebbe decidere quale processare prima.

Se voglio fare le cose secondo l'ordine giusto e devo convertire più segnali allora devo metterli in sequenza, programmando il multiplexer. Il risultato della conversione è salvato nel registro chiamato "ADC DATA REG". È una porta di 8 bit che contiene il dato convertito (1 byte).

L'altro registro dedicato all'A/D è l'"ADC control reg". Anche in tale registro c'è un'altra porta dove ci sono i settaggi. Possiamo dire che il tempo di conversione per singola conversione è di 7μs con 8MHz di clock.

Pertanto il valore nominale del clock di questo processore è 8M

Un'altra caratteristica importante è che non ha il simple and hold (s/h). Non ha quindi una memoria fisica non ha un buffer che tiene fermo il segnale analogico mentre viene convertito.

Idealmente, si è visto che il campionamento è l'hold di un certo valore analogico in un certo istante. Quindi devo memorizzare un certo istante e ho bisogno di un simple and hold, cioè un dispositivo che prende il campione (sample) e lo tiene (holdere). Ci vorrebbe un circuito che mi colleghi all'ingresso e poi quando ho acquisito il

c'è un multiplexer, con 4 piedini che sono i pin che indicano il canale tra dentro e fuori. La funzione che il mux svolge è quello di decidere chi dei quattro ingressi passa per primo. Poi c'è hardware che fa la conversione. Dopo l'ADC gli 8 bit vanno nel ADC DATA REGISTER. Abbiamo poi nello schema la f_{INT} che viene diviso /12 e troviamo la f_{ADC}.

f_{INT} è il clock interno mentre la periferica viaggia a f più bassa. Il blocco ADC lavora sincrono con una frequenza 12 volte più bassa. Il registro ADCCR comunica con il convertitore digitale e da info all'ADC su come funzionare. Definisce i 3 blocchi ← DDRx, DRx, ORx che comandano il mux.

AD CONTROL REGISTER

ha indirizzo 0D1h e i suoi bit sono così chiamati:

7	6	5	4	3	2	1	0
EAI	EOC	STA	PDS	AD CR3	OSC OFF	AD CR1	AD CR0

2 bit che sono (MSB) dello 0 al 3 (LSB) non vengono usati. Il bit 2 riguarda l'oscillatore

→ EAI (Enable Analog Interrupt): capacità di mandare un interrupt

→ EOC (End off conversion) (READ ONLY) questo pin lo leggo ma non posso scrivere → mi dice quando sono finiti i 70μs

→ STA START OFF CONVERSION (WRITE) quale dell'EOC scrivo quando vuole di iniziare la conversione.

→ PDS: POWER DOWN SELECTION Che se non uso la periferica la posso spegnere così non consumo.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 23 di 164

Il primo e il secondo vengono [illegible] I/O e quindi sono pestiti dal multiplexer.
Il 3° viene dal Timer
Il 4° " dall'ADC-EOC. Viene usato quando finito la conversione voglio che venga eseguito una parte di codice. Il convertitore fa la conversione con le approssimazioni successive.
Vuol dire che il segnale analogico bufferato viene confrontato per approssimazioni successive con dei livelli di Tensione (multipli e sottomultipli del range della Tensione di ingresso)

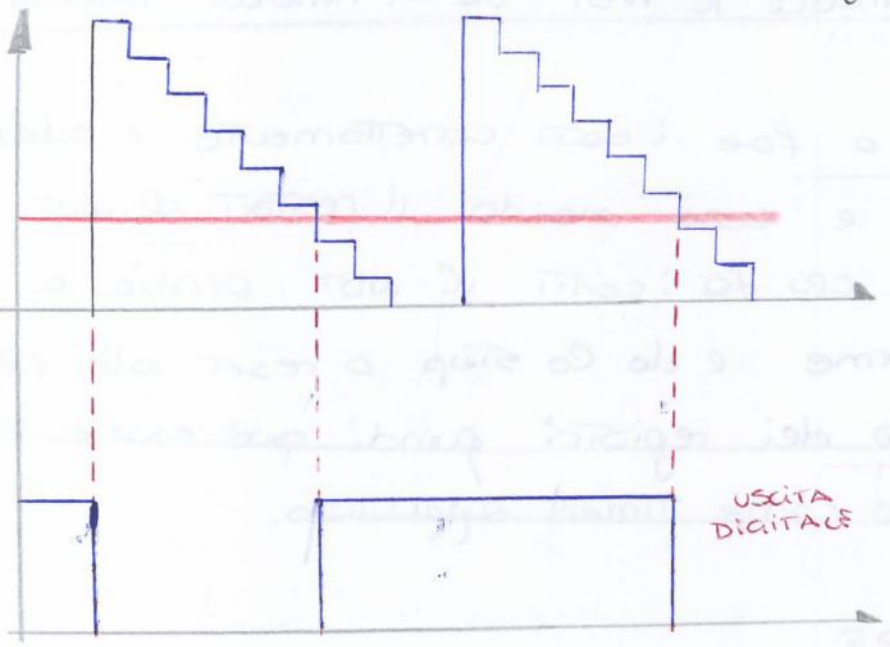
Dobbiamo immaginarci che il segnale che va in ingresso è un segnale analogico che può andare da 0 a 5VOLT.
Se ho valori a 5VOLT → 1
0VOLT → 0 sono i fondo scala.

Nel mezzo cosa succede?
Succede che i bit della parola devono essere scritti uno alla volta a partire dal MSB fino al LSB.
Prima questione dell'approssimazione successiva è quella di stabilire quello che sarà il MSB. Per stabilire se MSB è 0 o 1 devo vedere se il segnale è più grande o più piccolo di ½. Se è più grande di ½ allora sarà 1 se è piccolo sarà 0.

TIMER

Esso non è altro che un contatore. Esso serve a contare il tempo che passa e per aspettare il tempo necessario affinché siano state eseguite certe operazioni.
Un'altra utilità del contatore è quella per la costruzione di una base tempi per fare qualcosa per l'esterno (es. PWM)

modulare in uscita, come info, è un Timer PWM eseguito in digitale.
Faccio tutto questo settando qual'è l'uscita da voglio comandare e comandandolo con lo STATUS register del Timer. Invece con il control register del Timer stabilisco la base Tempi. Nello status leggero quando è arrivato a fine conteggio e quando è andato sopra e sotto un certo livello stabilito e scritto da quale altra parte.



Cos'è il WATCHDOG TIMER (WDT)

Evita i malfunzionamenti. Esso evita che se succede un malfunzionamento nell'esecuzione del programma, questo porti alla scrittura di risultati sbagliati in uscita.
Quindi è un'altro Timer (baseTempo) che è stabilito quanto deve durare e quando si arriva a zero blocca l'esecuzione del programma. Se TUTTO va bene il Timer del WDT non arriva mai a zero. Ma se entrasse un disturbo o non ho programato bene e quindi nei 100 µs non ottengo i risultati, ho il WDT che ha contato 99 µs e quando arrivo a zero riesce da bottar

e tutto per controllare in real-time.

E' però un microcontrollore specializzato per elaborare dei segni come audio, Telefonia, immagini ecc...

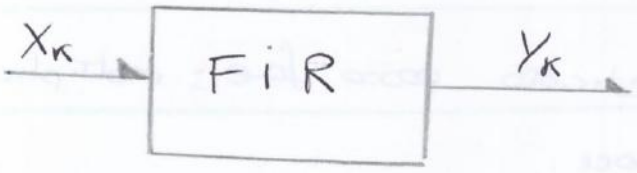
E' un dispositivo fatto per elaborare in modo veloce i segnali. Ha un architettura molto semplice e specializzata e le sue caratteristiche distintive rispetto al microcontrollore sono quelle di eseguire velocemente dei filtri digitali (FIR e IIR). Processare vuol dire prendere un segnale in ingresso manipolarlo e poi restituirlo manipolato come output

Allora FIR e IIR sono due Tipi di filtri digitali possibili. FIR sta per Finite impulse response e IIR invece sta per infinite impulse response.

Un fir vuol dire sostanzialmente che:

$$y_k = \sum_{i=0}^{n} (x_{k-1}\, a_i)$$

somma pesata di campioni



K = istante di campionamento

a_i = peso del filtro

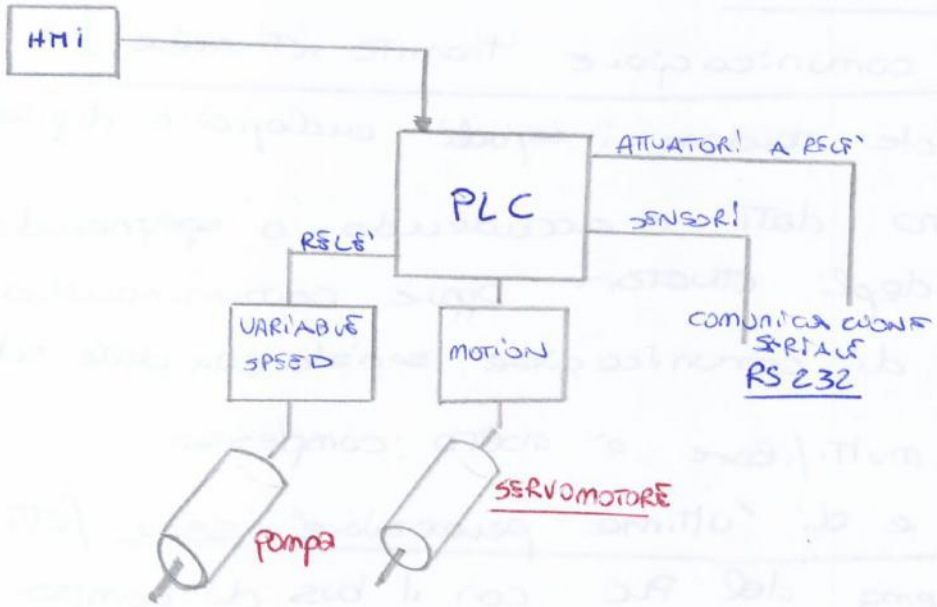
Il risultato è una somma pesata di n+1 campioni di ingresso intesi come campione attuale e campioni precedenti. Ci serve un Hard-ware capace di fare moltiplicazioni e somme. Il DSP è specializzato nel fare velocemente MAC = multiply and accumulate. Quindi ci va un componente che fa le moltiplicazioni.

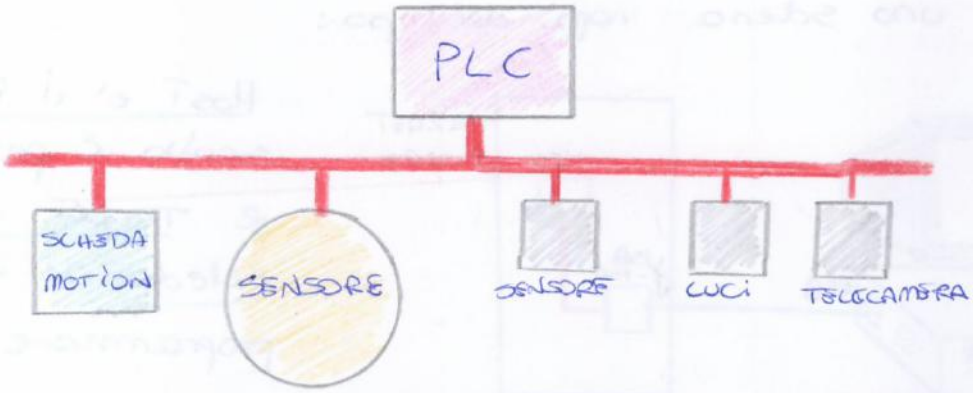
Il primo componente che metto è un ALU con il MAC dentro. Il secondo è un buffer circolare con una serie di registri dedicati alla ALU (non usati x altro)

Sia FIR che IIR servono per fare delle moltiplicazioni e delle somme. Sostanzialmente servono dei buffer circolari, uno dove c'è memorizzato l'ingresso e uno dove c'è l'uscita e poi basta farli girare.

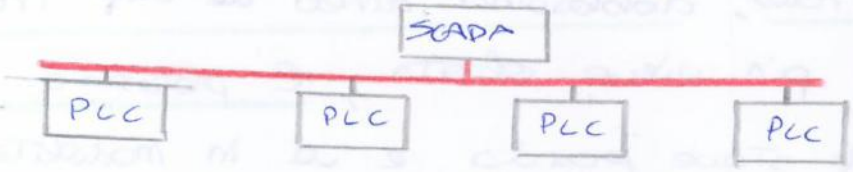
PLC (programmable logic controller)

Il plc dal nostro punto di vista è l'oggetto tipico dell'automazione industriale. È un controllore logico costituito da un processore specializzato che si chiama Asic, dalla memoria e da molti I/O logici e analogici. Ci sono schede varie di tipo motion, attuatori o sensori di qui fuori. Infine c'è anche una HMI (human machine interface). Negli azionamenti normalmente ci sono dei supervisori che dicono, attraverso la motion, allo azionamento di posizionare un pezzo per una lavorazione. Quel supervisore è il PLC. Uno schema tipico è il seguente





È un doppino come quello delle CAN. In questo modo gli attuatori e i sensori diventano delle periferiche esterne. Il pacchetto dati viene mandato avanti e indietro senza distinguere tra input e output.
Uno schema così, se è di piccole dimensioni, può gestire da solo un processo.
In passato gestivo i vari PLC dedicati con un computer detto SCADA (Supervisory Control and data acquisition



Lo scada doveva controllare che non c'erano inconvenienti tra le varie emissioni, memorizzare i dati con lo scopo di monitorare i guasti e mettere in sicurezza.
La criticità dello schema sono le linee di comunicazione seriale.
Il linguaggio di programmazione del PLC è grafico, chiamato LADDER. Esso deriva dalla logica cablata.

PROGRAMMAZIONE HOST-TARGET

Questa programmazione è effettivamente il modo in cui vedo a programmare un controllore embedded
Bisogna caricare il programma dentro la memoria del microcontrollore. Come lo carico? con il PC via

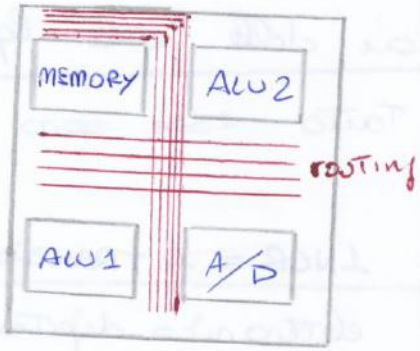
si comporta. Posso anche vedere se l'ingresso e le uscite si comportano come devono. Si può emulare anche il chip sull'host e non sul chip. Emulazione infatti significa eseguire l'algoritmo prima ancora di implementarlo.

ASIC e FPGA (dispositivi proprammabile)

ASIC è un acronimo che sta per "application specific integrated circuit". E'un chip fatto per uno specifico scopo e utilizzo. Inizialmente la divisione di memoria degli ASIC era:

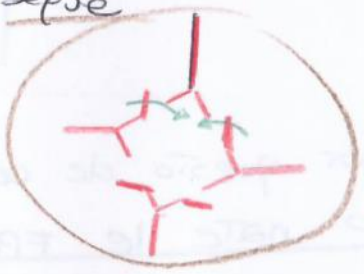
1) full custom ASIC
2) semi custom ASIC

Il primo è totalmente fatto su misura. Si prende un chip di silicio, si sceglie quanto è grande, poi si mettono le celle logiche dopote che servono per il chip stesso. Lo schema è il seguente:



Queste sono delle logic cells custom che inserisce nel chip e le inserisce nel silicio. Poi si dropano le maschere che fanno i collegamenti fra le celle. Normalmente c'è quello che è chiamato ROUTING. Il routing è l'insieme delle tracce metalliche fatte su multistrato per portare in giro i segnali. A secondo di dì deve essere collegato poi ci sono collegamenti Te d'uno e l'altro porte.
Nel caso del full custom si ha sia le logic cells che il routing di tipo custom fatti su misura. Quindi viene

Il routing è fatto dei Transistor. Come segue



Qui incrocio ne 4 Transistor.

Allora a seconda di come chiudo i Transistor ottengo la mia configurazione. Quindi la configurazione dell'hardware avviene attraverso la programmazione di chiusura e apertura di certi collegamenti di routing sia Tra blocchi di GA che all'interno del gate array.

Si capisce che da un lato l'ASIC ha le cose da solo e faccio quello che voglio. Questo è un vantaggio soprattutto in fase di inizio dello sviluppo di una scheda quando magari non so ancora qual'è il microcontrollore che servirà.

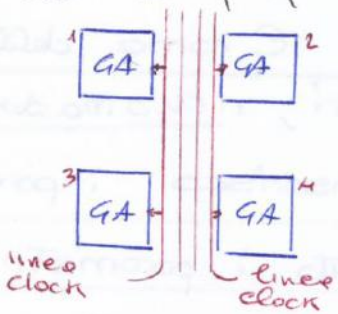
Le FPGA hanno il difetto di essere molto ridondanti e quindi diventano molto costose e per questo non vengono usate in ambito industriale. Inoltre ha un altro difetto ovvero che le FPGA è volatile, cioè quando spengo, tutto si apre e la FPGA dimentica come era programmata.

Praticamente il firmware risiede su una memoria non volatile di Tipo flash esterna (o interna) e viene configurato all'accensione. Ciò porta problemi:

- Il primo che FPGA si programma
- Secondo se la memoria dove risiede il programma e le configurazioni stanno fuori dal chip. Le questioni di segretezza industriale sono a rischio.

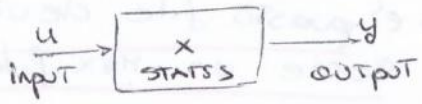
Inoltre se la flash è esterna quando all'on parte il programma c'è una serie di comunicazioni Tra la flash e la FPGA che devo criptare per evitare che a

a centinaia di Megahertz possono essere critiche. Quello che si chiama la sincronizzazione e la sequenzialità delle istruzioni devono essere garantite e testate facendo un layout fisico fatto bene. La prima cosa di cui si deve garantire la contemporaneità è il clock. Infatti c'è una linea dedicata che deve portare il clock a tutto il chip. Però ci sono dei ritardi da un gate all'altro, quindi il clock è lo stesso ma fino ad un certo punto.
Se il clock tra il primo e il numero 4 non è sincrono, perché il chip è stato fatto male, si dice che c'è lo SKEWING. Vuol dire che tra il primo e il secondo ci sono dei ritardi e così via. Magari è solo il primo ad essere sincrono con il clock.
Se c'è lo skewing limita le potenzialità hardware che le FPGA hanno. Le FPGA quindi è molto general purpose ma ha costi più alti e si deve stare attenti a come la si usa e la si programma.



Altra cosa critica è che la FPGA non ha A/D converter, quindi sono solo logiche. Se devo fare il controllo di un motore con FPGA ci devo poi mettere l'A/D esterno. Ci sono inoltre delle FPGA ibride con un A/D dentro e poi una parte programmabile.
Ci sono anche delle librerie nelle FPGA con cui uno si può costruire un microcontrollore dentro la FPGA stessa.

Un blocco Simulink consiste di una serie di ingressi, una serie di stati, una serie di uscite, dove le uscite sono funzione del tempo di simulazione, degli ingressi e degli stati.



Di seguito le relazioni matematiche:

$$\begin{cases} y = f_o(t, x, u) \\ \dot{x}_c = f_d(t, x, u) \quad \text{(variabili di stato continue)} \\ x_{d_{n+1}} = f_u(t, x_c, x_d, u) \quad \text{(" " " discrete)} \end{cases}$$

L'esecuzione di un modello Simulink procede a tappe

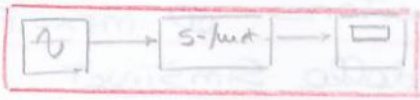
* FASE INIZIALIZZAZIONE. In questa fase Simulink incorpora blocchi dalle biblioteca nel modello.
* Dopo si PASSA ad una simulazione ed quello dove ogni passaggio attraverso il ciclo viene definito come un passo di simulazione.

Si esegue il loop fin che non si raggiunge il risultato.

Una S-function comprende un set callback che svolgono compiti necessari in qui fase di simulazione

Il MEX-file definisce un blocco S-function dove definisco informazioni sul modello Simulink

Un esempio di S-function:




```
# define S_FUNCTION_NAME your_sfunction_name_here
# define S_FUNCTION_LEVEL 2
# include "simstruc.h"

static void mdlInitializeSizes (SimStruct *S)
{
}
....
< additional S-function routines/code >
```

```
if (!ssSetNumInputPorts(S,1)) return;
  ssSetInputPortWidth(S,0, DYNAMICALLY_SIZED);
  ssSetInputPortDirectFeedThrough(S,0,1);
if(!ssSetNumOutputPorts(S,1) return;
  ssSetOutputPortWidth(S,0, DYNAMICALLY_SIZED);
 ssSetNumSampleTimes(S,1);
 ssSetOption(S, SS_OPTION_EXCEPTION_FREE_CODE);
 }
```

Simulink chiama "mdlInitializeSizes" per avere informazioni a riguardo del numero di Input e porte di uscita, dimensioni delle porte, il numero di Tempi di campionamento e tutte le altre informazioni necessarie per la S-function.

Per ottenere la S-function avrà le seguenti informazioni:

* Zero parametri: il campo dei parametri nella definizione della S-function sarà vuota.
* Una porta input e una output
* Lughezze delle porte dimensionate dinamicamente
* Un tempo di campionamento

Il metodo di callback "mdlInitializeSampleTimes" specifica il valore effettivo del tempo del campione. Definendo "eccezione del codice libero" occorre eseguire nella S-function.

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
  ssSetSampleTime(S,0, INHERITED_SAMPLE_TIME);
  ssSetOffsetTime(S,0,0.0);
}
```

Simulink chiama "mdlInitializeSampleTimes" per impostare i Tempi di campionamento del S-function che può essere uno solo (campione 0) come nell'esempio o multiplo (caso di blocco multirate)

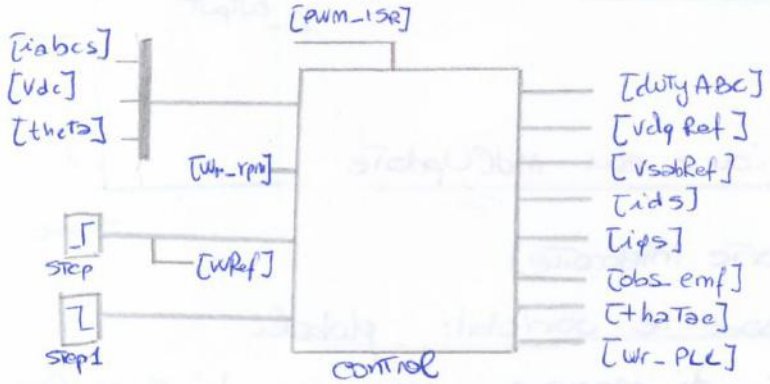
INHERITED_SAMPLE_TIME specifica che è il modello esterno che

nextTimeTWS.c

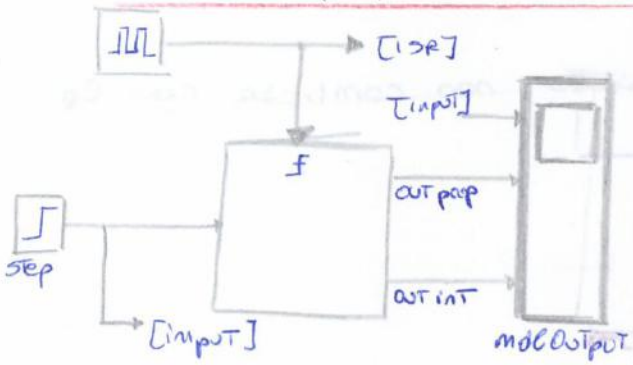
nello stile di comando.

La Sfunction per il controllo digitale è inclusa in un file TRIGGER SUBSYSTEM.

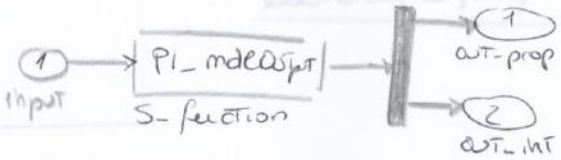
Il Trigger è chiamato PWM INTERRUPT SERVICE ROUTINE e corrisponde allo PWM Base Templ. La Sfunction viene chiamata dal risolutore Simulink ad ogni interrupt.



Il blocco proporzionale integrale:



La sottoscheda del Trigger



Un passo unitario viene applicato come input.

Tre diverse implementazioni vengono confrontate e discusse e una corretta viene scelta e viene usata per l'implementazione del controllo digitale, si sincronizzano le uscite sulle base temp:
* mdlOutput
* mdOutput + mdlUpdate
* mdlOutput + un ritardo del campione

Tempo di sincronizzazione con mdlOutput

Regole:
* No stati discreti
* la porta integrativa deve essere dichiarata come un GLOBAL.

Per controllo motore introduco un z^{-1} come ritardo.

```
#include "include\User_data_types.h"
#include "include\Costants.h"      → parametri costanti
#include "include\Variables.h"     → variabili
#include "include\MotorDefs.h"     → parametri costanti
#include "include\Usr_Macros.h"
#include "include\MotorCntrl.h"    → funzioni di controllo
```

Altre definizioni di pre-elaborazione

```
#define NINPUTS   9
#define NOUTPUTS  17
#define ACCEL     (mxGetPr(ssGetSFcnParam(S,0))[0])
#define NPARAMS   1
```

* dichiarare il numero di ingressi, uscite e parametri
* dichiarare il nome di parametri

mdlOutputs

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
  double      *y = ssGetOutputPortRealSignal(S,0);
  InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
  // copy parameters into variables
     accel = ACCEL * rpm2rad * TS;
  // NEVER ADJUST SCALE FACTOR IN DSP IMPLEMENTATION!!!
```

dichiarazione obbligatorie:
* vettore delle uscite (y), vettori di ingresso

copia parametri:
* l'accelerazione è definita in variables.h. ACCEL (il parametro) non può essere usato all'interno di una funzione durante Accel

Quindi in stazionario vado a lavorare ad una velocità che dipende dalla tensione. Se invece aumento la coppia di carico, un po la velocità si siede, mentre se la diminuisco mi avvicino alla velocità a vuoto.

E' la stessa cosa che succede con un motore sincrono alimentato in tensione, anche se lì c'è un elettrolito in più che è la frequenza.

Quindi a regime e a carico si ha

$$\omega = \frac{V_{arm} - R_a I_a}{K_E} = \frac{V_{arm} - R_a \frac{T}{K_T}}{K_E}$$

La differenza tra K_T e K_E è il fatto che K_T è più piccolo di K_E perché la coppia utile risente degli effetti delle perdite

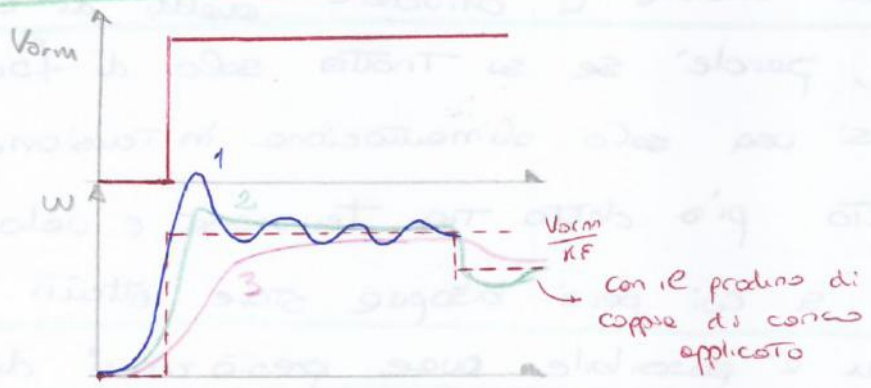
Si può dire che

$$K_E = k\phi_0$$

$$K_T \lesssim K_E$$

sono uguali quando non ci sono perdite

Se sono in transitorio e alimento in tensione, il motore risponde con un transitorio di velocità.



Se diamo un gradino di tensione ad un certo istante, la velocità del motore, arriverà lì solo dopo aver estinto il transitorio che può essere di 3 tipi.

Da cosa dipende il transitorio di ω del motore?

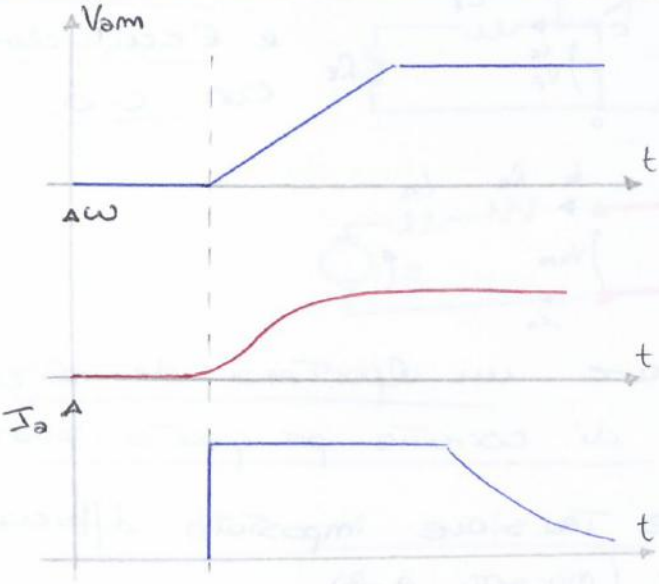
Dipende dalle costanti di tempo del motore. Ovvero da:

1) $\tilde{\tau}_a = \frac{L_a}{R_a}$ COST. Tempo elettrica

Le corrente all'istante iniziale può arrivare a valere anche all'incirca $\frac{Ve}{Ra}$, perché una volta estinto il Transitorio elettrico, se la velocità non si è ancora messa o si è messa poco, tutta la Tensione cade sulla resistenza. Può avere diversi andamenti a seconda delle costanti di Tempo ma comunque non è controllata da nessuno. Così si fa per mettere Tranquille la corrente?

Basta non dare gradini di Tensione. Allora si può pensare di fornire alla macchina della Tensione con andamento a rampa.

Se applico la rampa avrò la seguente situazione:



Se do la pista pendenza alla rampa, la velocità seguirà l'andamento rappresentato in maniera opportuna. Lo stesso farà la corrente.

Se la rampa è Troppo ripida ci si avvicina al gradino, se è rasoia si ha una corrente e la velocità cresce molto lentamente. Questo era per dire che ci concentriamo su un controllo semplice focalizzando la nostra attenzione sulle modalità di creazione di una Tensione di un convertitore DC/DC che vada dare in uscita una Vam.

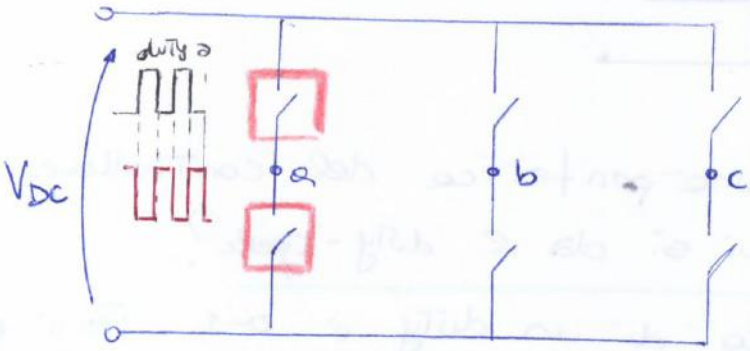
Le uscite output che vogliamo ottenere sono:

1) duty a;
2) duty b;
3) duty c;

Essi servono per comandare le 3 gambe. Sono i comandi compresi tra 0-1 che comanderemo gli IGBT alti delle 3 gambe.

Gli IGBT bassi verranno comandati in maniera complementare quindi il loro duty sarà (1 - duty delle parte alte)

Il nostro obiettivo è quello di dare duty non di creare le rettangole di comando degli switch, perché abbiamo un hardware e un modulatore che dato il duty di una gamba di inverter realizza i comandi.



Se do un duty di gamba a, vorrà dire che la parte alta verrà comandata da una rettangola che avrà un valor medio che sarà il proprio il duty a.

Lo switch basso verrà comandato con un comando rettangolare complementare con valor medio 1 - duty a

Ci sarà un modulatore che farà questo mestiere.

Quindi dentro il DSP c'è una periferica che si chiama PWM- UNIT. Questa periferica ha dentro un Timer up-down che è la Triangola, riceve un riferimento tramite la scrittura di un registro che si chiamerà duty-cycle

Preso questo livello, produrre su due pin d'uscita due rettangole di comando.

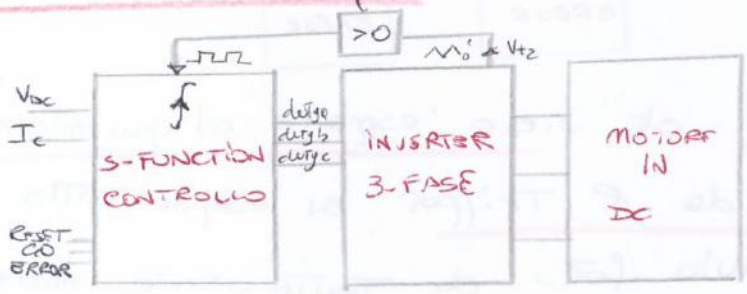
Quindi ogni gamba ha un solo comando ed è

interna nelle quali se ne informazioni sulla tripla. Il fondo scala della triangola dipende dalla risoluzione che voglio ottenere, ovvero da quanti gradini voglio. Può dipendere anche dal clock

Come è organizzato il codice?

Il codice sta dentro una S-function chiamata DC-MOTOR_ctrl_va.c

Il modello è il seguente



La S-function è triggerata dall'interrupt della pwm. Infatti quest'ultimo viene dal convertitore. L'inverter 3-fase, non è fatto con dei IGBT, ma con blocchi logici. Ha la proprietà di far uscire la Vtr e di produrre l'input.

La Vtr esce dall'inverter viene squadrata e produce le elaborazioni dell'algoritmo di controllo.

Visto che ho 0-1 metto un blocco che le squadra imponendo >0. 2 fronti di salita che formano portare il calcolo deve essere ogni istanti di vertice della triangola

Vediamo come è fatto la DC_MOTOR_ctrl_Va.c che fa il controllo del motore in Tensione.

Partiamo dal blocco di programmazione MdlOutputs. Vi è immediato Tutto un blocco di comunicazione che deve escludere cosa dicono i comandi fp, reset, error. Poi ci sono le misure.

* STATE = STOP
* ASPETTA IL pulsante go

Se entro nello STATO wake up non esco se non ho fatto 564 cicli. Quindi c'è un contatore che conta da conta do quando entro fino a 563 e a quel punto mi dice che ho finito. Mi fermo ed attendo nuove istruzioni ossia il GO.
Scrivono il codice di programmazione:

```
switch (STATE)
{
 case ERROR:
    pwm_stop = 1.0;
    go = 0;        → quello eventuali azioni da pulsante esterno
    counter = 0;
    Err = 0;
    if(restart >0,5);   ONE_HALF
    STATE = wake_up;   wake_up è un numero, quindi se STATE = quel numero allora calcolano:

    break;
 case wake_up;
   // if (counter < 1.0)
    {
      restart = 0.0;
      ResetVarCtrl();  → mette a zero TUTTE le variabili prima di partire
    }
```

Quando entro per la prima volta nel WAKE-UP me ne accorgo perché counter vale zero. Se vale zero allora annullo il pulsante di restart ed espleto il "reset_varcontrol()".
Questa è una function senza argomento ovvero è un richiamo ad una parte di codice scritta da qualche parte. Mette a zero tutte le variabili che Teyono memoria tipo le interprotive.

```
// offset measurement
  if((counter >= 1.0) && (counter < 64))
  {
```

```
// Enable PWM and modulate 5%
duty_abc.a = 0,05;
duty_abc.b = 0,05;
duty_abc.c = 0,05;
PWM_STOP = 0.0;
}
```

} definite in certe parti del codice

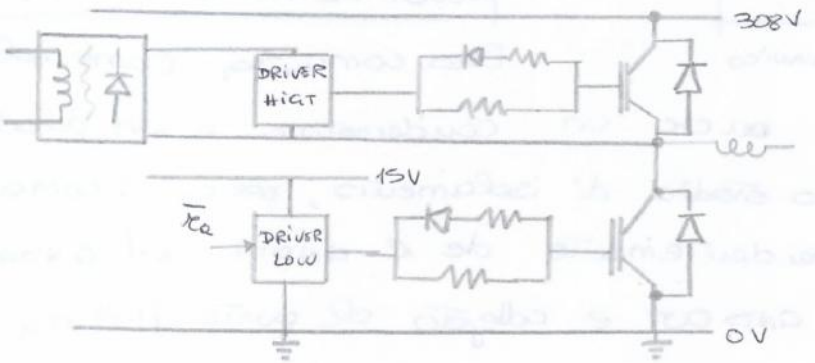
L'ultimo comando è quello de dalita la PWM. Da questo scatto si evince de le fasi vegono modulate dal ciclo successivo al 5%. Quindi 5% ON e il restante 95% OFF. Dopo di de si ha lo "switch To State STOP"

```
// SWITCH TO STATE STOP
if (counter > 563)
{
counter=0;
STATE= STOP;
}
counter++;
break;
```

Questo è la fine dello STATO WAKE-UP. Fino a counter 0-63 ha misurato. Per counter=64 ha calcolato gli offset e ha abilitato la modulazione. Poi ha contato fino a 564 e poi esce e torna su entrando poi in STOP.

A cosa serve modulare per 500 cicli e 5%?

Serve a caricare i condensatori di bootstrap. Essi sono i condensatori dei driver degli IGBT lato alto.



funziona quando c'è tensione su $V_{S,AT}$, ovvero quando è stato caricato quel condensatore a 15 Volt. Lo stadio di ingresso invece è sempre alimentato. Il diodo è chiamato di Boot Strap perché è il diodo che porta i 15 volt sui condensatori caricandoli. Lo carica quando il p.to basso del condensatore va al di sotto dei 15 volt.

Quando l'interruttore fosse chiuso e l'IGBT di sopra pure, e il punto fosse a 308V, allora il diodo blocca la carica. Ma se è chiuso quello di sotto e il p.to va zero volt, il condensatore può essere caricato, perché il meno del condensatore è a zero attraverso l'IGBT di potenza, e l'alimentazione a 15V fa scorrere una corrente fino a caricare completamente il condensatore. Il concetto di Bootstrap è di alimentare lo stadio di uscita a stratoni, quando l'IGBT basso è chiuso. Se quest'ultimo non si chiude mai o la pompa non si chiude non riuscirò mai a caricare il condensatore.

Il condensatore è un polmone di carica da usare per alimentare lo stadio di uscita e deve essere abbastanza grande da mantenere i 15 Volt entro le soglie richieste, almeno per tutta la durata del comando di gate dell'interruttore alto. Poi quando viene chiuso quello basso, essa torna più e viene ricaricato. Vedi come riduzione del circuito il circuito di potenza.

Questo, come detto, è una soluzione low-cost, perché non necessita di alimentazioni flottanti, ma necessita da qui tanto più se garantito il passaggio dello stato di duty o di gamba.

Se sull'eccitatrice do massima tensione per un secondo siccuro da dopo un po' si spegne.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 164

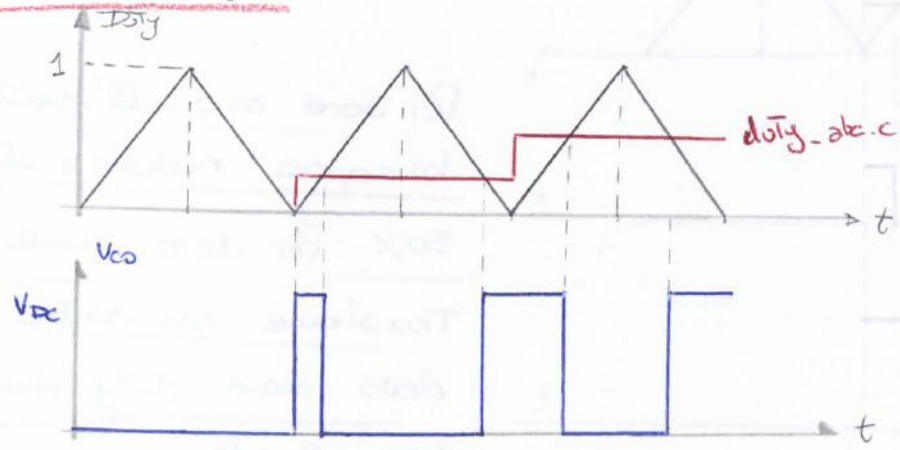
```
// PI_reg.
if_error = if_ref - iabcs.c;
   :
Vc = Vc + integral_f;   è la Tensione che mi serve per calcolare il duty delle pombe C.   // V*fref o Vc* in volt

// Armature control
Va = Vdc * 0,5 + Va_ref * 0,5;
Vb = Vdc * 0,5 - Vb_ref * 0,5;

// Duty
duty_abc.a = Va * Vdc_inv;
duty_abc.b = Vb * Vdc_inv;
duty_abc.c = Vc * Vdc_inv;
```

Ritorniamo un attimo alla schematizzazione a blocchi per capire cosa abbiamo fatto per calcolare duty_abc.c

Così facendo ho realizzato una modulazione unipolare e la scala della Tensione d'uscita va da 0 a Vdc ovvero tra "0 - Vdc" Quello che verrà realizzato in uscita sarà Vco in base al duty c.



Ad ogni interrupt verrà scritto un certo valore di duty e il comando eseguito in uscita fa sì che la Vco ha il seguente aspetto

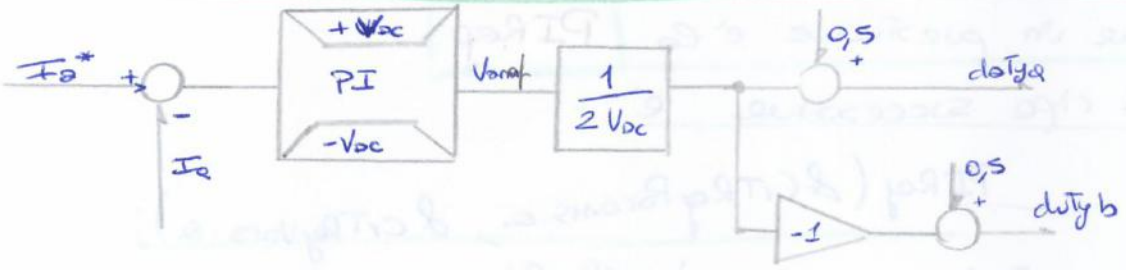
Ho realizzato un modulatore (convertitore) DC-DC unipolare.

Per quanto riguarda il controllo d'armatura ho uno schema diverso.

Ecco lo schema del controllo di armatura:

CONTROLLO DI CORRENTE DI ARMATURA - MOTORE IN DC

Da un punto di vista delle schematizzazioni a blocchi il controllo di corrente di armatura sarà:



A differenza del PI di eccitazione qui ci sono due limiti simmetrici. Questo PI produce una Tensione di riferimento alla luce dell'errore di corrente di ingresso. La Varm la noi l'abbiamo già vista nelle scorse lezioni. Le regole per tarare i Tre parametri del regolatore PI son le stesse. Per V_{MAX} si può impostare $\pm V_{dc}$ dipende dal convertitore. Per K_P e K_I si usano le seguenti formule per trovare valori di riferimento poi si affina il valore per tentativi. Si scrive:

$$\omega_b = \frac{K_P}{L_a} \Rightarrow K_P = \omega_b L_a$$

e

$$\frac{K_I}{K_P} = \omega_z < \omega_b = \frac{K_P}{L_a} \Rightarrow K_I < \frac{K_P^2}{L_a}$$

Dentro la S-function Troveremo allora

$$ia\_ref = U(6)$$

Vuol dire che sto dando il riferimento di corrente dall'esterno. Troveremo poi un poio di strutture che servono per entrare nelle function che fa il regolatore PI

```
CtrRegVars_a.Vmax = Vdc
CtrRegVars_a.ref = ia_ref
CtrRegVars_a.actual = iabcs_a
```

Sono solo delle assegnazioni, cioè vuol dire che sto impostando i dati che mi servono dentro ad una struttura di dati che si chiama CtrRegVars

troveremo tutte variabili definite dall'utente e in particolare troveremo le definizione delle due strutture di cui parlavamo prima. Allora troveremo `ctrRegVars_a` definito come:

```
// PI Regulator Variables structure
typedef struct {
    double ref;
    double out;
    double error;
    double actual;
    // integral;
    // Lim;
    // INT-Lim; } 
    XPI Reg Vars;
```

comando che definisce un tipo di variabile

queste sono tutte le variabili che servono dentro le PI.

Adesso se voglio definire una di queste variabili vado in

```
include \variables.h
// definisco la variable
double Temp;
XPI Reg Vars   e poi metto il nome che voglio
ctrRegVars_a;
(ctrRegVars_f;)
```

(tra parentesi se voglio una struttura che rimanda all'eccitazione). La struttura quindi avrà tre step da seguire per creare la funzione:

1) assegnare il Typedef;
2) definire la variabile;
3) assegnare i valori;

L'altra struttura che si chiama "ctrRegParams_a" è definita in modo analogo.

Ho usato le strutture per accorpare tutte variabili insieme per ragione di semplicità di scrittura nel codice.

```
//error computation
(RepVars -> Error) = (RepVars -> ref) - (RepVars -> actual);
```

Questa è la prima operazione che esegue dentro un PI e non è altro che quella di calcolare l'errore.
La freccetta è una sintassi che indica il contenuto di un campo di una struttura a partire dal puntatore a quella struttura. Per capirci: se RepVars è un puntatore ad una certa struttura, per avere il contenuto di quella struttura devo usare * RepVars. Ma poiché la struttura è un insieme di campi, e il contenuto è nei campi, per accedere al campo della struttura si può usare:

```
(* RepVars). campo      (caso peggiore)
```

oppure

```
RepVars -> campo
```

Dopo aver calcolato l'errore scrivo:

```
RepVars -> prop = (RepParams -> Kp) * (RepVars -> error);   (per)
if (RepVars -> prop > RepVars -> lim)
RepVars -> int_lim = RepVars -> lim - fabs(RepVars -> error)
RepVars -> integral += RgParams -> KI * RepVars -> error;
RepVars -> OUT = RepVars -> prop + RepVars -> integral; }
```

Questo significa che siamo andati proprio a modificare i contenuti di "Get RepVars_e" e poi usciamo.
Quello che una function non può fare è modificare il proprio argomento.
Ritorniamo all'uso che facciamo di questo PIReg.
Noi lo usiamo per il controllo. Noi controlliamo sia la I_p che la I_a d'armatura.
Nel nostro codice troveremo, quando arriveremo allo stato di START la seguente sintassi:

...

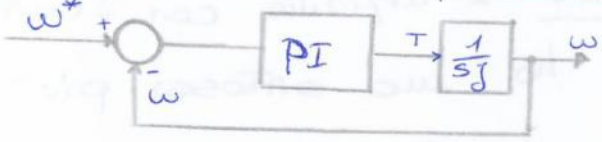
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 164

correttamente e poi accendiamo tutto nello stesso momento non funziona niente e non possono succedere dei casini. Le cose che può succedere, se porta il controllo mentre la macchina si sta flussando, è che porto un K_T variabile e questo un eccitazione bassa non ho coppia.

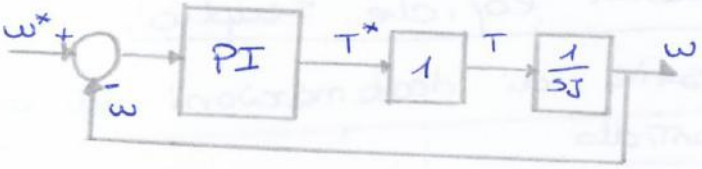
Quindi niente di prove. Però vedremo che nei motori asincroni con il field oriented se l'osservatore non ha avuto tempo di convergere al valore osservato, non si sa dove sono gli assi d e q e il controllo non funziona e scattano le protezioni.

- CONTROLLO VELOCITÀ

Il controllo di velocità presenta il seguente schema:



(schema semplificato)



1 controllo di macchina ideale

Si è trascurato tutto tranne l'inerzia dal punto di vista del carico. Da qui possiamo scrivere le eqn per la taratura del PI ovvero

$$\omega_b = \frac{K_p}{J} \Rightarrow K_p = \omega_b \cdot J \qquad \frac{K_I}{K_p} < \frac{K_p}{J} \Rightarrow K_I < \frac{K_p^2}{J}$$

Questa è quindi la teoria. Nella pratica (implementata) poi emergono alcune problematiche.

Il codice:

```
// speed control
omega_ref.in = U(6)*rpm2rad;
ramp(&omega_ref_ramp, accel, omega_ref_in);
```

(omega_ref_ramp: uscita di post; accel: accelerazione della rampa; omega_ref_in: variabile di ingresso)

Il risultato sarà contenuto in "omega_ref_ramp", di cui do l'indirizzo e quindi so che dentro la funzione ramp ci sta una dichiarazione di un puntatore.

Cerchiamo di capire una questione legata alle scelte prese nel codice più scritto prima; Parliamo nel riferimento di coppia. C'è questa scelta in più perché il feedback di coppia non è in realtà una coppia, ma una corrente di armatura. La scelta è data dal coefficiente di scelta che è K_T.

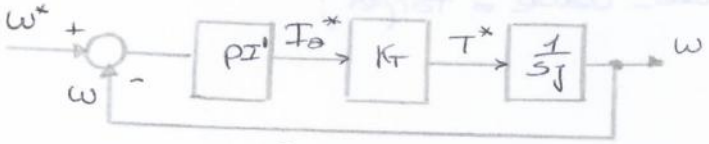
Il K_T di questa macchina, come visto in laboratorio è

$\varphi_S \geq 5 \frac{N_M}{A}$ a flux nominale ($I_f = 0{,}4A$)

Questo è il motivo per cui T_{ref} è moltiplicato per il valore due.

$$i_{d\_ref} = T_{ref} * \left(\frac{1}{K_T}\right) \quad \simeq 2$$

In questo caso posso anche evitare questo, facendo un'anello di velocità che esce direttamente con i riferimenti di corrente. Per fare questo devo evitare di porre il guadagno del controllo di macchina unitario, ma basta di poco quale K_T. Lo schema sarà il seguente:



Quindi il controllo di macchina, per avere le stesse prestazioni di prima, deve essere ritarato, avere ricalcolare i guadagni che non saranno più gli stessi.

Essi possono essere calcolati come:

$$\omega_b = K_p' \frac{K_T}{J} \Rightarrow K_p' = \frac{\omega_b J}{K_T} \quad \text{e} \quad \frac{K_I'}{K_p'} < K_p' \frac{K_T}{J} \Rightarrow K_I' < \frac{K_p'^2}{J} K_T$$

Vediamo ora come programmare la funzione (ramp) che si trova in `include\MotorControl.h`

Ramp è definita come:

```
void ramp ( double * actual_value, double delta, double target,
```

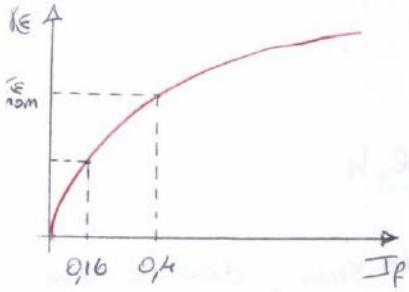
Con queste sintassi abbiamo filtrato con un filtro di 1° ordine queste due preudesse. Ovviamente la preudesse filtrata avrà un nome diverso da quella non filtrata.
Quindi la preudesse filtrata della function Filter è omega_m_filt. FILTER è un filtro in cui risultato dipende sia da campioni dell'ingresso da da campioni dell'uscita. Filter ha come ingressi omega_m (non filtrata) omega_m_filt (filtrata) e KFILT=25 da sta lì per ricordare che è il numero da fa venir fuori un filtro a 25 Hz.
Entrambe le istruzioni dove c'è Filter, mi indicano da le preudesse vengono filtrate entrambe a 25Hz.
Siamo in un motore in DC da ha a disposizione una dinamo tachimetrica e quindi abbiamo in ingresso una velocità. Quando, invece, andremo a lavorare in un motore in AC, avremo bisogno della posizione, quindi avremo l'encoder e non una velocità. Impareremo a calcolare la velocità per derivazione e poi la filtreremo.
La funzione filter si trova sempre in

include\MotorControl.h ed è così definita:

```
double Filter(double xK, double xf_k, double kfilt);
{
  xf_kt = kfilt (xk - xf_k);
  return xf_k;
}
```

È semplicemente una sommatoria di campioni vecchi dell'uscita e campioni vecchi più attuali dell'ingresso.
Il coefficiente Kfilt varia tra 0 e 1 ed è un peso da caratterizza la frequenza di Taglio del filtro.
Se il prodotto Kfilt fosse (0-) (filtro ottimale), il filtro avrebbe una frequenza di Taglio 0 e non si incrementerebbe

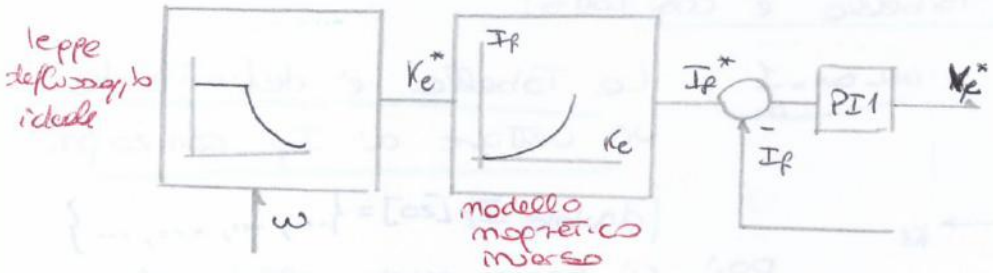
Ma per ridurlo non devo farlo linearmente, perché come sappiamo la caratteristica di eccitazione è non lineare



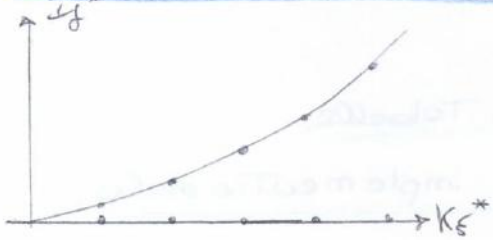
Facciamo girare, a vuoto, il motore a una determinata velocità e variando l'eccitazione si calcola K_E.

Infatti misuriamo la velocità e la Tensione di morsetti a vuoto e Troviamo K_E facendo il rapporto tra la Tensione e la velocità. Lo facciamo per diverse velocità e Troviamo la curva.

Se devo entrare nella legge di deflussaggio ideale e devo ridurre il flusso di una certa quantità e devo andare a metà vado a leggermi 0,16 A da Tabella e posso aggiornare il riferimento di corrente di campo. Lo schema di deflussaggio della macchina è il seguente



Avendo come obbiettivo quello di dare un riferimento di corrente di eccitazione al controllo della corrente di campo, il riferimento si trova dalla legge di deflussaggio entrando con la velocità e Trovando il K_E^*. Con questo K_E^* si entra dentro il modello magnetico inverso



Si può dire che per deflussare si può usare anche un altro metodo che non richiede l'uso della Tabella. Si può usare uno schema più semplice che ha vantaggi:

è limitato è impresso però con un piccolo rapporto di morsetto anche se quest'ultimo non è necessario.

Ora vediamo l'interpolazione: // interpolazione

```
ind_x_i = floor ((x-x_raw) * inv_Dx);
Ytab = Ytab + ind_x_i;
Yref = * Ytab;
Yref = (*(Ytab+1) - Yref) * inv_Dx * ((x-x_min) - Dx * ind_x_i) + Yref;
return Yref;
}
```

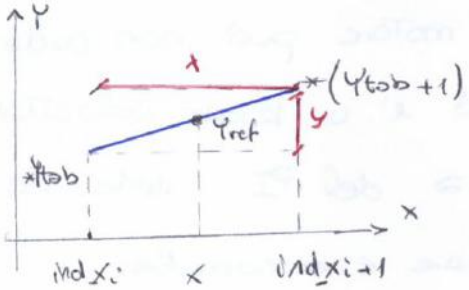
(la vera interpolazione)

per · per · per

La prima cosa da fare è calcolare il numero (indice) di elemento della Tabella che precede il valore di cui devo fare l'individuo anche, quello che segue il punto da trovo. Per meglio dire se mi trovo con un x tra due valori interi devo individuarli per poi fare l'interpolazione.

Poi vado a calcolare l'indirizzo di "ind_x_i" e lo sommo a "Ytab" per dare il vero contenuto dell'indirizzo "Ytab". E poi faccio l'interpolazione vera e propria.

In modo più schematico ho:



Interpolando linearmente tra ind_x_i e ind_x_i+1, o meglio tra *Ytab e *Ytab+1 trovo Yref nel modo così scritto:

$$Y_{ref} = Y_{ref} + \frac{\Delta Y}{\Delta x} = *\left(x - x(ind\_x_i)\right)$$

Dobbiamo trattare per il motore DC la tecnica di deflussaggio in quello chiuso.

DEFLUSSAGGIO DEL MOTORE DC TRAMITE CONTROLLO IN ANELLO CHIUSO DELLA FORZA CONTROELETTROMOTRICE (back-emf) ($f_{sc} = f_{ew}$)

Riportiamo di seguito lo schema di principio di questa tecnica:

Si effettua con le seguenti formule:

$$\tilde{E}_a = V_a^* - \underbrace{\Delta V_{inv}}_{\text{caduta inverter}} - \underbrace{R_a I_a}_{\text{caduta resistiva}} - \underbrace{L_a \frac{dI_a}{dt}}_{\text{caduta induttiva}}$$

Posso scrivere che

$$\Delta V_{inv} = 10\ V$$

Questo valore si potrebbe stimare più accuratamente guardando le caratteristiche degli IGBT, il tempo morto, e le cadute di ON. Il termine $R_a I_a$ contiene la stima della resistenza e quindi dell'azione della temperature. Come sappiamo R_a varia con la temperatura operativa della macchina. La $R_a I_a$ è una parte minoritaria della tensione, perché quando andiamo a deflussare la f.c.m. è grande e si prende quasi tutta la tensione della macchina. Il termine $L_a \frac{dI_a}{dt}$ si calcola come:

$$L_a \frac{dI_a}{dt} \simeq L_a \frac{I_a(k) - I_a(k-1)}{T_s}$$

Bisogna conoscere la L_a e devo memorizzare il campione vecchio di corrente $I_a(k-1)$. Il campo $I_a(k)$ è la misura non devo memorizzarla.

Per essere più preciso non dobbiamo parlare di $\tilde{E}_a$ ma del modulo di $\tilde{E}_a$, perché ad esempio non ci interesse se il motore gira avanti o indietro, ma che la f.e.m. non sia troppo elevata in modulo. Quindi questa tecnica è di facile implementazione, perché i parametri di cui dobbiamo bisogno sono solo due e facilmente calcolabili.

Il regolatore è lento rispetto ai fenomeni elettrici perché E_a si muove lentamente con le dinamiche meccaniche. Sarà solo integrativo e non proporzionale.

Non c'è più il K_s nello schema, ma c'è solo il concetto di tenere la f.e.m. saturata agendo direttamente sul canale

Quella che si usa nella pratica è la sviluppo in serie dell'esponenziale che sarà

$$e^{AT_s} = I + AT_s + \frac{(AT_s)^2}{2} + \dots$$

e alla fine scrivo $A_d = e^{AT_s} \cong I + AT_s$

Quindi si prende solo il termine lineare. È come dire che vicino a zero l'esponenziale è confondibile con la sua Taylor

Se Ad vale tanto Bd diventerà:

$$B_d \cong A^{-1}(I + AT_s - I)B = A^{-1}AT_sB = T_sB$$

Quindi si può scrivere

$$x(k+1) \cong (I + AT_s)\,x(k) + T_sBu(k)$$

Fatta tutta questa premessa rigorosa, si può notare che il risultato trovato si può ottenere anche tramite le differenze finite, confondendo la derivata con il rapporto incrementale

Quindi si avrà che $\dot{x}(t) = Ax(t) + Bu(t)$

ma se $\dot{x}(t) = \dfrac{x(k+1) - x(k)}{T_s}$

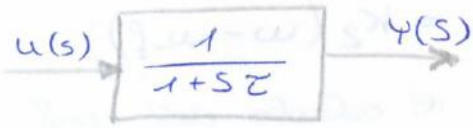
$$x(k+1) = x(k) + T_s(Ax(k) + Bu(k))$$

Scritta meglio

$$x(k+1) = (I + AT_s)\,x(k) + BT_s\,u(k)$$

Calcolo della discretizzazione della f.d.t. del filtro del 1° ordine

Lo schema del principio è il seguente:



$$\frac{dy}{dt} = -\tau^{-1}y + \tau^{-1}u$$

Nel tempo del continuo si ha $A = -\tau^{-1}$ $B = \tau^{-1}$

Nel discreto posso scrivere

$$A_d = e^{AT_s} = e^{-\frac{T_s}{\tau}}$$

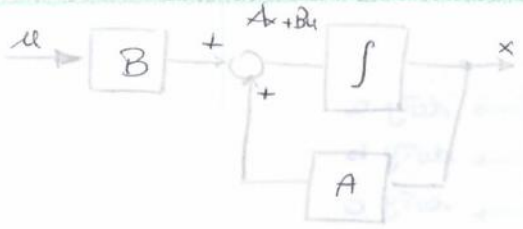
$$B_d = -\tau\left(e^{-\frac{T_s}{\tau}} - 1\right)\tau^{-1} = 1 - e^{-\frac{T_s}{\tau}}$$

In definitiva $y(k+1) = y(k)\,e^{-\frac{T_s}{\tau}} + \left(1 - e^{-\frac{T_s}{\tau}}\right)u(k)$

Un altro modo ancora più pratico è quello con gli schemi a blocchi. Le equazioni sono

$$\dot{x} = Ax + Bu$$
$$x = \int (Ax + Bu)\,dt$$

Lo schema a blocchi è il seguente



L'uscita è x e l'ingresso è $Ax + Bu$ allora si scriverà

$$x(k+1) = x(k) + T_s(Ax(k) + Bu(k))$$
$$x(k+1) = x(k)\cdot(1 + AT_s) + u(k)BT_s$$

($1 + AT_s = z^{-1}$, $BT_s = z^{-1}$)

Quindi se abbiamo una f.d.t qualunque è conveniente scriverla in forma di equazioni di STATO e poi disegnare lo schema a blocchi per vedere di cosa sono gli ingressi dell'integrale. Per ultimo caso si usa la formula calcolando la solita espressione facendo l'integrale discreto.

per avere duty tra 0 e 1.

Il Vdc deve essere il più accurato possibile per poter avere tensioni applicate al motore molto accurate. Tutto questo ci serve per osservare ad esempio il flusso da poter realizzare attraverso l'integrale di Tensione.

Il controllo vettoriale di corrente si presta a tante varianti, una di fatto è un controllo di coppia.

In un motore asincrono la id sarà tenuta ad un valore nominale e serve per flussare la macchina, mentre la iq viene usata per produrre e variare la coppia.

In un motore brushless SPM, la id verrà impostata nulla e la iq viene usata per produrre la coppia.

In un motore a riluttanza la id e la iq verranno mosse di volta in volta in base alla coppia di riferimento.

Posso anche fare come nel motore asincrono id fissa e iq variabile

In un motore IPM (→ ingressi accoppiati) la id e la iq si dovranno muovere assieme, perché non esiste la possibilità di flussare con una sola componente e di dare coppia nulla con il motore già flussato. Se do id mentre iq=0 il motore dà già coppia. Al di là della legge di controllo con cui posiziono le correnti, lo schema della pagina precedente è un controllo di coppia. Il controllo di coppia potrà prescindere al deflussaggio in quanto modo si varierà la costante di coppia della macchina, cioè il flusso della macchina. Quindi questo dipende dal Tipo di macchina.

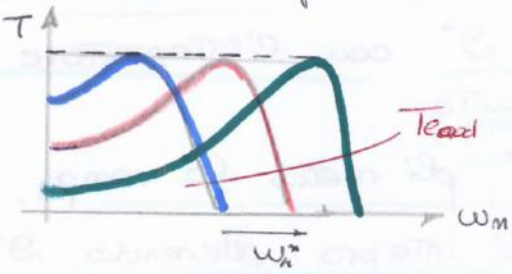
- Controllo scalare V/f del motore asincrono (detto Volt/Hertz)

È il controllo più semplice e il primo ad essere implementato. È l'analogo del controllo in tensione del motore in DC. Si impone al motore una velocità in anello

la I_{s0} è la corrente nominale a vuoto.

Quindi la legge di controllo definitiva sarà $V_s^* = V_0 + K\omega_m^*$ dove V_0 è la componente continua e vale $R_s I_{s0}$, mentre K sarà $K = p|\lambda_{s,nom}|$

Così facendo produrremo nel comp coppia-velocità la caratteristica ben nota del motore asincrono. Si lavorerà nel tratto a basso scorrimento, e se cambiamo la velocità ridotta e se abbiamo implementato la curva traslerà parallelamente



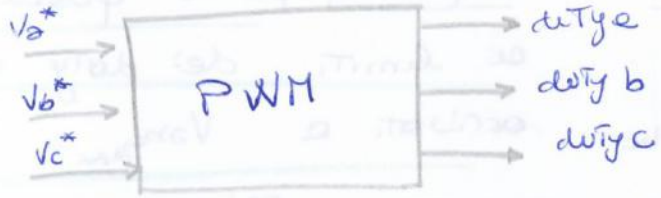
Questo vale in stazionario. Si potrà azionare un carico e regolare la velocità. Però va fatto in maniera lenta.

Come nel motore DC, anche in questo motore, il controllo di tensione ha delle prestazioni dinamiche scarse. Non tiene sotto controllo la corrente e se ci muoviamo velocemente si formano dei transitori di corrente. Si può muovere velocemente la frequenza in un motore asincrono e come alimentare un motore in cortocircuito. Vengono fuori degli scorrimenti elevati che fanno sì che il termine $\frac{R_R}{s}$ molto piccolo

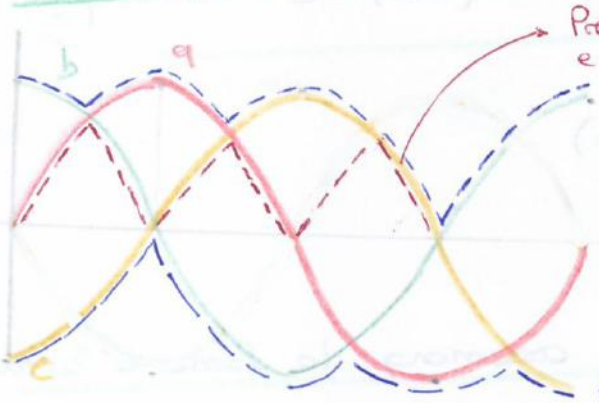
Se avvio di botto a salto ci sarà una corrente di spunto elevatissima. Lo stesso succede se diamo gradini di tensione frequenza. Ricordiamoci dunque che le regole sono le stesse del motore DC. L'unica differenza è che oltre al valore effettivo di tensione che va aggiornato con la velocità, si deve aggiornare la frequenza a cui gira il vettore tensione. Fin più le due cose sono correlate, perché se il vettore tensione gira troppo veloce rispetto alla sua ampiezza vuol dire che la macchina è poco flussata.

Modulatore PWM

Lo schema a blocchi è il seguente:



I suoi ingressi sono le Tensioni di fase di riferimento ed escono i duty. La funzione che fa questo mestiere si chiama PWM Compute( ). Esso Esegue il calcolo dei duty secondo la Tecnica degli impulsi bilanciati, ovvero calcola le modulanti della PWM applicando il modo comune che viene calcolato a partire dai segnali di riferimento. Questo viene fatto per andare a sfruttare pienamente la Tensione disponibile (dc-link). Quindi non è altro che sfruttare pienamente il cerchio inscritto nell'esagono descritto dagli stati dell'inverter. Immaginiamo di avere V_{abc}^* che sono delle sinusoidi.



Se produco : duty semplicemente riscalando per $\frac{2}{V_{dc}}$ e applicando 0,5

$$duty = \frac{2}{V_{dc}} V_{abc}^* + 0,5$$

ho le stesse sinusoidi riscalate tra 0 e 1

Vediamo adesso qual'è la sintassi di PWM compute

```
//sintassi delle PWMcompute
void PWMCompute (void)
{
if (Vdc > 0,25)
  Vdc_inv = 1/Vdc;
if (Vabcs_ref.a > Vabcs_ref.b)
  {
   temp1 = Vabcs_ref.a;
   temp2 = Vabcs_ref.b;
  }
else
  {
   temp 1 = Vabcs_ref.b;
   temp 2 = Vabcs_ref.a;
  }
```

temp1 è una variabile temporanea che contiene il candidato ad essere il massimo e Temp2 per il minimo. L'intermedio sarà quello che rimane. temp3 è la variabile intermedia che cerchiamo. Il codice prosegue così:

```
if (temp 1 < Vabcs_ref.c)
   temp 3 = temp 1;
else
  { if (temp 2 > Vabcs_ref.c)
     temp 3 = temp 2;
    else
     temp 3 = Vabc_ref.c;
  }
PWM_zero_seq = temp 3 * 0,5;
temp 1 = Vabcs_ref.a + PWM_zero_seq;
temp 2 = Vabcs_ref.b + PWM_zero_seq;
temp 3 = Vabcs_ref.c + PWM_zero_seq;
```

Mi calcolo le duty_cicle

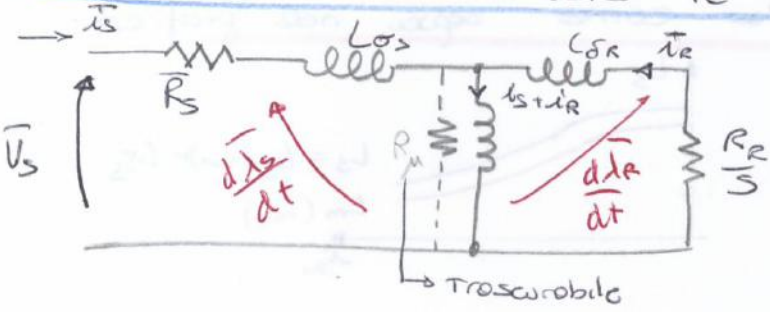
```
duty_abc.a = 0,5 + temp 1 * Vdc_inv;
duty_abc.b = 0,5 + temp 2 * Vdc_inv;
duty_abc.c = 0,5 + temp 3 * Vdc_inv;
```

Infine introduciamo il parametro σ che tiene conto della dispersione totale, da cui poi deriva il termine che rappresenta l'induttanza di c.to c.to $\sigma = 1 - K_S K_R$

Alle equazioni del modello magnetico sostituiamo i valori di L_S e L_R e si ottiene.

$$\left.\begin{aligned} \bar{\lambda}_S &= L_{\sigma S}\,\bar{i}_S + L_m\,(\bar{i}_S + \bar{i}_R) \\ \bar{\lambda}_R &= L_{\sigma S}\,\bar{i}_R + L_m\,(\bar{i}_S + \bar{i}_R) \end{aligned}\right\}$$ nuovo modello magnetico

Queste equazioni appena trovate con quelle del modello elettrico ci permettono di scrivere il circuito equivalente.



Devo ricordare che il termine resistivo di rotore si ha $\frac{R_R}{S}$, con S che è lo scorrimento definito come

$$S = \frac{\omega - \omega_R}{\omega} = 1 - \frac{\omega_R}{\omega}$$

Si è trascurato in questo modello il termine dovuto alle perdite nel ferro. Però per i nostri scopi quel termine non è utile. Il modello serve per dimensionare i regolatori che controllano la macchina e per implementare l'osservatore di flusso da cui parte tutto il controllo. Mentre nel motore sincrono il controllo può essere implementato senza l'osservatore, nel motore asincrono non si può fare.

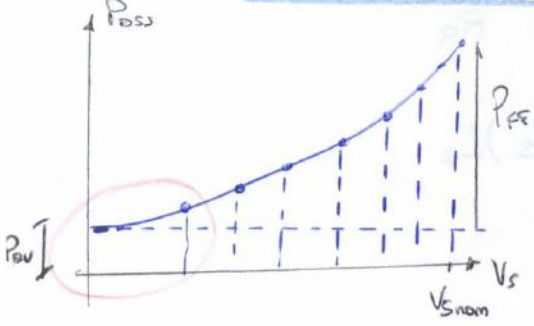
2 parametri necessari a controllare il motore sono tutti presenti nel circuito. Sono dei parametri variabili. Infatti le resistenze variano con la temperatura. Però il parametro più critico di tutti è l'induttanza L_m, perché dipende dal p.to di lavoro. Sappiamo che la caratteristica di magnetizzazione della macchina non è lineare, perché

Questo. Fatto questo si può determinare il contributo di P_{FF} e quello di L_s. Già per calcolare L_s ho fatto la prova a vari livelli di i_{sd} e quindi per diversi valori di livelli di Tensione V_s. Allora per calcolare le perdite nel ferro si determinano plottando la potenza assorbita in funzione della V_s meno il termine $3R_s I_{sd}^2$

Quindi passo a plottare le seguenti Termine

$$P_{ass} - 3R_s I_{sd}^2 = P_{FF} - P_{AV}$$ → Perdite attrito e ventilazione

Quindi viene fuori le seguente curva.



Dalla V_{snom} riduco la Tensione per Trovarmi vari campioni di potenza Però per Tensioni piccole i punti perdono di ripetibilità e la macchina perde il passo. Se però facciamo una spazzatura di Tensione ripetere nelle prove abbiamo per la potenza, un andamento quadratico. Quello sopra tracciato è l'andamento della potenza assorbita durante la prova a vuoto in funzione di V_s. Se a questa sottraggo le perdite Joule di statore, mi rimangono le perdite nel ferro, più un termine dovuto all'attrito e alla ventilazione.

E' però un andamento quadratico con uno "zoccolo" (in rosso). Questo però è solo intuibile da quel grafico perché a bassi valori di Tensione, la velocità non è costante e la corrente assorbita sale e quindi non è molto visibile. Se però si estrapola dai dati della misura trova lo "zoccolo" che sono le P_{AV} a 1500 rpm. Mentre la parte quadratica sarà tutta la quota delle perdite nel ferro 41