SAPIENZA
UNIVERSITÀ DI ROMA

Multimetro digitale basato su microcontrollore STM32 con rilevamento della forma d'onda

Facoltà di Ingegneria dell'informazione, informatica e statistica
Dipartimento di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni
Corso di laurea in Ingegneria Elettronica

Marco Cipriani
Matricola 1938396

Relatore
Prof. Emanuele Piuzzi

A.A. 2023–2024

Sommario

1 Premessa

I multimetri digitali (*DMM, Digital Multimeter*) attualmente sul mercato non sono in grado di determinare automaticamente o mostrare all'utente il tipo di forma d'onda della tensione o corrente sotto misura[1]. Possono misurare separatamente grandezze alternate o continue solo dopo la selezione manuale della modalità $\left(\tilde{V},\ \overline{\tilde{V}},\ \tilde{A},\ \overline{\tilde{A}}\right)$, il che comporta possibili errori di misura o rischi per l'operatore. Quest'ultimo potrebbe assumere erroneamente l'assenza di alta tensione su un cavo dopo aver effettuato una misura con la sola modalità tensione alternata.

Inoltre, i multimetri commerciali, *True RMS* e non, non possono distinguere tra onda sinusoidale pura, sinusoidale modificata o onda quadra, né rilevare la presenza di picchi o segnali sovrapposti. Disporre di queste informazioni consentirebbe a un tecnico di diagnosticare più rapidamente guasti in inverter, gruppi di continuità o altre apparecchiature ad essi collegate, nonché verificare le dichiarazioni dei produttori di tali dispositivi. I maggiori siti di *e-commerce* al giorno d'oggi offrono infatti un'ampia scelta di questi prodotti, ed è spesso difficile capire se un inverter sia effettivamente a sinusoidale pura o meno. Per comprenderlo, si avrebbe normalmente bisogno di un oscilloscopio con sonda differenziale ad alta tensione, uno strumento a disposizione solo di laboratori attrezzati.

L'obbiettivo di questa tesi è lo sviluppo di un multimetro *proof-of-concept* in grado di misurare e visualizzare contemporaneamente le componenti continua e alternata di una tensione, e al contempo fornire il maggior numero possibile di informazioni sulla forma d'onda in ingresso al multimetro. Ci si concentrerà dapprima sull'architettura generale di un multimetro, con lo scopo di individuare i cambiamenti necessari per permettere le misure richieste, poi sulla nuova architettura basata su microprocessore *STM32*. In seguito, si configureranno le periferiche del microcontrollore e l'*ADC* (*Analog-to-Digital Converter*) interno. Infine, verrà descritto il firmware basato su algoritmi di *DSP* (*Digital Signal Processing*) che è stato sviluppato.

[1] Si vedano, ad esempio, i cataloghi online di Fluke® e Amprobe®.

2 Funzionamento dei multimetri digitali

2.1 Caso studio: *Fluke® 87V*

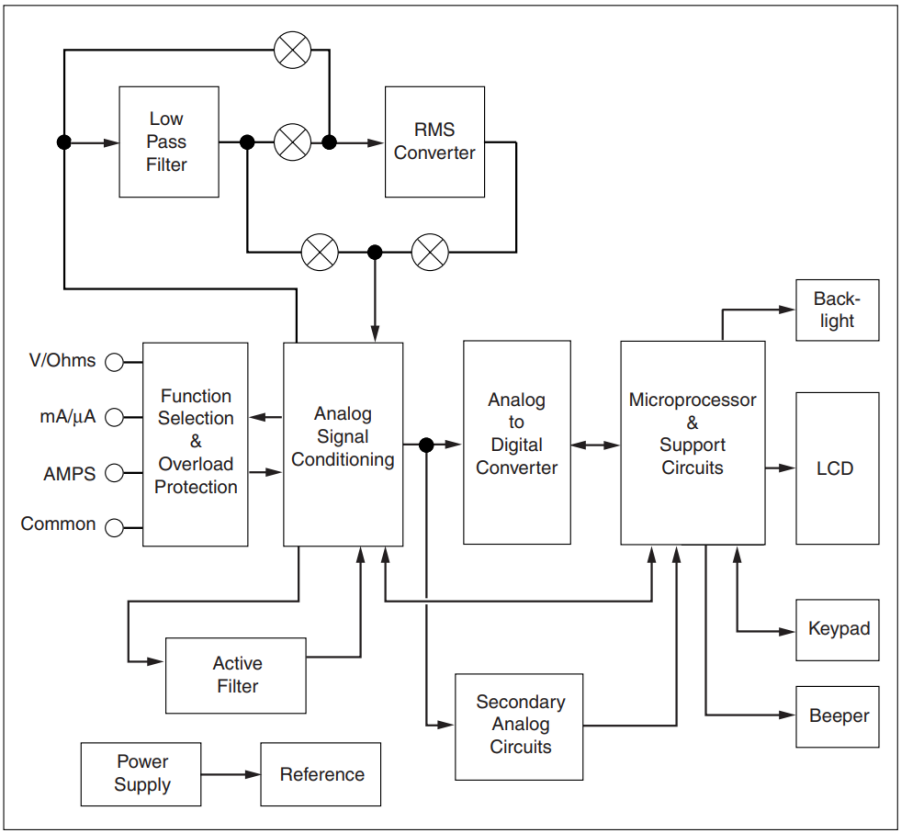

Figura 1: diagramma a blocchi del *Fluke® 87V* [1]

Per comprendere il funzionamento dei multimetri commerciali, si è deciso di prendere in esame il multimetro *Fluke® 87V* in quanto è uno dei pochi multimetri moderni per cui è ancora disponibile un manuale di servizio completo di spiegazioni e schema elettrico. L'analisi si concentrerà sul tragitto del segnale dai connettori di ingresso all'*ADC*.

2.1.1 Protezione ingressi

Come si può notare nello schema a blocchi in figura 1, collegato ai terminali di ingresso del multimetro è presente un circuito di protezione che in generale può contenere i seguenti componenti:

- *MOV* (*Metal-Oxide Varistor*), che proteggono le funzioni di misura di tensione e resistenza da fenomeni transitori di sovratensione. Sono spesso in serie con resistenze ad alto wattaggio, che limitano la corrente e prolungano la vita dei varistori.

- Fusibili *PPTC* (*Polymeric Positive Temperature Coefficient*): sono fusibili "resettabili" che interrompono una connessione quando si scaldano e la ripristinano quando si raffreddano.
- Fusibili *HRC* (*High Rupturing Capacity*), ossia fusibili in grado di interrompere forti correnti anche ad alte tensioni, che potrebbero non venir interrotte da normali fusibili di vetro a causa di archi elettrici. Proteggono il multimetro nelle misure di corrente.
- Un ponte di diodi, chiuso su un quinto diodo di *clamping*, che limita la tensione, positiva o negativa, ai capi delle resistenze di *shunt*.
- Diodi *TVS* (*Transient-Voltage-Suppression*), che svolgono un ruolo simile ai *MOV* ma con tempi di reazione più veloci ma capacità di dissipazione ridotta. Sostituiscono spesso il ponte di diodi in multimetri economici.

Dopo il circuito di protezione, la tensione proveniente dall'ingresso V e Ω raggiunge direttamente l'interruttore rotativo del multimetro, mentre le correnti degli ingressi A e mA passano per le due resistenze di *shunt* prima di uscire dal terminale COM. La tensione ai capi degli shunt, proporzionale alla corrente in esame, è quella che verrà effettivamente misurata dall'*ADC* e moltiplicata per un fattore di conversione per ottenere il valore numerico della corrente iniziale. Per questo motivo, ci si concentrerà da qui in poi sulle sole misure di tensione, senza perdita di generalità.

2.1.2 Condizionamento del segnale e filtraggio

Dopo il blocco di protezione e selezione della funzione, segue il blocco di condizionamento del segnale. Al suo interno, la tensione da misurare viene inizialmente ridotta nella gamma dinamica dell'*ADC* tramite un partitore resistivo (in questo caso il multimetro è *auto-ranging*, quindi l'attenuazione è scelta automaticamente dal multimetro). Successivamente, a seconda della posizione dell'interruttore, il segnale può attraversare due percorsi distinti prima di raggiungere l'*ADC*:

a) un filtro passa-basso attivo, nel caso di misure *DC*;
b) un condensatore di rimozione della continua, seguito da un filtro passa-basso a $800\ Hz$, poi dal convertitore *RMS* e infine nuovamente dal filtro attivo, per le misure *AC*.

Figura 2: circuito stampato del multimetro *Fluke® 87V*. Fonte: [25]

Il manuale del *Fluke® 87V* [1] descrive infatti:

> *After the Meter is set to the desired function, the signal is routed to the signal conditioning circuit. Either automatically or manually, a range is selected that puts the signal to be measured within the dynamic range of the analog-to-digital converter (ADC) or other signal conditioning circuits like the RMS-to-DC converter [...]. A scaled AC signal voltage is routed directly, or via an 800-Hz low-pass filter, to the RMS-to-DC converter circuit. A DC input signal or DC output of the RMS-to-DC converter (for AC functions) is routed to a low-pass, 6-Hz, 2-pole active filter to prepare it for ADC measurement. The conditioned analog input signal voltage is converted to a digital value by the ADC and sent to the microprocessor. The microprocessor converts this digital value for display on the LCD based on the function, range and keypad entered options.*

I filtri passa-basso hanno lo scopo di far raggiungere all'*ADC* solo un valore medio (in un tempo che dipende dalla costante di tempo del filtro stesso) del segnale sotto misura, nonché rimuovere componenti ad alta frequenza che potrebbero disturbare il convertitore *RMS*. Nell'utilizzo di un multimetro, infatti, si è spesso interessati solo a un valore medio di tensione o corrente. I filtri sono individuabili in parte in figura 2 e al

completo nello schema elettrico in fondo al Service Manual [1], insieme a tutti gli altri componenti descritti finora.

Ci si concentrerà ora sul blocco denominato *RMS converter*, per il quale passano i segnali nella modalità *AC* del multimetro, introducendo prima il concetto di valore efficace. Si passerà poi all'analisi del convertitore analogico-digitale, dove avviene la misura del segnale *DC* o del segnale *AC* convertito.

2.2 Valore efficace e convertitori *RMS*

Il valore efficace è il valore che avrebbe un segnale costante di pari potenza media:

$$\overline{P} = V_{rms} \cdot I_{rms} \tag{1}$$

È definito [2] come la media quadratica sul periodo della funzione stessa, ossia la radice della media dei quadrati. Nel caso di una tensione $v(t)$ variabile nel tempo e periodica di periodo T:

$$V_{rms} = \sqrt{\frac{1}{T}\int_0^T v^2(t)\ dt} \tag{2}$$

Nel caso di campionamento discreto della tensione, con n campioni v_i, si ha:

$$V_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} v_i^2} \tag{3}$$

Dalla sua definizione risulta ovvia l'utilità di questa quantità, per segnali sinusoidali o comunque non continui, per la determinazione della potenza elettrica utilizzata da un circuito in esame. Il valore efficace di un segnale risulta in generale diverso dalla media (evidente per segnali alternati a media nulla) e dalla media del valore assoluto. Nel caso di un segnale sinusoidale, il valore di picco V_p e il valore efficace sono legati dalla relazione $V_{eff} = V_p/\sqrt{2}$, ottenuta dall'equazione 1 sostituendo $v(t) = V_p \cdot \sin\left(\frac{2\pi}{T}t\right)$. Si riportano a seguire i fattori di conversione (*crest factor*, V_p/V_{rms}) tra i valori di picco ed efficaci di vari tipi di segnale:

Segnale continuo	1
Onda sinusoidale	$\sqrt{2}$
Onda quadra	1
Onda triangolare	$\sqrt{3}$

Per permettere al multimetro di calcolare il valore efficace della tensione che si vuole misurare, si utilizza dunque un componente chiamato *convertitore RMS*, che ha il compito di convertire il segnale in ingresso in una tensione continua proporzionale, a seconda del tipo di convertitore, al valore di picco, medio o efficace dell'ingresso stesso.

2.2.1 Convertitori *RMS* a valore di picco

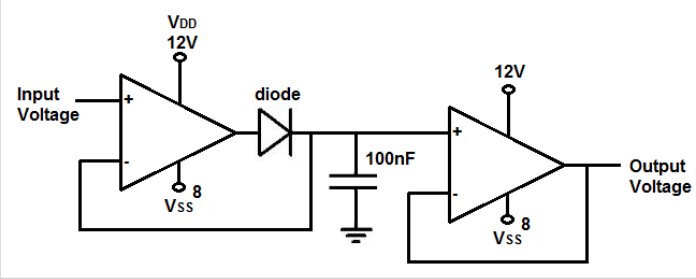

Figura 3: rivelatore di picchi [26].

I convertitori *RMS* più semplici si basano su un circuito rilevatore di picchi (*peak detector*) che sfrutta un diodo (o, comunemente, un circuito *super-diodo*) e un condensatore per ricavare V_p, e quindi V_{rms}, assumendo che il segnale sia sinusoidale. Per l'analisi circuitale si rimanda a [3]. Sono caratterizzati da grandi errori di misura quando la forma d'onda è diversa da quella per cui sono progettati e da scarsa reiezione dei disturbi.

2.2.2 Convertitori *RMS* a valore medio

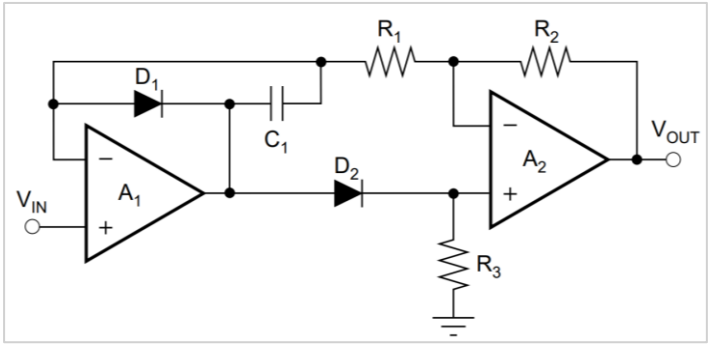

Figura 4: circuito *valore assoluto* di precisione [7].

A partire dal famoso *Fluke® 8020A* [4] [5], uno dei primi multimetri digitali di successo [6], si è preferito usare convertitori a valore medio. I multimetri di questo tipo sono detti *average responding*. Questo tipo di multimetri utilizza un circuito valore assoluto di precisione [7] [8], come quello mostrato in figura 4, per raddrizzare la tensione alternata senza la caduta di tensione tipica dei semplici ponti di diodi. Il segnale passa poi in un filtro passa-basso, che ne fa la media, per poi raggiungere l'*ADC* del multimetro. Il microprocessore effettua infine la

moltiplicazione per un fattore di conversione pari a $\pi/2\sqrt{2}$ tra valore medio ed efficace di un'onda sinusoidale [9]. Ne segue che, se la forma d'onda non è sinusoidale, il multimetro fornirà una lettura che può differire anche del 40% rispetto al valore corretto [10].

2.2.3 Convertitori *True RMS*

I convertitori *True RMS* sono spesso unificati in un unico circuito integrato, come l'*AD737* [11] utilizzato nel *Fluke® 87V* [1]. Sono anche detti *RMS-to-DC* converters in quanto la loro uscita è un segnale continuo proporzionale al valore efficace della tensione al loro ingresso. Il principio di funzionamento di questi convertitori si basa sulla realizzazione di circuiti in grado di effettuare, nel dominio analogico, il quadrato del segnale, poi una media per ottenere la media del quadrato, ed infine la radice quadrata del segnale. Un'ottima descrizione di un primo tipo di questo circuito, detto *convertitore RMS esplicito*, si trova nella documentazione [12] del produttore dell'*AD737*, *Analog Devices*:

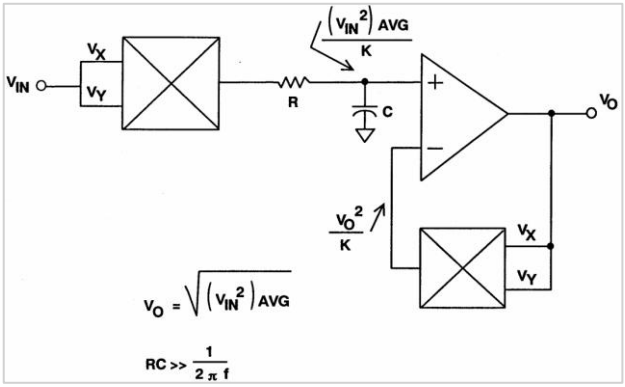

Figura 5: convertitore *RMS* esplicito [12].

The input signal is first squared by an analog multiplier. The average value is then taken by using an appropriate filter, and the square root is taken using an op amp with a second squarer in the feedback loop. This circuit has limited dynamic range because the stages following the squarer must try to deal with a signal that varies enormously in amplitude. This restricts this method to inputs which have a maximum dynamic range of approximately 10:1 (20 dB).

Il moltiplicatore analogico è spesso implementato come una cella di Gilbert [13], la cui uscita è proporzionale, con un fattore $1/K$, al prodotto delle tensioni in ingresso [14]. La retroazione negativa dell'amplificatore operazionale rende l'uscita V_o pari alla radice quadrata di $(V_{IN}^2)_{AVG}/K$ in quanto l'amplificatore reagisce in modo da annullare la differenza di potenziale tra l'ingresso invertente e quello non invertente. Quindi:

$$\frac{(V_{IN}^2)_{AVG}}{K} = \frac{{V_O}^2}{K} \;\Rightarrow\; V_O = \sqrt{(V_{IN}^2)_{AVG}}$$

Per risolvere il problema della gamma dinamica, si usa più spesso un cosiddetto *convertitore implicito*, descritto nel documento già citato:

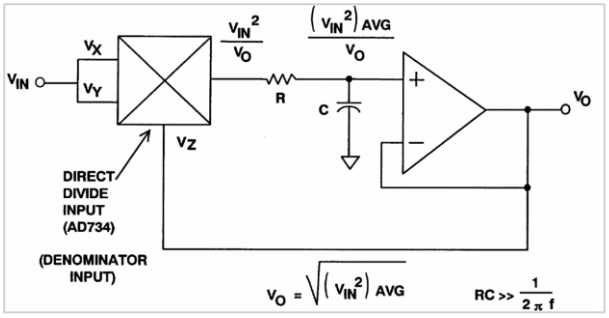

Figura 6: convertitore *RMS* implicito [12].

Here, the output is fed back to the direct-divide input of a multiplier such as the AD734. In this circuit, the output of the multiplier varies linearly (instead of as the square) with the rms value of the input. This considerably increases the dynamic range of the implicit circuit as compared to the explicit circuit. [...] The disadvantage of the implicit rms-to-dc approach is that it generally has less bandwidth than the explicit computation.

In questo caso, l'amplificatore operazionale ha solo il ruolo di *buffer* dopo il filtro *RC*, e il moltiplicatore analogico è sostituito con un circuito con funzione di trasferimento $X \cdot Y/Z$ (*four-quadrant analog multiplier/divider*). Con calcoli analoghi ai precedenti, si ottiene nuovamente $V_O = \sqrt{(V_{IN}^2)_{AVG}}$.

Curiosità: come è possibile vedere in figura 7, tratta dal datasheet dell'*AD737* [11], il condensatore C_A, che assume il ruolo del condensatore di figura 6, è esterno al convertitore in quanto non è possibile inserire grandi capacità all'interno di circuiti integrati. In assenza di C_A l'uscita V_O del convertitore è pari a V_{IN} poiché l'elevamento a potenza e la radice si semplificano (non avviene più l'operazione di media).

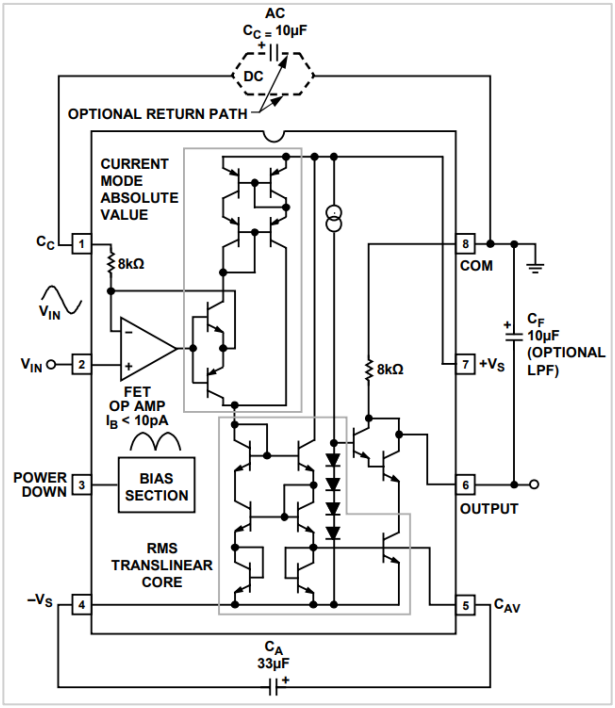

Figura 7: schema interno del convertitore AD737

Questo fatto è sfruttato dai produttori di multimetri, i quali possono utilizzare lo stesso circuito stampato per due modelli diversi di multimetri, uno *True RMS* e uno *average responding*. Il *Fluke® 83V* è infatti identico all'*87V*, ad eccezione per un condensatore mancante. In questi casi, è possibile convertire il multimetro in *True RMS*, con la conseguenza però di dover calibrare nuovamente lo strumento [15].

2.3 Convertitore analogico-digitale

I convertitori analogico-digitale sono componenti (o parte di circuiti integrati più complessi) che convertono un segnale analogico continuo in un segnale digitale, ossia una sequenza di bit o byte che rappresentano il segnale di partenza in forma quantizzata e discretizzata nel tempo. L'analisi dettagliata delle varie architetture di *ADC* è fuori dallo scopo di questa tesi; tuttavia, si riportano a seguire brevi descrizioni dei tipi più pertinenti all'uso in un multimetro digitale.

Nel caso del Fluke® 87V[1], è presente un *ADC* delta-sigma a 20 bit con un riferimento di tensione esterno di precisione, utilizzato insieme a un regolatore lineare a $5\ V$ per generare due tensioni di riferimento, $-2.5\ V$ e $+2.5\ V$ (*split rail power supply*), le quali alimentano l'*ADC* e gli permettono di effettuare misure di tensione sia positiva che negativa.

2.3.1 *ADC* ad integrazione (o doppia rampa, *dual slope*)

In questa architettura, la tensione da misurare è applicata all'ingresso di un integratore, precedentemente azzerato, per un periodo di tempo fisso detto *run-up*. Durante questa fase, il condensatore dell'integratore si carica. Successivamente, l'ingresso dell'integratore viene commutato su una tensione di riferimento di polarità opposta fino a che l'integratore non si scarica del tutto (*run-down*). Si può ricavare a questo punto la tensione da misurare in funzione del tempo di *run-up*, la tensione di riferimento e il tempo di *run-down* misurato. Questo tipo di convertitore offre un'elevata risoluzione, sacrificando però la frequenza di campionamento. Tuttavia, è molto comune nei multimetri in quanto restituisce la media del segnale durante l'intero periodo di *run-up*, migliorando il rapporto segnale-rumore. Inoltre, se il periodo di *run-up* è scelto in modo da essere un multiplo intero del periodo della tensione di rete alternata (50 o $60\ Hz$), si minimizzano i disturbi di misura dovuti a quest'ultima.

2.3.2 *ADC* ad approssimazioni successive (*SAR*)

L'*ADC SAR* è costituito da un circuito *sample and hold* in ingresso, seguito da un comparatore che confronta il segnale con l'uscita di un convertitore digitale-analogico (*DAC*). L'uscita del comparatore è collegata ad una logica di controllo, chiamata registro *SAR* (*successive-approximation register*), che a sua volta controlla il *DAC*. Questo tipo di *ADC* implementa un algoritmo di ricerca dicotomica attraverso tutti i possibili livelli di quantizzazione fino alla determinazione del valore di conversione finale. È il tipo di convertitore più comune nei microprocessori.

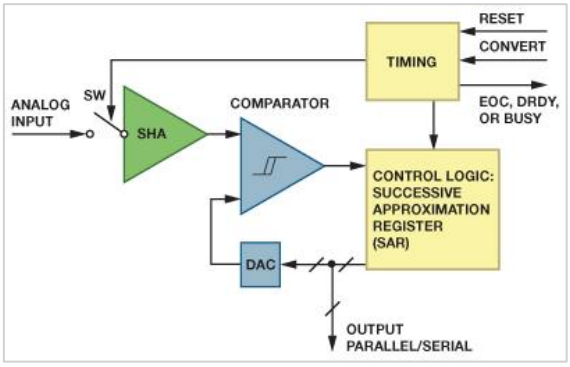

Figura 8: ADC SAR [27]

2.3.3 ADC delta-sigma ($\Delta\Sigma$)

Questo tipo di convertitore è composto da un integratore e un comparatore in retroazione negativa con un *DAC* ad 1 bit (ossia un circuito che commuta tra una tensione di riferimento positiva e una negativa). L'uscita di questa prima parte del convertitore, detta modulatore $\Delta\Sigma$, consiste in un *bit stream* la cui densità di probabilità è proporzionale all'ampiezza del segnale in ingresso, ossia si comporta come un convertitore tensione-frequenza. Il modulatore lavora in forte *oversampling*, ossia ad un multiplo della frequenza minima di campionamento che sarebbe necessaria per campionare correttamente il segnale in ingresso (frequenza di Nyquist). Il *bit stream* entra in un filtro passa-basso digitale *FIR* (*finite impulse response*), che include anche un decimatore, con il compito di rimuovere la maggior parte del rumore di quantizzazione, il quale è stato portato nelle alte frequenze dal *noise shaping* dovuto alla retroazione del modulatore. Tutto questo va a determinare l'ottima risoluzione, solitamente dai 16 ai 24 bit, degli *ADC* ΔΣ.

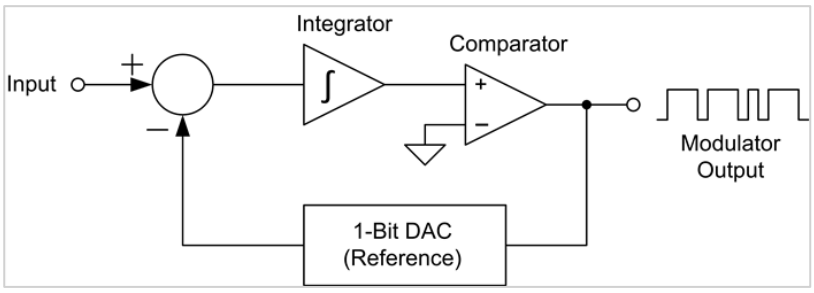

Figura 9: schema a blocchi del modulatore ΔΣ [28].

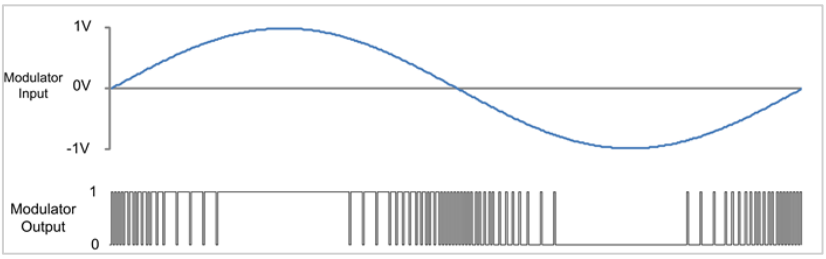

Figura 10: esempio di *bit stream* in uscita dal modulatore [28].

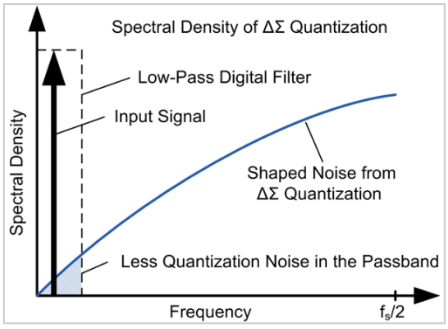

Figura 11: rumore di quantizzazione e banda del filtro *FIR* [28].

2.4 Logica di controllo del multimetro

L'uscita del convertitore analogico digitale, costituita solitamente da un *bus SPI* (*Serial Peripheral Interface*), è collegata direttamente al microprocessore o *chipset* del multimetro. Quest'ultimo ha il compito di:

- selezionare il tipo di misura in base all'input proveniente dall'interruttore rotativo e dai pulsanti,
- effettuare l'*auto ranging* se supportato,
- comandare l'acquisizione,
- leggere i dati sul *bus*,
- applicare eventuali fattori di conversione e calibrazione,
- visualizzare la misura sul *display*.

Tutte queste operazioni sono coordinate dal *firmware* del multimetro.

3 Architettura del multimetro proposto

3.1 Requisiti e funzionalità

Come anticipato, si vuole realizzare un multimetro digitale in grado di effettuare tutte le seguenti misure in contemporanea:

- Rilevamento della forma d'onda del segnale in ingresso, tra cui:
  - Sinusoidale
  - Quadra
  - Triangolare
  - Dente di sega
- Valore medio di tensione, ossia la componente *DC*
- Valore efficace di tensione (*RMS*)
- Valori minimo, massimo e picco-picco
- Frequenza e periodo

Il multimetro deve essere in grado di effettuare tutte queste misure con il minimo intervento dell'utente possibile. Si vuole avere una dinamica d'ingresso di $\pm 30\ V$. Infine, si desidera visualizzare tutti i dati su computer collegato tramite cavo USB.

3.2 Hardware

Il prototipo di multimetro è composto da due parti: un semplice front-end analogico e una scheda di sviluppo contenente un microprocessore. La scelta dei componenti è descritta a seguire.

3.2.1 Front-end analogico

Per soddisfare questi requisiti ma utilizzare il minor numero di componenti possibile, si è scelto di utilizzare un solo amplificatore operazionale e un partitore di tensione per generare una tensione di riferimento pari a metà della tensione di alimentazione del circuito. L'amplificatore operazionale (in questo caso un generico *rail-to-rail*, il *LMC6484* [16] della *Texas Instruments*), in configurazione invertente, ha il compito di scalare i $\pm 30\ V$ in ingresso nel range dell'*ADC*, ossia $0 \rightarrow 3.3\ V$. Inoltre, siccome il suo ingresso non

invertente è collegato al partitore di tensione, lo zero in ingresso viene portato a circa metà della dinamica d'uscita. Da una semplice analisi del circuito risulta:

$$V_- \cong V_+ = V_{Ref} = \frac{3.3\ V}{2}$$

$$I_{R1} = \frac{V_{IN} - V_{Ref}}{R_1} = I_{Rf} = \frac{V_{Ref} - V_{ADC}}{R_f} \quad \Rightarrow \quad (V_{IN} - V_{Ref}) \cdot R_f = V_{Ref} \cdot R_1 - V_{ADC} \cdot R_1$$

$$\Rightarrow \quad V_{ADC} = \frac{V_{Ref} \cdot (R_1 + R_f) - V_{IN} \cdot R_f}{R_1}$$

Si noti come, se $V_{Ref} = 0\ V$, $V_{ADC} = -\frac{R_f}{R_1} \cdot V_{IN}$, che coincide con la nota formula per la tensione d'uscita di un amplificatore operazionale in configurazione invertente. Facendo riferimento allo schema elettrico di figura 12, si è scelto $R_1 = 100\ k\Omega$ e $R_f = 4.7\ k\Omega$, in modo che:

$V_{IN} = -30\ V$	$\Rightarrow$	$V_{ADC} \cong 3.14\ V$
$V_{IN} = 0\ V$	$\Rightarrow$	$V_{ADC} \cong 1.73\ V$
$V_{IN} = +30\ V$	$\Rightarrow$	$V_{ADC} \cong 317.55\ mV$

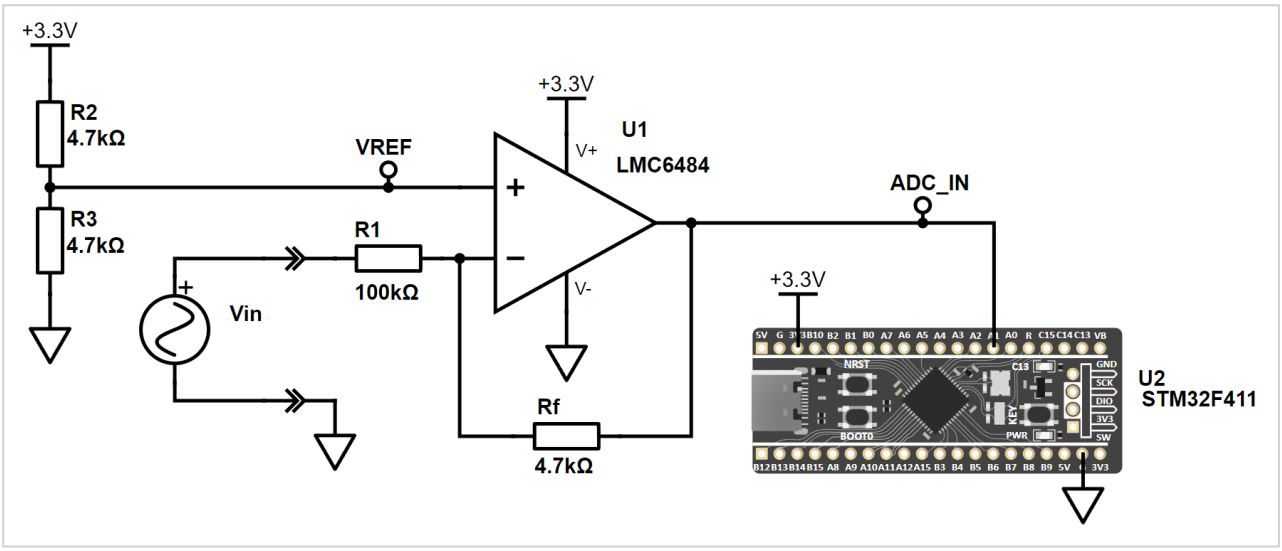

Figura 12: schema elettrico del prototipo.

Sebbene questo circuito, estremamente semplice, non presenti alcuna protezione né *auto-ranging*, né utilizzi componenti di precisione, è più che sufficiente per dimostrare il funzionamento di questo prototipo di multimetro. Possibili miglioramenti al front-end verranno discussi successivamente.

3.2.2 Microprocessore

Per supportare tutti gli algoritmi di *Digital Signal Processing* (*DSP*) che verranno utilizzati e implementare tutte le misure senza utilizzare ulteriori componenti, si è valutato che un microprocessore con le seguenti specifiche è necessario:

- *Floating Point Unit* (*FPU*), preferibilmente con estensioni *DSP*
- *Analog-to-Digital Converter* con almeno 12 bit di risoluzione
- *Direct Memory Access* (*DMA*) per trasferire i campioni dell'*ADC* direttamente in memoria senza l'intervento della *CPU*
- Abbastanza memoria *RAM* per contenere i buffer dell'*ADC* e della trasformata di Fourier
- Connettività *USB* per visualizzare i risultati delle misure su computer

Tra le maggiori *Instruction Set Architecture* (*ISA*) disponibili al giorno d'oggi, la più prominente nell'ambito dei microcontrollori è la famiglia *Cortex-M®* dell'azienda britannica *Arm*, la quale comprende diverse architetture *RISC* (*Reduced Instruction Set Computer*) a 32 bit, da varianti *low-power* ad altre più performanti. Tra queste [17], spicca l'architettura *Cortex-M4* come compromesso tra consumo energetico e prestazioni. Un esempio di microcontrollore *Cortex-M4* che soddisfa tutti i requisiti è l'*STM32F411* [18] dell'azienda europea *STMicroelectronics NV*, disponibile nella scheda di sviluppo *BlackPill* scelta per questo prototipo. Si riassumono a seguire le specifiche tecniche più importanti dell'*STM32F411*:

- Frequenza di *clock* di $100\ MHz$
- $512\ kB$ di flash, $128\ kB$ di *SRAM*
- Alimentazione da 1.7 a $3.6\ V$
- *ADC SAR* da 12 bit, $2.4\ MSPS$
- *DMA* con 16 stream
- 11 timer da 16 o 32 bit

Il primo ingresso libero dell'*ADC*, il pin *PA1*, è stato collegato direttamente dall'uscita dell'amplificatore operazionale. La configurazione del microprocessore e del suo convertitore analogico-digitale verrà discussa nei prossimi paragrafi.

4 Firmware e algoritmi di *DSP*

4.1 Configurazione dell'ambiente di sviluppo

Per poter scrivere il firmware per l'*STM32F411* è necessario installare l'ambiente di sviluppo di *STMicroelectronics*, *STM32CubeIDE*, disponibile sul sito del produttore. Per la programmazione della scheda è inoltre necessario un *debugger* e *programmer* compatibile, come l'*ST-Link*, collegato direttamente all'interfaccia *Serial Wire Debug* (*SWD*) della scheda *BlackPill*, ossia ai pin *GND*, *SCK* e *DIO*.

Il primo passo nello sviluppo del firmware, una volta aperto *STM32CubeIDE*, è stato creare un nuovo progetto per la variante *STM32F411CEU6* presente sulla scheda, selezionare l'interfaccia di debug *SWD* nella sezione *SYS* e abilitare i cristalli oscillatori esterni nella sezione *RCC* (*Reset and Clock Control*). Successivamente, si sono configurati i *Phase-Locked Loop* (*PLL*) interni nella sezione *Clock Configuration* in modo da fornire 96 MHz al core e 48 MHz al clock *USB* a partire dai 25 MHz dall'*High Speed External Oscillator* (*HSE*):

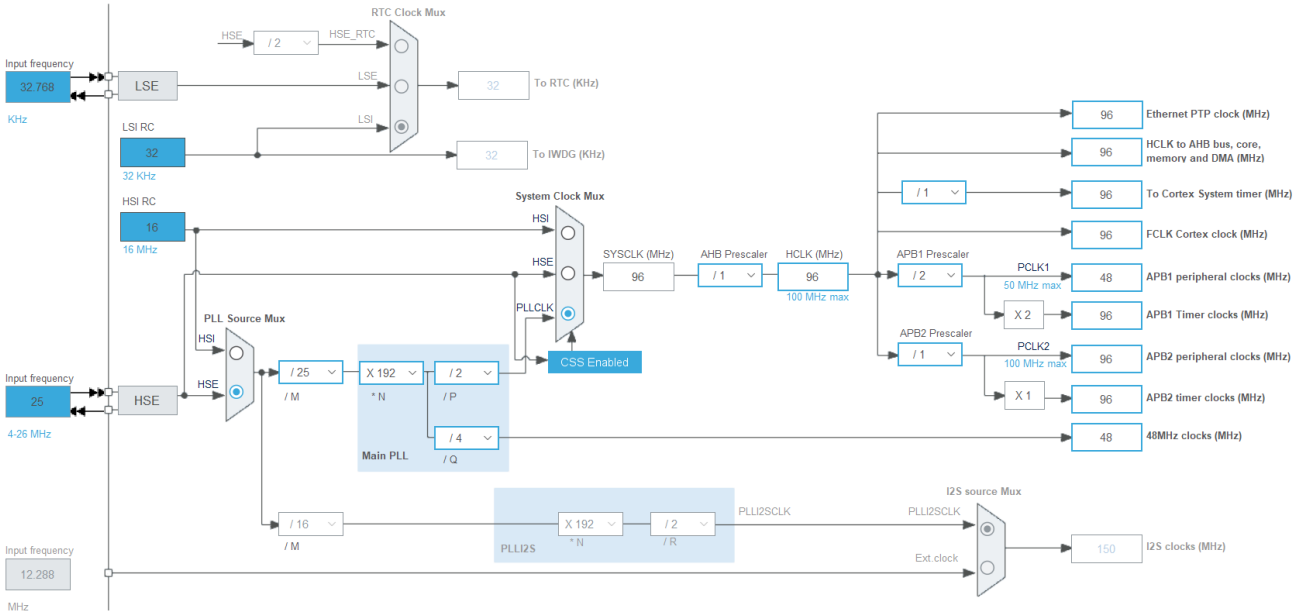

Figura 13: configurazione dei *PLL* del microprocessore.

È stato a questo punto possibile utilizzare la funzione *Generate Code* per far generare a *STM32CubeIDE* il codice minimo necessario per l'utilizzo dell'*STM32* e caricare il firmware per assicurarsi che la scheda funzioni correttamente.

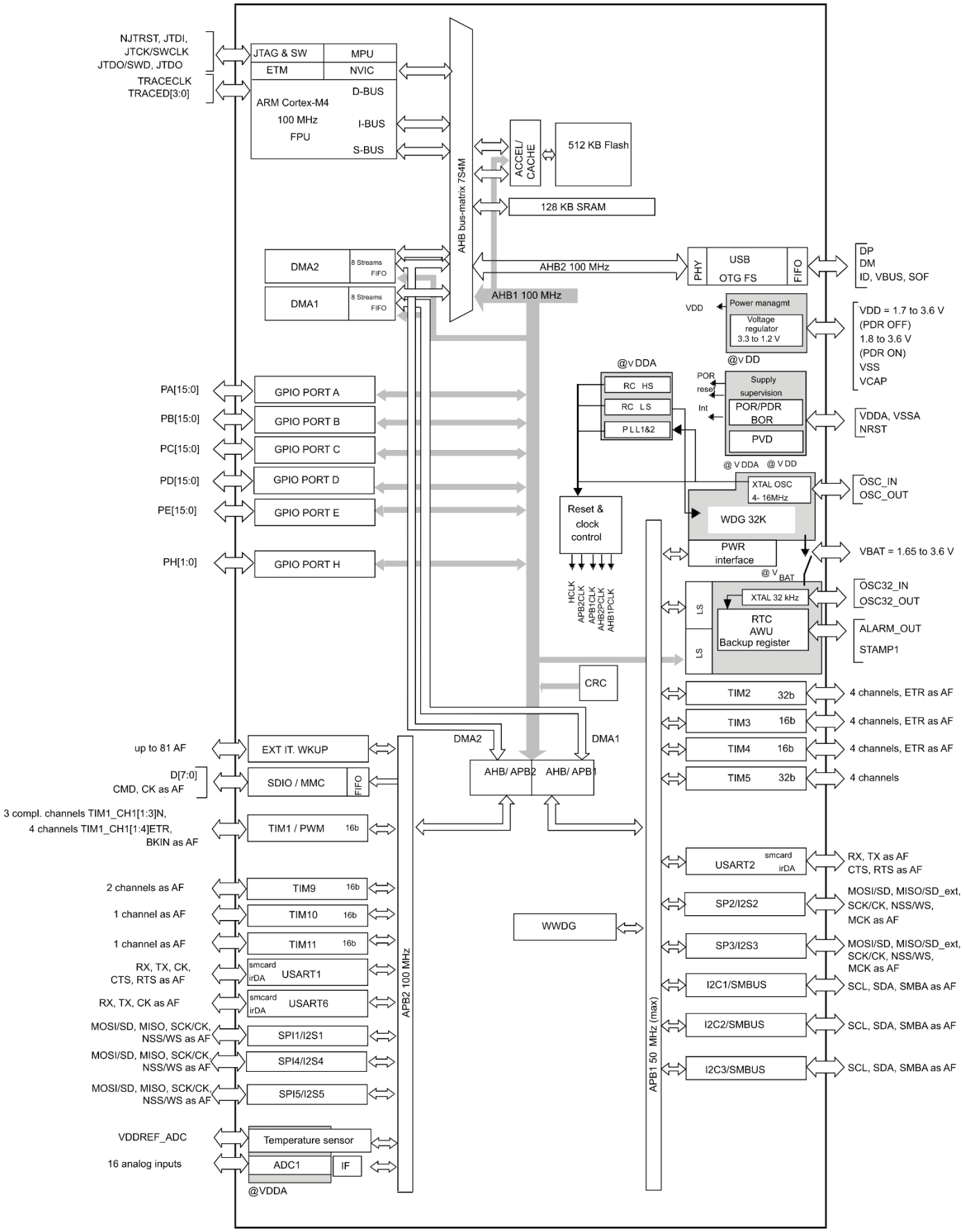

Figura 14: schema interno dell'*STM32F411* [18]. I collegamenti tra i bus interni e le periferiche saranno necessari durante la configurazione dell'*ADC*.

4.2 Configurazione di librerie e periferiche

4.2.1 Libreria *Arm® CMSIS-DSP*

Per implementare gli algoritmi di elaborazione digitale dei segnali si è scelto di utilizzare la libreria ***CMSIS DSP*** [19], parte del ***Common Microcontroller Software Interface Standard*** (*CMSIS*), un set di *API* e componenti *software* comune a tutti i microprocessori *Arm Cortex*. Per poter utilizzare la libreria è necessario compilare insieme *CMSIS 6* e *CMSIS DSP*, con determinate opzioni che permettono a *GCC* (lo *GNU Compiler*) di utilizzare istruzioni specifiche per la ***FPU*** e le operazioni ***SIMD*** (*Single Instruction/Multiple Data*).

Senza le istruzioni per operazioni a virgola mobile, il compilatore produce solamente istruzioni assembly per l'algebra intera, le quali vengono utilizzate per emulare tutte le operazioni che avrebbe invece svolto l'unità di calcolo in virgola mobile. Ogni calcolo su numeri di tipo *float(ing point)* è soggetto ad una notevole penalità in termini di velocità di calcolo se non si utilizza l'*FPU*. Confrontando l'assembly compilato a partire dalla stessa funzione di esempio, che calcola il quadrato di un numero, è evidente quale versione usa istruzioni *FPU* e quale no. Si noti la chiamata alla funzione `__aeabi_fmul` a cui è delegato il calcolo utilizzando l'algebra intera.

```
square:
  vmul.f32 s0, s0, s0
  bx lr
```

```
square:
  mov r1, r0
  push {r3, lr}
  bl __aeabi_fmul
  pop {r3, pc}
```

Le istruzioni *SIMD*, invece, sono particolari istruzioni che effettuano la stessa operazione su più dati contemporaneamente, velocizzando notevolmente i cicli e calcoli quali la trasformata di Fourier.

Le istruzioni per la compilazione di *CMSIS DSP* sono disponibili nel forum della *STMicroelectronics* [20] [21]. Tra le funzionalità della libreria che verranno utilizzate nel firmware, ci sono quelle per l'algebra complessa, la *Fast Fourier Transform* (*FFT*) e la generazione di funzioni finestra.

4.2.2 *Convertitore analogico-digitale* e *DMA*

Per effettuare conversioni con l'*ADC* del microprocessore è necessario configurare correttamente i registri della periferica e impostare le frequenze di acquisizione. Informazioni utili all'ottimizzazione dell'ADC sono reperibili nell'apposita *application note* della *STMicroelectronics* [22]. Il convertitore analogico-digitale dell'STM32 è di tipo *SAR*: la conversione è dunque divisa in *sampling* e *bit conversion*. Nella prima fase, di durata T_{SMPL}, il condensatore di campionamento viene caricato dalla tensione in ingresso, mentre nella seconda, di durata T_{SAR}, avviene l'effettiva conversione del segnale analogico in un valore digitale. La durata della conversione dipende dalla risoluzione selezionata: nel caso dell'*STM32F411*, selezionando la risoluzione massima di 12 bit, $T_{SAR} = 15\ ADC$ clock cycles. La massima *sampling rate* dell'*ADC* è data da:

$$ADC \text{ sampling rate} = \frac{1}{T_{CONV}} = \frac{1}{T_{SMPL} + T_{SAR}}$$

Supponendo di voler misurare frequenze fino a $f_{max} = 100\ kHz$, per il teorema di Nyquist è necessario campionare il segnale ad almeno $f_{Nyquist} = 2 \cdot f_{max} = 200\ kHz$. Si nota dallo schema interno del microprocessore in figura 14 che l'*ADC* è collegato al bus *APB2* (*Advanced Peripheral Bus*), il quale è stato configurato ad una frequenza $f_{APB2} = 96\ MHz$ (si veda figura 13). Tuttavia, siccome l'ADC può lavorare ad un massimo di $36\ MHz$ [18], è necessario impostare un *prescaler* di 4, quindi $f_{ADC} = 24\ MHz$.

Mettendo insieme queste informazioni, otteniamo che il valore massimo di T_{SMPL} che possiamo utilizzare è:

$$T_{CONV} = T_{SMPL} + T_{SAR} < \frac{1}{f_{Nyquist}} \quad \Rightarrow \quad \frac{15\ ADC \text{ clock cycles} + T_{SAR}}{f_{ADC}} < \frac{1}{f_{Nyquist}}$$

$$\Rightarrow T_{SMPL} < 105\ ADC\ clock\ cycles$$

Per trasferire le conversioni in memoria, è necessario configurare il *Direct Memory Access*, in modo da lasciare la *CPU* libera per l'elaborazione dei dati mentre l'*ADC* effettua le misure successive. Per fare ciò, bisogna creare un *DMA Stream* di tipo *Peripheral To Memory* e impostare la dimensione dei dati ad *Half Word*, ossia 16 bit (una *Word* è composta da 32 bit, corrispondenti alla larghezza dei *bus* dell'*Arm Cortex-M4*). Siccome si vuole avviare manualmente i cicli di conversione dell'*ADC* dopo aver

elaborato tutti i dati precedenti, è importante utilizzare la modalità normale del *DMA* (altrimenti, con la modalità circolare, le conversioni non si fermano e il buffer viene sovrascritto una volta pieno). Inoltre, dato che ogni conversione utilizza 12 su 16 bit, è importante specificare l'allineamento a destra dei dati. I bit restanti, più significativi, sono così impostati a zero.

Con il *DMA* attivo, è infine necessario abilitare un *timer hardware*, in questo caso il *timer 2*, che avvierà le singole conversioni dell'*ADC* una volta attivato. Si vuole infatti avviare manualmente solamente l'intero ciclo di acquisizione, non le singole conversioni. Gli interrupt dell'*ADC* e del *DMA* informeranno il firmware che tutte le conversioni richieste sono state effettuate e il *buffer* di dati è pieno. Gli interrupt devono essere abilitati tra le opzioni del *Nested Vectored Interrupt Controller* (*NVIC*). Concludendo, le seguenti figure mostrano le opzioni della periferica *ADC1* in *STM32CubeMX*:

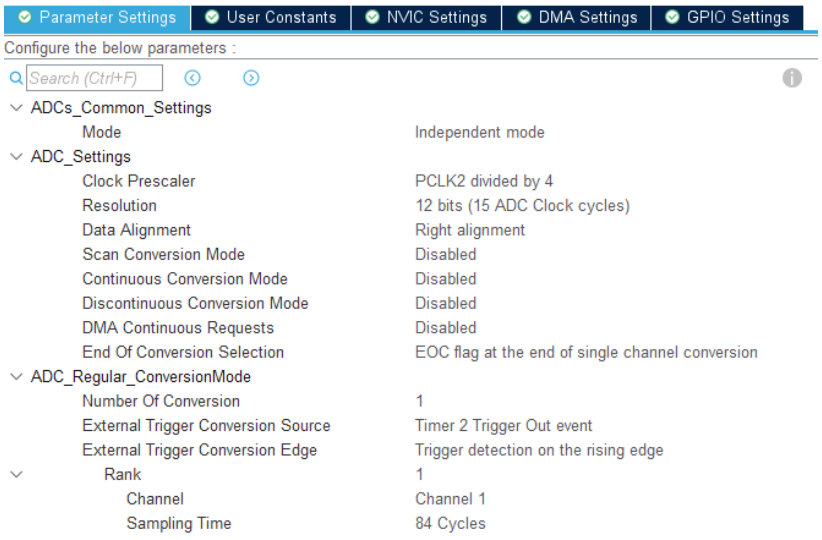

Figura 15: parametri dell'*Analog-to-Digital Converter*.

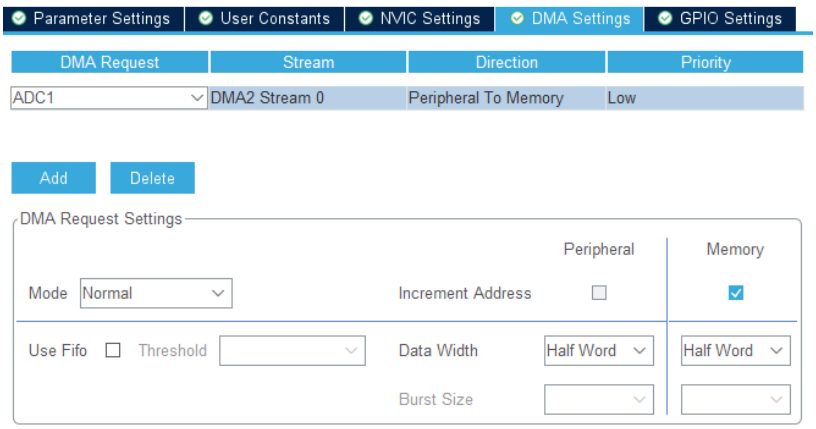

Figura 16: impostazioni del *Direct Memory Access*.

4.2.3 Timer come trigger dell'*ADC*

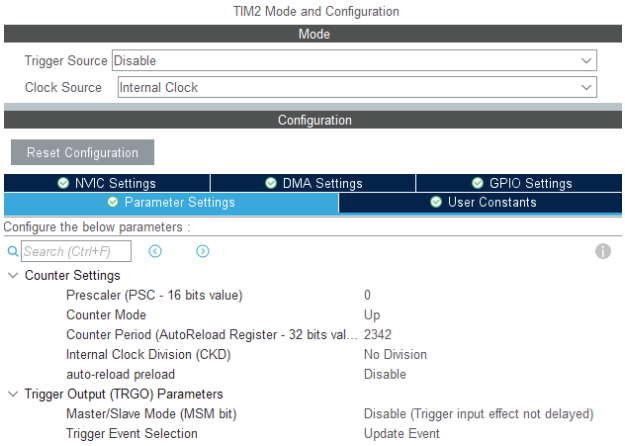

Figura 17: configurazione del timer.

Per configurare il *timer 2*, è sufficiente abilitare la *clock source* e selezionare *Update Event* come azione del trigger del timer. In questo modo, il *timer* fungerà da trigger per l'*ADC*.

L'intervallo di *tick* del timer verrà impostato dal firmware in base all'intervallo di frequenze che si vuole misurare. Per misurare frequenze elevate, verrà impostato un *tick* prossimo ai $200\ kHz$. Per frequenze basse sarà invece importante selezionare un *tick* che permette al sistema di avere più periodi del segnale in misura all'interno del *buffer*. In caso contrario, la misura sarà affetta da *aliasing* o, nella peggiore delle ipotesi, non sarà possibile.

La frequenza di aggiornamento del timer, per dei determinati *prescaler* (*PSC*) e *Auto-Reload Register* (*ARR*), è data da $f_{TICK} = \frac{f_{CLK}}{(PSC+1)\cdot(ARR+1)}$, dove f_{CLK} è la frequenza del *bus* interno al quale il *timer* è collegato. Poiché il *timer 2* si trova sul bus *APB1* a $48\ MHz$, ma è preceduto da un *PLL* che raddoppia la frequenza, si ha $f_{CLK} = 96\ MHz$. La seguente funzione *C* è stata sviluppata per calcolare e applicare nei registri dell'*STM32* sia il prescaler che l'*ARR* in modo da ottenere una determinata frequenza di campionamento dell'*ADC*:

```
void setSamplingFreq(const float f) {
    const float timFreq = ((float) HAL_RCC_GetPCLK1Freq()) * 2.0f;
    for (unsigned int psc = 0; psc < USHRT_MAX; psc++) {
        const float arrFloat = (timFreq / ((float) (psc + 1) * f)) - 1.0f;
        if ((arrFloat > 1.0f) && (arrFloat < (ULONG_MAX - 1.0f))) {
            __HAL_TIM_DISABLE(&htim2);
            __HAL_TIM_SET_COUNTER(&htim2, 0);
            __HAL_TIM_SET_PRESCALER(&htim2, psc);
            const uint32_t arr = (uint32_t) roundf(arrFloat);
            __HAL_TIM_SET_AUTORELOAD(&htim2, arr);
            __HAL_TIM_ENABLE(&htim2);
            return;
        }
    }
}
```

4.3 Algoritmi e codice *DSP*

Il seguente diagramma di flusso mostra tutti i passaggi che il *firmware* deve compiere per effettuare le misure richieste. Ogni *step* sarà analizzato in dettaglio nei prossimi paragrafi.

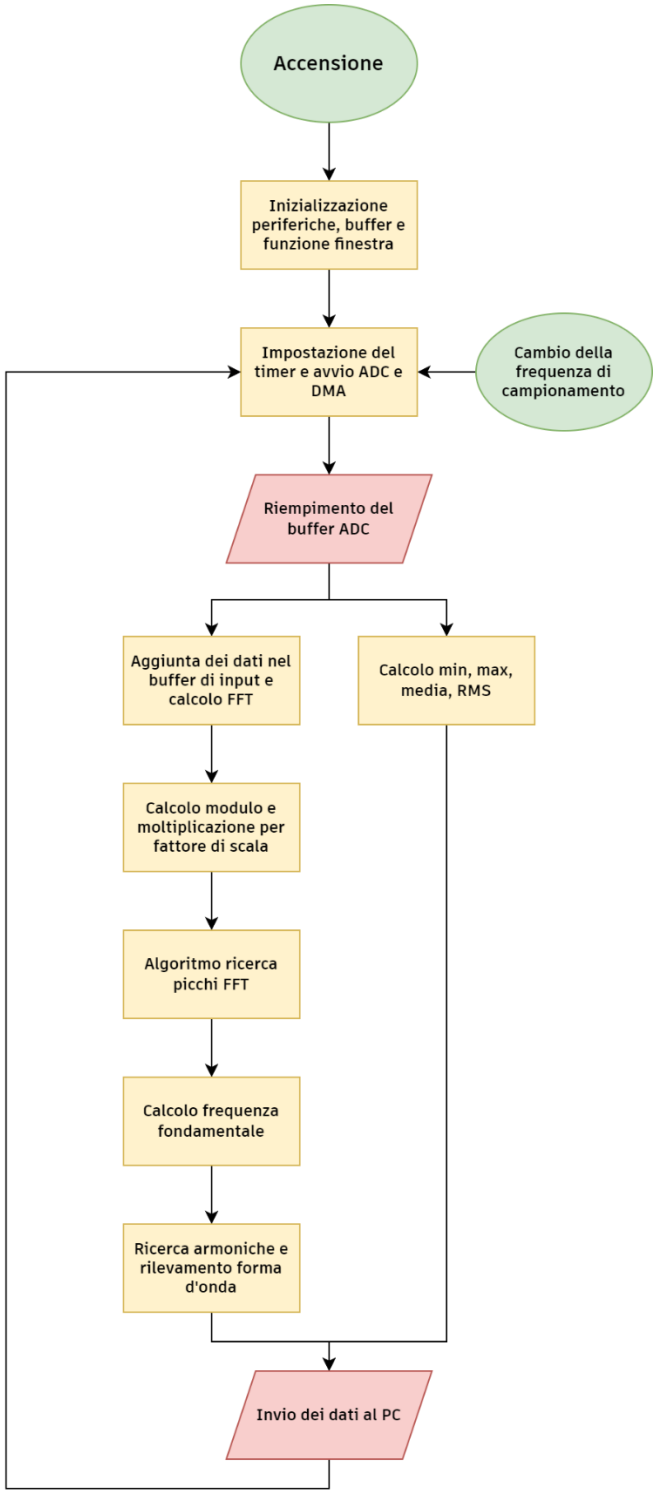

Figura 18: diagramma di flusso del *firmware*.

4.3.1 Creazione dei buffer

Il primo passo, prima di avviare il convertitore analogico-digitale, è creare tutti i *buffer* di memoria necessari:

```
volatile uint16_t adcData[4096];          // ADC data buffer
arm_rfft_fast_instance_f32 fftInst;       // FFT data structure instance
float32_t fftInputBuf[4096];              // FFT input buffer
float32_t fftMagnitudes[4096 / 2];        // FFT magnitudes (results)

void initBuffers() {
    arm_rfft_fast_init_f32(&fftInst, 4096);
    ...
}
```

È stata scelta la dimensione massima supportata per il *buffer* dell'*FFT*, ossia 4096, e quindi anche il *buffer* dell'*ADC* è stato fissato alla stessa dimensione. Quest'ultimo è stato dichiarato come *volatile*, il che segnala al compilatore che i dati al suo interno possono essere modificati fuori dal flusso principale del codice, ossia dal *Direct Memory Access*. Questo forza il *firmware* a leggere sempre i nuovi dati dell'*ADC*. Si noti inoltre che il *buffer* del risultato della trasformata di Fourier è grande la metà: la funzione di *FFT* di *CMSIS DSP* restituisce infatti un *array* contenente, in alternanza, la parte reale e la parte immaginaria per ogni punto dell'*FFT*. Queste verranno combinate in un solo numero per ogni *frequency bin* della trasformata tramite l'operazione di modulo.

4.3.2 Funzioni finestra e fattore di scala dell'*FFT*

Successivamente, è necessario inizializzare l'*array* contenente la funzione finestra scelta (*window function*, in questo esempio *Hanning*) da applicare ai dati prima di inserirli nel *buffer* di ingresso. Contestualmente, è importante calcolare il fattore di scala che, moltiplicato per il modulo di ogni punto della trasformata di Fourier, restituisce l'ampiezza (in *Volt*, se i dati in ingresso sono in *Volt*) della componente nel *frequency bin* in esame. Per la funzione di *FFT* di *CMSIS DSP*, il valore corretto è $k_1 = 2/N$, con $N = 4096$, ossia il numero di campioni scelto precedentemente. Un secondo fattore di scala, pari alla media di tutti i valori della funzione finestra, è inoltre calcolato dal firmware utilizzando la funzione di accumulo (somma) veloce di *CMSIS DSP*. Riassumendo:

$$k = k_1 \cdot k_2 = \frac{2}{N} \cdot \frac{\sum_{n=0}^{4095} \mathrm{Hann}_n}{N}$$

```
float32_t fftWindow[4096];                          // FFT window buffer
float32_t fftScaling;

void initBuffers() {
    ...
    arm_hanning_f32(fftWindow, 4096);               // Compute the Hanning window
    float32_t sum;
    arm_accumulate_f32(fftWindow, 4096, &sum);      // Sum of the window values
    fftScaling = sum * (2.0f / (4096 * 4096));      // Scaling factor
}
```

4.3.3 Avvio del ciclo di conversione

Per avviare il ciclo di conversione (e quindi *timer 2*, *DMA* e *ADC*), sono state create delle funzioni di supporto e un *callback*, ossia una funzione che viene chiamata dall'*Hardware Abstraction Layer* (*HAL*) dell'*STM32* una volta che il *DMA* ha riempito l'intero *buffer*.

```
volatile ADCStatus adcStatus = ADC_NOT_READY;

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
    // Conversion complete callback
    HAL_ADC_Stop_DMA(&hadc1);
    adcStatus = ADC_SUCCESS;
}

float setMeasurementTime(const float t) {
    const float f = 4096 / t;
    setSamplingFreq(f);
    return f;
}

void startSampling() {
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*) adcData, 4096);
    adcStatus = ADC_BUSY;
}

int main() {
    ...
    float samplingFreq = setMeasurementTime(0.1f);
    HAL_TIM_Base_Start(&htim2);

    startSampling();
    while (adcStatus == ADC_BUSY) {}
    ...
}
```

Importante è la funzione che imposta la durata dell'intero tempo di misura, ossia da quando il *buffer* è vuoto a quando viene riempito completamente. La frequenza di *sampling*, ossia quella di *tick* del timer, non è altro che la dimensione del *buffer* divisa per il tempo di misura richiesto. Se, ad esempio, si vuole misurare la frequenza della rete

elettrica, è consigliato impostare un multiplo dei periodi delle due più comuni frequenze, 50 e 60 Hz:

$$T_{TOT} = \text{mcm}\left(\frac{1}{50\ Hz}, \frac{1}{60\ Hz}\right) = \text{mcm}\left(20\ ms, \frac{50}{3}\ ms\right) = 100\ ms$$

Con $100ms$ che corrisponde a 5 periodi a 50 Hz e 6 periodi a 60 Hz.

Infine, nel ciclo principale del programma, il *main*, si attende la variabile di stato dell'*ADC* passi da *occupato* a *pronto* prima di proseguire con gli algoritmi di analisi del segnale.

4.3.4 Calcolo di *min*, *max*, media, *RMS* e *FFT*

In questa fase, il *firmware* compie tutte le seguenti operazioni in due cicli successivi:

- I dati dell'*ADC* vengono trasformati in valori di tensioni utilizzando le formule ricavate nel paragrafo "Front-end analogico", con $V_{IN,n} = ADC_n \cdot 3.3\ V/2^{12}$.
- Vengono calcolati i valori minimo, massimo, medio ed efficace (*RMS*).
- Dopo aver applicato la funzione finestra voluta, le tensioni $V_{IN,n}$ vengono copiate nel *buffer* d'ingresso per l'*FFT*, e viene chiamata l'apposita funzione di trasformata di Fourier discreta di *CMSIS DSP*.
- Per ogni numero complesso risultante della trasformata, viene calcolato il modulo, moltiplicato per il fattore di scala trovato precedentemente.

```
const unsigned int ADC_MAX_VALUE = 4096;       // = 2^(ADC_RESOLUTION)
const float ADC_VREF = 3.3f;
const float ADC_R1 = 100000.0f;
const float ADC_Rf = 4700.0f;

const float ADC_TO_VOLTAGE = (ADC_VREF / ADC_MAX_VALUE) * (-ADC_R1 / ADC_Rf);
const float ADC_OFFSET = ((ADC_VREF / 2.0f) * ADC_Rf / ADC_R1) *
                         (ADC_MAX_VALUE / ADC_VREF) +
                         (ADC_MAX_VALUE / 2.0f);

float min = FLT_MAX, max = FLT_MIN, avg = 0.0f, rms = 0.0f;
for (size_t i = 0; i < 4096; ++i) {
    const float val = (((float) adcData[i]) - ADC_OFFSET) * ADC_TO_VOLTAGE;
    if (val < min) min = val;
    if (val > max) max = val;
    avg += val;
    rms += (val * val);
    fftInputBuf[i] = fftWindow[i] * val;
}
```

```
avg /= 4096;
rms = sqrtf(rms / 4096);

float tempBuf[4096];
arm_rfft_fast_f32(&fftInst, fftInputBuf, tempBuf, 0);
for (size_t i = 0; i < (4096 / 2); ++i) {
    const float real = tempBuf[i * 2];
    const float imag = tempBuf[(i * 2) + 1];
    fftMagnitudes[i] = sqrtf((real * real) + (imag * imag)) * fftScaling;
}
```

4.3.5 Ricerca della fondamentale e delle armoniche

L'approccio scelto per il rilevamento della forma d'onda consiste nel confrontare l'ampiezza della componente fondamentale con l'ampiezza delle armoniche. Per fare ciò, il primo passo è utilizzare un algoritmo di rilevamento dei picchi (*peak detector*) nell'*array* dei moduli della trasformata di Fourier. È stato implementato un algoritmo che rileva con successo tutti i massimi locali, ad esclusione della componente *DC* a frequenza $0\ Hz$. La frequenza fondamentale è calcolata a partire dall'indice del picco più alto, utilizzando la seguente formula:

$$f = \text{index} \cdot f_{sampling} / N$$

Con $f_{sampling}$ pari alla frequenza di campionamento dell'*ADC*. $N = 4096$ è, come prima, il numero di campioni dell'*FFT*.

Per quanto riguarda le armoniche, è stata creata una funzione che ricerca, tra i picchi trovati precedentemente, quelli ad un multiplo intero n, ed effettua il confronto con l'ampiezza della fondamentale. Più chiamate a questa funzione sono state combinate per verificare se, ad esempio, la terza armonica ha un'ampiezza pari ad $1/3$ di quella della fondamentale. In tal caso, è molto probabile che il segnale in ingresso sia un'onda quadra. Sono state riportate in forma tabellare le ampiezze relative delle forme d'onda che il prototipo è in grado di rilevare.

Sono state infine create due funzioni, più semplici, che controllano se è presente un unico picco, rispettivamente, o frequenza $0\ Hz$ (quindi il segnale è *DC*) o a frequenza positiva (segnale sinusoidale puro).

Tipo di onda	Serie di Fourier	R_2	R_3
Quadra	$x(t) = \frac{4}{\pi}\sum_{k=1}^{+\infty}\frac{1}{2k-1}\sin(2\pi(2k-1)\cdot t)$	0	$1/3$
Dente di sega	$x(t) = -\frac{2}{\pi}\sum_{k=1}^{+\infty}\frac{(-1)^k}{k}\sin(2\pi k\cdot t)$	$1/2$	$1/3$
Triangolare	$x(t) = -\frac{8}{\pi^2}\sum_{k=1}^{+\infty}\frac{(-1)^k}{(2k-1)^2}\sin(2\pi(2k-1)\cdot t)$	0	$1/9$

con R_n rapporto tra l'ampiezza dell'n-esima armonica e l'ampiezza della componente fondamentale.

```
bool matchHarmonicCoeff(uint16_t fundamental, const uint16_t* peaks,
                        size_t count, uint16_t n, float coeff) {
    if (count < 2) return false;
    const float minIdx = ((float) (n * fundamental)) * (1.0f - FREQ_TOLERANCE);
    const float maxIdx = ((float) (n * fundamental)) * (1.0f + FREQ_TOLERANCE);
    const float minVal = (fftMagnitudes[fundamental] * coeff) *
                         (1.0f - AMPLITUDE_TOLERANCE);
    const float maxVal = (fftMagnitudes[fundamental] * coeff) *
                         (1.0f + AMPLITUDE_TOLERANCE);

    for (size_t i = 1; i < count; ++i) {
        const float peakIdx = peaks[i];
        if (peakIdx >= maxIdx) return false;
        if (peakIdx >= minIdx) {
            const float peakVal = fftMagnitudes[peaks[i]];
            if ((peakVal >= minVal) && (peakVal <= maxVal)) return true;
        }
    }
    return false;
}

Waveform getWaveform(uint16_t fundamental, const uint16_t* peaks, size_t count) {
    if (isDC(peaks, count)) return DC;
    if (isPureSine(fundamental, peaks, count)) return SINE;

    if (matchHarmonicCoeff(fundamental, peaks, count, 3, 1.0f / 3.0f)) {
        if (matchHarmonicCoeff(fundamental, peaks, count, 2, 1.0f / 2.0f))
            return SAWTOOTH;
        return SQUARE;
    }
    if (matchHarmonicCoeff(fundamental, peaks, count, 3, 1.0f / 9.0f))
        return TRIANGLE;
    return UNKNOWN_WAVE;
}
```

4.3.6 Trasmissione dei dati al PC

Per poter trasmettere le misure al computer, si è impostata la periferica *USB* in *STM32CubeMX* come *Device Only*, *Communication Class Device* (*CDC*). Si è poi implementato un *buffer* circolare in *C* e si è intercettato la *syscall* `_write` per reindirizzare i dati dallo *standard output* alla libreria *USB* di *STMicroelectronics* tramite il *buffer* circolare. Un altro *buffer* circolare è stato utilizzato per i *byte* ricevuti dalla *USB*. Il ciclo principale del *firmware* controlla il contenuto di quest'ultimo *buffer* per rilevare e processare eventuali comandi, tra cui ad esempio il comando di cambio del periodo di misura.

4.3.7 Debugging

Durante lo sviluppo, si è sfruttato molto il *debugging* tramite *GDB* (*GNU Debugger*) e l'*ST-Link*. Il *debugging* permette di fermare l'esecuzione del *firmware*, eseguire un'istruzione alla volta e ispezionare il contenuto delle variabili. È stato inoltre scritto uno *script Python* che, tramite *GDB*, estrae il contenuto di tutti i *buffer* del *firmware* e li trasforma in *file Excel* per controllare la correttezza dei calcoli.

Un altro *script* è stato scritto per generare dati sintetici, utilizzati al posto delle misure dell'*ADC* per effettuare prove esattamente ripetibili. Questo ha permesso di isolare e risolvere problemi con determinati segnali, e confrontare i calcoli del codice *C* con quelli di codice equivalente *Python*, linguaggio in cui l'elaborazione del segnale è più semplice.

L'ultima strategia utilizzata è stata quella di impostare un altro canale del *timer 2* del microprocessore per generare brevi impulsi ad ogni misura dell'*ADC*. Questi impulsi sono stati poi utilizzati come *trigger* per l'oscilloscopio e sovrapposti al segnale d'ingresso, permettendo di associare letture e misure errate a determinati cambiamenti nel segnale.

5 Prove e conclusioni

5.1 *Hardware* di testing

Figura 19: una foto del prototipo, in cui sono visibili l'*STM32*, l'amplificatore operazionale e il generatore di

In figura 19 è mostrato l'*hardware* del prototipo, sviluppato su *breadboard*, composto dalla scheda di sviluppo *BlackPill*, l'amplificatore operazionale, un condensatore di *decoupling* e le resistenze di *feedback*.

L'ingresso del dispositivo è connesso, in figura, ad un generatore di segnali di *test* audio, in grado di produrre onde sinusoidali, quadre, triangolari e a dente di sega a frequenze variabili da $20\,Hz$ a $20\,kHz$. Il generatore permette inoltre di variare l'ampiezza del segnale da pochi mV fino ad un massimo di $1.6\,V\,RMS$. Nel corso delle prove sono stati utilizzati anche un trasformatore con uscita a $12\,V\,AC$ e varie batterie.

L'*STM*32 è connesso, tramite *USB*, al computer per l'alimentazione e l'invio delle misure. Quando il firmware è in esecuzione, il consumo complessivo del circuito è di circa $30\,mA$, il che consentirebbe un utilizzo di almeno 4 giorni continuativi se alimentato da due pile stilo alcaline. Il costo complessivo del prototipo è inferiore a € 10,00.

5.2 Interfaccia grafica

Per semplificare l'esecuzione delle prove, una semplice interfaccia è stata sviluppata nel linguaggio di programmazione *Python*. Il *software* permette di selezionare la porta seriale e il tempo totale di campionamento, da cui il *firmware* calcola la frequenza di campionamento $f_s = N/T_{TOT}$, con $N = 4096$. In *background*, vengono letti i pacchetti dalla porta *USB* e tutte le misure vengono inserite nelle caselle di testo corrispondenti (minimo, medio, massimo, *RMS*, frequenza e periodo). Infine, viene visualizzato il tipo di forma d'onda.

5.3 Misure effettuate

Utilizzando il generatore audio, il trasformatore e le batterie, sono stati misurati numerosi segnali comprendenti tutte le forme d'onda che si è scelto al paragrafo 3.1 di voler misurare. Il prototipo è stato in grado di rilevare correttamente tutti i segnali nell'intervallo di frequenze audio e alla frequenza di rete di $50\ Hz$, e distinguere tra onde sinusoidali, quadre, triangolari, dente di sega o tensione continua.

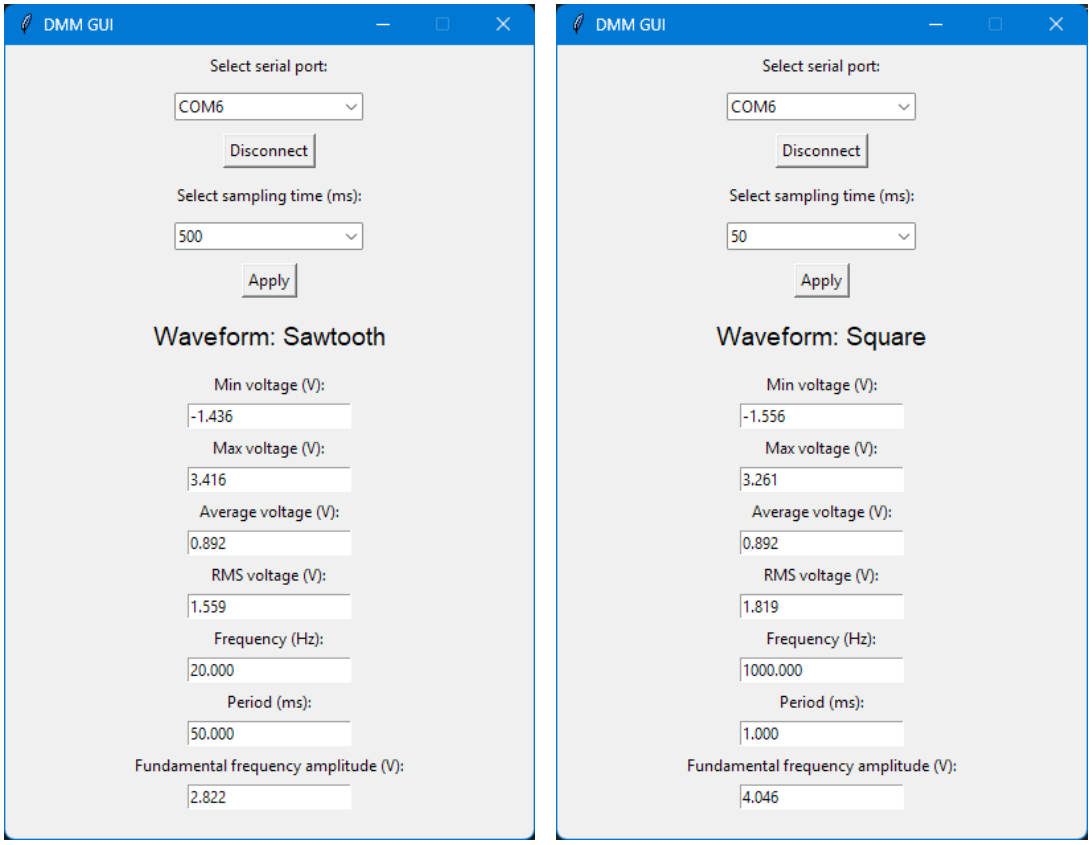

Figura 20: due screenshot dell'interfaccia grafica, connessa al prototipo, durante la misura delle forme d'onda del generatore di segnali audio.

Nel corso delle prove, i valori di tensione sono rimasti entro il $\pm 5\%$ dalle misure effettuate con multimetro e oscilloscopio, ad eccezione de i segnali di ampiezza inferiore ad $1\ V$, per i quali l'errore è aumentato a circa $\pm 10\%$ a causa della mancanza di *auto-ranging*, l'elevato rumore del semplice circuito, le capacità parassite della *breadboard* e la relativamente bassa risoluzione e precisione del convertitore analogico-digitale dell'*STM32F411*. Non si riportano le misure effettuate in quanto poco significative: è evidente la limitazione *hardware* in termini di precisione. Tuttavia, questo svantaggio dimostra le buone capacità dell'algoritmo di rilevare il segnale corretto anche in presenza di rumore. Una volta ricevuti tutti i dati dall'*ADC*, il tempo di calcolo dell'algoritmo è di solo $12\ ms$, a testimonianza della velocità della libreria *CMSIS DSP* e delle istruzioni *FPU* e *SIMD* dell'*Arm Cortex-M4*.

Per le misure di frequenza, invece, il fattore limitante è risultato essere il numero di campioni N della trasformata di Fourier discreta. Infatti, la risoluzione dell'*FFT* (ovvero la differenza tra le frequenze centrali di due *bin* successivi) è data da $R = f_S/N$. La scarsa risoluzione in frequenza è particolarmente evidente per frequenze di campionamento elevate. Un metodo comunemente utilizzato per migliorare la risoluzione della trasformata di Fourier discreta è lo *zero padding*, che consiste nell'aggiunta di zeri al segnale nel dominio del tempo per aumentarne artificialmente la durata. In questo modo, si può incrementare il numero di punti N dell'*FFT* e migliorare la risoluzione in frequenza. Nel caso di questo prototipo, tuttavia, si è limitati al massimo a 4096 punti: la libreria *CMSIS DSP* impedisce di utilizzare valori più grandi e, anche se venisse tolto il limite, passare a 8192 punti richiederebbe di riservare circa l'80% dell'intera memoria *SRAM* dell'*STM32F411* solo per gli *array* dell'*ADC* e di *output* dell'*FFT*, il che sarebbe inaccettabile. Il motivo per cui sarebbe necessario raddoppiare N, invece di scegliere un valore arbitrario, risiede nel particolare algoritmo utilizzato dalla libreria, di tipo *radix-2* (si veda l'algoritmo Cooley–Tukey per il calcolo della trasformata di Fourier [23]).

5.3.1 *Aliasing* nelle misure di frequenza

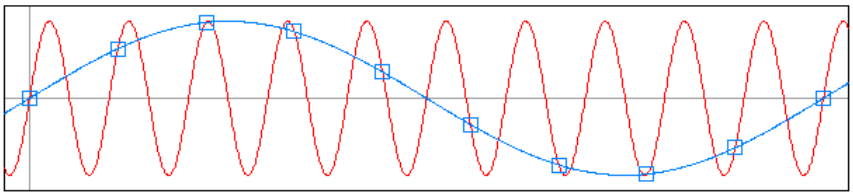

Figura 21: esempio di *aliasing* in caso di campionamento di un segnale a frequenza maggiore di $f_{Nyquist}$. Il segnale ricostruito a partire dai campioni ha una frequenza minore di quello originale.

Si è notato come il fenomeno dell'*aliasing*, o *distorsione da campionamento lento*, abbia una forte importanza in un dispositivo che dovrebbe essere in grado di misurare autonomamente un ampio spettro di frequenze. Nell'utilizzo del prototipo, è molto importante che la frequenza di campionamento impostata sia maggiore della frequenza di Nyquist per il segnale sotto misura. In caso contrario, verrà riportata una frequenza inferiore a quella reale. Possibili strategie di mitigazione saranno discusse nei paragrafi successivi.

Un fenomeno simile in risultato all'*aliasing*, ma opposto, è stato infine osservato: un campionamento troppo veloce (che normalmente comporta solo una bassa risoluzione in frequenza) impedisce una corretta misura della frequenza fondamentale se il *buffer* di input non contiene una frazione significativa del periodo del segnale. In tal caso, le armoniche inevitabilmente presenti anche in segnali sinusoidali risultano essere più ampie della componente fondamentale nell'*FFT*. Poiché il *firmware* calcola la frequenza fondamentale dal picco più alto nella trasformata di Fourier, il risultato è una frequenza misurata multipla di quella reale. Idealmente, nel *buffer* del convertitore analogico-digitale ci dovrebbe essere più di un periodo. Nella stessa situazione, anche le misure di tensione sono errate e instabili.

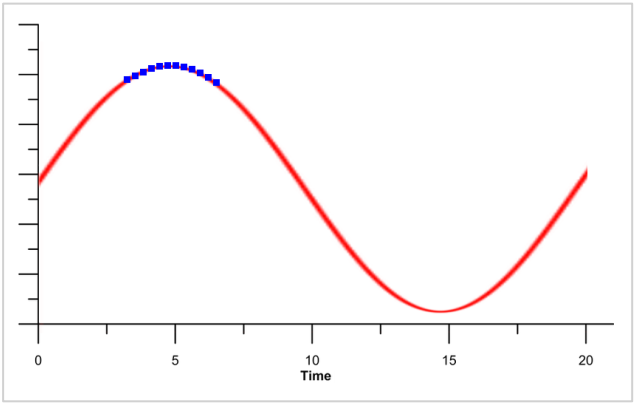

Figura 22: esempio di campionamento molto veloce con un *buffer* di dimensioni inadeguate. I campioni sono indicati in blu.

5.4 Possibili miglioramenti

5.4.1 Hardware

È evidente che uno sviluppo successivo di questo prototipo richiede un *front-end* analogico più avanzato, magari con amplificatori operazionali strumentali a basso rumore e bassa distorsione armonica, resistenze e *voltage reference* di precisione e un'impedenza d'ingresso maggiore (nel circuito attuale, è circa pari a $100\ k\Omega$, e in alcuni casi si è osservato un effetto di carico non trascurabile). Inoltre, la mancanza di protezioni in ingresso rende pericoloso l'utilizzo dello strumento nella misura della tensione di rete a $220\ V$. Per ovviare a queste mancanze, è necessario uno studio dello stato dell'arte e della letteratura sugli stadi d'ingresso dei multimetri.

In termini di facilità di utilizzo, un multimetro commerciale basato su questo progetto dovrebbe essere alimentato a batteria (preferibilmente ricaricabile a causa dell'elevato consumo del microprocessore durante i calcoli di *DSP*), e avere un *display LCD* e dei pulsanti per scorrere tra le misure e passare da misura di tensione a misura di corrente o resistenza. Questi miglioramenti garantirebbero inoltre il completo isolamento galvanico dello strumento, che allo stato attuale è collegato alla massa del computer dal quale è alimentato. La comunicazione con l'esterno potrebbe essere manutenuta con l'ausilio di una connessione ottica, soluzione già adottata da diversi multimetri commerciali di fascia alta. Oltre a ciò, l'aggiunta di *auto-ranging* sarebbe sicuramente necessaria per un multimetro moderno.

Per ridurre il consumo di batteria e migliorare le prestazioni, un possibile sostituto del microprocessore utilizzato è il nuovo *STM32U575*, dotato di *core Arm Cortex-M33* di ultima generazione, *ultra-low power*, sviluppato specificatamente per applicazioni performanti a batteria. Esso dispone inoltre di un *ADC* più performante da 14 bit, comparatori e amplificatori operazionali integrati e più memoria *SRAM* e *flash*. Non è stato utilizzato nel prototipo a causa del maggior costo della scheda di sviluppo e della maggiore complessità di programmazione. In alternativa all'*ADC* interno, si potrebbe utilizzare un convertitore esterno su *bus* I^2C o *SPI*, opportunamente configurato in *Direct Memory Access*.

Al fine di risolvere il problema dell'*aliasing* e della necessità di selezionare manualmente il tempo totale di campionamento, infine, si potrebbe implementare un semplice frequenzimetro utilizzando un comparatore e un *interrupt* del microprocessore. Sarebbe così possibile calcolare più accuratamente la frequenza fondamentale del segnale e impostare la frequenza di campionamento ad un multiplo di quest'ultima per rispettare il teorema del campionamento di Nyquist. L'unica difficoltà tecnica sarebbe implementare un frequenzimetro in grado di funzionare anche con una componente *DC* sovrapposta al segnale, magari con la soglia del comparatore programmabile dal microprocessore stesso in funzione del valore medio della tensione in ingresso.

5.4.2 Firmware

Sebbene il *firmware* si sia rivelato sufficientemente robusto, è presente ancora margine di ottimizzazione: ad esempio, si potrebbero unificare il fattore di scala dei dati dell'*ADC* e quello dei punti dell'*FFT*, riducendo il numero di moltiplicazioni a virgola mobile. L'algoritmo di rilevamento della forma d'onda potrebbe essere modificato in modo da testare tutte le possibilità in un unico ciclo. Si potrebbe inoltre tentare di processare la trasformata di Fourier con numeri interi e valutare se è possibile ridurre il tempo di calcolo e aumentare la precisione (i calcoli *floating point* sono soggetti ad errori di arrotondamento).

Per risolvere il problema della dimensione limitata dei *buffer* e ridurre l'*aliasing*, si potrebbe dedicare un'intera sezione della memoria *SRAM* del microprocessore ai dati dell'*ADC* e mantenere le conversioni sempre attive alla massima frequenza di campionamento. Si potrebbe a quel punto adoperare una decimazione digitale, di entità variabile a seconda della frequenza del segnale misurata con l'*interrupt*, per avere sempre diversi periodi del segnale nel *buffer* di ingresso dell'*FFT*. A causa di limitazioni interne a *CMSIS DSP*, il numero massimo di campioni per la trasformata di Fourier è fissato a 4096.

Infine, una volta aggiunto il *display*, sarebbe inoltre utile nascondere i valori non rilevanti, ad esempio non mostrando la frequenza se il *firmware* ha rilevato un segnale continuo.

5.5 Conclusione

Si è dimostrata con successo la realizzabilità e la funzionalità di un multimetro in grado di rilevare la forma d'onda di un segnale, e al contempo misurare tutti i parametri più importanti dello stesso. Si è inoltre mostrato come, anche con semplici microcontrollori di classe *mainstream*, è possibile implementare efficacemente complessi algoritmi di elaborazione digitale dei segnali, che fino a pochi anni fa erano accessibili solo ai computer. Il largo margine di prestazioni disponibile, opportunatamente sfruttato, apre le porte all'utilizzo di questi microprocessori anche in sistemi di acquisizione e monitoraggio, ad esempio con analisi frequenziale in tempo reale.

In termini personali, il lavoro svolto mi ha permesso di acquisire famigliarità con l'elaborazione dei segnali in sistemi con poche risorse *hardware* e un linguaggio di programmazione di basso livello, in un panorama *software* sempre più dipendente da astrazioni e grandi librerie ad alto livello che non permettono di sfruttare a pieno le prestazioni dei moderni processori. Mi ha inoltre consentito di mettere alla prova le mie competenze nel mondo *embedded* e dell'elettronica digitale.

Il progetto con tutto il codice sorgente del *firmware* è disponibile online [24] con licenza *open-source*.

6 Riferimenti

[1] Fluke Corporation (2004), *87 V/AN Service Manual*. https://xdevs.com/doc/Fluke/87V/Fluke_87V_Full_service_manual_with_schematic_and_electrical_parts_list_Service_Manual-87VANFullService.pdf

[2] Wikipedia, *Valore efficace*. https://it.wikipedia.org/wiki/Valore_efficace

[3] Rod Elliott (2017), *Peak Detection Circuits*. https://sound-au.com/appnotes/an014.htm

[4] Fluke Corporation (1977), *8020A Instruction Manual*. https://xdevs.com/doc/Fluke/FLUKE%208020A%20Instruction.pdf

[5] Youtuuba (2022), *Fluke 8020A Handheld Digital Multimeter: History, Overview, Demonstration, Theory.* https://www.youtube.com/watch?v=La7s1zZFbMA

[6] *Fluke Corporation*, The history of the multimeter. https://www.fluke.com/en/learn/blog/digital-multimeters/multimeter-history

[7] Burr-Brown Corporation, ora Texas Instruments (1997), *Precision absolute value circuits*. https://www.ti.com/lit/an/sboa068/sboa068.pdf

[8] Rod Elliott (2021), *Precision Rectifiers*. https://sound-au.com/appnotes/an001.htm

[9] Wellington Institute of Technology (2020), *RMS, Peak, and Peak-to-Peak Conversions*. https://moodle.weltec.ac.nz/mod/book/view.php?id=244070&chapterid=51920

[10] Fluke Corporation, *What is true-RMS?* https://www.fluke.com/en/learn/blog/electrical/what-is-true-rms

[11] Analog Devices, Inc. (2015), *AD737 Datasheet*. https://www.analog.com/media/en/technical-documentation/data-sheets/ad737.pdf

[12] Analog Devices, Inc. (2009), *RMS-to-DC Converters*. https://www.analog.com/media/en/training-seminars/tutorials/mt-081.pdf

[13] Analog Devices, Inc. (2009), *Analog Multipliers*. https://www.analog.com/media/en/training-seminars/tutorials/MT-079.pdf

[14] w2aew (2015), *Basics of the Gilbert Cell*. https://www.youtube.com/watch?v=7nmmb0pqTU0

[15] EEVblog (2022), *Convert a Fluke 77 IV to True RMS for 10 CENTS!* https://www.youtube.com/watch?v=X_1M3xgl4zo

[16] Texas Instruments, *LMC6484 CMOS Quad Rail-to-Rail Input and Output Operational Amplifier*. https://www.ti.com/lit/ds/symlink/lmc6484.pdf

[17] Arm, *Arm Cortex-M Processor Comparison Table*. https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Cortex-A%20R%20M%20datasheets/Arm%20Cortex-M%20Comparison%20Table_v3.pdf

[18] STMicroelectronics, *STM32F411 datasheet*. https://www.st.com/resource/en/datasheet/stm32f411ce.pdf

[19] Arm, *CMSIS DSP software library documentation*. https://arm-software.github.io/CMSIS-DSP/latest/index.html

[20] STMicroelectronics Community, *Configuring DSP libraries on STM32CubeIDE*. https://community.st.com/t5/stm32-mcus/configuring-dsp-libraries-on-stm32cubeide/ta-p/49637

[21] STMicroelectronics Community, *How to use CMSIS-DSP now that there are no precompiled libs?* https://community.st.com/t5/stm32cubeide-mcus/how-to-use-cmsis-dsp-now-that-there-are-no-precompiled-libs-in/m-p/637229/highlight/true#M24142

[22] STMicroelectronics, *How to optimize the ADC accuracy in the STM32 MCUs*. https://www.st.com/resource/en/application_note/an2834-how-to-optimize-the-adc-accuracy-in-the-stm32-mcus-stmicroelectronics.pdf

[23] Wikipedia, *Cooley–Tukey FFT algorithm*. https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

[24] Marco Cipriani (2024), Tesi di Laurea. https://github.com/marcocipriani01/Bachelor-Thesis

[25] EEVblog forum (2010), *What do you think of the new Fluke 87 V MAX?* https://www.eevblog.com/forum/testgear/what-do-you-think-of-the-new-fluke-87-v-max/

[26] Learning about Electronics (2018), *How to Build a Peak Precision Detector Circuit*. https://www.learningaboutelectronics.com/Articles/Precision-peak-detector-circuit.php

[27] Analog Devices, Inc. (2013), *Successive-Approximation ADCs*. https://www.analog.com/en/resources/analog-dialogue/articles/successive-approximation-adcs.html

[28] Texas Instruments (2015), *Understanding the delta-sigma modulator*. https://web.archive.org/web/20240411161741/https://e2e.ti.com/blogs_/archives/b/precisionhub/posts/delta-sigma-adc-basics-understanding-the-delta-sigma-modulator