UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

Realizzazione di un voltmetro digitale a valore efficace con STM32F334R8

Relatore: Prof. Simone Buso

Laureando: Jacopo Lucchini
1225222

ANNO ACCADEMICO 2022-2023

Data di laurea: 21/03/2023

Abstract

In questo documento viene sviluppata la base di un voltmetro a valore efficace in grado di elaborare la componente fondamentale di un segnale e alcune sue armoniche. La frequenza portante sarà nota a priori. Si potrà utilizzare l'algoritmo con armoniche superiori per avere un calcolo più preciso del valore efficace.
Per l'acquisizione e la manipolazione dei segnali verrà usata la scheda Nucleo64 equipaggiata con un microprocessore *ARM STM32F334R8*. Inoltre, si userà la scheda denominata *mCTb*, acronimo di microController Testbench, come supporto alla sperimentazione. La scheda è stata utilizzata come supporto ai laboratori svolti nel corso di Elettronica Industriale per progettare, ad esempio, filtri del primo o secondo ordine, PLL, o sistemi di comunicazione I^2C con periferiche disponibili sulla stessa.

L'obbiettivo principale di questa esperienza è l'approfondimento delle conoscenze relative alla piattaforma *STM32*, ma sopratutto quello di applicare e approfondire alcuni contenuti di corsi frequentati in questi anni.
Per questo non verranno usati circuiti di condizionamento del segnale concentrandosi sul processo utile nel raggiungimento dell'obbiettivo.

Per la programmazione e la configurazione viene usato l'IDE Keil uVision5 che ci permette di modificare il codice sorgente e di compilarlo per poi caricarlo sulla scheda usata. Viene inoltre usato il programma *STM32CubeMX* per la configurazione delle periferiche del microcontrollore ARM.

Indice

Capitolo 1

Panoramica

In questa sezione viene spiegata la base teorica che ha portato all'implementazione del voltmetro a valore efficace.
Sono presenti anche le simulazioni che sono state effettuate per la verifica del corretto funzionamento e interpretazione della teoria usata.

1.1 Introduzione

L'idea del progetto prevede di sviluppare un voltmetro in grado di misurare il valore efficace di un segnale sinusoidale a 50Hz. Sul mercato ne esistono di molte tipologie, nel nostro caso sarà un'implementazione base.
Il tutto consiste nell'acquisire il segnale con un micro-controllore. Attraverso la trasformata seno e coseno si riesce a trovare quanto vale la componente spettrale a una frequenza prefissata, quindi un valore proporzionale all'ampiezza della componente spettrale del segnale a quella frequenza. Queste funzioni si possono applicare a qualunque frequenza. Noi, per semplicità, fisseremo una frequenza fondamentale di 50Hz evitando quindi di dover trovare la frequenza con la componente spettrale più alta.
Per una maggiore precisione nel calcolo del valore efficace si può ripetere l'algoritmo su una o più armoniche. La maggior parte degli esempi sarà svolta solo sulla frequenza fondamentale per semplicità di visualizzazione e studio.

Il tutto può essere riassunto dal seguente schema a blocchi in figura 1.1.

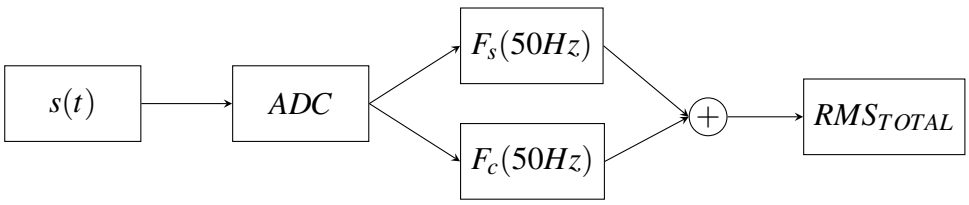


Figura 1.1: Schema di funzionamento

Il blocco *s(t)* è la sorgente del segnale che proviene dall'ambiente esterno. Può essere casuale o conosciuto, con anche delle interferenze.
Il blocco *ADC* è in realtà l'insieme di campionamento e quantizzazione, passaggi necessari per poter trasferire un segnale in un dominio discreto e con un numero finito di valori memorizzabili. Di questo blocco si parlerà nella sezione 2, dove verrà approfondito l'hardware utilizzato per questo progetto.
(I blocchi trasformata e l'operazione di somma verranno spiegati meglio nella prossima sottosezione 1.1.1).

Un esempio di utilizzo di questo algoritmo si può trovare nelle moderne "prese intelligenti" (smart plug) che sono in grado di monitorare, approssimativamente, i consumi di un apparato connesso. Esse sono capaci di misurare la tensione ai loro capi e la corrente che le attraversa. Dalle trasformate si può ricavare anche lo sfasamento tra corrente e tensione necessario per il calcolo della potenza attiva e reattiva. Le "prese intelligenti" si possono trovare come dispositivi "stand-alone" o integrate nei più moderni impianti casalinghi. Un esempio si può vedere nell'immagine 1.2.

Figura 1.2: Esempio smart plug "stand-alone"

1.1.1 Trasformata seno e coseno

Esistono molti modi per implementare le trasformate di cui andremo a parlare. Una delle caratteristiche delle trasformate è il poter risalire a un ingresso in maniera univoca dal loro risultato. Facendo ciò si può passare a un dominio in cui è più semplice effettuare alcune operazioni.
La trasformata coseno, semplificando, è la parte reale della trasformata di Fourier a meno di qualche costante.
La definiamo come:

$$F_c = \sum_{n=1}^{N-1} f_n \cdot cos(n\omega) \tag{1.1}$$

Il termine f_n è l'elemento del vettore di dati che abbiamo in ingresso nella posizione n. Mentre ω è la pulsazione alla quale viene calcolata la trasformata e si trova con la formula classica $\omega = 2\pi f$, dove f è la frequenza. Nel nostro caso N è il numero di campioni in cui viene diviso un periodo della funzione coseno. Quindi usando come esempio la frequenza di campionamento $f_s = 1kHz$ e la frequenza a cui calcoliamo la trasformata $f_{tra} = 50Hz$ si avrà:

$$N = \frac{f_s}{f_{tra}} = \frac{1 \cdot 10^3}{50} = 20 \; campioni \tag{1.2}$$

Si può dire quindi di avere una finestra di N campioni su cui viene effettuato il calcolo. Maggiore è la frequenza di campionamento, maggiore sarà la grandezza della finestra.
Applicando quindi la trasformata a un segnale in ingresso ci viene restituito un valore costante proporzionale all'ampiezza della componente spettrale scelta.

La trasformata seno, come per la precedente, può essere la parte immaginaria della trasformata di Fourier a meno di qualche costante.
La definiamo come:

$$F_s = \sum_{m=1}^{N-1} f_m \cdot sin(m\omega) \tag{1.3}$$

Come prima si ha un vettore di dati e la funzione principale, in questo caso il seno.

La loro proprietà di produrre come risultato un valore relativo alla quantità di una certa componente spettrale ci permette di usarle come *"filtro"* che permette il passaggio di una certa pulsazione (lo verificheremo nella sezione 1.2 con MATLAB). Inoltre, essendo simili alla parte reale e immaginaria della trasformata di Fourier si possono usare per trovare il valore di segnali anche sfasati rispetto alla funzione coseno o seno. Non è una problematica che abbiamo studiato ma ne vedremo gli effetti nella sezione di simulazione con MATLAB (1.2).

Alla fine delle operazioni il risultato delle due viene sommato attraverso il teorema di Pitagora generalizzato come di seguito:

$$F_t = \sqrt{F_S^2 + F_C^2} \tag{1.4}$$

vista la loro dualità con parte immaginaria e reale come nell'immagine 1.3.

1.1.2 Filtro passa basso del primo ordine

Il filtro passa basso del primo ordine ci servirà per sopprimere tutte quelle frequenze indesiderate introdotte dall'operazione di prodotto. Vedremo due tipologie di filtro digitale realizzabile. Entrambi hanno come base un'equazione alle differenze come la seguente:

$$y(k) = b_0x(k) + b_1x(k-1) + \cdots + b_nx(k-n) + +(k-1) + a_2y(k-2) + \cdots + a_my(k-m) \tag{1.5}$$

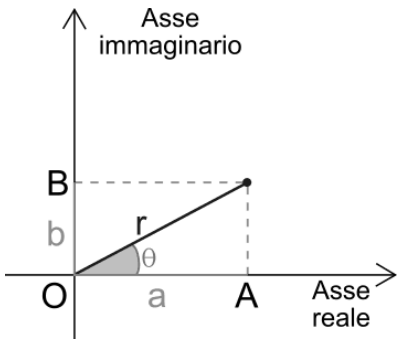


Figura 1.3: Dominio dei numeri complessi

La differenza principale, oltre alla loro risposta in frequenza, sono i coefficienti di questa equazione.

Filtro passa basso IIR

Si è scelto come primo studio di utilizzare la discretizzazione di un filtro a tempo continuo, che non approfondiremo in questo documento.
Quando uno dei coefficienti $a_i \neq 0$ il filtro è detto ricorsivo e quindi di tipo IIR, abbreviazione di *Infinite Impulse Response*. Questo rispecchia il caso della discretizzazione di un filtro passa basso.
Per un filtro passa basso del primo ordine avremo che l'equazione alle differenze risulta:

$$y(k) = b \cdot x(k) + a \cdot y(k-1) \tag{1.6}$$

viene detto ricorsivo perché il termine $y(k-1)$, il risultato prodotto dal filtro con il campione precedente, ha un coefficiente diverso da zero.
Ponendo

$$a = b - 1 \tag{1.7}$$

il filtro avrà guadagno unitario. Nel caso in cui questo non fosse valido bisogna comunque rispettare che il coefficiente a sia minore di 1 per avere una stabilità del filtro.
Lo schema a blocchi del filtro IIR trovato sarà il seguente

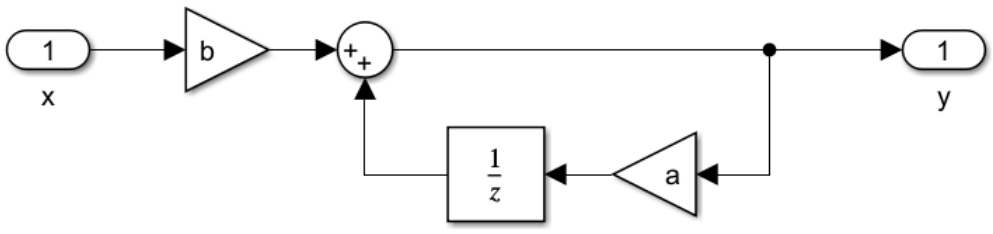


Figura 1.4: Schema filtro IIR

Vedremo le sue prestazioni nella sezione 1.2.

Filtro passa basso FIR

Un filtro FIR ha invece i coefficienti a_i, nell'equazione 1.5, tutti identicamente nulli. Un esempio di filtro passa basso FIR può avere la seguente equazione:

$$y(k) = \frac{1}{N} \cdot \sum_{i=0}^{N-1} x(k-i) \tag{1.8}$$

Viene detto filtro a media mobile proprio perché effettua la media su N campioni. Ogni campione nuovo inserito elimina quello più vecchio, come un buffer circolare di dimensione N.
Un filtro FIR necessita di un numero di coefficienti molto più elevato rispetto a un filtro IIR per ottenere la stessa risposta in frequenza.
Il suo schema a blocchi equivalente risulta:

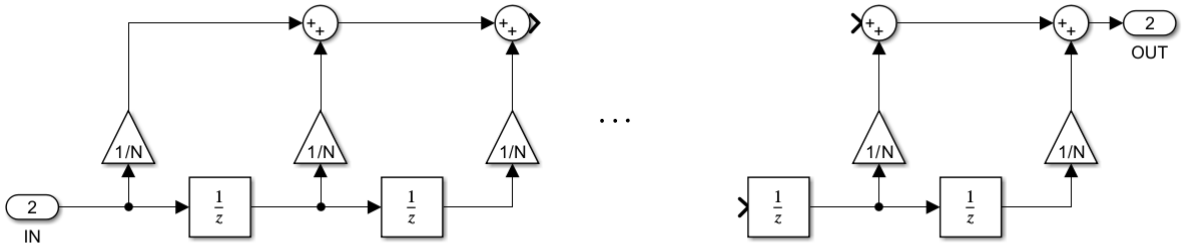


Figura 1.5: Schema filtro FIR con coefficiente N

Il progetto dei filtri verrà svolto nel paragrafo 4.

1.2 Simulazione in MATLAB

In questa sezione sarà spiegata l'analisi con *MATLAB* che ha portato alla verifica della teoria. Si è utilizzato *Simulink*, che ci permette di creare uno schema a blocchi che rappresenta, in maniera semplificata, il sistema composto dal micro-controllore. Per semplificare lo studio teorico del sistema si è inoltre scelto di utilizzare solo la trasformata a 50Hz.

1.2.1 Schema a blocchi Simulink

Lo schema a blocchi in *Simulink* del nostro micro-controllore è visibile in immagine 1.6.

Si ha un segnale in ingresso collegato all'ingresso **x** del nostro blocco **DSP**, che sta per Digital Signal Processor. All'uscita **y** del nostro DSP abbiamo invece una sonda utile per visualizzare il risultato del nostro sistema. Per concludere abbiamo un clock che serve per sincronizzare il momento in cui il sistema effettua le operazioni, come in un μC.

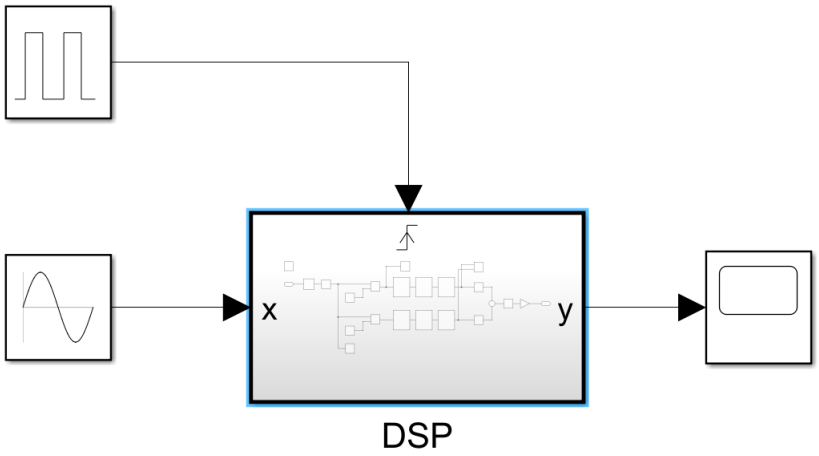


Figura 1.6: Schema a blocchi μC

Approfondiamo ora il blocco **DSP** che comprende le operazioni che noi andremo a scrivere nel codice.

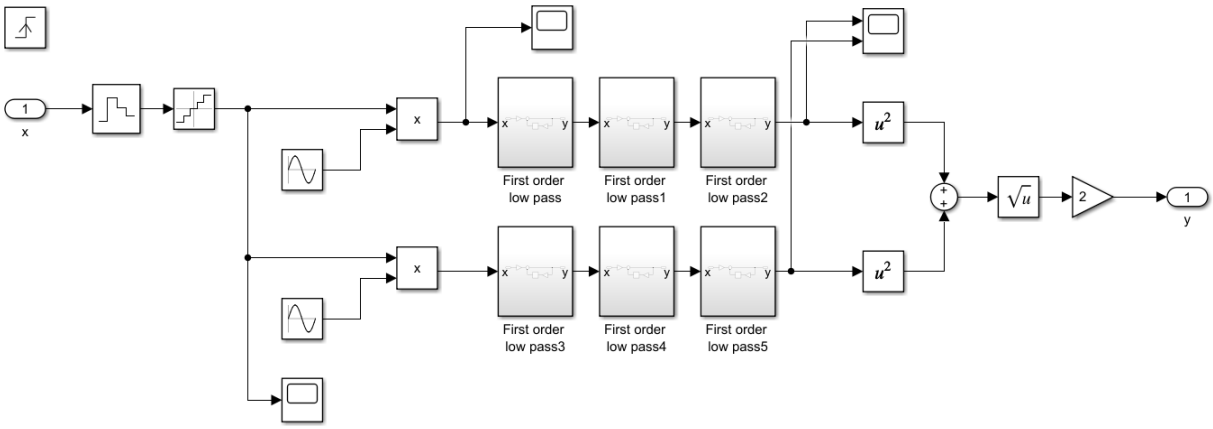


Figura 1.7: Schema a blocchi algoritmo

In questo caso l'ingresso e l'uscita del sistema sono individuati rispettivamente da x e y, mentre il segnale di clock viene individuato dal blocco in alto a sinistra, che indica anche quale fronte del segnale di sincronizzazione utilizziamo. Nel nostro caso il fronte di salita (quindi ogni qualvolta che il segnale cambia da uno stato "basso" a uno stato "alto").

I primi due blocchi rappresentano l'ADC. Il primo è detto "Hold and Sample", la traduzione ci suggerisce che "blocca e salva" il segnale in ingresso. Mentre il secondo è il quantizzatore che permette di mappare l'ingresso, continuo e con un numero infinito di possibili valori, in un numero finito di possibili valori.

Il tutto viene poi diviso in due rami, uno per svolgere i calcoli relativi alla trasformata seno e uno per la trasformata coseno. Di seguito un ingrandimento su uno dei due rami che rimangono uguali in ogni loro parte tranne per l'onda (seno o coseno) per cui vengono moltiplicati i campioni.

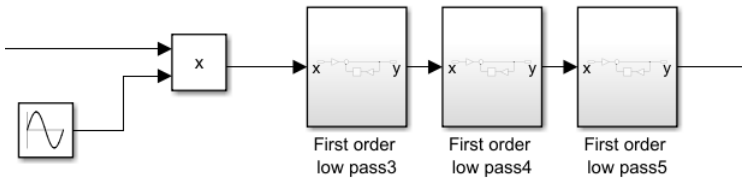


Figura 1.8: Operazioni della trasformata

Il blocco $\boxed{x}$ è la semplice operazione di prodotto tra due numeri che vengono forniti, uno dal blocco "segnale" e uno dal ADC. Successivamente il risultato prosegue attraverso un banco di filtri: in questo caso sono tre filtri IIR del primo ordine in cascata. Il blocco segnale sarà la funzione seno o coseno in base alla trasformata da calcolare.
Si conclude con la somma dei due vettori risultanti attraverso il teorema di Pitagora generalizzato (1.4). Da notare il blocco con un fattore moltiplicativo che ci aiuta a riportare il risultato al valore corretto.

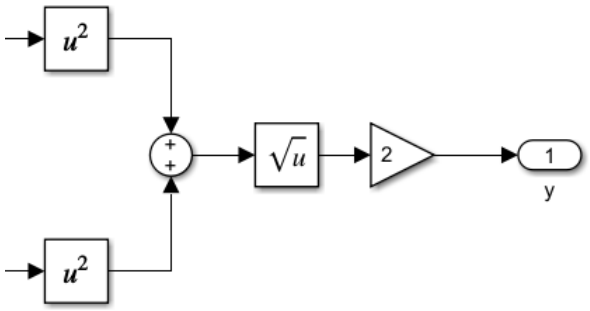


Figura 1.9: Schema a blocchi teorema di Pitagora generalizzato

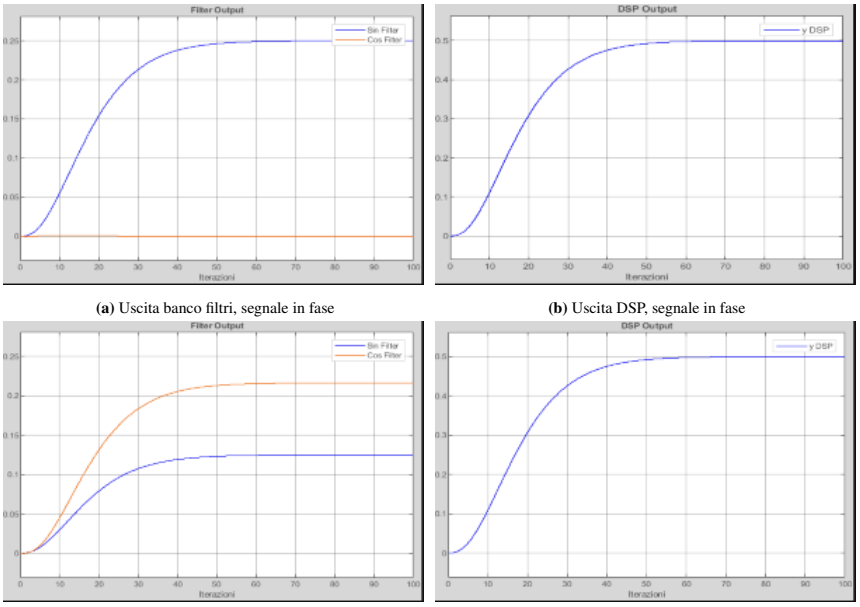
1.3 Risultati Teorici

Attraverso le sonde, che è possibile aggiungere allo schema a blocchi, siamo in grado di studiare il comportamento del sistema a diversi ingressi. Di seguito alcuni studi sul sistema.

Segnale in ingresso sfasato rispetto a una delle due trasformate

Come prima cosa si è deciso di controllare il suo comportamento con in ingresso un segnale sfasato di un valore casuale rispetto alle trasformate. Nel primo caso il segnale era quindi in fase rispetto al seno usato per il calcolo della trasformata. Mentre nel secondo caso era sfasato di una quantità non nulla.

Le due trasformate sono visibili nel grafico intitolato *Filter Output*, rispettivamente di colore blu, la trasformata seno, e arancione la trasformata coseno.
Il risultato qui sopra fa notare come nel caso di segnali sfasati le trasformate abbiano entrambe un risultato non nullo, mentre nel caso in cui il segnale sia in fase con una delle due trasformate il calcolo di una delle due tende a zero. (immagini 1.10a e 1.10c)



(a) Uscita banco filtri, segnale in fase

(b) Uscita DSP, segnale in fase

(c) Uscita banco filtri, segnale sfasato

(d) Uscita DSP, segnale sfasato

Figura 1.10: Risultati sonde con due fasi diverse.
A sinistra la sonda relativa all'uscita dei filtri.
A destra l'uscita del sistema.

Mentre è importante notare come l'andamento e il valore a cui tendono i grafici intitolati *DSP Output*, che come ci dice il nome sono l'uscita "y" del nostro sistema, si stabilizzano a un valore uguale in entrambi i casi.

Segnale in ingresso con frequenza diversa rispetto a quella della trasformata

Nel caso in cui invece il segnale in ingresso abbia una frequenza diversa da quella utilizzata per il calcolo delle trasformate si ha che l'uscita del nostro sistema tende a zero dopo un certo periodo. Questo accade per la proprietà filtrante delle trasformate che hanno uscita non nulla solo con segnali alla stessa frequenza.

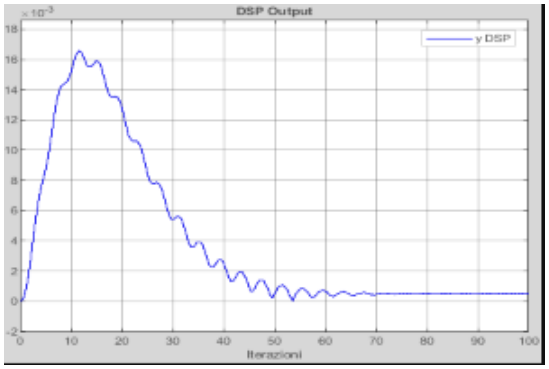


Figura 1.11: Uscita sistema con frequenza del segnale in ingresso diversa da quella delle trasformate

Come si può vedere il risultato in figura 1.11 è stato ingrandito per far notare come l'uscita del sistema non tende a zero immediatamente, ma dopo alcune iterazioni. In ogni caso il valore di picco raggiunto è piccolo rispetto al valore raggiunto con una frequenza in ingresso corretta.

Studio filtri IIR

Si è provato a cambiare la frequenza di taglio del banco dei filtri IIR studiandone l'uscita.
Il risultato in figura 1.10b ha una frequenza del banco dei filtri molto bassa. Questo porta in uscita solo la componente continua del segnale, ovvero l'informazione di cui abbiamo bisogno. Le frequenze indesiderate sono dovute al prodotto del segnale in ingresso con il seno o il coseno. Nel caso in cui si abbia una frequenza di taglio non adatta il risultato in uscita non sarà stabile, si avrà del ripple residuo. Sarà quindi difficile da visualizzare, non avendo un valore costante. Lo si può vedere in figura 1.12.

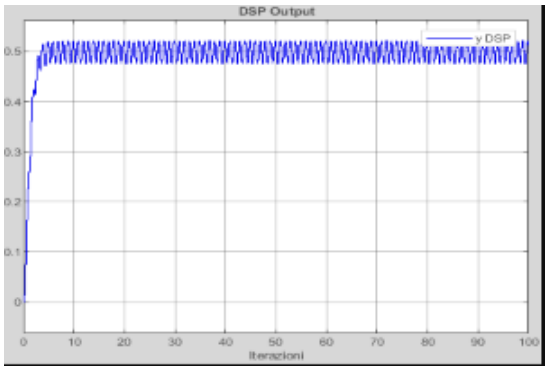


Figura 1.12: Uscita sistema con frequenza di taglio troppo alta

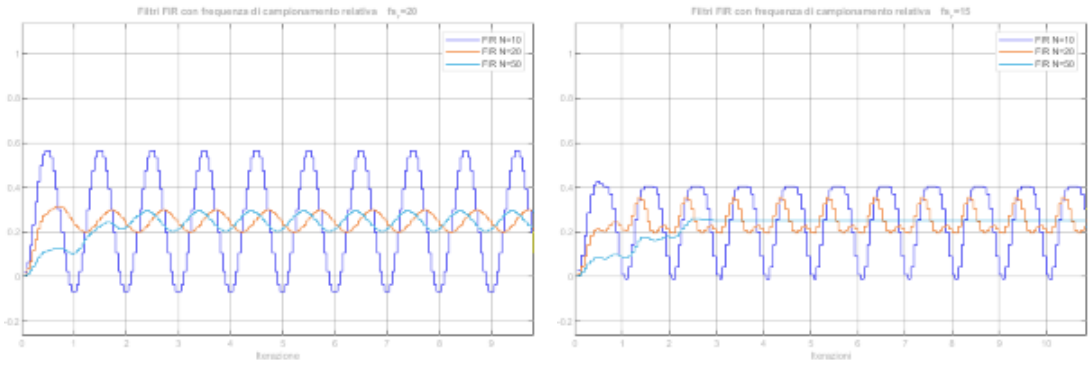
Si nota che la media del segnale a regime tende a un valore di 0.5.

Studio filtri FIR

Per concludere lo studio abbiamo sostituito il banco di filtri IIR passa basso del primo ordine con un filtro FIR di ordine $N = 50$. L'equazione alle differenze riprende la 1.8 lo schema a blocchi corrisponde a quello in immagine 1.5 con $N = 50$.
Nel primo caso (immagine 1.13a) abbiamo una frequenza relativa $f_r = 20$, questo significa che il periodo del segnale in ingresso è suddiviso in 20 campioni. Mentre nel secondo caso (immagine 1.13b) abbiamo diminuito la frequenza relativa a $f_r = 15$ il che migliora la risposta del filtro senza dover aumentare il numero di campioni e operazioni usati per il calcolo.

1.3.1 Conclusioni Teoriche

Utilizzando le trasformate riusciamo quindi a eliminare l'errore che introdurrebbe uno sfasamento del segnale d'ingresso. Questo semplifica il codice non dovendo aggiungere algoritmi per mantenere i segnali sincronizzati tra loro. Inoltre le trasformate ci forniscono l'informazione di sfasamento rispetto alle funzioni seno e coseno. Potrebbe esserci utile nel caso volessimo implementare uno strumento in grado di misurare la potenza attiva e reattiva assorbita da un circuito.
Dall'analisi precedente notiamo anche che un filtro IIR passa basso è più efficace nell'eliminare



(a) Risposta filtro FIR con $f_r = 20$ e $N = 50$

(b) Risposta filtro FIR con $f_r = 15$ e $N = 50$

Figura 1.13: Risposta filtro a media mobile con diverse frequenze di campionamento relative

le basse frequenze utilizzando meno coefficienti. Si nota che, a parità di frequenza di campionamento, un filtro a media mobile con 50 coefficienti non riesce a filtrare completamente il segnale in ingresso. Questo è uno degli aspetti che differenzia un filtro IIR da un filtro FIR. Un filtro FIR equivalente deve avere molti più coefficienti per avere la stessa efficacia.

Capitolo 2

Hardware

In questo capitolo parleremo del hardware utilizzato per la realizzazione del progetto. Inoltre si descriveranno, in breve, gli strumenti di misura utilizzati per la verifica del corretto funzionamento e l'analisi dell'algoritmo per il calcolo del valore efficace.

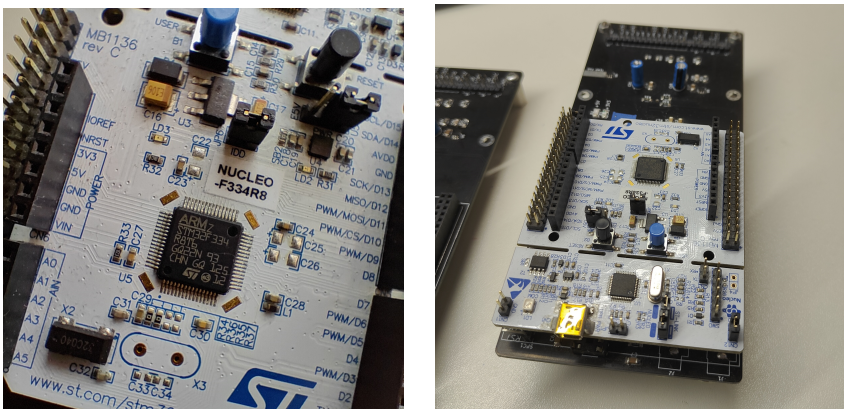
2.1 Microcontrollore STM32

2.1.1 Caratteristiche generali

Il micro-controllore STM32F334R8 (immagine 2.1a) fa parte della famiglia di micro-controllori ARM Cortex-M4 con aritmetica a 32 bit con floating point unit. Può essere alimentato con una tensione da 2.0 a 3.6 V. La sua frequenza di funzionamento può raggiungere 72 MHz.
Alcune delle sue caratteristiche sono:

- 64 Kbyte di memoria Flash;
- 12 Kbyte di memoria SRAM;
- timer ad alta risoluzione a 16-bit 10 canali;
- 5 timer general purpose (1 a 32 bit e 4 a 16-bit)
- 2 ADC a 12-bit con 21 canali;
- 3 canali DAC a 12 bit;

La scheda da noi usata, chiamata Nucleo64 (immagine 2.1b) usa il micro-controllore appena citato. Le schede Nucleo64 sono dette di sviluppo, cioè con la circuitazione di supporto necessaria per sperimentare con il micro-controllore. Sono nate per l'esigenza di facilitare la sperimentazione e la prototipazione.
Sulla scheda Nucleo64 in nostro possesso è presente un programmatore *ST-LINK*, che ci permette di caricare, tramite un cavo USB micro-B, nella memoria interna del micro-controllore il codice da eseguire. Per la comunicazione con l'ambiente circostante sono presenti dei connettori che possono svolgere diverse funzioni. Alcuni di essi sono compatibili con lo *"Standard Arduino-UNO"* e quindi con alcune daughter-board create per la scheda Arduino-UNO.



(a) Macro su μC STM32F334R8 **(b)** Nucleo-64 inserita su mCTb

Figura 2.1: Nucleo64 F334R8

Convertitore analogico digitale

Un convertitore analogico digitale (in inglese analog to digital converter) è un dispositivo in grado di convertire nel dominio discreto un segnale continuo.
Da ora abbrevieremo convertitore analogico digitale con ADC. Un ADC è composto da due blocchi fondamentali: un sample and hold(1) e un quantizzatore(2), come visualizzato nella figura 2.2.

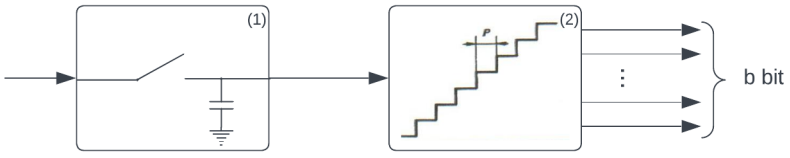


Figura 2.2: Schema a blocchi "Sample & Hold"(1) e quantizzatore(2)

Il blocco (1) garantisce che il segnale in ingresso al secondo blocco sia il più possibile costante nel momento in cui viene effettuata la conversione. Il blocco (2) riconduce a un numero b di bit il valore in ingresso approssimandolo alla soglia più vicina.
L'ADC implementato sul micro-controllore è di tipo ad approssimazioni successive. Viene spesso utilizzato per il suo ottimo compromesso fra prestazioni e complessità del circuito.
Il suo funzionamento consiste nel formare in uscita una tensione che a ogni iterazione viene comparata con il valore in ingresso. Inizia la conversione dal bit più significativo finendo con quello meno significativo. Se il valore generato è maggiore dell'ingresso si imposta il bit relativo a 0. Se invece è minore viene impostato a 1. Si procede così per ogni bit lasciando invariati i bit precedenti.

Con STM32F334R8 è possibile scegliere la risoluzione dell'ADC tra 12/10/8/6 bit, scegliendo il compromesso tra velocità e precisione.

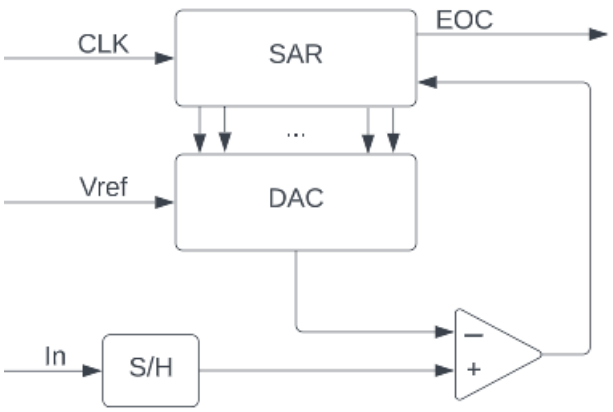


Figura 2.3: Schema a blocchi ADC SAR

Timer

Il timer di un micro-controllore è composto da un registro che viene incrementato a ogni impulso di clock. Il clock che accetta in ingresso il timer può essere diviso o moltiplicato, con un prescaler. Si possono avere frequenze diverse dal clock di sistema. Il registro del timer può essere utilizzato direttamente dall'utente richiamando un'opportuna funzione. Oppure può essere comparato con un altro registro contenente un valore scelto dall'utente. Se il valore salvato nel registro del timer risulta uguale al valore salvato nel registro di comparazione si può scegliere di lanciare un interrupt e/o resettare il timer. I timer sono delle unità autonome e quindi non influiscono sui cicli di esecuzione di un programma.
Nel nostro caso il timer verrà utilizzato come trigger della conversione dell'ADC, avendo così una frequenza di campionamento costante.

2.2 Generatore di funzioni

Il generatore di funzioni è uno strumento che ci permette di generare diverse forme d'onda per verificare il corretto funzionamento di un circuito da collaudare.
Verrà usato nella parte finale di questo documento per la verifica dell'algoritmo. I parametri più importanti che si possono variare sono:

- Forma d'onda
- Ampiezza;
- Frequenza;
- Offset;

Usando questo strumento possiamo evitare di aggiungere circuiti di condizionamento del segnale in ingresso semplificando lo studio del circuito interessato ed evitando distorsioni aggiuntive al segnale.

Figura 2.4: Generatore di funzioni Agilent 33220A

2.3 Oscilloscopio

L'oscilloscopio è uno strumento fondamentale per la visualizzazione dei segnali elettrici variabili nel tempo. Nel caso di un segnale periodico con un oscilloscopio si riescono a fare misure di periodo, duty cycle, sfasamento rispetto a un altro segnale e molto altro. Gli oscilloscopi moderni sono prevalentemente digitali e permettono di svolgere funzioni che un oscilloscopio analogico non sarebbe in grado di svolgere. Si possono usare, ad esempio, per trasmettere a un computer i dati acquisiti, potendo fare calcoli più complessi rispetto a quelli che è in grado di svolgere l'hardware interno all'oscilloscopio.

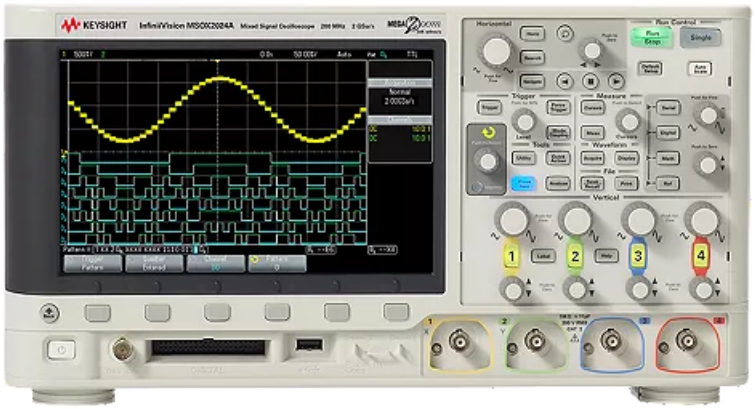


Figura 2.5: Esempio oscilloscopio digitale

La funzione che noi utilizzeremo consiste nel salvare l'immagine di quello che l'utente visualizza a schermo su un dispositivo di archiviazione di massa (es. chiavetta USB).

Capitolo 3

Software

3.1 Configurazione periferiche

In questa sezione verranno illustrate le scelte per le impostazioni delle periferiche nel codice C. L'impostazione è stata svolta con il programma *STM32CubeMX* che ci permette di modificare i parametri delle periferiche con un'interfaccia grafica. Il programma fornisce anche una visualizzazione dei reofori del chip. Questa funzione è importante dal momento che un piedino del micro-controllore può avere più funzioni diverse tra loro. Inoltre CubeMX ci permette di cambiare le impostazioni delle periferiche anche se l'utente ha modificato il codice. Per far si che il codice utente non venga modificato deve essere inserito tra i commenti `USER CODE BEGIN` e `USER CODE END`.

Le istruzioni in questo paragrafo sono le prime richiamate nel programma. Finita la configurazione il programma entra in un loop infinito dove non viene svolta alcuna operazione e si aspetta l'arrivo di un evento di interrupt.

3.1.1 Configurazione timer

Nel nostro caso è stato utilizzato il `TIMER1` dei 12 possibili timer di cui è equipaggiato il micro-controllore. Si è scelto questo timer perché non è necessaria un'alta velocità di conteggio, avendo scelto una frequenza di campionamento relativamente bassa.

```
1 void MX_TIM1_Init(void)
2 {
3
4   ...
5
6   htim1.Instance = TIM1;
7   htim1.Init.Prescaler = 1;
8   htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
9   htim1.Init.Period = 15000;
```

```
10   htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
11
12   ...
13
14   sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
15
16   ...
17
18   /* USER CODE BEGIN TIM1_Init 2 */
19   //Start the timer after config
20   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
21   /* USER CODE END TIM1_Init 2 */
22
23 }
```

Listing 3.1: Istruzioni impostazione TIMER1

L'istruzione `htim1.Instance = TIM1` assegna alla variabile `Instance` il valore `TIM1`. Si è così scelto il timer da utilizzare e inizializzare. L'assegnazione del valore `TIM_COUNTERMODE_UP` indica che il conteggio sarà crescente. Mentre l'assegnazione `Period` ci dice che il periodo, cioè il valore massimo che raggiunge il timer, sarà 15000. Infine `TIM_CLOCKDIVISION_DIV1` ci dice che è impostato un prescaler che divide il clock in ingresso per un fattore 2. Sapendo che il clock che viene portato al blocco dei timer ha una frequenza di $f_{clk} = 72MHz$ si trova

$$f_{timer} = \frac{f_{clk}}{2} \tag{3.1}$$

$$f_{reset} = \frac{f_{timer}}{Period} \tag{3.2}$$

Nel nostro caso $f_{reset} = 2400Hz$ corrisponde alla frequenza di campionamento. Essa soddisfa di gran lunga il teorema del campionamento supponendo che la frequenza massima del segnale in ingresso possa essere di 150Hz. Con questa frequenza si riescono a garantire anche abbastanza campioni per le trasformate.
Con l'istruzione `TIM_TRGO_UPDATE`, il timer lancia un trigger ogni volta che viene aggiornato. Questo trigger viene poi usato dall'ADC che inizia la conversione del dato in ingresso.
Con il comando `HAL_TIM_PWM_Start` viene avviato il timer una volta conclusa la sua configurazione.

3.1.2 Configurazione ADC

Per quanto riguarda l'ADC si è scelto di usare il secondo A/D, dei due a nostra disposizione. Usando il canale 4 del secondo ADC, come si vede dal codice di configurazione in 3.2, si ha una protezione in ingresso fornita dalla scheda mCTb. L'acquisizione viene triggerata con il segnale esterno `T1_TRGO` generato dal `TIMER1` (riga 10 listato 3.2).

```
1
```

```
void MX_ADC2_Init(void)
{

  ...

  /** Common config*/
  hadc2.Instance = ADC2;
  hadc2.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
  hadc2.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T1_TRGO;

  ...

  /** Configure Regular Channel*/
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;

  ...

  /* USER CODE BEGIN ADC2_Init 2 */
  //Calibrate the ADC before use
  HAL_ADCEx_Calibration_Start(&hadc2, ADC_SINGLE_ENDED);

  //Starting the adc with interrupt
  HAL_ADC_Start_IT(&hadc2);
  /* USER CODE END ADC2_Init 2 */

}
```

Listing 3.2: Istruzioni impostazione ADC2

Il canale viene impostato per un'acquisizione *SINGLE_ENDED*, quindi viene riferito alla massa del sistema invece che a un secondo ingresso con la modalità differenziale.
Il codice di inizializzazione del ADC si conclude con un comando che calibra il convertitore e lo avvia con la modalità interrupt. Viene generato un segnale di interrupt ogni volta che il convertitore finisce la conversione e viene richiamata la routine `HAL_ADC_ConvCpltCallback` in cui scriveremo il codice da eseguire (paragrafo 3.2.2).

3.2 Codice Principale

In questo paragrafo saranno spiegate le funzioni utilizzate per implementare lo schema a blocchi visto in immagine 1.7. Viene scelto di implementare una funzione di filtraggio per rendere il codice più leggibile e facile da modificare. Inoltre viene sovra scritta la funzione `HAL_ADC_ConvCpltCallback` come da istruzioni nelle libreria dell'ADC. Questa funzione è detta di "callback", viene richiamata ogni volta che l'ADC conclude la conversione.

3.2.1 Funzione Filtro del primo ordine

Per prima cosa abbiamo scritto la funzione del filtro del primo ordine. Si è deciso di utilizzare un passaggio con puntatori vista la sua maggiore efficienza rispetto a ritornare una variabile. Si è utilizzato questo metodo anche perchè il filtro usa una variabile in ingresso che poi viene modificata (*y[n-1]*).

```
/*Filter function implementation
  a = coefficient filter
  b = coefficient filter
  inS = input of the filter
  y = input/output array for y(n) e y (n-1) passed by address
*/
void lowPass_Filter(uint16_t a, uint16_t b, uint16_t inS, uint32_t* y)
{
  y[0] = (b * inS + a * y[1]) >> 16;
  y[1] = y[0];
}
```

Listing 3.3: Funzione filtro del primo ordine

La funzione segue lo schema in figura 1.4.

3.2.2 Callback ADC

Il codice che vedremo sarà relativo a una sola frequenza, ma è facilmente scalabile a frequenze diverse. In questa funzione sono presenti le operazioni per il calcolo della trasformata, come visualizzato in figura 1.7. Le variabili necessarie nella funzione di "Callback" sono visibili nel listato 3.4. Ogni vettore nella sezione "`bank filter variable`" rappresenta l'uscita del filtro $y[n]$ e $y[n-1]$. L'inizializzazione delle variabili è stata effettuata nella sezione "`Private Variable`" del file *main.c*.

```
//Sinusoidi calcolo della trasformata
const uint16_t sin5[] = { ... };
const uint16_t cos5[] = { ... };

uint16_t adOut, daIn;
uint32_t mul1, mul2;

//first bank filter variable
uint32_t y1[] = {0, 0};
uint32_t y2[] = {0, 0};
uint32_t y3[] = {0, 0};
uint32_t z1[] = {0, 0};
uint32_t z2[] = {0, 0};
uint32_t z3[] = {0, 0};

//other variable
```

```
uint32_t sPow, cPow, rmsPow;
uint8_t i=0;

//Filter Coefficient
uint16_t aCoef = 65364;
uint16_t bCoef = 171;
```

Listing 3.4: Inizializzazione varibili

Le variabili `sin5` e `cos5` sono una Look Up Table (LUT), che riporta rispettivamente seno e coseno nel dominio dei discreti senza segno. Sono state create grazie al sito "Sine LUT Generator". Sapendo che la frequenza di campionamento è $f_c = 2400Hz$ e che la frequenza dell'onda da generare è $f_{TRA} = 50Hz$, troviamo che il numero di campioni da salvare sono:

$$n = \frac{f_c}{f_{TRA}} = \frac{2400Hz}{50Hz} = 48 \text{ campioni} \tag{3.3}$$

Sono stati infine inizializzati i coefficienti dei filtri (`aCoef, bCoef`).

La funzione di callback dell'ADC viene dichiarata nella libreria (HAL) del convertitore. Viene esplicitato che la modifica di tale funzione è da effettuare nel file principale creato dall'utente e non nei file di libreria. Questo perchè la libreria può essere utilizzata in altri programmi. Il codice della funzione callback è il seguente:

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  HAL_GPIO_TogglePin(Green_Led_GPIO_Port, Green_Led_Pin);
  adOut = (HAL_ADC_GetValue(&hadc2))<<4;

  //Algoritmo calcolo 50 Hz
  mul1 = (adOut*sin5[i])>>16;
  mul2 = (adOut*cos5[i])>>16;
  //Filtro Seno
  lowPass_Filter(aCoef, bCoef, mul1, y1);
  lowPass_Filter(aCoef, bCoef, y1[0], y2);
  lowPass_Filter(aCoef, bCoef, y2[0], y3);
  //Filtro Coseno
  lowPass_Filter(aCoef, bCoef, mul2, z1);
  lowPass_Filter(aCoef, bCoef, z1[0], z2);
  lowPass_Filter(aCoef, bCoef, z2[0], z3);

  //Power of 2
  sPow = ((y3[0] * y3[0])+(w3[0] * w3[0])+(k3[0] * k3[0]))>>16;
  cPow = ((z3[0] * z3[0])+(x3[0] * x3[0])+(q3[0] * q3[0]))>>16;

  //Sum
  rmsPow = sPow + cPow;

  //Output
```

```
26    daIn = rmsPow;
27
28    HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, adOut);
29
30    //Aggiornamento indici
31    i++;
32    if(i>47)
33      i = 0;
34    HAL_GPIO_TogglePin(Green_Led_GPIO_Port, Green_Led_Pin);
35  }
```

Listing 3.5: Funzione di Callback dell'ADC

Viene prima salvato il valore convertito dall'ADC nella variabile *"adOut"*. Dal momento che il dato in uscita dall'adc è a 12 bit viene traslato di 4 bit a sinistra. Successivamente l'uscita traslata si moltiplica per il campione relativo della funzione seno e coseno. Per salvare il risultato abbiamo usato una variabile a 32 bit. Questo perché l'operazione di prodotto tra due dati binari ne produce uno con lunghezza doppia (se il numero di bit dei dati coincide). L'operazione si conclude con uno shift a destra di 16 bit così da avere la parte più significativa nei 16 bit meno significativi, che rimangono utilizzabili per altre operazioni in variabili di dimensioni minori. In questo modo si può utilizzare il risultato per effettuare altre operazioni, con meno bit, mantenendo la maggior parte dell'informazione.

L'operazione di prodotto produce in uscita delle frequenze indesiderate, per questo il dato deve poi essere filtrato. Concluse le operazioni di filtraggio le uscite vengono elevate al quadrato e sommate seguendo così lo schema in figura 1.9. Per effettuare l'operazione di radice quadrata si potrà usare una LUT, più veloce nell'esecuzione.

Capitolo 4

Realizzazione e risultati

In questo capitolo verranno illustrati i passaggi che hanno portato alla verifica del codice e dell'algoritmo. Verranno utilizzati come strumenti di supporto un oscilloscopio per visualizzare le forme d'onda in ingresso e uscita dal sistema, e un generatore di funzioni che genererà il segnale in ingresso al sistema.
All'interno del codice sono state inserite alcune funzioni che aiuteranno nella verifica. In particolare nel listato 3.5 riga 3 e riga 34 si manipola un piedino di uscita per indicarci la durata e la frequenza con cui l'ADC converte ed effettua le operazioni. Dal periodo del segnale riusciamo a capire a quale frequenza il convertitore sta campionando. Dalla lunghezza della parte alta capiremo invece la durata della subroutine.
Inoltre, sempre nel listato 3.5 riga 28, viene impostato nel registro del DAC (Digital to Analog Converter) un valore in uscita. Quest'ultima operazione ci consente di verificare l'andamento temporale del valore e sarà utile per verificare il funzionamento del banco di filtri. Non approfondiremo l'inizializzazione del DAC perché viene usato come supporto e non come parte integrante dei nostri calcoli.

4.1 Risulati pratici

Verifica filtri passa basso

Si è scelto di verificare prima il funzionamento del banco di filtraggio per confermarne l'efficacia. Come già detto precedentemente si utilizza il DAC per visualizzare l'uscita su un oscilloscopio. La frequenza di taglio è stata calcolata con le formule classiche dei filtri IIR per essere intorno a 1Hz. Il filtro passa basso di partenza è il seguente:

$$F(s) = \frac{1}{1 + sT_p} \tag{4.1}$$

Attraverso la trasformata z abbiamo che:

$$s = \frac{1 - z^{-1}}{T_c} \tag{4.2}$$

Calcoliamo prima T_p e T_c che ci serviranno successivamente per trovare i coefficienti dei filtri.

$$T_p = \frac{1}{2\pi \cdot f_t} \tag{4.3}$$

$$T_c = \frac{1}{f_c} \tag{4.4}$$

f_t corrisponde alla frequenza di taglio, mentre f_c è quella di campionamento. Conoscendo $f_c = 2.4kHz$ e scegliendo come frequenza di taglio una frequenza abbastanza bassa, ad esempio $f_t = 1Hz$, risulta:

$$T_p = 0.1591549431 \tag{4.5}$$

$$T_c = 0.0004166667 \tag{4.6}$$

Dalla discretizzazione si trova:

$$a = \frac{T_p}{T_p + T_c} \tag{4.7}$$

$$b = \frac{T_c}{T_p + T_c} \tag{4.8}$$

Usando una delle due formule sopra ricaviamo un coefficiente, ad esempio $a = 0.9973888421$. Ricordando che deve valere l'equazione 1.7 per avere un guadagno unitario del filtro ricaviamo anche $b = 0.002611157$. Per sfruttare al massimo l'aritmetica del nostro micro-controllore, rappresentiamo questi numeri su 16 bit.

$$a_{DEC} = 65364 \rightarrow a_{HEX16} = 0xFF54 \tag{4.9}$$

$$b_{DEC} = 171 \rightarrow b_{HEX16} = 0x00AB \tag{4.10}$$

I risultati ottenuti dovranno essere numeri interi. Questo introduce un errore di quantizzazione sulla frequenza di taglio. Attraverso la formula inversa troviamo il vero valore della frequenza di taglio del sistema.

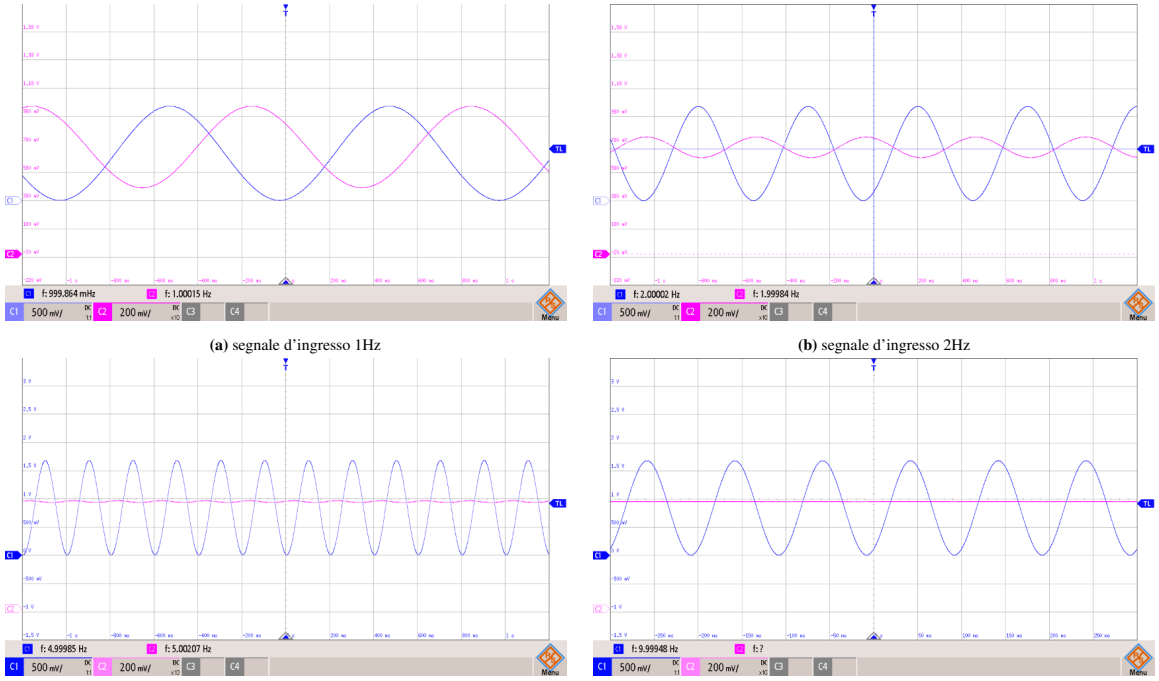
$$f_t = \frac{b}{2\pi \cdot T_c \cdot (1-b)} = 0.9999996Hz \tag{4.11}$$

Questo risultato si può approssimare a $1Hz$. In laboratorio si è verificata la risposta del sistema applicando in ingresso un segnale sinusoidale a frequenze diverse ($1Hz$, $2Hz$, $5Hz$, $10Hz$). Nelle figure 4.1 si nota come più la frequenza d'ingresso aumenta, più il segnale viene attenuato, mentre alla frequenza di taglio l'attenuazione è minima.

Questo rispecchia la risposta che deve avere un filtro passa basso.

Verifica totale algoritmo

La verifica del sistema è stata effettuata sul codice relativo alla componente fondamentale di 50Hz. Si è scelto di non utilizzare anche le componenti armoniche, che garantirebbero maggio-



(a) segnale d'ingresso 1Hz

(b) segnale d'ingresso 2Hz

(c) segnale d'ingresso 5Hz

(d) segnale d'ingresso 10Hz

Figura 4.1: Acquisizioni da oscilloscopio digitale, banco di filtri 1Hz.
In giallo l'ingresso, in verde l'uscita del sistema.

re precisione del risultato. Il programma che viene eseguito dal micro-controllore è quello nel listato 3.5. Applicando diversi ingressi al sistema si è riuscito a creare una tabella.

Frequenza Ingresso [Hz]	Ampiezza Ingresso [V_{pp}]	RMS Teorico [V]	Valore Uscita Sistema
50 Hz	2.0 V	0.70710 V	9159
60 Hz	2.0 V	0.70710 V	7119
100 Hz	2.0 V	0.70710 V	7108
50 Hz	2.5 V	0.88388 V	10433
60 Hz	2.5 V	0.88388 V	7166
100 Hz	2.5 V	0.88388 V	7124
50 Hz	3.0 V	1.06066 V	12390
60 Hz	3.0 V	1.06066 V	7379
100 Hz	3.0 V	1.06066 V	7251

Tabella 4.1: Tabella risultati uscita

I valori di frequenza e ampiezza sono stati scelti dal display del generatore di funzioni e controllati successivamente con l'aiuto dell'oscilloscopio. Il valore efficace in tabella 4.1 è stato calcolato attraverso la formula:

$$V_{RMS} = \frac{V_{pp}}{2 \cdot \sqrt{2}} \tag{4.12}$$

Questa vale solo per segnali sinusoidali e senza distorsioni, come nel nostro caso.

Notiamo come i segnali a frequenza diversa da quella base, cioè la frequenza delle trasformate, tendono a un valore simile. Questo valore è costante, ma diverso da zero e non coincide

con la simulazione. Per interpretarlo bisogna sapere che, nella simulazione, le funzioni seno e coseno per la trasformata avevano dinamica compresa tra ± 1. Questo permetteva all'uscita del prodotto di assumere valori negativi. Dopo il filtraggio si aveva quindi un valore nullo. Nel nostro caso il micro-controllore utilizza solo valori positivi senza segno. Si ha quindi un valore costante sommato all'uscita. Questo spiega perché i segnali a frequenza diversa da quella calcolata avessero un valore in uscita diverso da zero ma simile.

Assumo come valore medio il minimo tra gli ingressi trovati ($V_m = 7108$). Togliendo questo valore all'uscita a frequenza di 50Hz abbiamo:

Frequenza Ingresso [Hz]	Ampiezza Ingresso [V_{pp}]	Uscita - Valore Medio
50 Hz	2.0 V	2051
50 Hz	2.5 V	3325
50 Hz	3.0 V	5282

Tabella 4.2: Tabella risultati uscita traslati

Utilizzando i valori nella tabella 4.2 possiamo proseguire con i calcoli per la verifica del valore efficace. I passaggi rimanenti sono:

$$V_{RMS} = \sqrt{\text{output}} \cdot k \tag{4.13}$$

Usando il dato intermedio, cioè i risultati a 2.5V, calcolo il valore di k:

$$k \approx \frac{1}{65} \tag{4.14}$$

Applichiamo il risultato ai casi con ampiezza 3V e 2V per verificarlo. Troviamo:

$$V_{RMS}(3.0V) = \frac{\sqrt{5282}}{65} = 1.1096 \approx V_{RMS_{teorico}} \tag{4.15}$$

$$V_{RMS}(2.0V) = \frac{\sqrt{2051}}{65} = 0.6967 \approx V_{RMS_{teorico}} \tag{4.16}$$

Il risultato ottenuto è simile al valore teorico in tabella 4.1. Potremmo migliorare il risultato applicando l'algoritmo ad alcune armoniche avendo così una precisione migliore.

4.2 Conclusione

Come visto nei paragrafi precedenti il risultato finale è simile a quello reale. Il banco di filtri del primo ordine reagisce a ingressi con frequenze diverse come ci si aspettava. Si hanno comunque degli errori relativi alla quantizzazione e ai cicli limite del micro-controllore. Questo è migliorabile aumentando la precisione della piattaforma usata, ad esempio a 64 bit. Inoltre i filtri si possono migliorare usando più coefficienti o utilizzando un filtro FIR di ordine maggiore. In

questo caso possono essere d'aiuto per la progettazione programmi come MATLAB, che integrano algoritmi per il calcolo dei coefficienti dei filtri. Migliorare il filtraggio della componente continua del segnale in uscita aiuterebbe di gran lunga la lettura del dato finale che diventerebbe più stabile rispetto a quanto riscontrato. Non è stato migliorato questo aspetto data la grande quantità di tipologie diverse di filtri esistenti. Si è ritenuto opportuno dedicargli uno studio più approfondito.

Aumentare la frequenza di campionamento migliorerebbe la precisione del risultato. Nel nostro caso è stata scelta così bassa per avere pochi campioni nelle funzioni seno e coseno delle trasformate. Garantiva allo stesso tempo di avere abbastanza campioni per la trasformata a $150Hz$. Si è comunque scelto di non implementare il codice per le armoniche visti i buoni risultati ottenuti con la sola frequenza portante.

Con questa esperienza si puntava ad approfondire la programmazione della piattaforma STM32 e ad applicare alcuni contenuti dei corsi frequentati in questi anni. Al compimento di questo progetto ritengo di aver aumentato le mie conoscenze sul digital signal processing e sopratutto sulla programmazione che riguarda micro-controllori ARM.

Bibliografia

[1] Buso S., *Esercitazioni con il microcontrollore STM32F334R8 per il corso di Microcontrollori e DSP*, 2018.

[2] STMicroelctronics, *How to get the best ADC accuracy in STM32Fx Series, application note (AN2834)*, 2013.

[3] STMicroelctronics, *STM32F334x8 Datasheet (Rev 9)*, 2018.

[4] "Keysight Technologies", `https://www.keysight.com/it/en/home.html`

[5] "TP-Link Corporation Limited", `https://www.tp-link.com/it/home-networking/smart-plug/tapo-p100/`

[6] Daycounter, Inc., "Sine Look Up Table Generator", `https://www.daycounter.com/Calculators/Sine-Generator-Calculator.phtml`