Politecnico di Torino

Dipartimento di ELETTRONICA e TELECOMUNICAZIONI (DET)
Corso di Laurea Magistrale in Ingegneria Elettronica

Sviluppo di una scheda elettronica digitale nell'ambito della sicurezza industriale

Tesi sperimentale redatta per il conseguimento della Laurea Magistrale in Ingegneria Elettronica, in cooperazione con D.M.D. Computers s.r.l.

Relatore
Prof. Massimo Ruo Roch

Laureando
Simone Ceglia s287852

Anno Accademico 2022 - 2023

Dichiarazione di paternità

Io, Simone Ceglia, dichiaro che questa tesi intitolata, "Sviluppo di una scheda elettronica digitale nell'ambito della sicurezza industriale" e il lavoro in essa contenuto sono miei. Inoltre confermo che:

- questo lavoro è stato svolto interamente o principalmente durante la candidatura per il titolo di Laurea Magistrale presso questa Università;
- dove ho consultato il lavoro pubblicato da altri, questo è sempre chiaramente specificato;
- dove ho citato il lavoro di altri, la fonte è sempre indicata;
- ho riconosciuto tutte le principali fonti di aiuto;
- laddove la tesi si basa su un lavoro svolto da me stesso insieme ad altri, ho chiarito esattamente cosa è stato fatto dagli altri e a cosa ho contribuito io stesso.

Firma: *Simone Ceglia*

Data: *05/04/2023*

Alla mia famiglia, che crede e ha sempre creduto in me.

Abstract

Lo sviluppo dell'economia ha da sempre richiesto un incremento del ritmo di produzione in ambito industriale per soddisfare la crescente domanda di mercato. La nascita di macchine utensili ha permesso di velocizzare la quasi totalità dei processi produttivi garantendo nel contempo un livello di precisione impareggiabile se confrontato al lavoro manuale. Al fine di marginare il verificarsi di incidenti in sede di lavoro, tali macchine sono state dotate di sistemi di sicurezza, generalmente passivi, il cui uso e posizionamento sono subordinati alla responsabilità dell'operatore. L'obiettivo della tesi consta nella realizzazione di una scheda elettronica digitale che si interfaccia con una macchina utensile preesistente al fine di fornire un livello di sicurezza attivo, ovvero, che prevenga l'incidente disattivando automaticamente l'utensile in caso di mancanza da parte dell'operatore.

Il dispositivo è basato sul TriCore™ di Aurix, un microcontrollore che, nonostante sia stato concepito per applicazioni automotive, viene impiegato nella sicurezza industriale per i suoi alti standard di sicurezza. Il sistema è in grado di identificare la presenza di una qualsiasi parte dell'operatore in una zona della macchina ritenuta pericolosa tramite l'analisi in frequenza dei segnali interposti tra trasmettitori e ricevitori. Alla descrizione del sistema segue uno studio approfondito sul microcontrollore. In particolare, si analizzano i moduli del timer e del convertitore analogico-digitale rispettivamente coinvolti nel funzionamento di trasmettitori e ricevitori. Si determinano le impostazioni compatibili con le specifiche che il sistema deve essere in grado di soddisfare e si procede con la stesura del firmware, il codice che consente di configurare le periferiche del microcontrollore permettendo il loro funzionamento al livello fisico. Conclude una fase di sviluppo hardware delle schede di trasmissione e ricezione, le quali consentono la realizzazione di un primo prototipo e la sua conseguente validazione attraverso la realizzazione di un applicativo di test e la successiva analisi dei risultati ottenuti.

Indice

Lista delle abbreviazioni

ADC	Analog to Digital Converter.
ANCFG	Analog Function Configuration Register.
BJT	Bipolar Junction Transistor.
CAD	Computer Aided Design.
CCU	Counter Compare Unit.
CPU	Central Processing Unit.
DAC	Digital to Analog Converter.
DFT	Discrete Fourier Transform.
DMA	Direct Memory Access.
EDSADC	Enhanced Delta Sigma ADC.
EOC	End Of Conversion.
EVADC	Enhanced Versatile ADC.
FET	Field Effect Transistor.
FFT	Fast Fourier Transform.
FIFO	First In First Out.
FR4	Flame Resistant 4.
GND	Ground.
GPIO	General Purpose Input Output.
GTM	Generic Timer Module.
ICLASS	Input Class Register.
IDE	Integrated Development Environment.
IFX	Infineon Technologies AG.
ILLD	Infineon Low Level Driver.
MCU	Micro Controller Unit.
MUX	Multiplexer.
OP-AMP	Operational Amplifier.
PCB	Printed Circuit Board.
PTH	Pin Through Hole.
PWM	Pulse Width Modulation.
Q0R	Queue 0 Register.
QCTRL	Queue Source Control Register.
QINR	Queue Input Register.

QMR	Queue Mode Register.
QRS	Queued Request Source.
RAM	Random Access Memory.
REQTM	Request Timer Mode Register.
ROM	Read Only Memory.
RT	Request Timer.
RX	Ricevitore.
S/H	Sample and Hold.
SAR	Successive Approximation Register.
SEQTIM	Sequence Timer.
SMD	Surface Mounting Device.
SOU	Signal Output Unit.
TOM	Timer Output Module.
TX	Trasmettitore.

Capitolo 1

Introduzione

1.1 Motivazione dello studio

Lo sviluppo dell'economia ha da sempre richiesto un incremento del ritmo di produzione in ambito industriale per soddisfare la crescente domanda di mercato. La nascita di macchine utensili ha permesso di velocizzare la quasi totalità dei processi produttivi garantendo nel contempo un livello di precisione impareggiabile se confrontato al lavoro manuale. Al fine di marginare il verificarsi di incidenti in sede di lavoro, tali macchine sono state dotate di sistemi di sicurezza, generalmente passivi, il cui uso e posizionamento sono subordinati alla responsabilità dell'operatore.

Di inestimabile importanza è la realizzazione di una scheda elettronica digitale che si interfaccia con una macchina utensile preesistente al fine di fornire un livello di sicurezza attivo, ovvero, che prevenga l'incidente disattivando automaticamente l'utensile in caso di mancanza da parte dell'operatore.

1.2 L'oggetto della tesi

Lo studio di questa tesi affonda le sue radici in un principio di funzionamento ideato dell'azienda ospitante D.M.D. Computers s.r.l. circa la rilevazione di una qualsiasi parte del corpo dell'operatore in una zona ritenuta pericolosa di una macchina utensile ad avanzamento manuale. A seguito di uno studio di fattibilità eseguito da parte dell'azienda nel periodo antecedente a questo lavoro, è stata elaborata una una bozza del dispositivo il cui schema di principio è illustrato di seguito.

Come è possibile osservare dalla figura 1.1, la macchina utensile è caratterizzata da un'area di lavoro (tratteggiata in nero) alla quale possono liberamente accedere l'operatore e i materiali da lavoro, e da una zona di pericolo (tratteggiata in rosso) dove è prevista la sola presenza di materiali da lavoro.

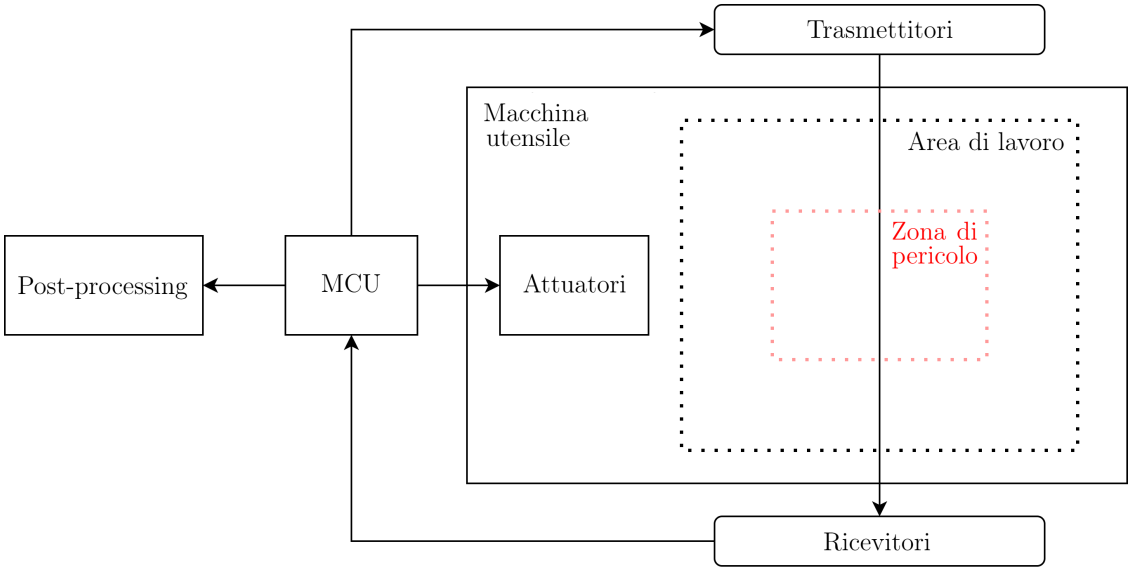


Figura 1.1: Diagramma del sistema

Il sistema è gestito da un microcontrollore (MCU) il quale pilota sensori ed attuatori. I primi sono responsabili del sorvegliamento della zona di pericolo e in caso di rilevamento di una qualsiasi parte dell'operatore i secondi sono in grado di disabilitare automaticamente l'utensile. I sensori, o più correttamente trasduttori, alloggiano su due schede elettroniche le quali vengono poste l'una al di sopra dell'altra (rispettivamente trasmettitori in alto e ricevitori in basso) e sono separati da un dielettrico (aria) il quale costituisce il canale di trasmissione dei segnali. Una terza unità è in grado di elaborare dati per applicazioni di statistiche real-time e di diagnostica.

1.3 Obiettivi del lavoro

Il principio fisico secondo il quale il sistema è in grado di riconoscere la presenza dell'operatore nei pressi dell'utensile non rientra nel progetto di tesi ed è protetto da accordi di non divulgazione. Ciononostante, nella fase di validazione verranno comunque fornite tutte le informazioni sufficienti e necessarie per la corretta interpretazione dei risultati ottenuti.

L'obiettivo della tesi si focalizza, invece, sulla realizzazione di un primo prototipo del sistema, semplificato rispetto alla versione finale, che possa confermare il corretto funzionamento del principio fisico precedentemente concepito. In particolare:

- si ricercheranno le **funzionalità** del microcontrollore e le relative **configurazioni** dei moduli che gestiscono le schede di trasmissione e ricezione;

› si svilupperà il **firmware** (driver dei moduli) per implementare le configurazioni trovate al punto precedente;

› si svilupperà l'**hardware** (schede di trasmissione e ricezione) per la realizzazione di un primo prototipo;

› si validerà il **prototipo** tramite l'analisi dei dati ottenuti.

1.4 Struttura dell'elaborato

Analogamente alla progettazione di dispositivo elettronico, la dissertazione segue un flusso di progetto le cui fasi vengono divise in capitoli per facilitare la comprensione al lettore. Il capitolo 2 ha l'obiettivo di introdurre il lettore alle tecniche e ai dispositivi attualmente utilizzati per la realizzazione di sistemi elettronici. Il capitolo 3 presenta le analisi preliminari che vengono effettuate sul progetto precedenti all'implementazione. Si indaga sulle funzionalità che il dispositivo deve essere in grado di fornire e le specifiche che derivano da quest'ultime. Segue un'analisi dettagliata sulle periferiche del microcontrollore utilizzato per la realizzazione del dispositivo. Il capitolo 4 contiene l'implementazione delle configurazione ricercate nel capitolo precedente la quale si traduce nella stesura del firmware del sistema. Il capitolo 5 espone la progettazione di due schede elettroniche, rispettivamente dei trasmettitori e dei ricevitori, le quali consentono la realizzazione di un primo prototipo. Il capitolo 6 conduce alla validazione del sistema attraverso l'esecuzione di alcuni test che portano ai primi risultati desiderati in fase prototipale. Infine, il capitolo 7 riassumerà gli obiettivi raggiunti dell'attività di tesi e offrirà una prospettiva sul futuro sviluppo del dispositivo.

1.5 Notazioni

Le seguenti notazioni saranno utilizzate:

› ***sans serif*** - carattere grassetto corsivo senza grazie per i nomi dei file;

› **`typewriter`** - carattere grassetto monospaziato per il codice;

› (l.x) - per un riferimento alla linea x del codice sottostante;

› (l.x → l.y) - per un riferimento dalla linea x alla linea y del codice sottostante.

› i campi dei registri negli schemi saranno riportati in colorazione verde se modificati manualmente e in colorazione gialla se modificati da una funzione predefinita degli iLLD. I campi non modificati compariranno su sfondo bianco.

Capitolo 2

Stato dell'arte

È il capitolo per introdurre ai lettori meno esperti i concetti, i dispositivi e le tecniche attualmente utilizzati per la realizzazione di sistemi elettronici. A tale scopo si attingerà materiale da fonti esterne le quali saranno regolarmente citate nelle note a piè di pagina e visibili nella bibliografia.

2.1 L'elettronica digitale

I fenomeni del mondo reale sono caratterizzati da grandezze fisiche di diversa natura che seguendo le leggi della fisica interagiscono tra di loro assumendo determinati valori nel tempo. La variazione di tali grandezze contiene intrinsecamente una certa informazione e viene astratta dal concetto di segnale.

In genere, le grandezze osservabili in natura rispettano un andamento continuo o graduale nel tempo: è il caso dei **segnali analogici** i quali possono assumere infiniti valori reali in ogni istante di tempo, dunque, vengono modellizzati da funzioni continue nel tempo e nelle ampiezze. A questi si contrappongono i **segnali discreti** i quali sono definiti, a partire da un segnale analogico, solo in determinati istanti di tempo (discretizzazione del tempo) per mezzo di un'operazione chiamata **campionamento**. Una particolare categoria di segnali discreti è costituita dai **segnali digitali** (dall'inglese "digit", in italiano "cifra") i quali possono assumere soltanto una serie di valori predefiniti e sono ottenuti a partire da un segnale discreto per mezzo di un'operazione chiamata **quantizzazione**.

La cui continuità nel tempo e nelle ampiezze dei segnali analogici consente un'accurata rappresentazione della realtà. Ciononostante, una serie di motivazioni hanno portato alla diffusione sempre più capillare dei segnali digitali e allo sviluppo della disciplina che li governa, ovvero, l'elettronica digitale.

2.1.1 Vantaggi del digitale[1]

Rispetto agli apparati analogici i vantaggi dei sistemi digitali sono così definiti:

- miglior tolleranza al rumore e ad altri disturbi elettromagnetici;
- possibilità di integrare in un unico sistema di trasmissione l'invio di informazioni di diversa tipologia (video, audio, dati, ...);
- maggiore efficienza e flessibilità dei sistemi;
- elaborazione del segnale in tempo reale;
- semplicità di memorizzazione dei dati;
- costo ridotto dei sistemi.

Tutti questi vantaggi costituiscono le motivazioni dello sviluppo degli attuali dispositivi elettronici digitali ivi compreso il sistema oggetto di questa tesi. Infatti quest'ultimo, essendo un apparecchio incentrato sulla sicurezza, deve poter garantire:

- il funzionamento indipendentemente da fonti esterne di disturbo;
- l'elaborazione di segnali in tempo reale per la discriminazione di una situazione di pericolo;
- un costo contenuto per essere diffuso in larga scala sul mercato;

in riferimento alle caratteristiche enunciate nella suddetta lista.

[1] **La stesura della sezione trae ispirazione dalla fonte [8]**

2.2 Microcontrollore[2]

Un **microcontrollore** o **MCU**, dall'inglese Microcontroller Unit, è un dispositivo elettronico integrato su un singolo supporto (**chip**), nato come evoluzione alternativa al microprocessore e utilizzato generalmente in sistemi dedicati (**embedded**) per applicazioni di controllo digitale.

Il microprocessore integra sul chip solo la logica di elaborazione delle istruzioni e si affida a delle unità esterne quali memorie e dispositivi periferici per poter scambiare informazioni e interagire con l'esterno. Questa architettura è tipica dei sistemi general purpose ed è in grado di raggiungere velocità di elaborazione molto elevate seppur consumando molta energia e occupando un notevole spazio.

Il microcontrollore è invece un sistema completo, che integra in uno stesso chip il processore (**CPU**), la memoria permanente (**ROM**), la memoria volatile (**RAM**) e i pin generici di input/output (**GPIO**), oltre ad eventuali altri blocchi specializzati, riducendo eventualmente a zero il numero di componenti esterni, rendendo il sistema più compatto e di facile realizzazione. Queste caratteristiche sono raggiunte a scapito della potenza di calcolo che tuttavia, nella maggior parte delle applicazioni embedded, non è richiesta. Al contrario, queste ultime privilegiano il risparmio di potenza che permette ai microcontrollori di essere implementati in sistemi alimentati a batteria. Si evince che il microcontrollore è appositamente progettato per essere funzionalmente autosufficiente, ottimizzando nel contempo il costo del dispositivo rispetto alle sue prestazioni in un determinato ambito.

Come anticipato precedentemente, un microcontrollore implementa al suo interno blocchi specializzati che permettono di interfacciarsi con il mondo esterno. All'interno del progetto di tesi particolare attenzione verrà rivolta al convertitore analogico-digitale e alle unità di temporizzazione.

2.2.1 Convertitore analogico-digitale

Il **convertitore analogico-digitale** o **ADC**, dall'inglese Analog to Digital Converter, è un blocco fondamentale nelle applicazioni di controllo digitale. L'ADC costituisce l'interfaccia tra il mondo reale, rappresentato da segnali analogici (continui nel tempo e nelle ampiezze) e il mondo digitale caratterizzato dai segnali discreti (discontinui nel tempo e nelle ampiezze). Un segnale digitale è ottenuto a partire da un segnale analogico per mezzo di un processo chiamato campionamento

[2] **La stesura della sezione trae ispirazione dalla fonte [10]**

il quale verrà descritto nella sezione 2.5.1. Ogni ADC, nonostante possa essere implementato differentemente in base al principio di funzionamento, è caratterizzato da due parametri fondamentali:

› una tensione di alimentazione V_{AL};

› il numero di bit di risoluzione N_{bit};

i quali, come è enunciato nella sezione 2.5.2, definiscono alcune proprietà dei convertitori analogico digitali. L'attenzione è riposta sul **convertitore ad approssimazioni successive** o **SAR**, dall'inglese Successive Approximation Register, il cui schema a blocchi è visibile in figura 2.1.

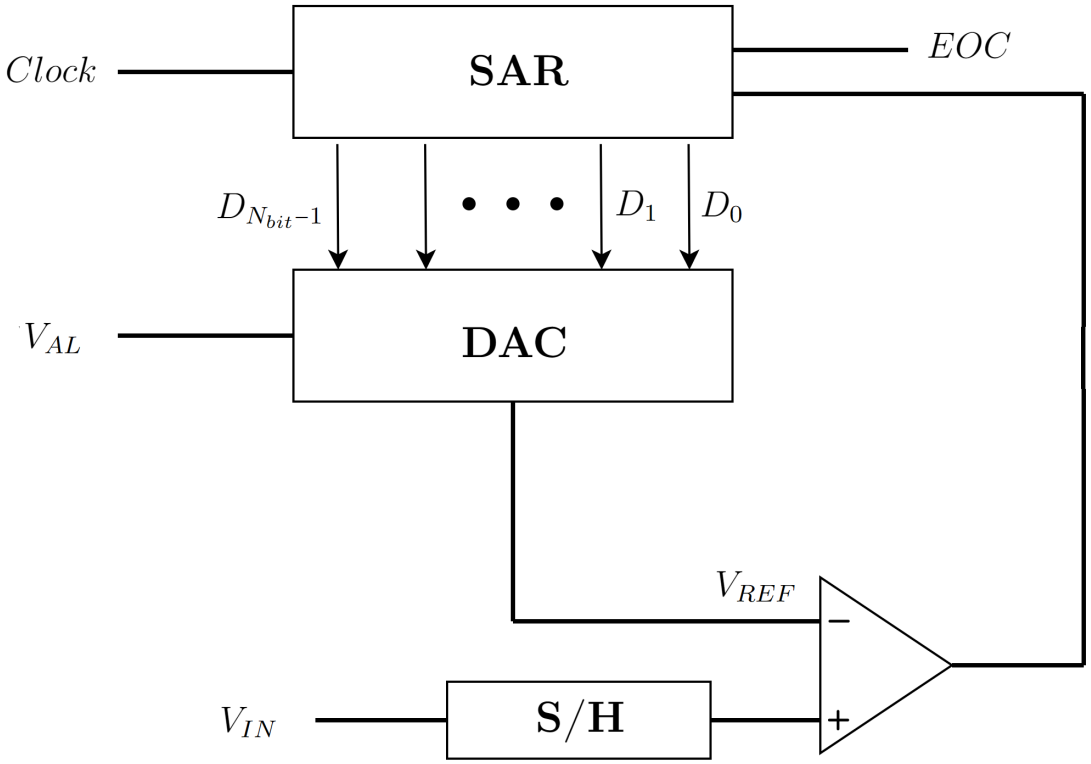


Figura 2.1: Schema a blocchi di un convertitore SAR

Il processo di campionamento è effettuato dal componente di **Sample and Hold** o **S/H** il quale mantiene costante per un certo periodo di tempo il segnale in ingresso V_{IN} da quantizzare fornendo tutto il tempo necessario alla logica che lo segue di effettuare le'operazione di quantizzazione: Quest'ultima si basa sull'utilizzo di un convertitore digitale-analogico o **DAC**, dall'inglese Digital to Analog Converter, il quale è in grado di compiere l'operazione inversa di un ADC per generare la tensione

analogica V_{REF} proporzionale al contenuto del registro SAR (che indica l'attuale approssimazione della conversione) da comparare con il valore V_{IN}. Il valore del registro SAR viene aggiornato a partire dal bit più significativo D_{N-1} ad ogni ciclo dal risultato della suddetta comparazione: il bit è impostato ad 1 se $V_{IN} > V_{REF}$ o a 0 in caso contrario. Dopo N_{bit} cicli il risultato è pronto e viene comunicato tramite l'attivazione di un flag di fine conversione **EOC**, dall'inglese End Of Conversion.

2.2.2 Unità di temporizzazione

Le unità di temporizzazione sono dei timer presenti su ogni MCU che assolvono le più disparate funzioni. In genere si distinguono in timer di sistema, utilizzati per lo più come dei veri e propri orologi e in timer di output, i quali sono impiegati per generare particolari onde quadre che sfruttano la **modulazione di larghezza d'impulso** o **PWM**, dall'inglese Pulse-Width Modulation. I segnali PWM sono caratterizzati da due parametri principali quali:

- T che indica il periodo dell'onda quadra;
- DC o duty cycle che indica la frazione del periodo per cui l'onda è nello stato logico alto.

Nel lavoro di tesi i segnali PWM verranno trasmessi e ricevuti dalle varie antenne al fine di sorvegliare l'area della macchina utensile ed individuare la presenza di una parte del corpo dell'operatore in una zona ritenuta pericolosa.

2.3 Circuito stampato[3]

Un **circuito stampato** o **PCB**, dall'inglese Printed Circuit Board, è un supporto impiegato per installare e collegare i diversi componenti di un circuito elettronico. Il PCB è costituito da un materiale dielettrico (isolante) con **proprietà di resistenza al fuoco** denominato **FR4**, dall'inglese Flame Resistant 4, sul quale viene depositato un sottile strato di rame conduttore che assolverà la funzione di interconnessione tra i componenti.

Nel caso di circuito complessi, il PCB può essere composto da più strati o layer ognuno con una funzione specifica. I layer più interni sono originati dalla porzione di dielettrico centrale (**core**) sul quale vengono stese le prime superfici conduttive in rame, mentre, gli strati più esterni sono formati depositando ulteriore materiale dielettrico detto **preimpregnato** o **prepreg** sui layer sottostanti che funge da separatore per poi ripetere il processo di deposizione del rame. I vari layer comunicano tra di loro attraverso particolari fori chiamati **via**. In passato i componenti venivano installati sul PCB tramite la tecnologia a **fori passanti** o **PTH**, dall'inglese Pin Through Hole, la quale prevedeva l'utilizzo di componenti con lunghi pin da inserire nella scheda e da saldare sul lato posteriore. In tale ottica i PCB erano spesso costituiti da due soli layer di cui il superiore era definito *lato componenti* e l'inferiore *lato saldature*. La crescente complessità dei circuiti ha portato allo sviluppo dei PCB multi-strato e all'impiego dei componenti a **montaggio superficiale** o **SMD**, dall'inglese Surface Mount Device, i quali possono essere saldati direttamente sullo strato in cui vengono posizionati.

[3] **La stesura della sezione trae ispirazione dalla fonte [13]**

2.4 Trasduttore[4]

Il trasduttore è un qualsiasi dispositivo in grado di *convertire* una grandezza fisica in una seconda che presenta caratteristiche diverse rispetto all'originaria. In letteratura, la distinzione tra i termini "trasduttore" e "sensore" non è molto marcata e sovente i due termini vengono utilizzati come sinonimi nonostante quest'ultimo sia tecnicamente un dispositivo in grado di *percepire le variazioni* di una grandezza fisica.

Un particolare tipo di trasduttore è l'**antenna**, un apparato in grado di trasmettere o ricevere onde elettromagnetiche veicolate nello spazio. Le antenne si dividono in trasmittenti e riceventi e spesso, grazie al principio di reciprocità, le due sono identiche ed intercambiabili. Un'antenna trasmittente converte un segnale elettrico in un campo elettromagnetico che irradia nello spazio, mentre, un'antenna ricevente svolge la funzione opposta, ovvero, converte le onde elettromagnetiche in un segnale elettrico. Questo permette ai segnali di viaggiare anche in assenza di un conduttore come ad esempio accade nell'aria. Per far si che i segnali elettrici vengano trasmessi nello spazio in modo efficace questi ultimi devono essere amplificati in modo da assumere una potenza elevata che subirà delle perdite inevitabili durante la trasmissione. Allo stesso modo questa amplificazione è richiesta alla ricezione in quanto il segnale captato non è sufficientemente ampio da poter essere elaborato dalle logiche digitali.

Per semplicità di comprensione, in questa tesi il termine "sensore" o "sensoristica" sarà utilizzato per indicare i trasmettitori ed i ricevitori i quali comprendono un'antenna ed un circuito di amplificazione del segnale.

2.4.1 Amplificatori

Gli amplificatori sono particolari circuiti elettronici attivi (alimentati) i quali, dato un segnale in ingresso con potenza P_{IN}, sono in grado di fornire un segnale con potenza P_{OUT} tale che:

$$P_{OUT} > P_{IN}. \tag{2.1}$$

I circuiti di amplificazione assumono diverse topologie e utilizzano differenti componenti in base agli ambiti di applicazione. Tra i componenti impiegati vi è il **transistore bibolare** o **BJT**, dall'inglese Bipolar Junction Transistor, un dispositivo originariamente utilizzato come amplificatore analogico e successivamente come interruttore logico adoperato nella logica digitale. L'evoluzione tecnologica ha portato allo sviluppo di una nuova tipologia di transistori detti **ad effetto di**

[4] **La stesura della sezione trae ispirazione dalle fonti [11], [9], [6]**

campo o **FET**, dall'inglese Field Effect Transistor, molto facilmente integrabili nell'elettronica moderna.

Amplificatori basati su singolo FET

Un circuito di amplificazione utilizzabile in un trasmettitore è la configurazione a **source comune** di un FET osservabile in figura 2.2. Il guadagno in tensione di

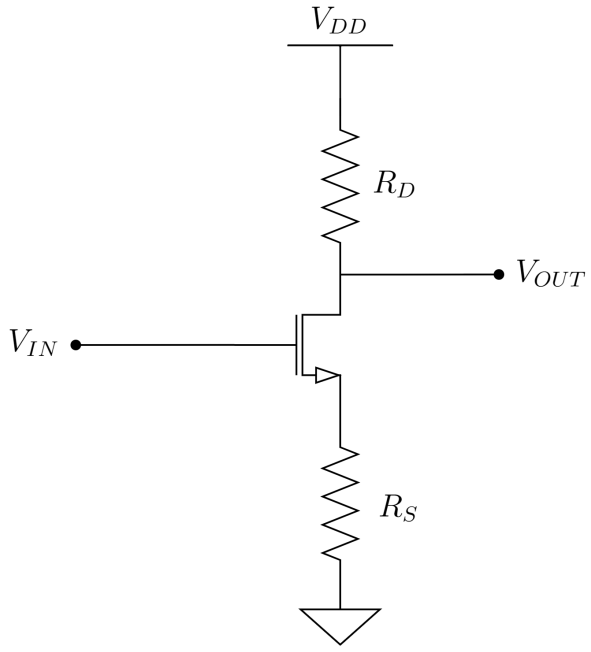


Figura 2.2: Schema circuitale della configurazione a source comune

questa configurazione è definito da:

$$G = \frac{V_{OUT}}{V_{IN}} = -\frac{g_m R_D}{1 + g_m R_S}, \tag{2.2}$$

dove g_m è la trans-conduttanza del FET ovvero il parametro di proporzionalità tra la tensione in ingresso al gate e la corrente che scorre nel canale del FET. Come si nota dalla formula 2.2 il guadagno dell'amplificatore può essere aumentato, a scapito della stabilità del punto di funzionamento (BIAS), rimuovendo la resistenza di source R_S.

Il common source è ampiamente utilizzato per la sua semplicità di implementazione e per l'alto gradagno in tensione e corrente, dunque, in potenza che riesce ad erogare.

Amplificatori basati su OP-AMP

Un componente molto utilizzato in elettronica è l'**amplificatore operazionale** o **OP-AMP**, dall'inglese Operational Amplifier, il quale è composto da numerosi transistor e viene utilizzato in configurazioni con retroazione per aumentare la stabilità dei circuiti. Una topologia molto diffusa è quella a **reazioni multiple** osservabile in figura 2.3 nella sua forma di filtro amplificatore **passa-banda**.

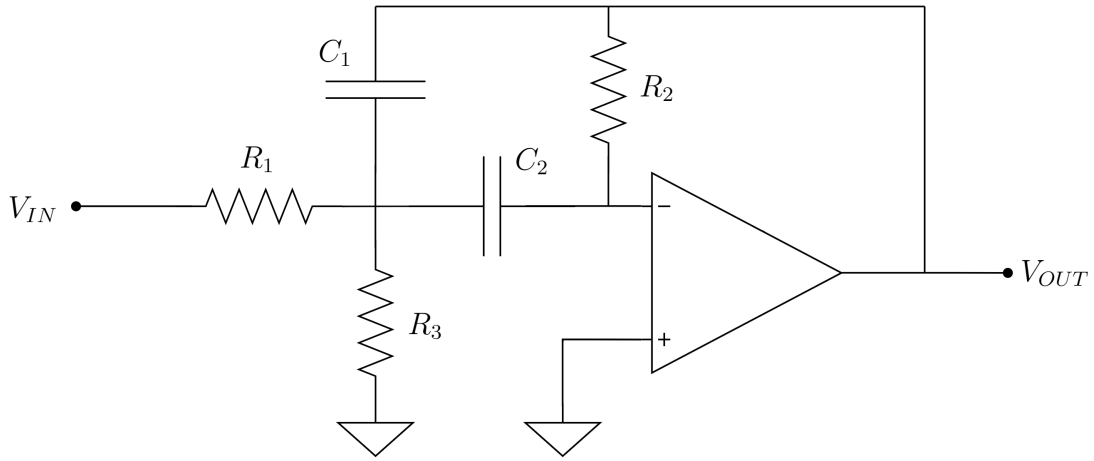


Figura 2.3: Schema circuitale del filtro amplificatore passa-banda

I parametri fondamentali dell'amplificatore sono la frequenza centrale del filtro f_0, il guadagno in continua H_0 e il fattore di qualità Q e si può dimostrare che dipendono dai componenti del circuito secondo le relazioni:

$$\begin{aligned} f_0 &= \frac{1}{2\pi\sqrt{C_1 C_2 R_2 (R_1 \| R_3)}}, \\ H_0 &= \frac{C_2 R_2}{(C_2 + C_1) R_1}, \\ Q &= \frac{\sqrt{C_1 C_2 R_2 (R_1 \| R_3)}}{(R_1 \| R_3)(C_2 + C_1)}. \end{aligned} \tag{2.3}$$

Si tratta di una configurazione invertente in quanto la tensione in uscita ha segno opposto rispetto a quella in ingresso. Per annullare tale effetto si è soliti introdurre un ulteriore stadio amplificatore di tipo invertente. Il circuito verrà impiegato nel seguente progetto come amplificatore del blocco ricevitore.

2.5 Analisi in frequenza dei segnali[5]

Un segnale può essere analizzato nel dominio del tempo o nel dominio della frequenza. Le due modalità costituiscono un modo diverso di vedere lo stesso fenomeno offrendo spesso un'alternativa efficace quando una delle due risulta essere troppo complessa. Il passaggio tra i due domini è consentito dalla **Trasformata di Fourier** uno strumento matematico che ha trovato numerosissime applicazioni nel mondo attuale.

Data una grandezza fisica espressa da una funzione continua nel tempo $x(t)$ la sua trasformata nel dominio delle frequenze $X(f)$ è così definita:

$$X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft}dt, \tag{2.4}$$

e risulta essere una funzione complessa caratterizzata da un modulo ed una fase. Nella presente tesi verrà implementato lo studio in frequenza del solo modulo della trasformata, dunque, verranno tralasciate le informazioni relative alla fase. Inoltre, la relazione descritta in formula 2.4 non è direttamente implementabile da un dispositivo elettronico digitale in quanto quest'ultimo è in grado di processare esclusivamente segnali digitali. Risulta dunque necessario effettuare le operazioni preliminari di campionamento e quantizzazione per trasformare il segnale in questione da analogico a digitale.

2.5.1 Campionamento[6]

Il campionamento è una tecnica che consiste nel convertire un segnale analogico $x(t)$ in un segnale discreto $x(kT_s)$, valutandone l'ampiezza ad intervalli temporali regolari di ampiezza T_s denominati anche periodi di campionamento. La frequenza di campionamento F_s si ottiene da reciproco del periodo di campionamento:

$$F_s = \frac{1}{T_s}. \tag{2.5}$$

Dato un segnale analogico originario a frequenza f, al fine di poterlo ricostruire senza perdite di informazioni a partire dal segnale campionato, è necessario che la frequenza di acquisizione F_s sia almeno pari a due volte la frequenza f del segnale. Il risultato è noto come teorema di **Shannon-Nyquist** ed enuncia in formule:

$$F_s \geq 2f. \tag{2.6}$$

Il mancato rispetto della condizione comporta la distorsione del segnale analogico ricostruito a causa di un effetto noto con il termine di **aliasing**.

---

[5] **La stesura della sezione trae ispirazione dalla fonte [7]**

[6] **La stesura della sezione trae ispirazione dalla fonte [12]**

Campionamento di segnali reali

Come è possibile osservare dall'integrale indefinito della formula 2.4, la trasformata di Fourier costituisce un'analisi che si protrae all'infinito e che richiederebbe un numero infinito di campioni. Dal punto di vista reale questa è una condizione non applicabile sia perché nella realtà non esistono segnali di durata illimitata, sia perché l'acquisizione di un segnale è caratterizzata da un tempo definito T_w (e quindi da un numero limitato di campioni) che prende il nome di finestra di osservazione. Un importante risultato della teoria dei segnali afferma che:

- un segnale limitato nel dominio del tempo ha un supporto illimitato nel dominio delle frequenze e viceversa;
- un segnale illimitato nel dominio del tempo ha un supporto limitato nel dominio delle frequenze e viceversa.

Ne consegue che il segnale reale di interesse, essendo limitato nel tempo presenta un'estensione illimitata nel dominio delle frequenze: questo spargimento di frequenze è riconosciuto con il nome di **dispersione spettrale**. Tale effetto, una volta applicato il campionamento porterà inevitabilmente al fenomeno dell'aliasing.

Campionamento di segnali periodici

Di particolare interesse risultano essere i segnali periodici i quali verranno trattati all'interno del lavoro di tesi. Al fine di poter riprodurre esattamente il segnale analogico a partire da quello campionato non è sufficiente solo rispettare il teorema di Shannon-Nyquist precedentemente enunciato. Per questa categoria di segnali, di fondamentale importanza risulta essere la condizione di **campionameto coerente** il quale consiste nel prelevare un numero intero N di campioni tali che costituiscano un numero intero di forme d'onda. In altre parole, la finestra di osservazione T_w deve contenere un numero intero di periodi del segnale da campionare.

2.5.2 Quantizzazione[7]

All'interno di un qualsiasi calcolatore elettronico digitale, i dati vengono rappresentati sotto forma di valori numerici da un numero finito di cifre binarie N_{bit} denominati **bit**, dall'inglese "binary digit". Ne risulta che un dato non può assumere qualsiasi valore ma solo un set discreto di M numeri pari a:

$$M = 2^{N_{bit}}. \tag{2.7}$$

[7] **La stesura della sezione trae ispirazione dalla fonte [14]**

L'operazione di quantizzazione è tipicamente svolta da un convertitore analogico-digitale il quale legge la tensione del segnale discreto $x(kT_s)$ in ingresso come frazione della sua tensione di alimentazione V_{AL} e gli assegna il valore quantizzato proporzionale. Ad ogni numero descritto dalla sequenza di N_{bit} corrisponde un intervallo di tensione la cui ampiezza q risulta:

$$q = \frac{V_{AL}}{M}. \tag{2.8}$$

All'interno di un determinato intervallo il valore reale del segnale discreto $x(kT_s)$ verrà tradotto con lo stesso valore quantizzato che può assumere nel dominio digitale. Pertanto, la quantizzazione è un processo irreversibile che introduce una perdita di informazioni in quanto genera degli errori in base alla differenza tra il valore reale del segnale discreto $x(kT_s)$ ed il valore quantizzato.

2.5.3 Trasformata di Fourier tempo-discreta

Dopo aver ottenuto un segnale digitale è possibile applicare una versione alternativa al calcolo della trasformata nota come **Traformata di Fourier tempo-discreta** o **DFT** (dall'inglese Discrete-time Fourier Transform), in formule:

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-j2\pi \frac{k}{N} n}, \tag{2.9}$$

Avendo a disposizione un numero di campioni N del segnale analogico originario equispaziati di un tempo T_s, la durata dell'osservazione T_w risulta essere:

$$T_w = NT_s, \tag{2.10}$$

dalla quale è possibile ricavare la frequenza di osservazione F_w:

$$F_w = \frac{1}{T_w} = \frac{1}{NT_s} = \frac{F_s}{N}, \tag{2.11}$$

la quale costituisce la risoluzione in frequenza della DFT.

Algoritmi alternativi al calcolo della DFT

La **Trasformata veloce di Fourier** o **FFT**, dall'inglese Fast Fourier Transform, rappresenta un modo molto efficiente di calcolare la trasformata di Fourier di un segnale discreto. È un algoritmo particolarmente vantaggioso quando il numero di campioni N è una potenza di 2. In questo caso il costo computazionale risulta essere di:

$$\begin{aligned} C(DFT) &\sim N^2, \\ C(FFT) &\sim N log_2 N, \end{aligned} \tag{2.12}$$

il che porta ad un notevole risparmio di operazioni per N elevati.

Capitolo 3

Analisi preliminari

3.1 Specifiche del sistema

Dai documenti relativi allo studio di fattibilità precedentemente condotto e dalla teoria dei segnali vengono ricavate le specifiche del sistema legate al flusso di trasmissione e ricezione. In particolare si richiede di:

- programmare fino a quattro frequenze di trasmissione;
- acquisire il segnale tramite campionamento coerente.

Il teorema di Shannon-Nyquist descritto nella formula 2.6 afferma che, per ricostruire un segnale dopo il campionamento, è necessario acquisirlo ad almeno il doppio della sua frequenza f. Tuttavia, da un punto di vista pratico si applica un leggero sovracampionamento in modo che:

$$F_s = 2.6f. \tag{3.1}$$

Ipotizzando di acquisire il segnale ad una frequenza di campionamento $F_s = 500\ kHz$, si ottiene che la massima frequenza per il segnale di trasmissione sia:

$$f_{max} \approx 190\ kHz \tag{3.2}$$

Equispaziando per ipotesi i vari segnali di 30 kHz si ottengono le seguenti frequenze:

$$\begin{aligned} f_1 &= 100\ kHz, \\ f_2 &= 130\ kHz, \\ f_3 &= 160\ kHz, \\ f_4 &= 190\ kHz. \end{aligned} \tag{3.3}$$

Per risolvere correttamente tali segnali nel dominio delle frequenze si imposta una risoluzione minima $\Delta f_{min} = 10\ kHz$: in questo modo le frequenze dei quattro segnali sono tutte multiple della risoluzione e le componenti non vengono separate in

diversi campioni nel dominio della frequenza.

La relazione che lega la frequenza di campionamento alla risoluzione in frequenza si ottiene dalla formula 2.11 ed enuncia che:

$$\Delta f = \frac{F_s}{N_{fft}}, \tag{3.4}$$

dove N_{fft} è il numero di campioni necessari ad ottenere la risoluzione voluta. Invertendo la relazione si ottiene che:

$$N_{fft} = \frac{F_s}{\Delta f} = \frac{500\ kHz}{10\ kHz} = 50. \tag{3.5}$$

Al fine di effettuare un campionamento coerente si dovrà impostare il convertitore analogico-digitale in modo da acquisire un numero intero di forme d'onde per ciascuna frequenza di trasmissione. Ad esempio, per la frequenza di trasmissione f_1 si otterranno un numero di campioni Npp per periodo pari a:

$$Npp = \frac{F_s}{f_1} = 5. \tag{3.6}$$

Si noti come il numero di campioni minimo N_{fft} soddisfa già la condizione di campionamento coerente per il segnale a frequenza f_1 essendo un multiplo intero di Npp.

3.2 Il microcontrollore Aurix

Nella seguente sezione verranno indagate le funzionalità del microcontrollore Aurix basato su architettura Tri-Core™ di Infineon per soddisfare le specifiche precedentemente enunciate. A tale scopo l'architettura relativa ai blocchi del MCU verrà presentata attingendo informazioni dalla documentazione di Infineon relative al manuale dell'utilizzatore [4] e alla relativa appendice del dispositivo [2].

3.2.1 EVADC

L'**EVADC**, acronimo di **Enhanced Versatile Analog-to-Digital Converter**, utilizza il principio SAR per convertire le tensioni analogiche di ingresso in valori discreti o digitali. Come è possibile osservare dalla figura 3.1, il modulo dispone di tre macro-sezioni, i **cluster**, i quali si dividono ulteriormente in **gruppi**, ciascuno dotato di una logica di controllo che definisce la sequenza di conversioni consecutive (in coda) pilotando tramite un multiplexer o **MUX** i canali di ingresso al blocco convertitore.

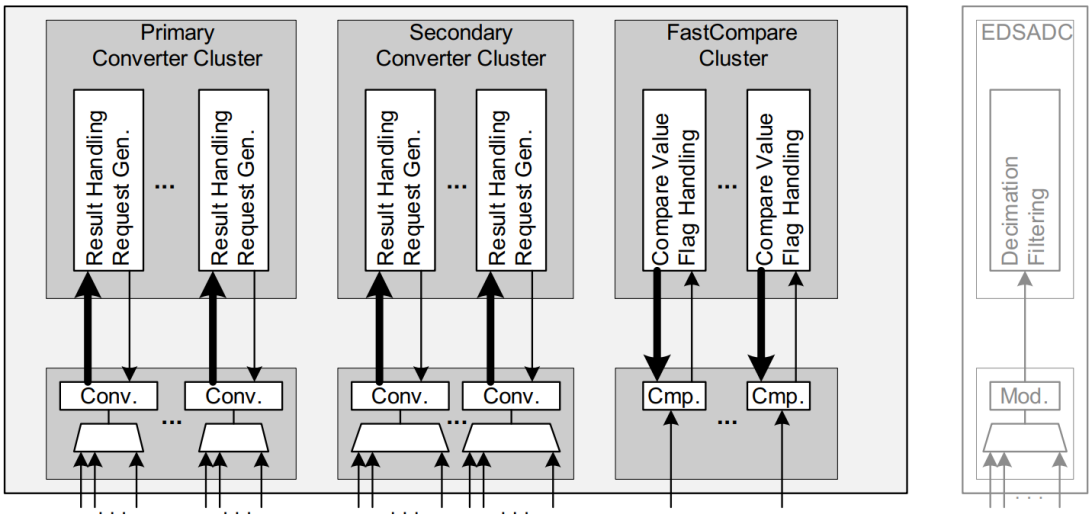


Figura 3.1: Raggruppamento in cluster e gruppi dell'EVADC

I gruppi possiedono caratteristiche diverse in base al cluster di appartenenza, ovvero:

- il cluster primario, dotato di MUX 8:1 e code a 8 stadi, è caratterizzato da un tempo di conversione inferiore a 0.5 μs;
- il cluster secondario: dotato di MUX 16:1 e code a 16 stadi, è caratterizzato da un tempo di conversione inferiore a 1 μs;

› il cluster a rapide comparazioni, il quale effettua solo delle comparazioni e presenta ingressi non multiplexati (canali singoli) per raggiungere una velocità di aggiornamento inferiore a 0.2 μs.

Oltre all'EVADC anche l'**EDSADC** (**Enhanced Delta-Sigma Analog-to-Digital Converter**) il quale si basa sull'omonimo principio di conversione effettua conversioni analogico-digitali ma non verrà impiegato nel progetto che invece si focalizzerà sui convertitori del modulo EVADC.

Queued Request Source

La **Queued Request Source** o **QRS**, visibile in figura 3.2 è dotata di un buffer FIFO composto da più registri 3.1 **QxR** impilati i quali gestiscono sequenze di conversioni lunghe fino alla dimensione dello stack. I gruppi primari sono dotati di code a 8 stadi, i secondari, invece, sono dotati di code a 16 stadi. Ogni coda può quindi gestire tutti i canali associati al gruppo corrispondente. I numeri dei canali per questa sequenza possono essere programmati liberamente, inoltre, più conversioni dello stesso canale all'interno di una sequenza sono supportati. Gli identificativi dei canali vengono immessi nel primo stadio del buffer (**QINR** o **Queue Input Register**), mentre, l'ultimo livello (**Q0R** o **Queue 0 Register**) definisce il prossimo canale da convertire. Ogni voce può essere convertita una sola volta o può essere reinserita automaticamente per una nuova conversione (**refill**). Una richiesta di conversione viene inviata all'arbitro solo se una voce valida è memorizzata nello stadio 0 del buffer. Il suo compito è quello di decidere quale conversione avviare tra i vari stadi 0 di diverse QRS. Essendo coinvolta una sola QRS nell'applicazione, il compito dell'arbitro è marginale. L'unità di trigger e gating gestisce i segnali, generati via software o hardware (interni o esterni all'EVADC), che attivano una richiesta di conversione per ogni valore valido giunto all'ultimo stadio. In tale ottica, modificando il campo **EXTR = 1_B** ogni elemento della coda attenderà un evento di trigger prima effettuare una richiesta di conversione. Nell'applicazione in esame, il suddetto evento di trigger verrà generato da un componente integrato nell'EVADC, chiamato **Request Timer** o **RT** e, come visibile in figura 3.6, la sua azione è resa effettiva solo modificando il campo **ENTR = 1_B** del registro 3.2 **QMR**.

Request Timer

Il RT integrato può richiedere la conversione di una sequenza programmata a intervalli configurabili in modo da ottenere campioni equispaziati nel tempo. La sua configurazione avviene modificando i campi del registro 3.3 **REQTM**. È dotato di un prescaler 16:1 il quale divide la frequenza f_{ADC} = 160 MHz del modulo per generare step da 0.1 μs, e di un timer (**Sequence Timer** o **SEQTIM** a 10 bit che crea intervalli fino a 2^{10} = 102.4 μs.

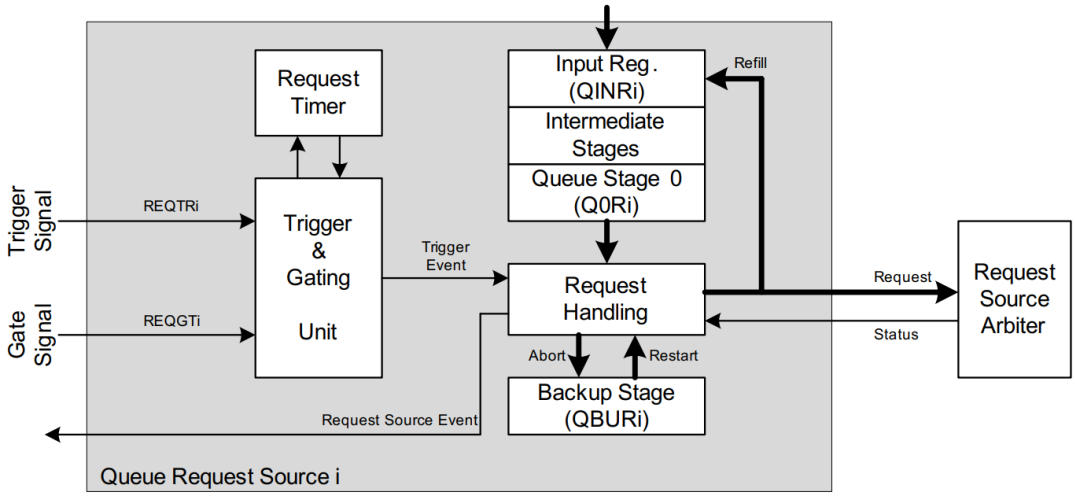


Figura 3.2: Architettura della Queue Request Source

Dalle specifiche si evince che è la modalità di funzionamento più adatta per il RT è la prima visualizzabile in figura 3.3. Il suo nome è **Pause after each conversion** ed è impostata modificando il campo **SEQMOD = 01_B**. Questa permette di iniziare a campionare il segnale PWM all'inizio della sua forma d'onda e di continuare secondo intervalli regolari. Si vuole, inoltre, che il RT venga avviato da un segnale di trigger esterno all'EVADC e lo si ottiene impostando il campo **ENTR = 1_B**.

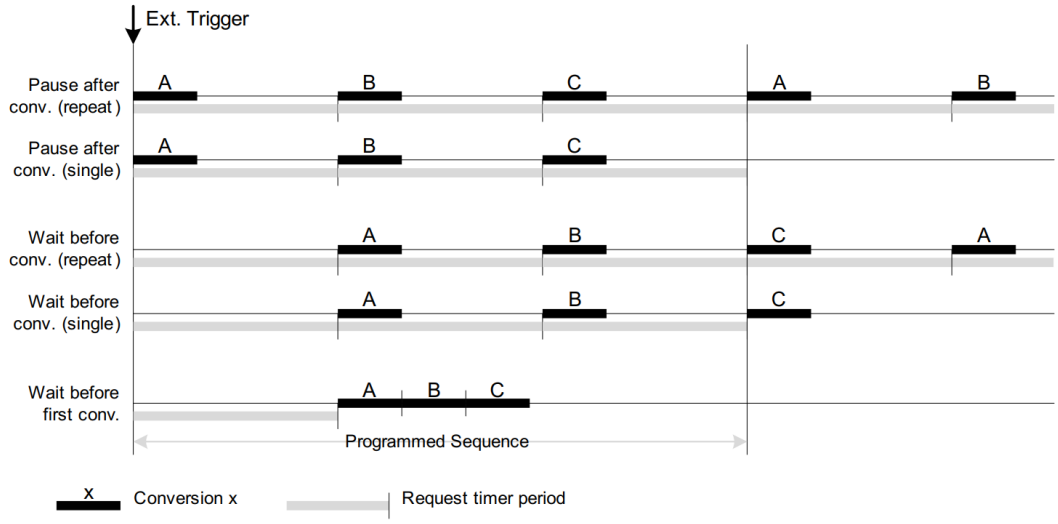


Figura 3.3: Modalità di funzionamento del Request Timer

Si definiscono gli intervalli regolari a partire dalla frequenza di campionamento desiderata $f_s = 500\ kHz$ ($T_s = 2\ \mu s$). Si ottiene il valore iniziale del SEQTIM **SEQTIMSET = 20_D** ($20 \cdot 0.1\ \mu s = 2\ \mu s$), mentre, per quanto concerne la soglia di spegnimento si imposta un valore **SEQTIMOFF = 0_D** in modo tale che il segnale di trigger generato dal RT abbia l'andamento descritto in figura 3.4.

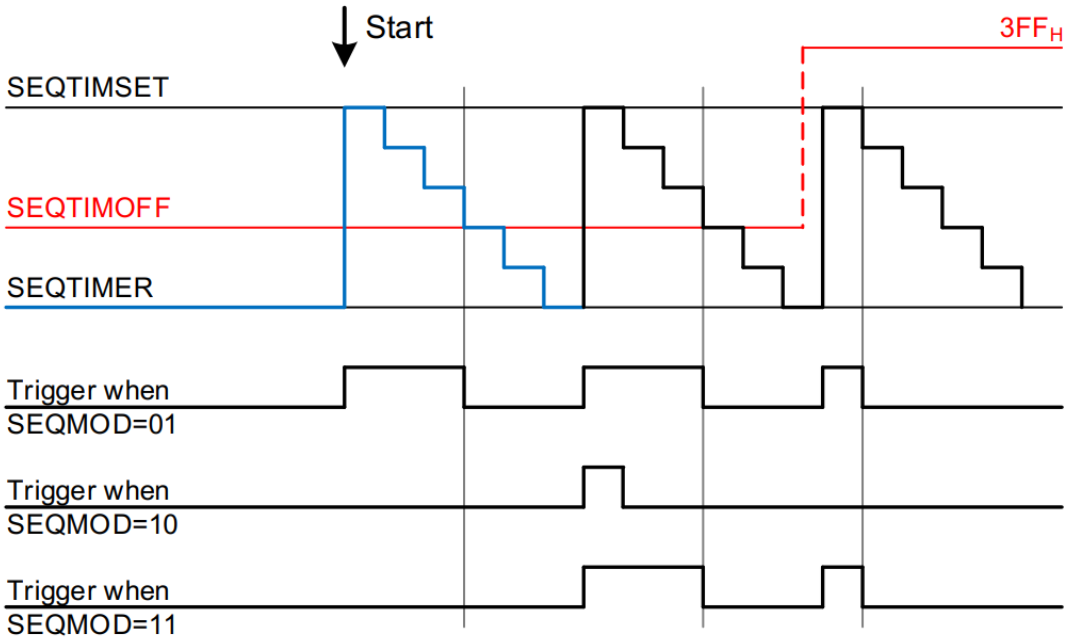


Figura 3.4: Trigger generato dal Request Timer

Segnali di trigger e di gating

La QRS può essere avviata da trigger interni o esterni e il suo funzionamento può essere controllato tramite segnali di gating. Questi segnali possono essere selezionati individualmente attraverso dei MUX da una serie di ingressi configurando opportunamente i campi del registro 3.4 **QCTRL**. Come è possibile osservare in figura 3.5, per far si che il RT sia avviato dall'onda quadra del PWM è necessario innanzitutto abilitare l'ingresso **REQTRI** per il segnale di trigger esterno **GTMADC0TRIG0** proveniente dal PWM modificando il campo **XTSEL = 8_D**. Successivamente si imposta **XTMODE = 2_D** per azionare il trigger sul fronte di salita del segnale e **TRSEL = 0_D** per instradare il segnale di trigger esterno sul RT. Infine, l'abilitazione della modalità di campionamento attraverso il RT è gestita dal campo **TMEN = 1_B**.

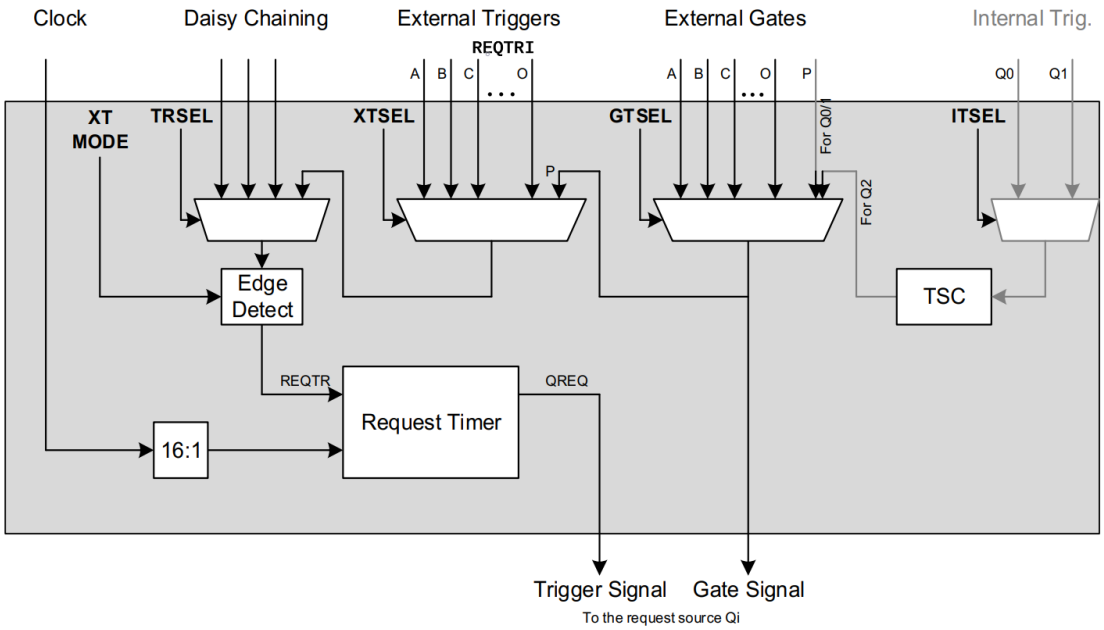


Figura 3.5: Segnali di trigger e di gating

Quando il RT è abilitato, il suo funzionamento è controllato dal segnale di trigger esterno selezionato, mentre, il segnale di trigger per la QRS corrispondente viene generato localmente dal RT. La figura 3.6 riassume i percorsi dei segnali di controllo nei blocchi impiegati.

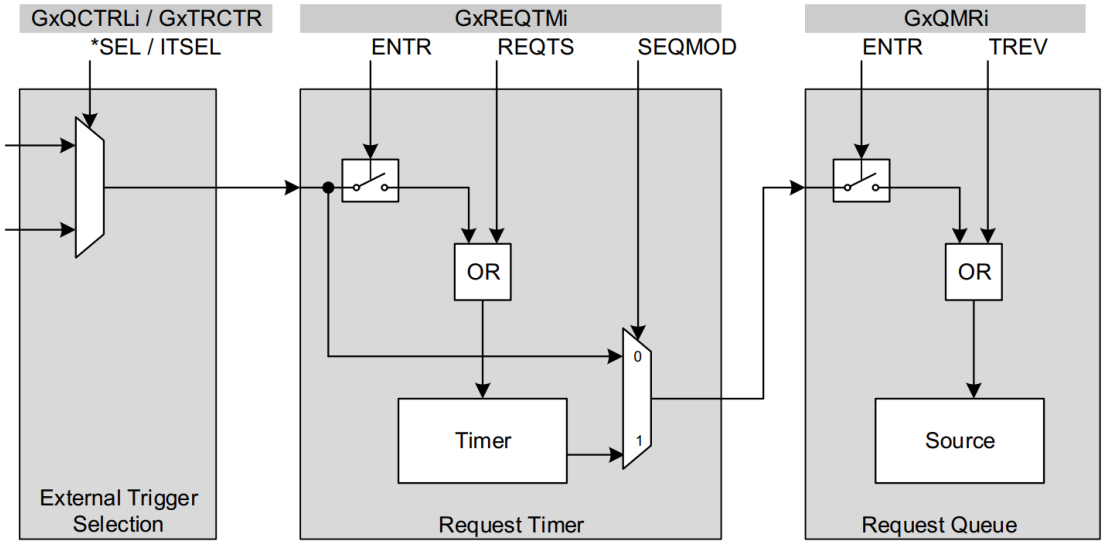


Figura 3.6: Percorsi dei segnali di trigger e di gating

Calcolo dei tempi di conversione

Un requisito fondamentale per il corretto funzionamento dell'applicazione è rappresentato dal tempo di conversione il quale deve essere minore del periodo di campionamento scandito dal RT. Come è possibile osservare in figura 3.7, il tempo di conversione totale t_{TC} (**Total Conversion Time**) è costituito da un tempo di conversione minimo (**Minimum Conversion Time**) diviso in tempo di campionamento del segnale t_S (**Sampling**) e in conversione t_C (**Conversion**) la quale sfrutta il principio del SAR per la generazione delle cifre digitali e da un tempo addizionale composto da operazioni come la riduzione del rumore t_N (**Noise Reduction**), la post-calibrazione t_P (**Post-Calibration**) e altri passaggi interni t_I (non visibili in figura) necessari per la sincronizzazione di una macchina a stati.

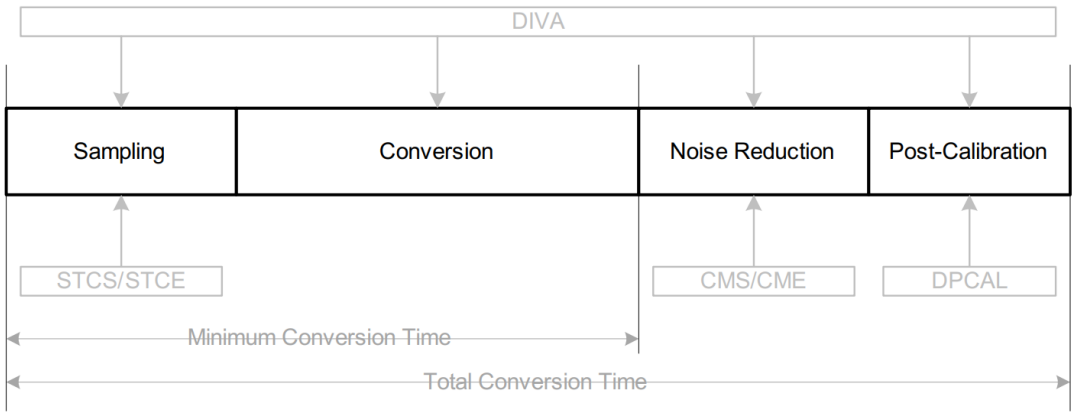


Figura 3.7: Suddivisione del tempo di conversione

In formule:

$$t_{TC} = t_S + t_C + t_N + t_P + t_I, \tag{3.7}$$

i cui addendi nello specifico constano di:

$$\begin{aligned} t_S &= (2 + STC) \cdot t_{ADCI}, \\ t_C &= 13 \cdot t_{ADCI}, \\ t_N &= NRS \cdot (4 \cdot t_{ADCI} + 3 \cdot t_{ADC}), \\ t_P &= (4 + 2 \cdot CALSTC) \cdot t_{ADCI} + 5 \cdot t_{ADC}), \\ t_I &= 3 \cdot t_{ADC}. \end{aligned} \tag{3.8}$$

I parametri di base dell'EVADC utili per il calcolo dei tempi sono:

$$\begin{aligned} f_{ADC} &= 160\ MHz, \\ t_{ADC} &= 6.25\ ns. \end{aligned} \tag{3.9}$$

Il prescaler che gestisce la velocità di conversione è controllato dal campo **DIVA** del registro 3.5 **ANCFG** il quale riduce la frequenza di base f_{ADC} del modulo in:

$$f_{ADCI} = \frac{f_{ADC}}{DIVA + 1}. \tag{3.10}$$

La frequenza massima di funzionamento del convertitore si ottiene per il valore **DIVA = 0**. Impostando il parametro **DIVA = 2** si ottiene:

$$\begin{aligned} f_{ADCI} &= 53.3\ MHz, \\ t_{ADCI} &= 18.75\ ns. \end{aligned} \tag{3.11}$$

Il tempo di campionamento t_S è controllato dal parametro **STC** (**Sample Time Control**) il quale aggiunge tempo addizionale al minimo tempo di campionamento previsto $2 \cdot t_{ADCI}$. Il suddetto parametro è modificabile agendo sul campo **STCS** del registro 3.6 **ICLASS**. I convertitori del gruppo primario, se alimentati da una tensione $4.5\ V < V_{AL} < 5.5\ V$ necessitano di un tempo di campionamento minimo $t_{S_{1,MIN}} = 100\ ns$, ottenibile impostando **STCS = 4**, mentre, gli appartenenti al cluster secondario nelle stesse condizioni di alimentazione necessitano di un tempo di campionamento minimo $t_{S_{2,MIN}} = 500\ ns$, ottenibile impostando **STCS = 32**. In dettaglio si calcola:

$$\begin{aligned} t_{S_1} &= (2 + 4) \cdot 18.75\ ns = 112.5\ ns, \\ t_{S_2} &= (2 + 32) \cdot 18.75\ ns = 637.5\ ns. \end{aligned} \tag{3.12}$$

I tempi di riduzione del rumore t_N e di post-calibrazione t_P sono invece gestiti dai parametri **NRS** (**Noise Reduction Step**) il quale definisce il numero di step da applicare per il processo di riduzione del rumore e **CALSTC** (**Calibration Sample Time Control**). Si imposta il campo **CMS = 0** del registro 3.6 **ICLASS** per disattivare la funzione di riduzione del rumore e il campo **CALSTC = 1** del registro 3.5 **ANCFG** per ridurre al minimo la post-calibrazione (per $f_{ADCI} > 40\ MHz$ è necessario impostare tale valore del parametro per garantire un adeguato funzionamento). Si ottengono i seguenti valori:

$$\begin{aligned} t_C &= 13 \cdot 18.75\ ns = 243.75\ ns, \\ t_N &= 0 \cdot (4 \cdot 18.75\ ns + 3 \cdot 6.25\ ns) = 0\ ns, \\ t_P &= (4 + 2 \cdot 1) \cdot 18.75\ ns + 5 \cdot 6.25\ ns = 143.75\ ns, \\ t_I &= 3 \cdot 6.25\ ns = 18.75\ ns \end{aligned} \tag{3.13}$$

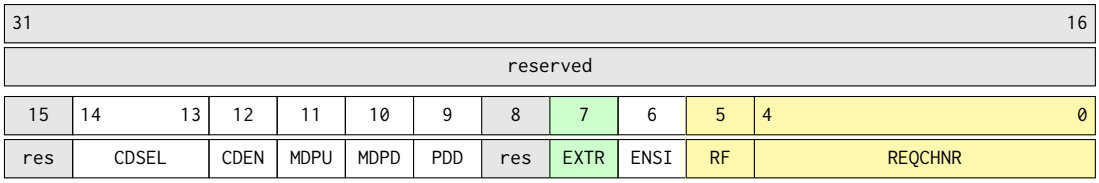
che portano ad un tempo di conversione totale di

$$\begin{aligned} t_{TC_1} &= 518.75\ ns, \\ t_{TC_2} &= 1043.75\ ns, \end{aligned} \tag{3.14}$$

rispettivamente con i convertitori del gruppo primario e secondario. Dato l'ampio margine rispetto al tempo impiegato da RT ($t_{RT} = 2\ \mu s$), ne consegue che possono essere sfruttati anche gli ADC secondari, i quali aumentano il numero totale di gruppi di convertitori per meglio predisporre una configurazione multi-canale.

Registri QINR e Q0R

Registro 3.1: **QINR**/**Q0R** (Queue Input Register/Queue 0 Register)



REQCHNR **Request Channel Number**
Definisce il numero del canale da convertire. I canali non disponibili vengono interpretati come canale 0.

RF **Refill**

0_B Nessuna ricarica: questa voce della coda viene convertita una sola volta una volta giunta nello stadio 0 della coda

1_B Ricarica automatica: questa voce della coda viene ricaricata automaticamente nello stadio di input della coda

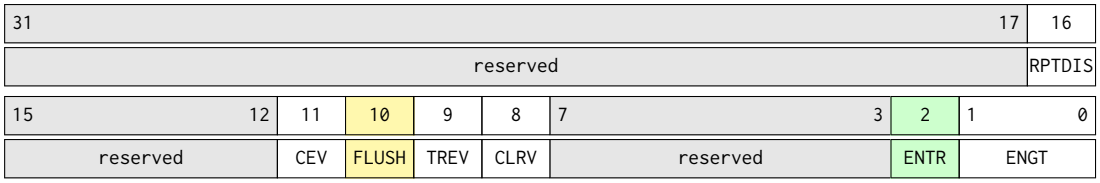
EXTR **External Trigger**

0_B Una voce valida in coda effettua immediatamente una richiesta di conversione

1_B Una voce valida in coda attende che si verifichi un evento trigger prima di effettuare una richiesta di conversione

Registro QMR

Registro 3.2: **QMR** (Queue Mode Register)



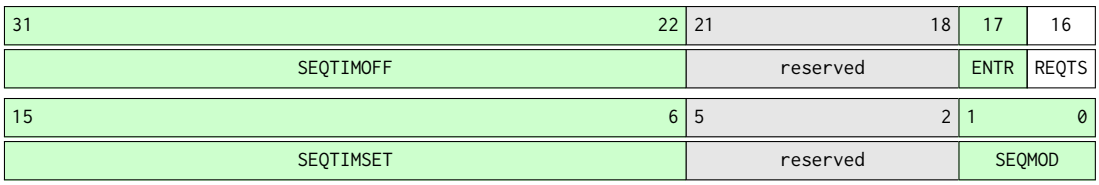
ENTR **External Trigger**

0_B Trigger esterno disabilitato

1_B Il fronte selezionato al segnale di trigger REQTR selezionato genera l'evento di trigger

Registro REQTM

Registro 3.3: **REQTM** (Request Timer Mode Register)



SEQMOD **Sequence Mode**
Seleziona il modo in cui il RT controlla la sequenza di conversione in base al trigger ricevuto (figura 3.3) e al trigger generato (figura 3.4).

- 00_B RT off
- 01_B Pause after each conversion
  Dopo un evento di trigger viene immediatamente richiesta la prima conversione. Ogni conversione successiva viene richiesta dopo che RT ha raggiunto il numero configurato di step.

SEQTIMSET **Sequence Timer, Set Value**
Valore iniziale per il SEQTIM in step da 0.1 µs. Questo valore è caricato quando viene avviato un nuovo periodo del RT.

SEQTIMOFF **Sequence Timer, Switch Off Value**
Il segnale di trigger generato viene disabilitato quando il valore del timer è uguale o al di sotto di questa soglia.

ENTR **Enable External Trigger**

- 0_B Trigger esterno disabilitato
- 1_B Il fronte selezionato sul segnale input di trigger selezionato avvia il RT

Registro QCTRL

Registro 3.4: **QCTRL** (Queue Source Control Register)

31	30 29	28	27 24	23	22 21	20	19 16
TMWC	reserved	TMEN	reserved	GTWC	reserved	GTLVL	GTSEL

15	14 13	12	11 8	7 6	5 4	3 0
XTWC	XTMODE	XTLVL	XTSEL	TRSEL	reserved	SRCRESREG

TRSEL **Trigger Source Selection**

00_B Trigger esterno, come selezionato da **XTSEL**

XTSEL **Trigger Source Selection**

8_D Trigger esterno proveniente da **GTMADC0TRIG0**

XTMODE **Trigger Operating Mode**

00_B Nessun trigger esterno

01_B Trigger sul fronte di discesa

10_B Trigger sul fronte di salita

11_B Trigger su entrambi i fronti

XTWC **Write Control for Trigger Configuration**

0_B Nessun accesso alla configurazione dei trigger

1_B I campi **XTMODE**, **XTSEL**, **TRSEL** possono essere scritti.

TMEN **Timer Mode Enable**

0_B Timer Mode disattivata

1_B Timer Mode per il campionamento equidistante è abilitata

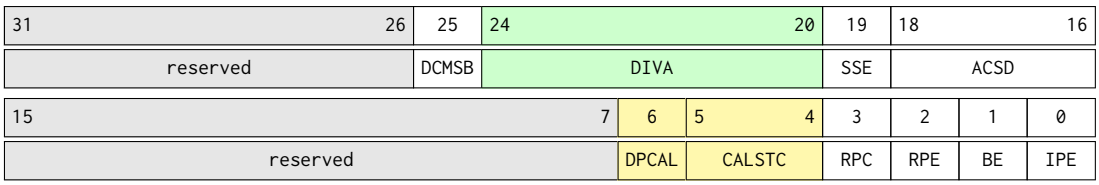
XTWC **Write Control for Timer Mode**

0_B Nessun accesso alla Timer Mode

1_B Il campo **TMEN** può essere scritto.

Registro ANCFG

Registro 3.5: **ANCFG** (Analog Function Configuration Register)



CALSTC **Calibration Sample Time Control**

00_B $2 \times t_{ADCI}$

01_B $4 \times t_{ADCI}$

10_B $6 \times t_{ADCI}$

11_B $8 \times t_{ADCI}$

DPCAL **Disable Post-Calibration**

0_B Post-calibrazione automatica dopo ogni conversione

1_B Post-calibrazione disattivata

DIVA **Divider Factor for the Analog Internal Clock**
Definisce la frequenza interna del convertitore f_{ADCI} derivata dal clock periferico: $f_{ADCI} = f_{ADC}/CP$.

00_H CP = 1 (frequenza massima)

01_H CP = 2

...

$1F_H$ CP = 32

Registro ICLASS

Registro 3.6: **ICLASS** (Input Class Register)

31 27	26	25 24	23 22	21	20 16
reserved	SESPE	CME	AIPE	res	STCE

15 11	10	9 8	7 6	5	4 0
reserved	SESPS	CMS	AIPS	res	STCS

STCS **Sample Time Control for Standard Conversions**
Numero di cicli di clock aggiuntivi rispetto al tempo di campionamento minimo di 2 cicli di clock.

CMS **Conversion Mode for Standard Conversions**

00_B Riduzione del rumore disattivata

01_B Riduzione del rumore di livello 1 (1 step addizionale)

10_B Riduzione del rumore di livello 2 (3 step addizionali)

11_B Riduzione del rumore di livello 3 (7 step addizionali)

3.2.2 TOM

Il **Generic Timer Module** o **GTM** è una piattaforma timer generica che assolve le più disparate funzioni grazie all'ampia configurabilità dei suoi sotto-moduli. Tra questi ci sono diversi **Timer Output Module** o **TOM** i quali dispongono di 16 canali indipendenti per generare semplici segnali PWM sui loro pin di output.

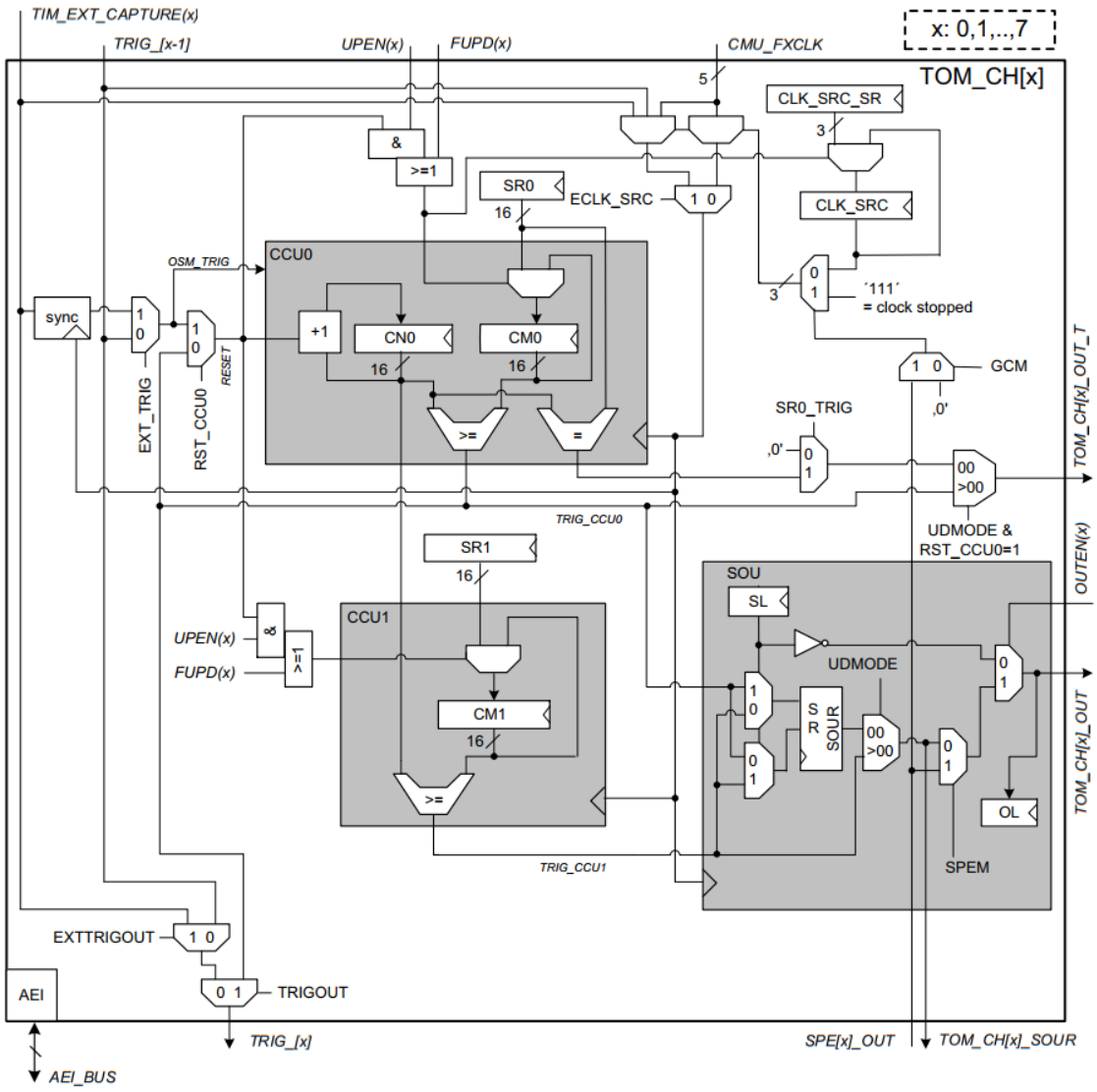


Figura 3.8: Architettura dei canali TOM

Canali TOM

Ogni canale TOM comprende una **Counter Compare Unit 0** (**CCU0**) che contiene il registro contatore **CN0** e il registro del periodo **CM0**, ed una **Counter Compare Unit 1** (**CCU1**) che ingloba il registro del duty cycle **CM1** e la **Signal Output Generation Unit** (**SOU**) che include il registro di uscita **SOUR**. L'architettura è illustrata in figura 3.8 per i canali da 0 a 7.

Nella CCU0 il registro **CN0** è aggiornato alla frequenza di ingresso (**CMU_FXCLK**) e quando è maggiore o uguale a **CM0** (di fatto anche a **CM1**) la CCU0 attiva la SOU tramite il segnale **TRIG_CCU0**. Nella CCU1 il registro contatore **CN0** viene confrontato con **CM1**. Se **CN0** è maggiore o uguale a **CM1**, la CCU1 attiva la SOU tramite il segnale **TRIG_CCU1**. Se il registro contatore **CN0** viene azzerato dalla propria unità CCU0 (configurazione tramite **RST_CCU0 = 0**), sono valide le seguenti istruzioni:

- **CN0** conta da zero a **CM0-1** e viene quindi azzerato;
- quando **CN0** viene reimpostato da **CM0** a zero, viene generato un fronte su **SL**;
- quando **CN0 > CM1**, viene generato un fronte su **!SL**;
- se **CM0 = 0** o **CM0 = 1**, il contatore **CN0** rimane fisso a zero;
- se **CM0 = 0**, l'uscita è **!SL = 0%** duty cycle;
- se **CM1 >= CM0** e **CM0 > 1**, l'uscita è **SL = 100%** duty cycle.

Segnali di trigger e di gating

Il GTM può comunicare attraverso dei collegamenti dedicati con il modulo EVADC, in particolare, come è visibile in figura 3.9, gli output del TOM possono essere instradati tramite i campi **SELx** del registro di selezione **ADCTRIG0OUTx** in modo tale da triggerare il Request Timer e far si che si sincronizzi con il segnale PWM, in accordo con quanto specificato nella sezione 3.2.1 relativa al RT.

Per il convertitore del gruppo primario si imposta il segnale di trigger **ADC_TRIG0[0]**, mentre, per il convertitore secondario si utilizza il segnale di trigger **ADC_TRIG0[8]**.

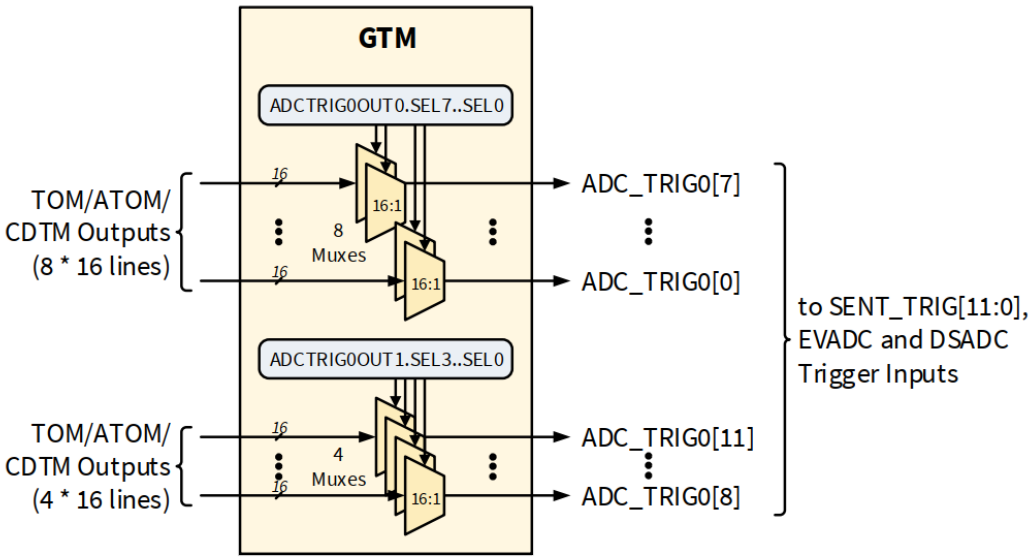


Figura 3.9: Collegamenti dedicati tra moduli GTM ed EVADC

Capitolo 4

Sviluppo firmware

4.1 Panoramica

4.1.1 Premessa

Rispetto alla versione definitiva del firmware, si svilupperà una versione semplificata la quale prevede l'utilizzo di uno solo trasmettitore ed un sola coppia di ricevitori (un primario ed un secondario). Successivamente, le configurazioni potranno essere estese al supporto di più sensori in contemporanea.

4.1.2 Ambiente di sviluppo integrato

L'ambiente di sviluppo integrato utilizzato per la scrittura del codice sorgente ed il suo relativo debugging è AURIX™ Development Studio, un software di proprietà di Infineon realizzato appositamente per la programmazione della famiglia di microcontrollori Aurix di Infineon. Il codice è sviluppato in C, un linguaggio ampiamente diffuso nel mondo dei microcontrollori, i quali non possedendo un sistema operativo, necessitano di comandi per l'accesso diretto all'hardware. Inoltre, il C è compilato in un codice macchina molto efficiente e che si adatta bene a dispositivi che possiedono risorse limitate.

4.1.3 Gerarchia del progetto

Il progetto è preventivamente suddiviso dall'IDE in diversi file e cartelle per renderlo modulare e facilitarne la comprensione.

Sono presenti 6 file sorgenti ***CpuX_Main.c*** i quali contengono la funzione di main relativa ad ogni core della CPU. L'applicazione in esame è piuttosto esigua dal punto di vista computazionale e non richiede l'esecuzione di processi in parallelo, pertanto, verrà utilizzata solo la Cpu0 (single-core). I file ***EVADC.c***

e ***GTM_TOM_PWM.c*** contengono rispettivamente i driver per la gestione dei moduli dell'ADC e del PWM.

I file di header con estensione .h contengono invece costanti e prototipi di funzioni che vengono chiamati in file sorgenti diversi da quelli in cui è contenuta la loro definizione.

4.1.4 Livello di astrazione del codice

La programmazione del MCU può essere effettuata secondo diversi stili ognuno avente dei vantaggi e svantaggi. L'approccio più diretto, spesso definito a "basso livello" è caratterizzato da un livello di astrazione nullo e prevede la scrittura manuale di tutti i registri che coinvolgono l'applicazione. Sebbene lo stile sia molto semplice dal punto di vista sintattico, porta ad una notevole complicazione dell'aspetto gestionale: infatti, la scrittura manuale di numerosi registri, ognuno identificato dal proprio indirizzo fisico, è difficile da controllare in quanto troppo onerosa e ciò si traduce di sovente con la nascita di errori software funzionali non voluti. Inoltre, la scrittura a basso livello richiede una conoscenza approfondita dell'hardware che si dovrà programmare in quanto spesso l'attivazione di alcuni registri è vincolata da meccanismi di protezione di scrittura o è subordinata al funzionamento di altri blocchi del MCU.

Un approccio decisamente più semplice e di facile gestione, consta nell'impiego degli **iLLD**, dall'inglese **infineon Low Level Driver**, ovvero driver software che traducono un comando più complesso o ad "alto livello" in istruzioni più semplici e trasparenti al programmatore. Questo stile di programmazione richiede inizialmente di familiarizzare con strutture dati e funzioni presenti nelle librerie di Infineon, ma spesso porta alla composizione di un codice più ordinato e di facile controllo in quanto tutte le dipendenze di un registro, gli indirizzi fisici e i suoi meccanismi di protezione vengono gestiti dalla funzione predefinita stessa.

4.1.5 iLLD

Per ogni modulo sono disponibili i file sorgente degli iLLD i quali contengono codice utile a semplificare il processo di configurazione. Qui di seguito verrà elargita una breve spiegazione dei tipi di dati utilizzati per accedere e configurare le risorse hardware del MCU.

Handle

Gli iLLD contengono handle, variabili che permettono di "maneggiare" le risorse hardware del MCU. Sono presenti sotto forma di strutture, ovvero, aggregazioni

di dati eterogenei utili a gestire un certo modulo. Al loro interno possiedono generalmente uno o più puntatori ai registri fisici, fondamentali per permettere la comunicazione tra il software e l'hardware e altri campi che contengono attributi relativi alla configurazione di un determinato modulo (identificativi e opzioni di configurazione).

Strutture di configurazione

Le strutture di configurazione sono delle variabili di aggregazione contenenti un dato corrispondente ad ogni campo di un indirizzo fisico il quale gestisce le funzionalità del modulo a livello hardware. Per mezzo di queste strutture è possibile specificare, in modo ordinato, tutte le opzioni di configurazioni di una periferica che verranno coinvolte nel processo di inizializzazione.

Funzioni

Le funzioni presenti all'interno degli iLLD si adattano a numerosi campi di applicazione. Tra queste, una particolare attenzione è rivolta alle funzioni di inizializzazione, le quali avviano le risorse applicando alle handle le informazioni contenute nelle strutture di configurazione. Ove non sono presenti funzioni predefinite per la configurazione dei moduli è necessario manipolare manualmente le handle.

Registri

Un generico registro **`REG`** viene rappresentato attraverso una unione, ovvero una variabile che può contenere in momenti diversi, tipi di dato differenti. Attraverso questa astrazione il registro può essere interpretato come un valore numerico, unsigned (**`REG.U`**) o signed (**`REG.S`**), o come un insieme di campi, ovvero bit a bit (**`REG.B`**). Alcuni registri possiedono dei meccanismi di protezione che impediscono la modifica non intenzionale di alcuni campi "critici" per il funzionamento del dispositivo. La modifica del valore di un campo di un particolare registro **`REG.B.FIELD`** può avvenire solo nel caso in cui il rispettivo abilitatore **`REG.B.EN_FIELD`** sia attivo contemporaneamente alla modifica. Grazie all'interpretazione del registro come valore numerico, è possibile effettuare una scrittura contemporanea di due campi e dunque soddisfare i criteri di protezione del MCU.

Acronimi

All'interno del codice che verrà presentato nella successiva sezione, i tipi di dato e le funzioni degli iLLD sono sempre anteceduti dall'acronimo **Ifx** il quale sta per **Infineon Technologies AG**.

4.2 Realizzazione dei driver dei moduli

Seguendo le specifiche enunciate nel capitolo 3 dedicato alla ricerca, è possibile procedere con la realizzazione dei driver per il TOM e l'EVADC, ovvero i blocchi funzionali del MCU che piloteranno i trasmettitori e i ricevitori.

4.2.1 Driver dell'EVADC

Nella seguente sezione il codice relativo al driver dell'EVADC verrà presentato attingendo informazioni dalla documentazione di Infineon [3]. In particolare, di ogni riga presente nel file di sorgente ***EVADC.c*** riportato in appendice A.1 nella sua versione integrale, sarà riportato un estratto di codice arricchito delle relative spiegazioni.

Macro

Si definiscono le seguenti MACRO:

- **`GROUPS_NUM`** (l.40) per stabilire in numero di gruppi utilizzati;
- **`CHANNELS_NUM`** (l.41) per stabilire in numero di canali utilizzati;
- **`ISR_PRIORITY_ADC_1st`** (l.43) per assegnare priorità massima (**`1`**) all'interrupt richiesto dal convertitore primario;
- **`ISR_PRIORITY_ADC_2nd`** (l.44) per assegnare priorità immediatamente inferiore (**`2`**) all'interrupt richiesto dal convertitore secondario;
- **`IFX_INTERRUPT(p1,p2,p3)`** (l.47 → l.48) per comunicare alla CPU il nome delle funzioni di interrupt (**`p1`**), le quali saranno successivamente descritte, e le loro priorità (**`p3`**).

```
40 #define GROUPS_NUM              2       // Number of used groups (group 0, 8)
41 #define CHANNELS_NUM            2       // Number of used channels
42
43 #define ISR_PRIORITY_ADC_1st    1       // Interrupt priority number 1st group
44 #define ISR_PRIORITY_ADC_2nd    2       // Interrupt priority number 2nd group
45
46 // Macro to define the Interrupt Service Routine
47 IFX_INTERRUPT(interruptADC_1st, 0, ISR_PRIORITY_ADC_1st);
48 IFX_INTERRUPT(interruptADC_2nd, 0, ISR_PRIORITY_ADC_2nd);
```

Variabili globali

La direttiva **extern** permette di utilizzare le variabili globali definite nel file sorgente principale ***Cpu0_Main.c*** in altri file sorgenti. In questo modo è possibile aggiornare il valore dei vettori **ticks_1st[]** (l.55) e **buffer_1st[]** (l.56), i quali contengono rispettivamente i campioni prelevati dall'ADC del gruppo primario e gli istanti di tempo in cui vengono campionati, attraverso la routine di interrupt che non può restituire alcun valore essendo una funzione di tipo **void**. Allo stesso modo è definito come **extern** il flag **end_acquisition_1st** (l.57), il cui compito è quello di segnalare il riempimento del buffer primario di acquisizione non appena il contatore **j** (l.58) raggiunga la soglia prevista. La variabile **Ifx_EVADC_G_RES_1st** (l.59) permette infine di accedere al registro dei risultati del convertitore ed è dichiarata come variabile globale in quanto deve essere visibile alla funzione di interrupt.

Analogamente vengono inizializzate le variabili relative al gruppo di convertitori secondario (l.62 → l.66). Le strutture (l.69 → l.71) svolgono le funzioni di handle del modulo, gruppo e canale dell'ADC e sono globali in quanto memorizzano il valore dei registri dell'EVADC. Le ultime due strutture hanno un formato vettoriale in quanto devono contenere rispettivamente la configurazione di due gruppi e due canali.

```
54 // 1st group variable
55 extern uint64 ticks_1st[N_SAMPLES];              // sample time instants
56 extern uint32 buffer_1st[N_SAMPLES];             // sample buffer
57 extern int end_acquisition_1st;                  // buffer is full
58 uint32  j = 0;                                  // buffer counter
59 Ifx_EVADC_G_RES conversionResult_1st;           // EVADC result register variable
60
61 // 2nd group variable
62 extern uint64 ticks_2nd[N_SAMPLES];              // sample time instants
63 extern uint32 buffer_2nd[N_SAMPLES];             // sample buffer
64 extern int end_acquisition_2nd;                  // buffer is full
65 uint32  k = 0;                                  // buffer counter
66 Ifx_EVADC_G_RES conversionResult_2nd;           // EVADC result register variable
67
68 // EVADC handles
69 IfxEvadc_Adc         g_evadc;                   // EVADC module  handle variable
70 IfxEvadc_Adc_Group   g_adcGroup[GROUPS_NUM];    // EVADC group   handle variable
71 IfxEvadc_Adc_Channel g_adcChannel[CHANNELS_NUM]; // EVADC channel handle variable
```

Configurazione dell'EVADC

La configurazione dell'EVADC è effettuata nella funzione **initEVADC()** ed è divisa in cinque fasi:

› configurazione del modulo dell'EVADC (l.89);

› configurazione del gruppo dell'EVADC (l.90);

› configurazione dei canali dell'EVADC (l.91);

› configurazione del Request Timer (l.92).

› configurazione della Queued Request Source (l.93).

```
87  void initEVADC(void)
88  {
89      initEVADCModule();           // Initialize the EVADC module
90      initEVADCGroup();            // Initialize the EVADC group
91      initEVADCChannels();         // Initialize the channel
92      configRequestTimer();        // Configure  the Request Timer
93      configQueueRequestSource();  // Configure  the Queued Request Source
94  }
```

Configurazione del modulo dell'EVADC

Come visibile nell'estratto di codice in basso, la configurazione di default del modulo EVADC, data dagli iLLD viene impiegata inizializzando un'istanza della struttura **`IfxEvadc_Adc_Config`** (l.100) e applicando i valori predefiniti ai suoi campi attraverso la funzione **`IfxEvadc_Adc_initModuleConfig()`** (l.101), fatta eccezione per i campi **`supplyVoltage`** (l.104) e **`startupCalibrationControl`** (l.106). Successivamente, la configurazione può essere applicata al modulo EVADC con la funzione **`IfxEvadc_Adc_initModule()`** (l.109).

```
97  void initEVADCModule(void)
98  {
99      // Create and initializes module configuration with default values
100     IfxEvadc_Adc_Config adcConfig;
101     IfxEvadc_Adc_initModuleConfig(&adcConfig, &MODULE_EVADC);
102
103     // Need 5V for reducing the sampling time of converters
104     adcConfig.supplyVoltage = IfxEvadc_SupplyVoltageLevelControl_upperVoltage;
105     // Start up calibration takes 36.7 us (only at power on)
106     adcConfig.startupCalibrationControl = TRUE;
107
108     // Initialize module
109     IfxEvadc_Adc_initModule(&g_evadc, &adcConfig);
110 }
```

Configurazione del gruppo dell'EVADC

La configurazione dei gruppi EVADC avviene inizializzando un'istanza vettoriale della struttura **IfxEvadc_Adc_GroupConfig** (l.116). I valori di default del gruppo primario sono impostati tramite la funzione **IfxEvadc_Adc_initGroupConfig()** (l.120) e in seguito si modificano i seguenti campi:

- **groupId** (l.121) - per selezionare il gruppo di convertitori da configurare;
- **master** (l.122) - per indicare quale convertitore è il master (in questo caso vi è un solo convertitore per gruppo che è anche il master);
- **arbiter** (l.123) - una sotto-struttura per abilitare le QRS desiderate;
- **queueRequest[0]** (l.124) - una sotto-struttura per gestire il segnale di gating della QRS0;
- **calibrationSampleTimeControlMode** (l.125) - per selezionare il tempo aggiuntivo per il campionamento;
- **disablePostCalibration** (l.126) - per disabilitare la post-calibrazione;
- **inputBufferEnabled** (l.127) - per abilitare il buffer di input;
- **inputClass[0]** (l.128 → l.129) - una sotto-struttura che gestisce il tempo di campionamento minimo del segnale (**sampleTime**) e la riduzione del rumore (**conversionMode**).

Successivamente, la configurazione viene applicata tramite la funzione **IfxEvadc_Adc_initGroup()** (l.130). Nella struttura di configurazione non è presente un campo per la gestione del fattore **DIVA** del prescaler il quale genera la frequenza di funzionamento interna dell'ADC, pertanto, è necessario modificare il valore del registro manualmente sfruttando la handle del gruppo primario **g_adcGroup[0]** (l.131) dichiarata come variabile globale.

```
113 void initEVADCGroup(void)
114 {
115     // Create and initializes group configuration with default values
116     IfxEvadc_Adc_GroupConfig adcGroupConfig[GROUPS_NUM];
117
118     // Primary group
119
120     IfxEvadc_Adc_initGroupConfig(&adcGroupConfig[0], &g_evadc);
121     adcGroupConfig[0].groupId = IfxEvadc_GroupId_0;
122     adcGroupConfig[0].master = IfxEvadc_GroupId_0;
123     adcGroupConfig[0].arbiter.requestSlotQueue0Enabled = TRUE;
124     adcGroupConfig[0].queueRequest[0].triggerConfig.gatingMode =
↳    IfxEvadc_GatingMode_always;
```

```
125     adcGroupConfig[0].calibrationSampleTimeControlMode =
    ↪ IfxEvadc_CalibrationSampleTimeControl_4;
126     adcGroupConfig[0].disablePostCalibration = TRUE;
127     adcGroupConfig[0].inputBufferEnabled = FALSE;
128     adcGroupConfig[0].inputClass[0].sampleTime = 100.0e-9;
129     adcGroupConfig[0].inputClass[0].conversionMode =
    ↪ IfxEvadc_ChannelNoiseReduction_standardConversion;
130     IfxEvadc_Adc_initGroup(&g_adcGroup[0], &adcGroupConfig[0]);
131     g_adcGroup[0].group->ANCFG.B.DIVA=2;
132
133     // Secondary group
134
135     IfxEvadc_Adc_initGroupConfig(&adcGroupConfig[1], &g_evadc);
136     adcGroupConfig[1].groupId = IfxEvadc_GroupId_8;
137     adcGroupConfig[1].master = IfxEvadc_GroupId_8;
138     adcGroupConfig[1].arbiter.requestSlotQueue0Enabled = TRUE;
139     adcGroupConfig[1].queueRequest[0].triggerConfig.gatingMode =
    ↪ IfxEvadc_GatingMode_always;
140     adcGroupConfig[1].calibrationSampleTimeControlMode =
    ↪ IfxEvadc_CalibrationSampleTimeControl_4;
141     adcGroupConfig[1].disablePostCalibration = TRUE;
142     adcGroupConfig[1].inputBufferEnabled = FALSE;
143     adcGroupConfig[1].inputClass[0].sampleTime = 500.0e-9;
144     adcGroupConfig[1].inputClass[0].conversionMode =
    ↪ IfxEvadc_ChannelNoiseReduction_standardConversion;
145     IfxEvadc_Adc_initGroup(&g_adcGroup[1], &adcGroupConfig[1]);
146     g_adcGroup[1].group->ANCFG.B.DIVA=2;
147 }
```

Analogamente è effettuata la configurazione del gruppo secondario (l.135 → l.146) la quale differisce esclusivamente per il parametro **`sampleTime`** (l.143) a causa delle caratteristiche hardware del convertitore.

Configurazione dei canali dell'EVADC

La configurazione dei canali EVADC viene eseguita inizializzando un'istanza vettoriale della struttura **`IfxEvadc_Adc_ChannelConfig`** (l.153). I valori di default del canale del gruppo primario sono impostati tramite la funzione **`IfxEvadc_Adc_initChannelConfig()`** (l.157) e in seguito si modificano i seguenti campi:

- **`resultServProvider`** (l.159) – per designare la CPU0 come gestore dell'interrupt;
- **`resultPriority`** (l.161) – per assegnare la priorità dell'interrupt;
- **`channelId`** (l.163) – per selezionare il canale da configurare;
- **`resultRegister`** (l.164) – per indicare il registro nel quale è memorizzato il valore di conversione A/D.

Infine, la configurazione viene applicata nuovamente al canale con la funzione **`IfxEvadc_Adc_initChannel()`** (l.166).

```
150 void initEVADCChannels(void)
151 {
152     // Create and initializes channel configuration with default values
153     IfxEvadc_Adc_ChannelConfig adcChannelConfig[CHANNELS_NUM];
154 
155     // Primary group
156 
157     IfxEvadc_Adc_initChannelConfig(&adcChannelConfig[0], &g_adcGroup[0]);
158     // Assign the CPU0 as service provider for the interrupt
159     adcChannelConfig[0].resultServProvider = IfxSrc_Tos_cpu0;
160     // Set maximum priority for the ADC result event interrupt
161     adcChannelConfig[0].resultPriority = ISR_PRIORITY_ADC_1st;
162     // Select the channel ID and the respective result register
163     adcChannelConfig[0].channelId = IfxEvadc_ChannelId_0;
164     adcChannelConfig[0].resultRegister = IfxEvadc_ChannelResult_0;
165     // Initialize the channel
166     IfxEvadc_Adc_initChannel(&g_adcChannel[0], &adcChannelConfig[0]);
167 
168     // Secondary group
169 
170     IfxEvadc_Adc_initChannelConfig(&adcChannelConfig[1], &g_adcGroup[1]);
171     // Assign the CPU0 as service provider for the interrupt
172     adcChannelConfig[1].resultServProvider = IfxSrc_Tos_cpu0;
173     // Set maximum priority for the ADC result event interrupt
174     adcChannelConfig[1].resultPriority = ISR_PRIORITY_ADC_2nd;
175     // Select the channel ID and the respective result register
176     adcChannelConfig[1].channelId = IfxEvadc_ChannelId_0;
177     adcChannelConfig[1].resultRegister = IfxEvadc_ChannelResult_0;
178     // Initialize the channel
179     IfxEvadc_Adc_initChannel(&g_adcChannel[1], &adcChannelConfig[1]);
180 }
```

Analogamente è effettuata la configurazione del canale del gruppo secondario (l.170 → l.179).

Configurazione del Request Timer

Per quanto concerne il Request Timer, non è presente una funzione predefinita appartenente agli iLLD che possa configurarlo interamente, pertanto, ogni modifica ai registri deve essere effettuata manualmente sfruttando la handle dei gruppi **`g_adcGroup`**. In aggiunta, il modulo è protetto, in alcuni suoi registri, da criteri di protezione sulla scrittura. A tale scopo è necessario innanzitutto dichiarare un'istanza del registro 3.3 **`REQTM`** delle modalità del Request Timer rappresentato dall'unione **`Ifx_EVADC_G_Q_REQTM`** (l.185) alla quale viene assegnata tramite il puntatore dei registri di gruppo **`group`** della handle di gruppo **`g_adcGroup[0]`** (sarebbe

possibile utilizzare anche la handle del gruppo secondario) il valore corrente dei registri del Request Timer della QRS0 **`Q[0].REQTM.U`** (l.186). Successivamente si modificano bit a bit i valori dei seguenti campi dell'unione:

- **`reqtm.B.SEQMOD`** (l.187) - per impostare la modalità del timer su "Pause after each conversion";
- **`reqtm.B.SEQTIMSET`** (l.188) - per impostare il periodo del timer;
- **`reqtm.B.SEQTIMOFF`** (l.189) - per impostare la soglia di spegnimento del segnale di trigger generato dal RT;
- **`reqtm.B.ENTR`** (l.190) - per abilitare il trigger esterno del Request Timer.

Infine, i campi del registro **`REQTM`** vengono aggiornati contemporaneamente sia per il gruppo primario che per il secondario essendo le due configurazioni identiche (l.191 → l.192).

```
183 void configRequestTimer(void)
184 {
185     Ifx_EVADC_G_Q_REQTM reqtm;
186     reqtm.U                 = g_adcGroup[0].group->Q[0].REQTM.U;
187     reqtm.B.SEQMOD          = 1;  // sequence mode
188     reqtm.B.SEQTIMSET       = 20; // set value for the timer (20x0.1us=2us -> 500kHz)
189     reqtm.B.SEQTIMOFF       = 0;  // switch off value for the timer (no switch off)
190     reqtm.B.ENTR            = 1;  // enable external trigger for the timer
191     g_adcGroup[0].group->Q[0].REQTM.U = reqtm.U;
192     g_adcGroup[1].group->Q[0].REQTM.U = reqtm.U;
193 }
```

Configurazione della Queued Request Source

Analogamente al RT si interviene sul registro 3.4 **`QCTRL`** delle sorgenti del RT per configurare la QRS0 dei rispettivi gruppi. I campi modificati servono:

- **`qctrl.B.TMWC`** (l.200) - per abilitare la scrittura sul campo **`TMEN`**;
- **`qctrl.B.TMEN`** (l.201) - per abilitare il timer mode;
- **`qctrl.B.XTWC`** (l.202) - per abilitare la scrittura sui campi **`TRSEL`**, **`XTSEL`**, **`XTMODE`**;
- **`qctrl.B.XTMODE`** (l.203) - per impostare il trigger esterno del timer sul fronte di salita;
- **`qctrl.B.XTSEL`** (l.204) - per impostare la sorgente del trigger esterno;

› **qctrl.B.TRSEL** (l.205) - per impostare l'ingresso del trigger esterno.

Infine, i campi del registro **QCTRL** vengono aggiornati contemporaneamente sia per il gruppo primario che per il secondario essendo le due configurazioni identiche (l.206 → l.207). L'attivazione del trigger esterno sulla QRS0 viene effettuata da due scritture manuali sul campo **ENTR** del registro 3.2 **QMR** delle modalità della QRS (l.210 → l.211).

```
196 void configQueueRequestSource(void)
197 {
198     Ifx_EVADC_G_Q_QCTRL qctrl;
199     qctrl.U                 = g_adcGroup[0].group->Q[0].QCTRL.U;
200     qctrl.B.TMWC            = 1; // enable writing on TMEN field
201     qctrl.B.TMEN            = 1; // enable timer mode
202     qctrl.B.XTWC            = 1; // enable writing on TRSEL, XTSEL, XTMODE fields
203     qctrl.B.XTMODE          = 2; // set trigger on rising edge (10)B
204     qctrl.B.XTSEL           = 8; // set trigger on GTMADC0TRIG0 (1000)B
205     qctrl.B.TRSEL           = 0; // trigger is selected by XTSEL
206     g_adcGroup[0].group->Q[0].QCTRL.U = qctrl.U;
207     g_adcGroup[1].group->Q[0].QCTRL.U = qctrl.U;
208
209     // Queued Request Source can be triggered by Request Timer
210     g_adcGroup[0].group->Q[0].QMR.B.ENTR = 1;
211     g_adcGroup[1].group->Q[0].QMR.B.ENTR = 1;
212 }
```

Funzione per avviare l'acquisizione del buffer

Prima dell'acquisizione del buffer è necessario azzerare i contatori **j**, **k** e i flag **end_acquisition_1st**, **end_acquisition_2nd** (l.218 → l.221) di riempimento dei due buffer. Tramite la handle di gruppo è necessario modificare manualmente il campo **EXTR** (l.226) del registro 3.1 **QINR** il quale stabilisce che una richiesta di conversione da parte di un canale venga avviata solo dopo la ricezione di un evento di trigger esterno (in questo caso dal RT). Infine si riempie la coda del buffer della QRS0 chiamando la funzione **IfxEvadc_Adc_addToQueue()** (l.229) la quale applica la condizione di REFILL. Questa opzione permette di ricaricare la QRS con la stessa sequenza di conversioni effettuando delle letture ripetute dallo stesso canale (autoscan).

```
215 void initAcquisitionBuffer()
216 {
217     // Reset the counter and the flag
218     j = 0;
219     k = 0;
220     end_acquisition_1st = 0;
221     end_acquisition_2nd = 0;
```

```
222
223     uint8 grp;
224     for(grp = 0; grp < GROUPS_NUM; grp++){
225         // A conversion request is only issued by a trigger event
226         g_adcGroup[grp].group->Q[0].QINR.B.EXTR = 1;
227
228         // Set autoscan mode for the channel
229         IfxEvadc_Adc_addToQueue(&g_adcChannel[grp], IfxEvadc_RequestSource_queue0,
    ↪  IFXEVADC_QUEUE_REFILL);
230     }
231 }
```

Routine di interrupt

La routine di interrupt contiene le operazioni che la CPU0 deve effettuare quando viene generato un interrupt a seguito della scrittura di un nuovo valore sul registro dei risultati dell'ADC. Le istruzioni differiscono a seconda del gruppo in esame in quanto scrivono su variabili diverse: per semplicità verrà descritta solo la routine di interrupt del gruppo primario, mentre, la seconda può essere visionata in appendice A.1.

Una condizione **`if then else`** discrimina se il buffer ha raggiunto il livello di riempimento massimo. In caso contrario preleva il valore dal timer di sistema **STM** attraverso la funzione **`IfxStm_get()`** e la scrive sul vettore della base tempi **`ticks_1st[]`** (l.242). Tramite la funzione **`IfxEvadc_Adc_getResult`** (l.245) si preleva il risultato della nuova conversione che ha scaturito l'evento di trigger e lo si memorizza nel buffer dei campioni **`buffer_1st[]`** (l.248). Infine il contatore **`j`** (l.251) viene incrementato. Nel caso in cui il buffer sia pieno il flag di riempimento **`end_acquisition_1st`** (l.256) deve essere impostato e il contenuto della QRS0 è svuotato tramite la funzione **`IfxEvadc_Adc_clearQueue`** (l.259). Al termine, il flag di interrupt generato dalla scrittura di un nuovo valore sul registro dei risultati viene preventivamente azzerato attraverso la scrittura manuale sul campo **`REV0`** (l.262) del registro **`REFCLR`** per evitare che la routine di interrupt venga richiamata erroneamente.

```
234 void interruptADC_1st(void)
235 {
236     // when buffer is not full
237     if (j<N_SAMPLES){
238         // interrupts are generated continuously even if flag REV0 is kept high
239         // there is no need to reset it
240
241         // take the time
242         ticks_1st[j] = (uint64)IfxStm_get(IFXSTM_DEFAULT_TIMER) & TIME_INFINITE;
```

```
243
244         // access to ADC result register
245         conversionResult_1st = IfxEvadc_Adc_getResult(&g_adcChannel[0]);
246
247         // save the result in the buffer
248         buffer_1st[j] = (uint32)conversionResult_1st.B.RESULT;
249
250         // increment the counter
251         j = j+1;
252     }
253     // when buffer is full
254     else {
255         // trigger the main function write_files()
256         end_acquisition_1st = 1;
257
258         // clearing the queue no entry can issue a conversion and no result event
    ↪  can trigger a new interrupt
259         IfxEvadc_Adc_clearQueue(&g_adcGroup[0], IfxEvadc_RequestSource_queue0);
260
261         // clearing REV0 avoids running the interrupt one last time if still active
262         g_adcGroup[0].group->REFCLR.B.REV0 = 1;
263     }
264 }
```

4.2.2 Driver del TOM

Nella seguente sezione il codice relativo al driver del TOM verrà presentato attingendo informazioni dalla documentazione di Infineon [1]. In particolare, di ogni riga presente nel file di sorgente ***GTM_TOM_PWM.c*** riportato in appendice A.2 nella sua versione integrale, sarà riportato un estratto di codice arricchito delle relative spiegazioni.

Macro

Si definiscono le seguenti MACRO:

- **`TOM_OUTPUT_PIN`** (l.40) una struttura che identifica gli output del TOM i cui campi collezionano diversi attributi quali l'unità (**`tom`**), il canale (**`channel`**) e il pin di output (**`pin`**) che verranno utilizzati in fase di configurazione;
- **`PWM_PERIOD`** (l.41) per selezionare il numero di colpi di clock che scandiscono un periodo di oscillazione del segnale PWM;
- **`DUTY_CYCLE`** (l.42) per selezionare il numero di colpi di clock in cui il segnale è al livello logico alto (DUTY CYCLE).

```
40 #define TOM_OUTPUT_PIN      IfxGtm_TOM0_13_TOUT15_P00_6_OUT
41 #define PWM_PERIOD          1000   // PWM period for the TOM (in ticks)
   ↳  100MHz / 100kHz = 1000 ticks
42 #define DUTY_CYCLE          500   // PWM DC     for the TOM (in ticks)
```

Variabili globali

La struttura **`IfxGtm_Tom_Pwm_Config`** (l.47) ha la funzione di collezionare nei suoi campi i parametri di configurazione per il modulo TOM il cui stato è descritto dalla variabile globale **`IfxGtm_Tom_Pwm_Driver`** (l.48).

```
47 IfxGtm_Tom_Pwm_Config g_tomConfig;  // Timer configuration structure
48 IfxGtm_Tom_Pwm_Driver g_tomDriver;  // Timer Driver structure
```

Configurazione del PWM

La configurazione del TOM avviene chiamando la funzione di inizializzazione **`init GtmTomPwm()`** (l.54), la quale preliminarmente:

- abilita il modulo GTM tramite la funzione **`IfxGtm_enable()`** (l.57);
- abilita gli orologi FXU tramite la funzione **`IfxGtm_Cmu_enableClocks()`** (l.58).

Successivamente, la funzione **IfxGtm_Tom_Pwm_initConfig()** (l.61) crea un'istanza della struttura **IfxGtm_Tom_Pwm_Config** con i suoi valori predefiniti. Prima di inizializzare il modulo TOM che funge da generatore di segnali PWM, alcuni parametri della struttura di configurazione sono modificati al fine di:

- **tom** (l.63) – selezionare il TOM operativo;
- **tomChannel** (l.64) – selezionare il canale del TOM che piloterà il pin d'uscita;
- **period** (l.65) – impostare il periodo per il segnale PWM al valore desiderato;
- **dutyCycle** (l.66) – impostare il duty cycle per il segnale PWM al valore desiderato;
- **pin.outputPin** (l.67) – selezionare il pin d'uscita dell'onda quadra;
- **synchronousUpdateEnable** (l.68) – abilitare l'aggiornamento sincrono del timer.

Dopo la configurazione, la funzione **IfxGtm_Tom_Pwm_init()** (l.70) inizializza e attiva il TOM con la configurazione applicata. Infine, il modulo è avviato con la funzione **IfxGtm_Tom_Pwm_start()** (l.71).

Le funzioni predefinite degli iLLD **IfxGtm_Trig_toEVadc** (l.74 → l.75) consentono di instradare un segnale di trigger verso i Request Timer del gruppo primario e secondario: in questo modo le acquisizioni partono nel momento esatto in cui i segnali da campionare sono disponibili.

```
54 void initGtmTomPwm(void)
55 {
56     // Enable GTM and FXU clock (fixed clock unit 100MHz)
57     IfxGtm_enable(&MODULE_GTM);
58     IfxGtm_Cmu_enableClocks(&MODULE_GTM, IFXGTM_CMU_CLKEN_FXCLK);
59 
60     // Initialize the configuration structure with default parameters
61     IfxGtm_Tom_Pwm_initConfig(&g_tomConfig, &MODULE_GTM);
62 
63     g_tomConfig.tom = TOM_OUTPUT_PIN.tom;            // Select the TOM depending on
   ↪ the TOM_OUTPUT_PIN
64     g_tomConfig.tomChannel = TOM_OUTPUT_PIN.channel; // Select the channel
   ↪ depending on the TOM_OUTPUT_PIN
65     g_tomConfig.period = PWM_PERIOD;                 // Set the timer period
66     g_tomConfig.dutyCycle = DUTY_CYCLE;              // Set the duty cycle
67     g_tomConfig.pin.outputPin = &TOM_OUTPUT_PIN;     // Set the TOM_OUTPUT_PIN port
   ↪ pin as output
68     g_tomConfig.synchronousUpdateEnabled = TRUE;     // Enable synchronous update
69 
```

```
70      IfxGtm_Tom_Pwm_init(&g_tomDriver, &g_tomConfig); // Initialize the GTM TOM
71      IfxGtm_Tom_Pwm_start(&g_tomDriver, TRUE);        // Start the PWM
72
73      // Trigger the Request Timer of the ADC
74      IfxGtm_Trig_toEVadc(&MODULE_GTM, IfxGtm_Trig_AdcGroup_0, IfxGtm_Trig_AdcTrig_0,
    ↪  IfxGtm_Trig_AdcTrigSource_tom0, IfxGtm_Trig_AdcTrigChannel_13);
75      IfxGtm_Trig_toEVadc(&MODULE_GTM, IfxGtm_Trig_AdcGroup_8, IfxGtm_Trig_AdcTrig_0,
    ↪  IfxGtm_Trig_AdcTrigSource_tom0, IfxGtm_Trig_AdcTrigChannel_13);
76  }
```

Capitolo 5

Sviluppo hardware

5.1 Panoramica

5.1.1 Premessa

Il sistema è costituito da diverse schede elettroniche dedicate ai moduli:

- dei trasmettitori o **TX**;
- dei ricevitori o **RX**;
- del microcontrollore o **MCU**.

Le schede di trasmissione e ricezione sono divise in due aree differenti che si differenziano in base alla posizione in cui verranno installate. Nell'attività svolta ci si focalizzerà sulla sezione contenente la sensoristica di trasmissione e ricezione di una delle due aree. In fase prototipale la scheda ospitante il MCU sarà la TriBoard™, una evaluation board sviluppata da Aurix per testare le funzionalità di TriCore™.

5.1.2 Ambiente di sviluppo schede

L'ambiente di progettazione adottato per la realizzazione dei PCB è Cadstar di Zuken. Si tratta di un software CAD che integra diversi tool tra cui:

- editor dello schematico;
- editor del PCB;
- editor del routing.

Tutti gli editor sono in comunicazione in modo da visionare il progetto sotto i tre punti di vista precedentemente elencati.

5.2 Schemi elettrici

5.2.1 Scheda di trasmissione

Trasmettitori

Il trasmettitore è costituito da un FET di tipo n implementato nella configurazione a source comune per assolvere la funzione di amplificatore di tensione. Come è possibile osservare dalla figura 5.1, il segnale PWM proveniente dal TOM del MCU (**TP30**) e diretto sul gate del transistore, è in grado di modulare la corrente che scorre nel canale del transistore **Q7**, limitata dalla resistenza **R272** che converte la corrente in tensione e la invia sull'antenna trasmittente (**OUT_TX12**). Il valore della suddetta resistenza è stato scelto per ottenere un fronte di salita del segnale ragionevole in quanto le due antenne (placche di metallo) formano assieme al canale di comunicazione (dielettrico) un condensatore e dunque originano assieme ad **R272** un circuito RC del primo ordine caratterizzato da una certa costante di tempo. Come espresso in formula 2.2 il guadagno in tensione della configurazione

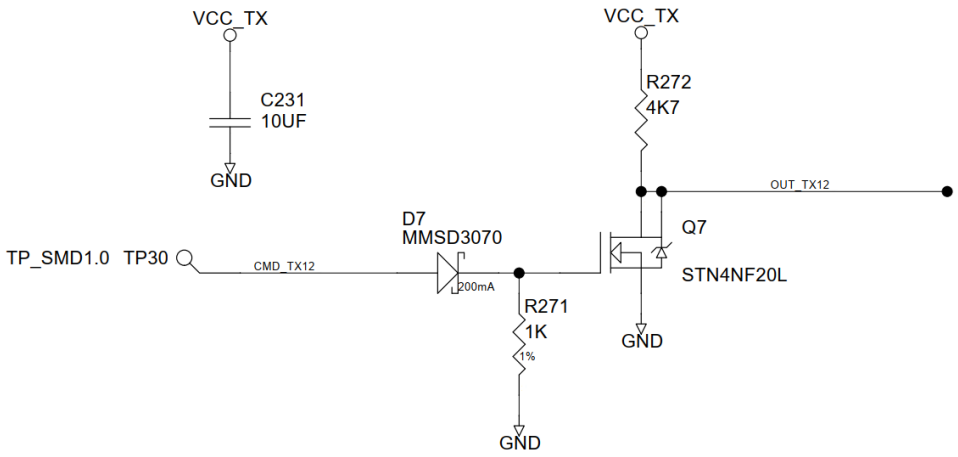


Figura 5.1: Schematico del trasmettitore

a source comune dipende dalla resistenza R_S che in tal caso è stata omessa per aumentare il guadagno (a scapito della stabilità del BIAS). Una tensione del valore di V_{CC} = 70 V è stata impiegata per garantire una potenza adeguata del segnale da trasmettere. Il transistore in esame è un particolare tipo di FET, chiamato STripFET, il quale presenta un diodo di body integrato di protezione. Un secondo diodo protettivo è collegato sul suo gate ed evita che il suo guasto possa mettere in contatto l'alta tensione con il MCU, un componente molto delicato e il cui danneggiamento comprometterebbe il funzionamento dell'intero sistema. Questo diodo Schottky è dunque dimensionato per supportare fino a 200 V di tensione in polarità

inversa garantendo un'adeguata protezione. Infine, il condensatore **C231** dal valore di 10 μF filtra eventuali disturbi presenti sulla linea di alimentazione.

5.2.2 Scheda di ricezione

Ricevitori

L'antenna ricevente connessa a **TP93** cattura il campo elettrico dalla placca trasmettitrice la quale è responsabile della generazione di una corrente. Il primo stadio, osservabile in figura 5.2, è costituito da un amplificatore di trans-resistenza che converte il segnale in ingresso da corrente a tensione tramite l'impedenza in retroazione **C34 || R50**.

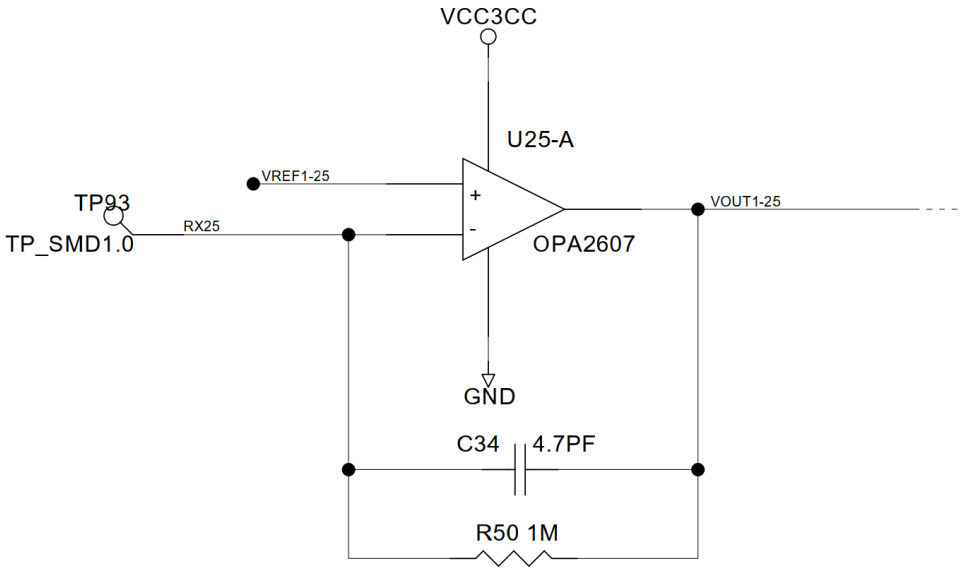


Figura 5.2: Schematico del primo stadio del ricevitore

Il secondo stadio, osservabile in figura 5.3 svolge le funzioni di amplificazione e di filtro. Si tratta di una cella a reazioni multiple del secondo ordine con configurazione in passa-banda i cui parametri principali sono definiti dalle formule 2.3 da cui si ricava:

$$\begin{aligned} f_0 &= 103.3\ kHz, \\ H_0 &= 100.0, \\ Q &= 10.2. \end{aligned} \tag{5.1}$$

La frequenza centrale del filtro f_0 e dal guadagno dell'amplificatore in continua H_0 sono stati resi modificabili a posteriori tramite l'implementazione di una resistenza di valore nullo la quale corrisponde all'aggiunta di una piazzola sul circuito stampato che potrà ospitare eventuali resistenze, **R45** e **R1**, che in base al loro valore

effettueranno un'operazione di tuning sui suddetti parametri.

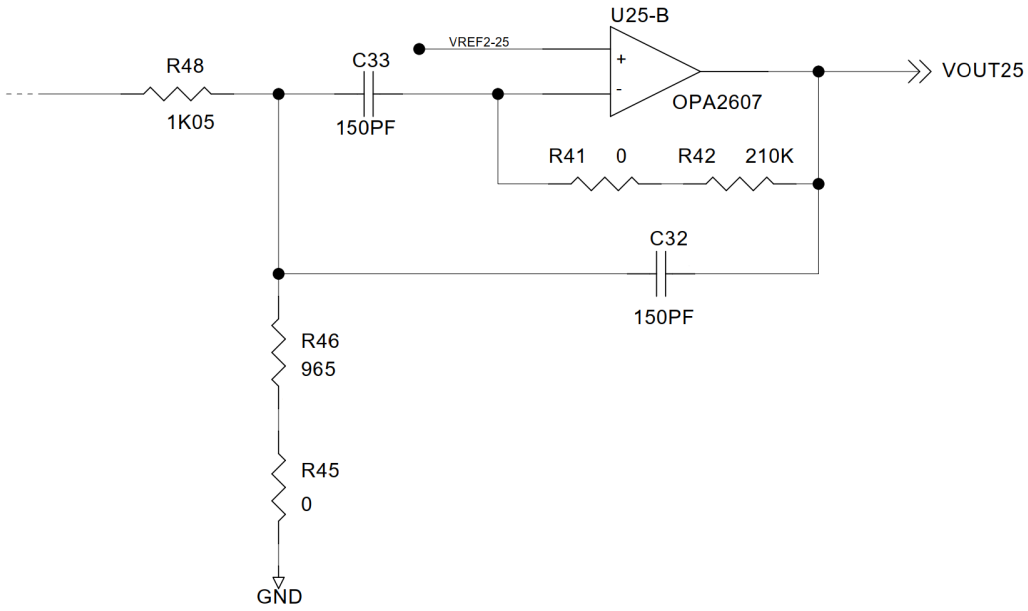


Figura 5.3: Schematico del secondo stadio del ricevitore

Filtro CLC

È un filtro simmetrico costituito da condensatori ed induttanze il quale blocca eventuali disturbi sia dall'esterno verso la scheda, dovuti ad esempio a fluttuazioni della tensione di alimentazione, sia dalla scheda verso l'esterno.

Partitori di tensione

I partitori offrono nuovi livelli di riferimento all'interno della scheda a partire dalla tensione di alimentazione.

Connettore

Attraverso il connettore la scheda di ricezione viene alimentata dall'esterno e comunica al MCU i segnali ricevuti amplificati e filtrati (**`VOUTX`**). Il connettore possiede 26 pin di cui 10 vengono connessi agli stadi di uscita dei ricevitori e altri 10 sono connessi alternatamente al riferimento in modo tale che tra ogni segnale ed il suo adiacente sia presente una pista di GND. I pin restanti restano non connessi oppure sono utilizzati come controllo elettrico per la corretta connessione dello spinotto. Maggiori dettagli sulle funzioni sopracitate verranno presentato nella sezione 5.3.5 relativa ai criteri di sicurezza.

Fiducial marker

I **Fiducial Marker** sono particolari riferimenti utilizzati dai macchinari **pick & place** per la collocazione dei componenti SMD in fase di produzione del PCB.

5.3 Printed Circuit Board

Come è possibile osservare dalla figura 5.4, i PCB delle schede di trasmissione e ricezione sono collocati secondo una disposizione spaziale che posiziona la prima al di sopra della seconda. Lo spazio interposto tra le due schede è costituito da aria e costituisce il canale di trasmissione dei segnali (freccia in nero). Su ciascuna delle due schede vengono posizionate delle antenne (viola) e dei circuiti per la trasmissione e ricezione (rosso) circa specularmente.

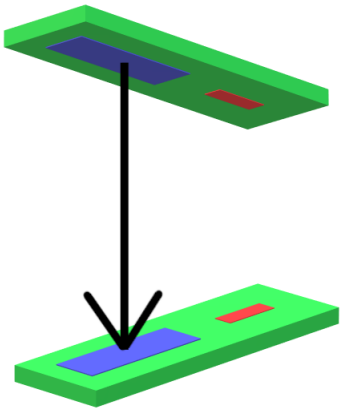


Figura 5.4: Disposizione spaziale delle schede di trasmissione e ricezione

5.3.1 Stack Layer

Come visibile in figura 5.5, il PCB di entrambe le schede è composto da quattro layer:

- layer 1 (top);
- layer 2;
- layer 3;
- layer 4 (bottom).

Il PCB viene prodotto partendo dalla porzione centrale (core), una piastra spessa 42.7 *th* in materiale isolante FR4 (dielettrico) le cui superfici superiore ed inferiore sono rivestite da due lamine in rame che costituiranno i layer 2 e 3. In entrambe le direzioni su questi ultimi si pongono successivamente nuovi strati denominati preimpregnati (prepreg), ugualmente costituiti da isolante FR4 ma dallo spessore ridotto. A differenza del core possiedono una sola superficie conduttrice il quale

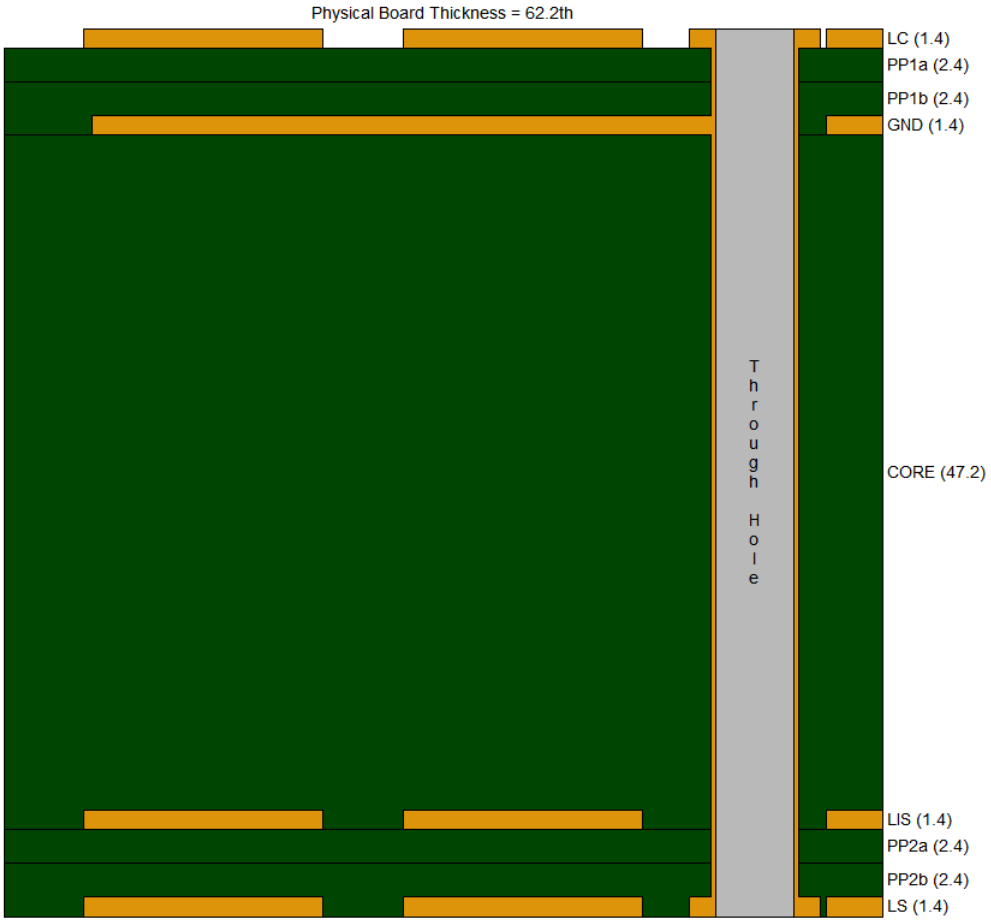


Figura 5.5: Stack Layer del PCB e relative misure in th

darà origine al layer top e bottom rispettivamente se incollato sul layer 2 o 3. All'interno del progetto Cadstar il layer top è denominato con la sigla LC o "Lato Componenti" in quanto originariamente su questo strato si era soliti collocare i componenti, mentre il layer bottom è chiamato "Lato Saldature", giacché su tale lato si effettuava la saldatura degli obsoleti componenti PTH. Per vicinanza spaziale, il layer 2 e 3 sono denominati rispettivamente LIC e LIS i quali stanno per "Lato Interno Componenti" e "Lato Interno Saldature". In questo progetto il secondo strato ha generalmente la funzione di ospitare un piano diffuso di massa, utilizzato per schermare le interferenze EMI di alcuni elementi della scheda, mentre il layer 3, è tendenzialmente utilizzato per ospitare le piste di collegamento o di alimentazione. Tuttavia, a seconda della tipologia di scheda (trasmissione o ricezione) i sopracitati layer verranno impiegati con alcune variazioni in quanto quest'ultime saranno collocate in posizioni spaziali differenti.

5.3.2 Scheda di trasmissione

Nella scheda di trasmissione i layer sono impiegati come seguentemente descritto:

- layer 1 per il piazzamento dei componenti SMD e per le piste di segnali e GND;
- layer 2 per il piano diffuso di ground;
- layer 3 per l'alimentazione ($70\,V$) dei trasmettitori;
- layer 4 per la placca trasmittente (antenna).

Al fine di ridurre l'accoppiamento e per facilitare la gestione dei guasti (vedi sezione 5.3.5 sulla sicurezza), i vari segnali nel layer top sono circondati da piste di riferimento. In aggiunta, nel layer 2 un piano di GND è collocato al di sotto delle linee di segnale e dei i componenti SMD per schermarli: per quanto concerne i segnali infatti, seppur la loro frequenza sia bassa, le piste da loro percorse sono relativamente lunghe e potrebbero fungere da antenna.

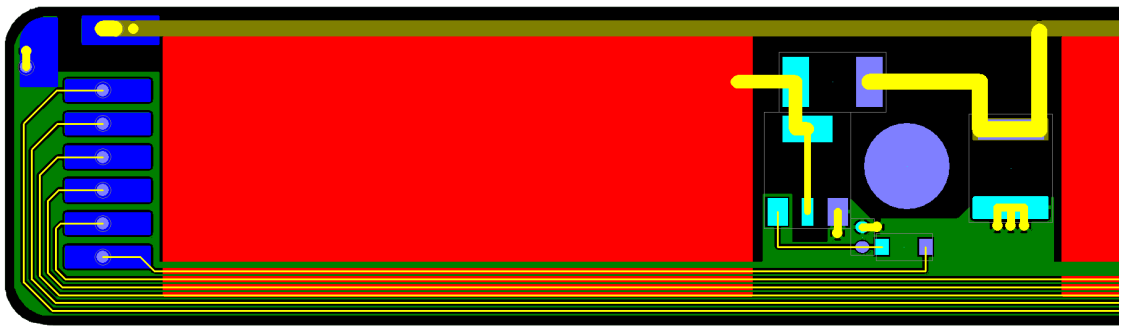


Figura 5.6: Parte del PCB della scheda di trasmissione

Il riferimento di ground è portato attraverso un filo conduttore saldato su una piazzola del layer top dal quale viene rimosso il solder layer top, e giunge al layer 2 tramite due fori di via. I fori di via sono annegati nel piano di ground diffuso. Le "croci" visualizzabili in figura 5.7 sono degli artefatti di progettazione per migliorare la qualità della saldatura. Le clearance, ovvero le zone in cui il rame non è presente, diminuiscono la superficie di conduzione e proporzionalmente la conducibilità termica che in un piano conduttore è molto elevata: questo permette di mantenere la temperatura alta ed evitare una saldatura "fredda" di scarsa qualità.

Il layer 3 è impiegato per l'alimentazione ad alta tensione delle antenne ($70\,V$). A quest'ultima non è in genere applicato uno schermo in quanto il potenziale seppur alto è costante. Eventuali oscillazioni della tensione di alimentazione sono inoltre

Figura 5.7: Struttura di un via

filtrate da un condensatore del valore di 10 μF. Le proprietà fisiche del dielettrico FR4 garantiscono inoltre che anche in condizioni di invecchiamento la rigidità elettrica del materiale non vada mai al di sotto di 100 V/mil [5]: questo evita la creazione di un arco elettrico tra i diversi layer anche in presenza di eventuali differenze di potenziale ΔV dell'ordine della tensione di alimentazione. Analogamente a quanto accade per il riferimento, anche i segnali e l'alta tensione vengono portati alla scheda attraverso la rimozione del solder (zone in blu) e la saldatura di fili conduttori sui pad del PCB.

5.3.3 Scheda di ricezione

Nella scheda di ricezione i layer sono impiegati come seguentemente descritto:

- layer 1 per la placca ricevente (antenna);
- layer 2 per il piano diffuso di ground;
- layer 3 per le piste dei segnali;
- layer 4 per il piazzamento dei componenti SMD.

In questo caso, il piano diffuso di riferimento del layer 2 è impiegato per schermare le placche riceventi (layer 1) dagli output del filtro amplificatore (piste nel layer 3) i quali raggiungono valori di 3.3 V e potrebbero fungere da piccole antenne. Analogamente alla scheda di trasmissione tra le piste di segnale nel layer 3 vengono interposte linee di GND per la gestione dei guasti (vedi sezione 5.3.5 sulla sicurezza). Come è possibile osservare dalla figura 5.8 ai bordi della scheda viene lasciato dello spazio per incollare la scheda alla macchina utensile: questo costituisce il motivo per cui il primo ricevitore presenta un'orientazione diversa. Un'altra osservazione di notevole importanza è rappresentata dalla collocazione dei circuiti di ricezione, i quali sono posti sempre alla stessa distanza dalle placche riceventi: in assenza di informazioni specifiche sugli effetti dei campi elettromagnetici, è buona norma far lavorare i componenti nelle stesse condizioni.

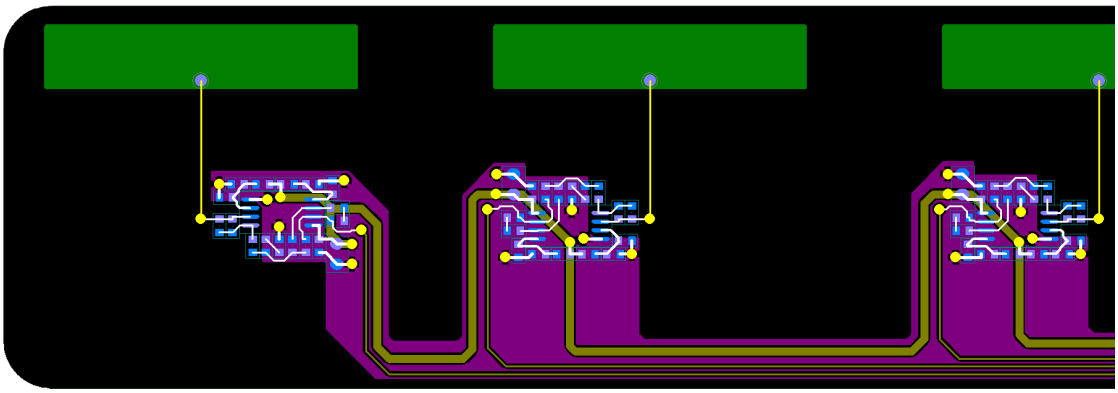
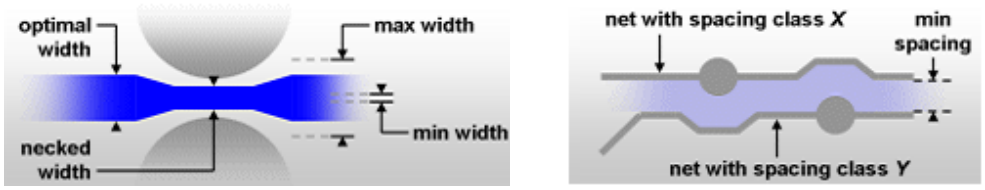


Figura 5.8: Parte del PCB della scheda di ricezione

5.3.4 Routing

Come è possibile osservare rispettivamente in figura 5.9a e 5.9b, ad ogni segnale si attribuiscono le proprietà di spessore e spaziatura. Lo spessore minimo di ogni pista è di 6 *th*, uno standard che garantisce un'impedenza controllata di 50 Ω, mentre, lo spessore massimo varia in base all'utilizzo della linea. Le piste dedicate all'alimentazione trasportano molta corrente e presentano una sezione incrementata mentre ciò non è necessario per le linee di segnale. All'interno delle larghezze massime e minime rientrano gli spessori ottimali che vengono adottati in condizioni ideali e nacked (spogliati) quando gli spazi sono ristretti.



(a) Spessore (b) Spaziatura

Figura 5.9: Proprietà delle piste per i segnali

Anche per quanto concerne la spaziatura è possibile definire un limite inferiore ma non uno superiore. All'interno del software CAD i segnali vengono catalogati per categorie e per ognuna di queste vengono definite le distanze dai segnali di tutte le altre categorie compilando delle corrispondenze come visibile in tabella 5.1.

Spacing Class	unclassed	GND	High_Voltage	Power	VOUT_18
unclassed	6	6	24	6	18
GND		ND	24	6	6
High_Voltage			24	24	24
Power				6	18
VOUT_18					18

Tabella 5.1: Tabella delle spaziature dei segnali in *th*

Prendendo ad esempio in esame i segnali di output degli amplificatori (categoria VOUT_18), questi distano 18 *th* da tutti gli altri segnali fatta eccezione per l'alta tensione e GND data la presenza di piste di riferimento tra di loro. In particolare queste ultime (categoria High_Voltage) devono mantenere una certa distanza in quanto potrebbero creare archi elettrici con layer o piste adiacenti. Questa norma è rispettata ad esempio nei collegamenti del FET inquadrati in rosso in figura 5.10 del trasmettitore dove la sua piazzola centrale (drain) è stata ridotta in larghezza per ridurre la distanza tra pista di alimentazione e le linee adiacenti.

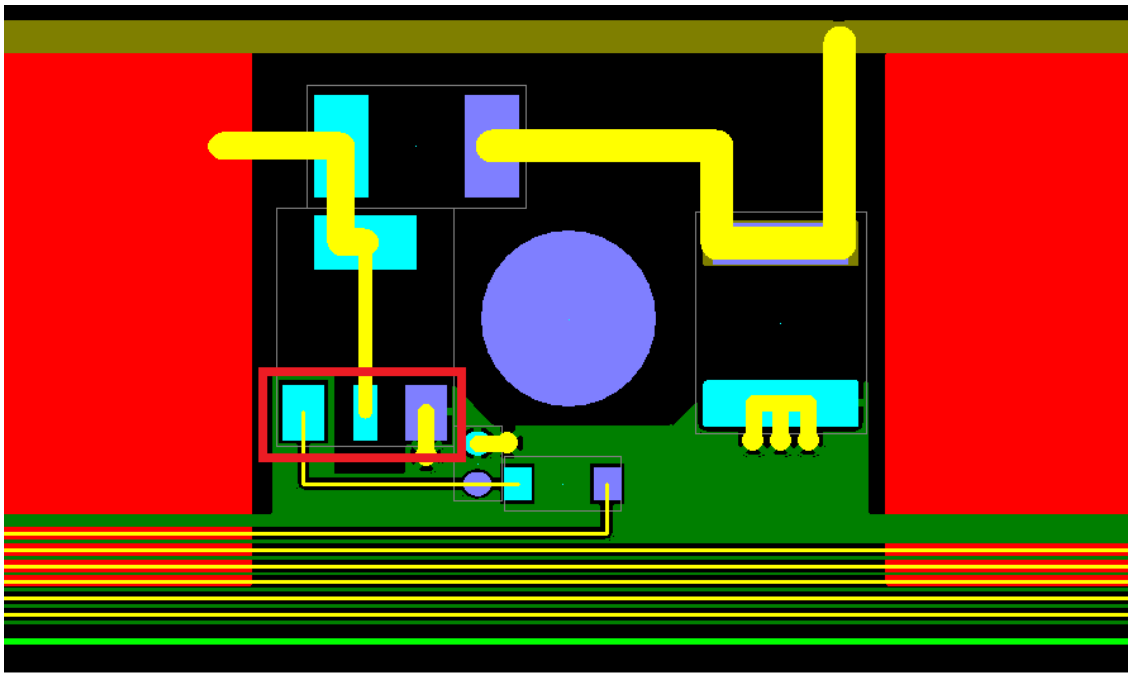


Figura 5.10: Criteri di distanziamento nel routing del PCB

5.3.5 Sicurezza

Data la natura dell'applicazione, determinate misure sono state adottate al fine di incrementare il livello di sicurezza. Per quanto concerne il punto di vista meccanico, banalmente, la forma del PCB è stata arrotondata in prossimità dei bordi per evitare spigoli vivi attraverso l'opzione "*fillet corner*" del software CAD.

Per quanto riguarda il routing dei segnali, tra di essi è presente una pista di ground: in questo modo oltre a ridurre l'accoppiamento capacitivo tra le linee si facilita la gestione dei guasti. Il malfunzionamento più noto all'interno di un dispositivo elettronico è rappresentato dall'**elettromigrazione**, un fenomeno secondo il quale il passaggio di corrente elettrica continuato può portare allo spostamento di particelle di conduttore che, depositandosi altrove, possono generare un circuito aperto o un corto circuito. Sebbene entrambi i casi conducono al malfunzionamento dell'intero sistema, in genere il secondo è più problematico in quanto crea danni maggiori e le sue cause potrebbero essere difficili da prevedere in fase di progettazione. Attraverso l'introduzione di una pista di riferimento tra ogni segnale, si riducono il numero di test da effettuare per coprire tutte le combinazioni di guasto, in quanto ogni segnale può corto-circuitarsi sempre e solo con la sua adiacente linea di ground.

Anche il connettore assolve una funzione di sicurezza, in particolare, si assicura che

il suo collegamento con lo spinotto esterno avvenga nella giusta direzione grazie ad un controllo elettrico. A tale scopo impiega due pin connessi opportunamente a VCC o GND, secondo una combinazione prescelta, il cui valore è letto dal MCU che assicura il corretto collegamento attraverso una routine software. Essendoci tre connettori nell'intero sistema, dunque, il numero minimo di pin necessari per garantire la copertura di tutti è: $2 = \lceil log_2 3 \rceil$.

Capitolo 6

Validazione prototipo

6.1 Realizzazione del prototipo

La progettazione delle schede per i trasmettitori e i ricevitori ha permesso la realizzazione di un primo prototipo del sistema osservabile in figura 6.1. Con alcuni materiali di fortuna disponibili in azienda è stato possibile disporre le due schede in una configurazione qualitativamente simile alla definitiva. Invece, come verrà spiegato di seguito, dal punto di vista quantitativo alcune grandezze fisiche come la distanza tra le due schede non rispecchia il valore reale di progetto per far fronte alla momentanea mancanza della strumentazione adeguata.

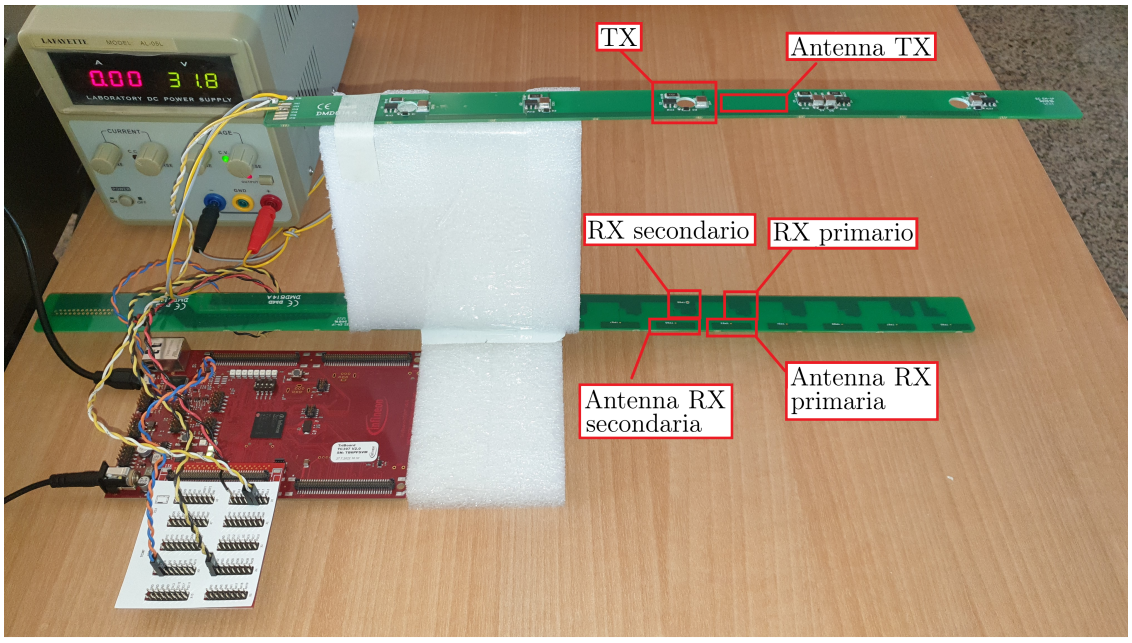


Figura 6.1: Setup del prototipo

TriBoard

Per la realizzazione del prototipo è stata impiegata una scheda AURIX™ Evaluation Board (**KIT_A2G_TC397_5V_TRB**), uno strumento versatile il quale consente, in fase prototipale, l'interfacciamento alle funzionalità dell'architettura TriCore™ del microcontrollore utilizzato. Come è possibile visualizzare in figura 6.2, la TriBoard è stata arricchita con una scheda di espansione la quale permette di avere facile accesso ai pin delle periferiche del microcontrollore. A scopo esemplificativo un jumper è stato impiegato per connettere il pin multifunzione P00.6 del modulo TOM con il pin AN0, ovvero, l'ingresso del primo canale del gruppo primario dell'ADC. La scheda è inoltre collegata ad un PC per il trasferimento dei campioni tramite una porta USB.

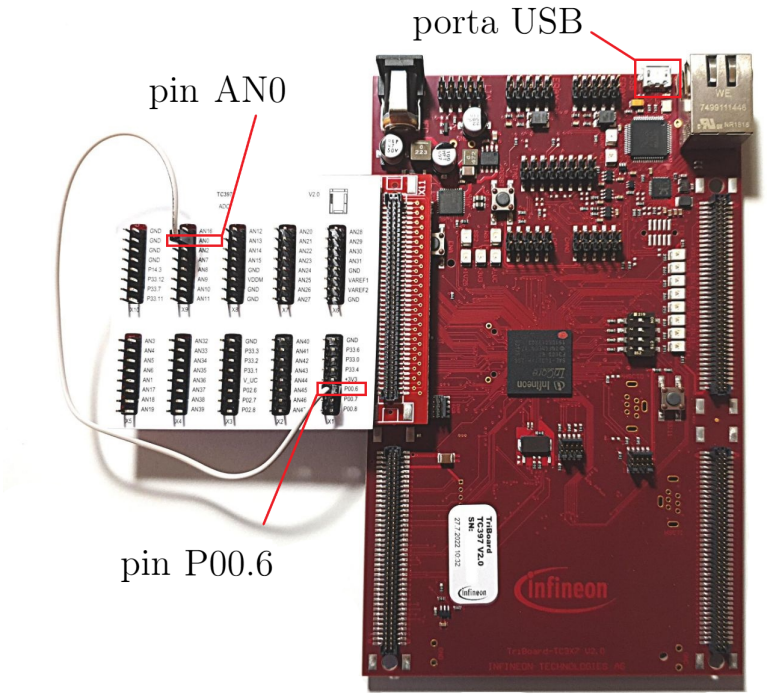


Figura 6.2: Scheda TriBoard assieme alla scheda di espansione utilizzata per il test di validazione del prototipo

Alimentatore

Un alimentatore da banco **LAFAYETTE AL-05L** è stato impiegato per fornire energia alla scheda trasmittente. Il valore massimo di tensione erogabile dal dispositivo è di 32 V, lontani dai 70 V previsti dal progetto finale; pertanto, la distanza spaziale tra le due schede è stata ridotta per non causare un eccessiva riduzione della potenza del segnale ricevuto con la conseguente non osservabilità dai risultati prodotti.

6.2 Test per la validazione del prototipo

La validazione del prototipo avviene tramite la scrittura in linguaggio C di un programma di test direttamente eseguito dalla CPU0 del microcontrollore Aurix. L'applicativo consiste nella generazione da parte del modulo TOM di un segnale PWM ad una frequenza di $f = 100\ kHz$ e la successiva acquisizione tramite l'ADC di un numero prestabilito di campioni **`N_SAMPLES = 50`** (per garantire una risoluzione in frequenza $\Delta f = 10\ kHz$) di un segnale PWM (onda quadra). Il buffer di campioni viene infine trascritto su un PC assieme agli istanti di tempo delle acquisizioni per analizzare successivamente in modalità post-processing il segnale originario tramite un applicativo MatLab.

6.2.1 CPU0

Nella seguente sottosezione il codice relativo alla CPU0 verrà presentato attingendo informazioni dalla documentazione di Infineon. In particolare, di ogni riga presente nel file di sorgente ***Cpu0_Main.c*** riportato in appendice B.1 nella sua versione integrale, sarà riportato un estratto di codice arricchito delle relative spiegazioni.

Variabili globali

La variabile globale **`IfxCpu_syncEvent`** (l.57) traccia l'evento di sincronizzazione tra i 6 core della CPU. Sebbene l'applicazione in esame sia del tipo single-core, si effettua comunque tale operazione per predisporla all'utilizzo di più core qualora l'applicazione dovesse prevedere funzionalità aggiuntive da svolgere in concorrenza. Come anticipato nel capitolo 4 relativo al firmware, le variabili del buffer di campioni del gruppo primario (l.60 → l.63) sono definite nel main per ovviare ad alcune limitazioni del debugger, il quale effettua il watching delle sole variabili presenti nel main. Analogamente sono definite le variabili del gruppo secondario (l.66 → l.69). Le variabili **`dt_in_ticks`**, **`ticks_per_us`** e **`dt_in_us`** (l.72 → l.74) sono impiegate per la deduzione della base tempi del segnale a partire dagli istanti di tempo in cui sono state effettuate le acquisizioni. Infine sono definite le variabili per la scrittura dei file in cui vengono memorizzati i campioni (l.77 → l.83) che comprendono: un indice **`i`** per i cicli for, due puntatori ai file dei campioni e dei tempi, due percorsi per ciascuno dei gruppi dei convertitori.

```
57 IFX_ALIGN(4) IfxCpu_syncEvent g_cpuSyncEvent = 0; // synch the cores
58
59 // 1st group buffer variables
60 uint64  ticks_1st[N_SAMPLES] = {0};              // sample time instants
61 float64 times_1st[N_SAMPLES] = {0};              // base time vector
62 uint32  buffer_1st[N_SAMPLES] = {0};             // sample buffer
63 int end_acquisition_1st;                         // buffer is full
64
```

```
65 // 2nd group buffer variables
66 uint64  ticks_2nd[N_SAMPLES] = {0};              // sample time instants
67 float64 times_2nd[N_SAMPLES] = {0};              // base time vector
68 uint32  buffer_2nd[N_SAMPLES] = {0};             // sample buffer
69 int end_acquisition_2nd;                         // buffer is full
70
71 // base time variables
72 uint64 dt_in_ticks = 0;                          // ticks between samples
73 uint64 ticks_per_us = 0;                         // ticks in 1 us
74 float64 dt_in_us = 0;                            // us between samples
75
76 // file variables
77 int i = 0;                                       // index for file writing
78 FILE *fp_samples;                                // sample file pointer
79 FILE *fp_times;                                  // time   file pointer
80 const char samples_path_1st[] =
   ↪ "S:/Users/.../code/SC_TC397_TOM_ADC/samples_1st/samples_1st.txt";
81 const char   times_path_1st[] =
   ↪ "S:/Users/.../code/SC_TC397_TOM_ADC/samples_1st/times_1st.txt";
82 const char samples_path_2nd[] =
   ↪ "S:/Users/.../code/SC_TC397_TOM_ADC/samples_2nd/samples_2nd.txt";
83 const char   times_path_2nd[] =
   ↪ "S:/Users/.../code/SC_TC397_TOM_ADC/samples_2nd/times_2nd.txt";
```

Funzione di main

La memorizzazione dei campioni sul buffer viene effettuata attraverso un Result Event, un evento di interrupt generato dall'EVADC sulla CPU0 quando un nuovo risultato valido viene scritto sul registro dei risultati del convertitore. A tale scopo è fondamentale abilitare gli interrupt attraverso la funzione **`IfxCpu_enableInterrupts()`** (l.96). Inoltre, si disattivano i watchdogs della CPU0 (l.99 → l.100) in quanto l'esecuzione dell'applicativo di test è molto breve e non necessita di tali meccanismi di controllo. Segue la sincronizzazione dei vari core attraverso la coppia di funzioni **`IfxCpu_emitEvent`** e **`IfxCpu_waitEvent`** (l.103 → l.104). Successivamente attraverso una funzione appartenente agli iLLD si aggiorna la variabile **`ticks_per_us`** (l.107) la quale restituisce il numero di colpi di clock presenti in un μs alla frequenza di clock del sistema. Dopo la fase preliminare è possibile inizializzare i moduli del TOM e dell'EVADC attraverso le funzioni **`initGtmTomPwm()`** e **`initEVADC`** (l.110 → l.111) le quali contengono i driver delle periferiche descritti nel capitolo 4 relativo al firmware. L'acquisizione del buffer è avviata per la prima volta attraverso la funzione **`initAcquisitionBuffer()`** (l.114) al di fuori del ciclo infinito (tipico dell'esecuzione del programma di un microcontrollore) il quale è implementato da un ciclo **`while`** la cui condizione è sempre soddisfatta (l.117). Impostando un breakpoint alla linea 122 del codice, responsabile dell'avvio di una nuova acquisizione del buffer, ed eseguendo l'applicativo di test in modalità debug è possibile:

› cancellare i file relativi alla vecchia acquisizione tramite la funzione **delete_files()** (l.120);

› scrivere su file i nuovi buffer tramite la funzione **write_files()** (l.121);

qualora entrambi i buffer dei gruppi primari e secondari siano completamente pieni e soddisfino la condizione presente all'interno dell'**if**.

```
94  void core0_main(void)
95  {
96      IfxCpu_enableInterrupts();
97
98      // DISABLING WATCHDOG0 AND SAFETY WATCHDOG
99      IfxScuWdt_disableCpuWatchdog(IfxScuWdt_getCpuWatchdogPassword());
100     IfxScuWdt_disableSafetyWatchdog(IfxScuWdt_getSafetyWatchdogPassword());
101
102     // Waiting for CPU sync event
103     IfxCpu_emitEvent(&g_cpuSyncEvent);
104     IfxCpu_waitEvent(&g_cpuSyncEvent, 1);
105
106     // Computing how many ticks in 1 us
107     ticks_per_us = IfxStm_getTicksFromMicroseconds(BSP_DEFAULT_TIMER,1);
108
109     // Initializing GTM_TOM and EVADC modules
110     initGtmTomPwm();
111     initEVADC();
112
113     // Starting the buffer acquisition
114     initAcquisitionBuffer();
115
116     // endless MCU loop
117     while(1)
118     {
119        if(end_acquisition_1st==1 && end_acquisition_2nd==1){ // if buffer is full...
120             delete_files();          // ... delete the old results on PC
121             write_files();           // ... write  the new results on PC
122             initAcquisitionBuffer(); // ... restart the acquisition
123         }
124     }
125 }
```

Funzione di scrittura

La funzione di scrittura ha il compito di trasferire sul Personal Computer connesso alla Evaluation Board il buffer di campioni acquisiti. Attraverso il primo ciclo **for** (l.133 → l.136) viene ricostruita la base tempi, ovvero gli istanti di tempo in cui vengono acquisiti i campioni. Il secondo gruppo di codice (l.139 → l.143) e il terzo (l.145 → l.149) effettuano un'operazione di scrittura sui file allocati sul PC

i quali verranno successivamente analizzati per stabilire la correttezza del flusso di trasmissione e acquisizione. Analogamente viene ripetuta la stessa sequenza di istruzioni per il convertitore del gruppo secondario (l.154 → l.171).

```
128 void write_files(void){
129 
130     // 1st group buffer
131 
132     // Reconstruct the base time
133     for(i=1; i<=N_SAMPLES; i++) {
134         dt_in_ticks = ticks_1st[i] - ticks_1st[i-1];
135         dt_in_us = (float64)dt_in_ticks/(float64)ticks_per_us;
136         times_1st[i] = dt_in_us + times_1st[i-1];
137     }
138     // Write the buffer on a file
139     fp_samples = fopen(samples_path_1st,"a");
140     for(i=0; i<N_SAMPLES; i++) {
141         fprintf(fp_samples,"%d\n", (int)buffer_1st[i]);
142     }
143     fclose(fp_samples);
144     // Write the base time on a file
145     fp_times = fopen(times_path_1st,"a");
146     for(i=0; i<N_SAMPLES; i++) {
147         fprintf(fp_times,"%f\n", times_1st[i]);
148     }
149     fclose(fp_times);
150 
151     // 2nd group buffer
152 
153     // Reconstruct the base time
154     times_2nd[0] = 0;
155     for(i=1; i<=N_SAMPLES; i++) {
156         dt_in_ticks = ticks_2nd[i] - ticks_2nd[i-1];
157         dt_in_us = (float64)dt_in_ticks/(float64)ticks_per_us;
158         times_2nd[i] = dt_in_us + times_2nd[i-1];
159     }
160     // Write the buffer on a file
161     fp_samples = fopen(samples_path_2nd,"a");
162     for(i=0; i<N_SAMPLES; i++) {
163         fprintf(fp_samples,"%d\n", (int)buffer_2nd[i]);
164     }
165     fclose(fp_samples);
166     // Write the base time on a file
167     fp_times = fopen(times_path_2nd,"a");
168     for(i=0; i<N_SAMPLES; i++) {
169         fprintf(fp_times,"%f\n", times_2nd[i]);
170     }
171     fclose(fp_times);
172 }
```

Funzione di cancellazione

La funzione di cancellazione rimuove i file agli indirizzi specificati che contengono precedenti acquisizioni relative alle esecuzioni antecedenti dell'applicativo. Essendo la funzione **write_files** impostata per scrivere in modalità "append", ovvero concatenando i nuovi dati in coda allo stesso file, tale provvedimento evita, in caso di errore da parte dell'utente, di contaminare i buffer più recenti con acquisizioni precedenti.

```
175 void delete_files(void){
176     remove(samples_path_1st);
177     remove(times_path_1st);
178     remove(samples_path_2nd);
179     remove(times_path_2nd);
180 }
```

6.3 Analisi dei dati raccolti

Attraverso la memorizzazione delle variabili **buffer_x** e **times_x** è possibile ricostruire il segnale campionato. A tale scopo è stato realizzato un applicativo Matlab, visualizzabile nella sua versione integrale in appendice C.1, il quale permette di visualizzare su una finestra plot il segnale nel dominio del tempo affiancato dalla sua trasformata di Fourier ottenuta implementando la trasformata di Fourier di un segnale discreto.

6.3.1 Implementazione

La trasformata di Fourier è implementata tramite l'algoritmo della Fast Fourier Transform (FFT), descritto dalla routine integrata **fft(x)** il cui parametro **x** rappresenta il segnale campionato nel dominio del tempo. Si noti come nel codice i vari parametri vengono denominati con un suffisso numerico il quale indica la relativa appartenenza ai convertitori del gruppo primario (**1**) e secondario (**2**).

Normalizzazione del segnale

Prima di applicare l'algoritmo della FFT vengono effettuate alcune operazioni preliminari sul segnale acquisito. Nella configurazione del prototipo, per questioni limitate dalla strumentazione disponibile in azienda, si utilizza un'alimentazione differente per la scheda ricevente e per la TriBoard. Il sistema di power management della Evaluation Board è in grado di alimentare il modulo ADC ad una tensione $V_{ADC} = 5\ V$ per rendere più veloce il tempo di campionamento, mentre, è in grado di fornire verso l'esterno (tramite la scheda di espansione) una tensione $V_{RX} = 3.3\ V$. Quest'ultima verrà utilizzata per alimentare la scheda dei ricevitori. Avendo due valori di tensione differenti per l'ADC e i ricevitori si applica una proporzione per ricostruire correttamente il fondo scala della lettura dell'ampiezza del segnale acquisito. Disponendo di registri a 12 bit per il campionamento, il valore massimo di lettura **full_scale** è definito:

$$full\_scale = 2^{12} \cdot \frac{V_{RX}}{V_{ADC}}. \tag{6.1}$$

Al segnale viene rimosso il BIAS (l.37) in quanto nell'analisi la componente continua non è importante. Successivamente viene applicata una normalizzazione rispetto al valore del fondo scala (l.38) in modo che la sua ampiezza picco-picco sia unitaria.

```
37 x1 = x1-(full_scale/2); % removing BIAS
38 x1 = x1/full_scale; % normalizing wave amplitude
```

Scalamento della trasformata

La routine integrata di Matlab **`fft()`** per il calcolo della FFT risulta così definita:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi \frac{k}{N} n}, \tag{6.2}$$

differente dalla formula 2.9 per il fattore di scala che precede la sommatoria. A tale scopo, come è possibile evincere dall'estratto di codice riportato in basso, è necessario dividere la trasformata **`X(k)`** per il numero di campioni **`N`** (l.50) al fine di ottenere un risultato significativo. Inoltre, essendo in presenza di un segnale reale, la sua trasformata di Fourier è simmetrica rispetto alla metà della frequenza di campionamento F_s: si tronca il vettore **`X(k)`** per ottenere la trasformata single-side e si moltiplica per due il suo valore per trasferire l'energia della componente rimossa su quella rimanente (l.51). Segue il calcolo del modulo della trasformata (l.52) il quale costituisce la grandezza rilevante in esame (la fase non viene analizzata dal sistema). Infine si effettua la conversione per passare dal dominio dei campioni al dominio delle frequenze (l.53) le quali saranno visualizzate sull'asse delle ascisse.

```
50 X1 = fft(x1)/N1; % computing the FFT
51 X1 = 2*X1(1:N1/2); % truncating due to symmetry
52 Mag1 = abs(X1); % FFT amplitude
53 f1 = (0:N1/2-1)/N1*fs; % from bins to frequency
```

6.3.2 Scenari

Il prototipo viene sottoposto a tre diversi scenari d'utilizzo per stabilire l'efficacia del suo principio di funzionamento. È previsto l'utilizzo di un trasmettitore a frequenza $f = 100\ kHz$ e di due ricevitori adiacenti appartenenti ad un gruppo primario e secondario: il primo verrà ostacolato dalla presenza di un certo materiale, mentre, il secondo sarà lasciato libero. Per ogni caso l'intensità del segnale in ricezione ai due convertitori viene valutata in base al modulo del picco della FFT corrispondente alla frequenza del segnale trasmesso. I tre scenari verranno denominati:

- **riposo**, condizione nella quale l'operatore non è prossimo alla macchina utensile;
- **lavoro**, situazione in cui l'operatore introduce del materiale all'interno dell'area di lavoro;
- **pericolo**, frangente in cui una parte del corpo dell'operatore si trovi all'interno della zona di pericolo.

6.3.3 Risultati

Di seguito si riportano i risultati per i tre diversi scenari descritti nella sezione precedente.

Riposo

Durante il riposo l'aria costituisce l'unico elemento interposto tra le schede di trasmissione e ricezione. Il modulo del picco della FFT registrato in questa condizione viene catalogato come valore di riferimento per le casistiche successive. Come è possibile evincere dalla figura 6.3 il segnale campionato, in origine un'onda quadra, è circa diventato una sinusoide: questo è dovuto ai vari componenti delle schede di trasmissione e ricezione che agiscono tutti, come ogni componente reale, da filtro passa basso. In termini minori è responsabile la limitatezza nel tempo del segnale acquisito che dunque non può avere una banda illimitata tipica delle onde quadre.

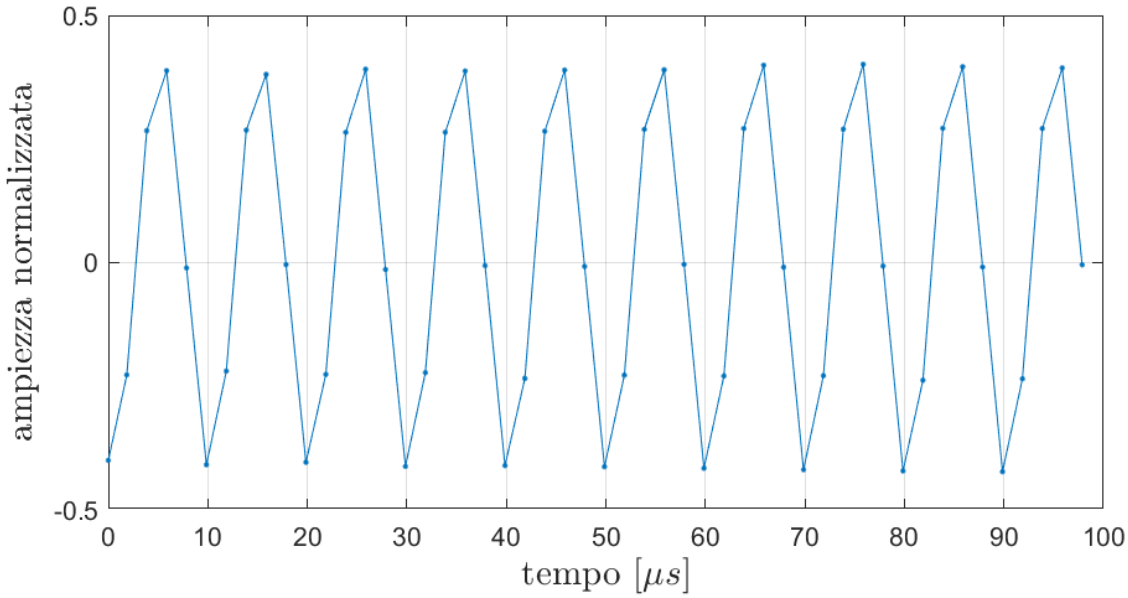


Figura 6.3: Segnale campionato dal convertitore del gruppo primario

Dallo spettro di frequenza, visibile in figura 6.4, è possibile osservare un picco relativo alla frequenza originaria del segnale di $f = 100\ kHz$. Tuttavia, non sono riportate le armoniche dispari tipiche di un'onda quadra in quanto la finestra utili di visibilità è limitata dalla frequenza di campionamento a $F_s/2 = 250\ kHz$. Dalla comparazione dei dati ottenuti dai convertitori del gruppo primario e secondario, riportati in tabella 6.1 è possibile affermare che questi ultimi siano equivalenti dal punto di vista operativo. È ammesso dunque l'utilizzo dei convertitori secondari con le stesse performance dei primari al fine di sfruttare la loro superiorità numerica che meglio può adattarsi alle configurazioni multi canale.

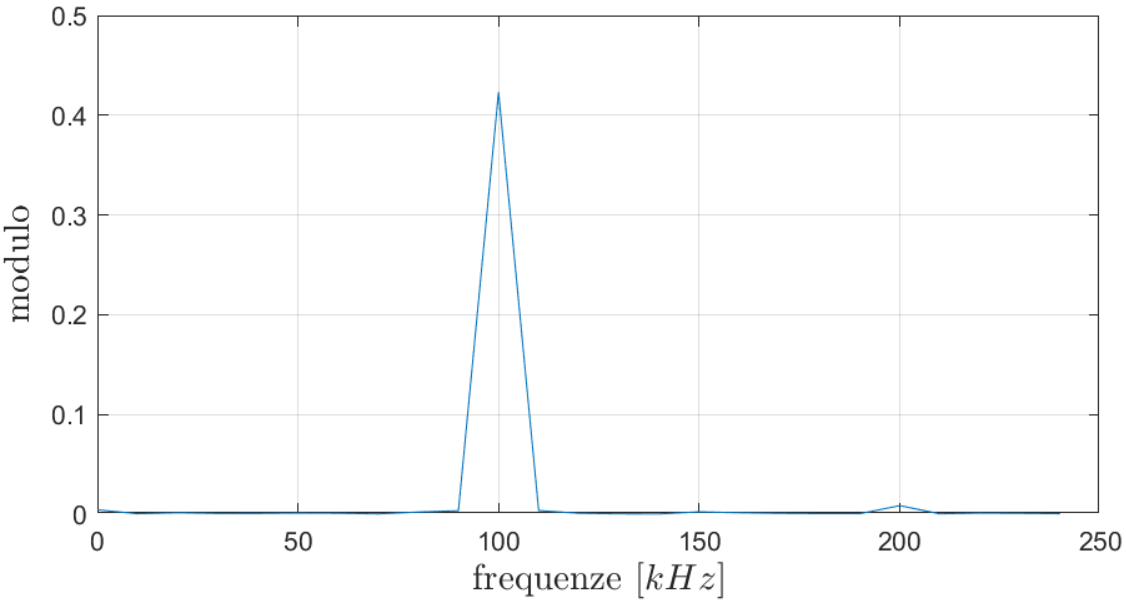


Figura 6.4: Ampiezza della trasformata del segnale acquisito dal convertitore del gruppo primario

Lavoro

Nel caso in cui venga interposto un materiale da lavoro, tipicamente un isolante, non si osservano riduzioni del picco della trasformata, anzi, a causa della costante dielettrica del materiale più alta rispetto a quella dell'aria, si osserva un incremento del campo elettrico pari 10 % rispetto alla condizione di riposo. In questo caso il ricevitore secondario mostra anch'esso una riduzione del picco della trasformata in quanto il materiale da lavoro, per le sopracitate proprietà dielettriche confina maggiormente le linee del campo elettrico attorno a se e dunque in corrispondenza del ricevitore primario.

Pericolo

Rispetto al caso precedente, il valore di picco della trasformata è notevolmente ridotto (di circa l'80 %) dalla presenza del corpo umano che, in qualità di conduttore, è in grado di drenare il campo elettrico verso terra. Anche il trasmettitore secondario, seppur abbia solo aria nel suo canale di trasmissione, inizia a risentire della vicinanza del corpo umano mostrando in questo caso un campo elettrico il cui picco della trasformata è circa il 60 % rispetto alla condizione di riposo.

Riepilogo

In tabella 6.1 vengono riportate le misurazioni relative al picco della trasformata per tutti i casi in esame.

SCENARIO	MATERIALE	Modulo del picco della FFT	
		RX primario	RX secondario
Riposo	Aria	0.4218	0.4161
Lavoro	Aria + Materiale da lavoro	0.4658	0.3897
Pericolo	Aria + Corpo umano	0.0879	0.2470

Tabella 6.1: Risultati raggiunti in termini di picco della FFT

In tabella 6.2, invece, i precedenti valori vengono messi in relazione tra di loro, esprimendoli come percentuale del modulo del picco della FFT raggiunto nella condizione di riposo.

SCENARIO	MATERIALE	Modulo del picco della FFT	
		RX primario	RX secondario
Riposo	Aria	100.0 %	100.0 %
Lavoro	Aria + Materiale da lavoro	110.5 %	93.7 %
Pericolo	Aria + Corpo umano	20.9 %	59.4 %

Tabella 6.2: Comparazione dei picchi della FFT nei vari scenari

Capitolo 7

Conclusioni

7.1 Considerazioni sul risultato raggiunto

I risultati elencati in tabella 6.1 ottenuti tramite la realizzazione di un primo prototipo del sistema costituiscono la prova concreta dell'efficacia del principio di funzionamento ideato dall'azienda circa la rilevazione di una qualsiasi parte del corpo dell'operatore in una zona ritenuta pericolosa di una macchina utensile ad avanzamento manuale.

In particolare, come dimostrano i risultati elencati in tabella 6.2, i sensori sono chiaramente in grado di discriminare i tre diversi scenari di utilizzo ipotizzati nel capitolo precedente. Dall'analisi congiunta tra le intensità dei segnali sia nei convertitori primari, direttamente esposti alla perturbazione di un materiale, sia dei convertitori secondari adiacenti, si deduce che l'applicazione di una soglia ipotetica pari al 70 % del modulo della condizione di riferimento, sarebbe sufficiente nel determinare l'avvicinamento della situazione di pericolo (già al livello del ricevitore secondario adiacente) senza generare falsi allarmi dovuti all'assorbimento del campo elettrico da parte del materiale da lavoro con costante dielettrica maggiore.

Inoltre, è stato possibile dimostrare l'uguaglianza operativa di convertitori primari e secondari per far fronte alle esigenze implementative di una futura configurazione multicanale. Avere molti gruppi a disposizione facilita anche il soddisfacimento delle norme di sicurezza le quali prevedono l'utilizzo di convertitori appartenenti a gruppi diversi per far fronte ad eventuali guasti del MCU senza incorrere in situazioni di incidente.

7.2 Implementazioni future

Considerato l'esito positivo della prima fase prototipale si elencano i miglioramenti da introdurre nelle versioni successive del sistema. In particolare per quanto concerne il firmware si interverrà:

- sull'implementazione multi-canale con trasmettitori e ricevitori multipli;
- sullo sviluppo del driver del modulo di **diretto accesso alla memoria** o **DMA**, dall'inglese Direct Access Memory, che possa effettuare i trasferimenti dei risultati campionati senza interrompere la CPU;
- sull'implementazione da parte della CPU dell'algoritmo della FFT;
- sull'implementazione del calcolo della soglia del modulo e la relativa attivazione dei comandi sugli attuatori per disabilitare l'utensile in caso di superamento della stessa.

Per quanto riguarda l'hardware si procederà con la realizzazione:

- di ulteriori schede di trasmissione e ricezione per aumentare la copertura dell'area sorvegliata;
- della scheda di alloggiamento del MCU ottimizzata per il caso in questione al fine di abbassare i costi di produzione del dispositivo.

Ringraziamenti

A conclusione del manoscritto di tesi è doveroso rammentare che lo stesso non avrebbe potuto avere seguito senza l'aiuto di chi mi è stato accanto durante questo lungo percorso di laurea magistrale.

In primis, un ringraziamento a D.M.D. Computers in qualità di soggetto ospitante per la realizzazione di questo lavoro, in particolare, all'Ing. Spiridione De Micheli, amministratore delegato dell'azienda e fautore dell'idea dalla quale il progetto ha avuto inizio e all'Ing. Fabrizio Gallo, tutore aziendale il quale mi ha accompagnato dal punto di vista tecnico nell'intero percorso di tesi offrendo disponibilità e gentilezza in ogni occasione.

Un ringraziamento al Prof. Massimo Ruo Roch per aver svolto l'attività di supervisione del manoscritto e per avermi trasmesso parte del mio sapere in qualità di professore.

Di inestimabile importanza, l'aiuto e la costante presenza della mia famiglia che mi ha accompagnato in questo lungo percorso di laurea e a cui in parte devo l'ottenimento del relativo titolo di studio. Un'immensa gratitudine a mio padre, l'uomo che più stimo nella mia vita e che ha costituito esempio di ispirazione per me, seppur indiretta, dandomi la forza di proseguire nonostante le difficoltà e che ha fornito il sostentamento economico per la mia realizzazione. Un incommensurabile ringraziamento a mia madre, che mi ha sostenuto emotivamente in tutti i momenti più difficili e mi ha accudito permettendo di concentrare i miei sforzi sui miei obiettivi. Un insostituibile riconoscimento ai miei fratelli che sono stati anche miei migliori amici nel divertimento, miei genitori nel bisogno, e miei consiglieri grazie alla loro esperienza. Ringrazio anche i miei nonni che mi sono stati sempre vicini sin dalla mia infanzia.

Un ringraziamento speciale alla mia ragazza Benedetta che negli ultimi due anni anni è stata sempre motivo di felicità nelle mie giornate più buie e che mi ha fatto vedere la realtà sotto un punto di vista più positivo.

Infine, un pensiero anche a tutti i miei amici, vicini e non, che sono stati presenti durante questo lungo percorso universitario, in particolare, ai miei compagni di corso che hanno condiviso le difficoltà accademiche e ai miei amici che sin dall'adolescenza mi hanno fatto sentire apprezzato regalandomi felici ricordi.

Bibliografia

[1] INFINEON TECHNOLOGIES AG. Gtm_tom_pwm_1 for kit_aurix_tc397_tft. https://www.infineon.com/dgdl/Infineon-AURIX_GTM_TOM_PWM_1_KIT_TC397_TFT-Training-v01_01-EN.pdf?fileId=5546d462766cbe860176862e3ac85a64&redirId=134912, 2020. (Consultato in data 04/01/2023).

[2] INFINEON TECHNOLOGIES AG. Tc39x-b user's manual appendix. https://www.infineon.com/dgdl/Infineon-AURIX_TC39x-UserManual-v01_00-EN.pdf?fileId=5546d462712ef9b7017182d371dc1d95, 2020-04-16. (Consultato in data 22/01/2023).

[3] INFINEON TECHNOLOGIES AG. Adc_single_channel_1_kit_tc397_tft. https://www.infineon.com/dgdl/Infineon-ADC_Single_Channel_1_KIT_TC397_TFT-Training-v01_02-EN.pdf?fileId=5546d4627883d7e00178a2d649793920, 2021. (Consultato in data 03/01/2023).

[4] INFINEON TECHNOLOGIES AG. Aurix tc3xx user's manual part 2. https://www.infineon.com/dgdl/Infineon-AURIX_TC3xx_Part2-UserManual-v02_00-EN.pdf?fileId=5546d462712ef9b701717d35f8541d94, 2022-09-21. (Consultato in data 22/01/2023).

[5] ROBERT TARZWELL, KEN BAHL. High voltage printed circuit design & manufacturing notebook. https://www.magazines007.com/pdf/High-Voltage-PCDesign.pdf, 2004. (Consultato in data 29/03/2023).

[6] FRANCO MARIA BOSCHETTO. Antenne. http://www.fmboschetto.it/didattica/90Radio/Radio_GL/Pagine_menu/antenna.html. (Consultato in data 23/12/2022).

[7] UNIVERSITA' DI CATANIA. Analisi in frequenza di segnali campionati. https://www.unica.it/static/resources/cms/documents/Analisi_di_segnali_campionati.pdf. (Consultato in data 17/03/2023).

[8] DIDATTICAINFO. Vantaggi dell'impiego di segnali digitali. http://www.didatticainfo.altervista.org/Quinta/Vantaggi_digitale.pdf. (Consultato in data 16/03/2023).

[9] WALTER FIORIO. Sensori e trasduttori. https://www.itaerferrarin.edu.it/pasw4/didattica/Trasduttori.pdf. (Consultato in data 23/12/2022).

[10] CONTRIBUTORI DI WIKIPEDIA, L'ENCICLOPEDIA LIBERA. Microcontrollore. http://it.wikipedia.org/w/index.php?title=Microcontrollore&oldid=124768100, 2021. (Consultato in data 20/12/2022).

[11] CONTRIBUTORI DI WIKIPEDIA, L'ENCICLOPEDIA LIBERA. Trasduttore. http://it.wikipedia.org/w/index.php?title=Trasduttore&oldid=129975783, 2021. (Consultato in data 23/12/2022).

[12] CONTRIBUTORI DI WIKIPEDIA, L'ENCICLOPEDIA LIBERA. Campionamento (teoria dei segnali). http://it.wikipedia.org/w/index.php?title=Campionamento_(teoria_dei_segnali)&oldid=130102793, 2022. (Consultato in data 23/12/2022).

[13] CONTRIBUTORI DI WIKIPEDIA, L'ENCICLOPEDIA LIBERA. Circuito stampato. http://it.wikipedia.org/w/index.php?title=Circuito_stampato&oldid=131113682, 2022. (Consultato in data 23/12/2022).

[14] CONTRIBUTORI DI WIKIPEDIA, L'ENCICLOPEDIA LIBERA. Quantizzazione (elettronica). //it.wikipedia.org/w/index.php?title=Quantizzazione_(elettronica)&oldid=129789369, 2022. (Consultato in data 17/03/2023).

Appendice A

Driver dei moduli

A.1 *EVADC.c*

```
28 //*****************************************************************************
29 //----------------------------------Includes-----------------------------------
30 //*****************************************************************************
31 #include "EVADC.h"           // file header
32
33 #include "Ifx_Types.h"       // for iLLDS types
34 #include "IfxEvadc_Adc.h"    // for iLLDs structures
35 #include "IfxStm.h"          // for iLLDs time count functions
36
37 //*****************************************************************************
38 //-----------------------------------Macros------------------------------------
39 //*****************************************************************************
40 #define GROUPS_NUM              2        // Number of used groups (group 0, 8)
41 #define CHANNELS_NUM            2        // Number of used channels
42
43 #define ISR_PRIORITY_ADC_1st    1        // Interrupt priority number 1st group
44 #define ISR_PRIORITY_ADC_2nd    2        // Interrupt priority number 2nd group
45
46 // Macro to define the Interrupt Service Routine
47 IFX_INTERRUPT(interruptADC_1st, 0, ISR_PRIORITY_ADC_1st);
48 IFX_INTERRUPT(interruptADC_2nd, 0, ISR_PRIORITY_ADC_2nd);
49
50 //*****************************************************************************
51 //-------------------------------Global Variables------------------------------
52 //*****************************************************************************
53
54 // 1st group variable
55 extern uint64 ticks_1st[N_SAMPLES];              // sample time instants
56 extern uint32 buffer_1st[N_SAMPLES];             // sample buffer
57 extern int end_acquisition_1st;                  // buffer is full
58 uint32  j = 0;                                  // buffer counter
```

```
59  Ifx_EVADC_G_RES conversionResult_1st;            // EVADC result register variable
60
61  // 2nd group variable
62  extern uint64 ticks_2nd[N_SAMPLES];              // sample time instants
63  extern uint32 buffer_2nd[N_SAMPLES];             // sample buffer
64  extern int end_acquisition_2nd;                  // buffer is full
65  uint32  k = 0;                                   // buffer counter
66  Ifx_EVADC_G_RES conversionResult_2nd;            // EVADC result register variable
67
68  // EVADC handles
69  IfxEvadc_Adc         g_evadc;                    // EVADC module  handle variable
70  IfxEvadc_Adc_Group   g_adcGroup[GROUPS_NUM];     // EVADC group   handle variable
71  IfxEvadc_Adc_Channel g_adcChannel[CHANNELS_NUM]; // EVADC channel handle variable
72
73  //*****************************************************************************
74  //------------------------------Function Prototypes------------------------------
75  //*****************************************************************************
76  void initEVADCModule(void);
77  void initEVADCGroup(void);
78  void initEVADCChannels(void);
79  void configRequestTimer(void);
80  void configQueueRequestSource(void);
81
82  //*****************************************************************************
83  //----------------------------Function Implementations---------------------------
84  //*****************************************************************************
85
86  // Function to initialize the EVADC
87  void initEVADC(void)
88  {
89      initEVADCModule();           // Initialize the EVADC module
90      initEVADCGroup();            // Initialize the EVADC group
91      initEVADCChannels();         // Initialize the channel
92      configRequestTimer();        // Configure  the Request Timer
93      configQueueRequestSource();  // Configure  the Queued Request Source
94  }
95
96  // Function to initialize the EVADC module
97  void initEVADCModule(void)
98  {
99      // Create and initializes module configuration with default values
100     IfxEvadc_Adc_Config adcConfig;
101     IfxEvadc_Adc_initModuleConfig(&adcConfig, &MODULE_EVADC);
102
103     // Need 5V for reducing the sampling time of converters
104     adcConfig.supplyVoltage = IfxEvadc_SupplyVoltageLevelControl_upperVoltage;
105     // Start up calibration takes 36.7 us (only at power on)
106     adcConfig.startupCalibrationControl = TRUE;
107
108     // Initialize module
```

```
109     IfxEvadc_Adc_initModule(&g_evadc, &adcConfig);
110 }
111
112 // Function to initialize the EVADC group
113 void initEVADCGroup(void)
114 {
115     // Create and initializes group configuration with default values
116     IfxEvadc_Adc_GroupConfig adcGroupConfig[GROUPS_NUM];
117
118     // Primary group
119
120     IfxEvadc_Adc_initGroupConfig(&adcGroupConfig[0], &g_evadc);
121     adcGroupConfig[0].groupId = IfxEvadc_GroupId_0;
122     adcGroupConfig[0].master = IfxEvadc_GroupId_0;
123     adcGroupConfig[0].arbiter.requestSlotQueue0Enabled = TRUE;
124     adcGroupConfig[0].queueRequest[0].triggerConfig.gatingMode =
    ↳ IfxEvadc_GatingMode_always;
125     adcGroupConfig[0].calibrationSampleTimeControlMode =
    ↳ IfxEvadc_CalibrationSampleTimeControl_4;
126     adcGroupConfig[0].disablePostCalibration = TRUE;
127     adcGroupConfig[0].inputBufferEnabled = FALSE;
128     adcGroupConfig[0].inputClass[0].sampleTime = 100.0e-9;
129     adcGroupConfig[0].inputClass[0].conversionMode =
    ↳ IfxEvadc_ChannelNoiseReduction_standardConversion;
130     IfxEvadc_Adc_initGroup(&g_adcGroup[0], &adcGroupConfig[0]);
131     g_adcGroup[0].group->ANCFG.B.DIVA=2;
132
133     // Secondary group
134
135     IfxEvadc_Adc_initGroupConfig(&adcGroupConfig[1], &g_evadc);
136     adcGroupConfig[1].groupId = IfxEvadc_GroupId_8;
137     adcGroupConfig[1].master = IfxEvadc_GroupId_8;
138     adcGroupConfig[1].arbiter.requestSlotQueue0Enabled = TRUE;
139     adcGroupConfig[1].queueRequest[0].triggerConfig.gatingMode =
    ↳ IfxEvadc_GatingMode_always;
140     adcGroupConfig[1].calibrationSampleTimeControlMode =
    ↳ IfxEvadc_CalibrationSampleTimeControl_4;
141     adcGroupConfig[1].disablePostCalibration = TRUE;
142     adcGroupConfig[1].inputBufferEnabled = FALSE;
143     adcGroupConfig[1].inputClass[0].sampleTime = 500.0e-9;
144     adcGroupConfig[1].inputClass[0].conversionMode =
    ↳ IfxEvadc_ChannelNoiseReduction_standardConversion;
145     IfxEvadc_Adc_initGroup(&g_adcGroup[1], &adcGroupConfig[1]);
146     g_adcGroup[1].group->ANCFG.B.DIVA=2;
147 }
148
149 // Function to initialize the EVADC channel
150 void initEVADCChannels(void)
151 {
152     // Create and initializes channel configuration with default values
```

```
153     IfxEvadc_Adc_ChannelConfig adcChannelConfig[CHANNELS_NUM];
154
155     // Primary group
156
157     IfxEvadc_Adc_initChannelConfig(&adcChannelConfig[0], &g_adcGroup[0]);
158     // Assign the CPU0 as service provider for the interrupt
159     adcChannelConfig[0].resultServProvider = IfxSrc_Tos_cpu0;
160     // Set maximum priority for the ADC result event interrupt
161     adcChannelConfig[0].resultPriority = ISR_PRIORITY_ADC_1st;
162     // Select the channel ID and the respective result register
163     adcChannelConfig[0].channelId = IfxEvadc_ChannelId_0;
164     adcChannelConfig[0].resultRegister = IfxEvadc_ChannelResult_0;
165     // Initialize the channel
166     IfxEvadc_Adc_initChannel(&g_adcChannel[0], &adcChannelConfig[0]);
167
168     // Secondary group
169
170     IfxEvadc_Adc_initChannelConfig(&adcChannelConfig[1], &g_adcGroup[1]);
171     // Assign the CPU0 as service provider for the interrupt
172     adcChannelConfig[1].resultServProvider = IfxSrc_Tos_cpu0;
173     // Set maximum priority for the ADC result event interrupt
174     adcChannelConfig[1].resultPriority = ISR_PRIORITY_ADC_2nd;
175     // Select the channel ID and the respective result register
176     adcChannelConfig[1].channelId = IfxEvadc_ChannelId_0;
177     adcChannelConfig[1].resultRegister = IfxEvadc_ChannelResult_0;
178     // Initialize the channel
179     IfxEvadc_Adc_initChannel(&g_adcChannel[1], &adcChannelConfig[1]);
180 }
181
182 // Function to configure the Request Timer
183 void configRequestTimer(void)
184 {
185     Ifx_EVADC_G_Q_REQTM reqtm;
186     reqtm.U                 = g_adcGroup[0].group->Q[0].REQTM.U;
187     reqtm.B.SEQMOD          = 1;  // sequence mode
188     reqtm.B.SEQTIMSET       = 20; // set value for the timer (20x0.1us=2us -> 500kHz)
189     reqtm.B.SEQTIMOFF       = 0;  // switch off value for the timer (no switch off)
190     reqtm.B.ENTR            = 1;  // enable external trigger for the timer
191     g_adcGroup[0].group->Q[0].REQTM.U = reqtm.U;
192     g_adcGroup[1].group->Q[0].REQTM.U = reqtm.U;
193 }
194
195 // Function to configure the Queued Request Source
196 void configQueueRequestSource(void)
197 {
198     Ifx_EVADC_G_Q_QCTRL qctrl;
199     qctrl.U                 = g_adcGroup[0].group->Q[0].QCTRL.U;
200     qctrl.B.TMWC            = 1; // enable writing on TMEN field
201     qctrl.B.TMEN            = 1; // enable timer mode
202     qctrl.B.XTWC            = 1; // enable writing on TRSEL, XTSEL, XTMODE fields
```

```
203     qctrl.B.XTMODE          = 2; // set trigger on rising edge (10)B
204     qctrl.B.XTSEL           = 8; // set trigger on GTMADC0TRIG0 (1000)B
205     qctrl.B.TRSEL           = 0; // trigger is selected by XTSEL
206     g_adcGroup[0].group->Q[0].QCTRL.U = qctrl.U;
207     g_adcGroup[1].group->Q[0].QCTRL.U = qctrl.U;
208
209     // Queued Request Source can be triggered by Request Timer
210     g_adcGroup[0].group->Q[0].QMR.B.ENTR = 1;
211     g_adcGroup[1].group->Q[0].QMR.B.ENTR = 1;
212 }
213
214 // Function to initialize the acquisition of the samples
215 void initAcquisitionBuffer()
216 {
217     // Reset the counter and the flag
218     j = 0;
219     k = 0;
220     end_acquisition_1st = 0;
221     end_acquisition_2nd = 0;
222
223     uint8 grp;
224     for(grp = 0; grp < GROUPS_NUM; grp++){
225         // A conversion request is only issued by a trigger event
226         g_adcGroup[grp].group->Q[0].QINR.B.EXTR = 1;
227
228         // Set autoscan mode for the channel
229         IfxEvadc_Adc_addToQueue(&g_adcChannel[grp], IfxEvadc_RequestSource_queue0,
    ↳ IFXEVADC_QUEUE_REFILL);
230     }
231 }
232
233 // Interrupt Service Routine for the ADC result event
234 void interruptADC_1st(void)
235 {
236     // when buffer is not full
237     if (j<N_SAMPLES){
238         // interrupts are generated continuously even if flag REV0 is kept high
239         // there is no need to reset it
240
241         // take the time
242         ticks_1st[j] = (uint64)IfxStm_get(IFXSTM_DEFAULT_TIMER) & TIME_INFINITE;
243
244         // access to ADC result register
245         conversionResult_1st = IfxEvadc_Adc_getResult(&g_adcChannel[0]);
246
247         // save the result in the buffer
248         buffer_1st[j] = (uint32)conversionResult_1st.B.RESULT;
249
250         // increment the counter
251         j = j+1;
```

```
252     }
253     // when buffer is full
254     else {
255         // trigger the main function write_files()
256         end_acquisition_1st = 1;
257 
258         // clearing the queue no entry can issue a conversion and no result event
    ↪  can trigger a new interrupt
259         IfxEvadc_Adc_clearQueue(&g_adcGroup[0], IfxEvadc_RequestSource_queue0);
260 
261         // clearing REV0 avoids running the interrupt one last time if still active
262         g_adcGroup[0].group->REFCLR.B.REV0 = 1;
263     }
264 }
265 
266 // Interrupt Service Routine for the ADC result event
267 void interruptADC_2nd(void)
268 {
269     // when buffer is not full
270     if (k<N_SAMPLES){
271         // interrupts are generated continuously even if flag REV0 is kept high
272         // there is no need to reset it
273 
274         // take the time
275         ticks_2nd[k] = (uint64)IfxStm_get(IFXSTM_DEFAULT_TIMER) & TIME_INFINITE;
276 
277         // access to ADC result register
278         conversionResult_2nd = IfxEvadc_Adc_getResult(&g_adcChannel[1]);
279 
280         // save the result in the buffer
281         buffer_2nd[k] = (uint32)conversionResult_2nd.B.RESULT;
282 
283         // increment the counter
284         k = k+1;
285     }
286     // when buffer is full
287     else {
288         // trigger the main function write_files()
289         end_acquisition_2nd = 1;
290 
291         // clearing the queue no entry can issue a conversion and no result event
    ↪  can trigger a new interrupt
292         IfxEvadc_Adc_clearQueue(&g_adcGroup[1], IfxEvadc_RequestSource_queue0);
293 
294         // clearing REV0 avoids running the interrupt one last time if still active
295         g_adcGroup[1].group->REFCLR.B.REV0 = 1;
296     }
297 }
```

A.2 GTM_TOM_PWM.c

```
28 //*****************************************************************************
29 //----------------------------------Includes-----------------------------------
30 //*****************************************************************************
31 #include "GTM_TOM_PWM.h"          // for iLLDS types
32
33 #include "Ifx_Types.h"            // file header
34 #include "IfxGtm_Tom_Pwm.h"       // for iLLDs GTM functions and types
35 #include "IfxGtm_Trig.h"          // for iLLDs GTM triggers functions and types
36
37 //*****************************************************************************
38 //-----------------------------------Macros------------------------------------
39 //*****************************************************************************
40 #define TOM_OUTPUT_PIN      IfxGtm_TOM0_13_TOUT15_P00_6_OUT
41 #define PWM_PERIOD          1000   // PWM period for the TOM (in ticks)
   ↳  100MHz / 100kHz = 1000 ticks
42 #define DUTY_CYCLE          500   // PWM DC     for the TOM (in ticks)
43
44 //*****************************************************************************
45 //-------------------------------Global Variables------------------------------
46 //*****************************************************************************
47 IfxGtm_Tom_Pwm_Config g_tomConfig;  // Timer configuration structure
48 IfxGtm_Tom_Pwm_Driver g_tomDriver;  // Timer Driver structure
49
50 //*****************************************************************************
51 //---------------------------Function Implementations--------------------------
52 //*****************************************************************************
53 // This function initializes the TOM
54 void initGtmTomPwm(void)
55 {
56     // Enable GTM and FXU clock (fixed clock unit 100MHz)
57     IfxGtm_enable(&MODULE_GTM);
58     IfxGtm_Cmu_enableClocks(&MODULE_GTM, IFXGTM_CMU_CLKEN_FXCLK);
59
60     // Initialize the configuration structure with default parameters
61     IfxGtm_Tom_Pwm_initConfig(&g_tomConfig, &MODULE_GTM);
62
63     g_tomConfig.tom = TOM_OUTPUT_PIN.tom;            // Select the TOM depending on
   ↳  the TOM_OUTPUT_PIN
64     g_tomConfig.tomChannel = TOM_OUTPUT_PIN.channel; // Select the channel
   ↳  depending on the TOM_OUTPUT_PIN
65     g_tomConfig.period = PWM_PERIOD;                 // Set the timer period
66     g_tomConfig.dutyCycle = DUTY_CYCLE;              // Set the duty cycle
67     g_tomConfig.pin.outputPin = &TOM_OUTPUT_PIN;     // Set the TOM_OUTPUT_PIN port
   ↳  pin as output
68     g_tomConfig.synchronousUpdateEnabled = TRUE;     // Enable synchronous update
69
70     IfxGtm_Tom_Pwm_init(&g_tomDriver, &g_tomConfig); // Initialize the GTM TOM
```

```
71      IfxGtm_Tom_Pwm_start(&g_tomDriver, TRUE);        // Start the PWM
72
73      // Trigger the Request Timer of the ADC
74      IfxGtm_Trig_toEVadc(&MODULE_GTM, IfxGtm_Trig_AdcGroup_0, IfxGtm_Trig_AdcTrig_0,
    ↪  IfxGtm_Trig_AdcTrigSource_tom0, IfxGtm_Trig_AdcTrigChannel_13);
75      IfxGtm_Trig_toEVadc(&MODULE_GTM, IfxGtm_Trig_AdcGroup_8, IfxGtm_Trig_AdcTrig_0,
    ↪  IfxGtm_Trig_AdcTrigSource_tom0, IfxGtm_Trig_AdcTrigChannel_13);
76  }
```

Appendice B

Applicativo di test

B.1 *Cpu0_main.c*

```
41 //*****************************************************************************
42 //-----------------------------------Includes--------------------------------
43 //*****************************************************************************
44 #include "Ifx_Types.h"     // for iLLDS types
45 #include "IfxCpu.h"        // for iLLDs CPU functions and types
46 #include "IfxScuWdt.h"     // for iLLDs watchdogs functions
47
48 #include "EVADC.h"         // file header
49 #include "GTM_TOM_PWM.h"   // file header
50
51 #include "Bsp.h"           // for iLLDs timer
52 #include <stdio.h>         // for file functions
53
54 //*****************************************************************************
55 //-------------------------------Global Variables----------------------------------
56 //*****************************************************************************
57 IFX_ALIGN(4) IfxCpu_syncEvent g_cpuSyncEvent = 0; // synch the cores
58
59 // 1st group buffer variables
60 uint64  ticks_1st[N_SAMPLES] = {0};              // sample time instants
61 float64 times_1st[N_SAMPLES] = {0};              // base time vector
62 uint32  buffer_1st[N_SAMPLES] = {0};             // sample buffer
63 int end_acquisition_1st;                         // buffer is full
64
65 // 2nd group buffer variables
66 uint64  ticks_2nd[N_SAMPLES] = {0};              // sample time instants
67 float64 times_2nd[N_SAMPLES] = {0};              // base time vector
68 uint32  buffer_2nd[N_SAMPLES] = {0};             // sample buffer
69 int end_acquisition_2nd;                         // buffer is full
70
71 // base time variables
```

```
72  uint64 dt_in_ticks = 0;                                 // ticks between samples
73  uint64 ticks_per_us = 0;                                // ticks in 1 us
74  float64 dt_in_us = 0;                                   // us between samples
75
76  // file variables
77  int i = 0;                                              // index for file writing
78  FILE *fp_samples;                                       // sample file pointer
79  FILE *fp_times;                                         // time   file pointer
80  const char samples_path_1st[] =
    ↪  "S:/Users/.../code/SC_TC397_TOM_ADC/samples_1st/samples_1st.txt";
81  const char   times_path_1st[] =
    ↪  "S:/Users/.../code/SC_TC397_TOM_ADC/samples_1st/times_1st.txt";
82  const char samples_path_2nd[] =
    ↪  "S:/Users/.../code/SC_TC397_TOM_ADC/samples_2nd/samples_2nd.txt";
83  const char   times_path_2nd[] =
    ↪  "S:/Users/.../code/SC_TC397_TOM_ADC/samples_2nd/times_2nd.txt";
84
85  //*************************************************************************************
86  //-------------------------------Function Prototypes-----------------------------------
87  //*************************************************************************************
88  void write_files(void);
89  void delete_files(void);
90
91  //*************************************************************************************
92  //---------------------------Function Implementations----------------------------------
93  //*************************************************************************************
94  void core0_main(void)
95  {
96      IfxCpu_enableInterrupts();
97
98      // DISABLING WATCHDOG0 AND SAFETY WATCHDOG
99      IfxScuWdt_disableCpuWatchdog(IfxScuWdt_getCpuWatchdogPassword());
100     IfxScuWdt_disableSafetyWatchdog(IfxScuWdt_getSafetyWatchdogPassword());
101
102     // Waiting for CPU sync event
103     IfxCpu_emitEvent(&g_cpuSyncEvent);
104     IfxCpu_waitEvent(&g_cpuSyncEvent, 1);
105
106     // Computing how many ticks in 1 us
107     ticks_per_us = IfxStm_getTicksFromMicroseconds(BSP_DEFAULT_TIMER,1);
108
109     // Initializing GTM_TOM and EVADC modules
110     initGtmTomPwm();
111     initEVADC();
112
113     // Starting the buffer acquisition
114     initAcquisitionBuffer();
115
116     // endless MCU loop
117     while(1)
```

```
118     {
119         if(end_acquisition_1st==1 && end_acquisition_2nd==1){ // if buffer is full...
120             delete_files();          // ... delete the old results on PC
121             write_files();           // ... write  the new results on PC
122             initAcquisitionBuffer(); // ... restart the acquisition
123          }
124     }
125 }
126 
127 // Function to write new buffers on PC
128 void write_files(void){
129 
130    // 1st group buffer
131 
132    // Reconstruct the base time
133    for(i=1; i<=N_SAMPLES; i++) {
134        dt_in_ticks = ticks_1st[i] - ticks_1st[i-1];
135        dt_in_us = (float64)dt_in_ticks/(float64)ticks_per_us;
136        times_1st[i] = dt_in_us + times_1st[i-1];
137    }
138    // Write the buffer on a file
139    fp_samples = fopen(samples_path_1st,"a");
140    for(i=0; i<N_SAMPLES; i++) {
141        fprintf(fp_samples,"%d\n", (int)buffer_1st[i]);
142    }
143    fclose(fp_samples);
144    // Write the base time on a file
145    fp_times = fopen(times_path_1st,"a");
146    for(i=0; i<N_SAMPLES; i++) {
147        fprintf(fp_times,"%f\n", times_1st[i]);
148    }
149    fclose(fp_times);
150 
151    // 2nd group buffer
152 
153    // Reconstruct the base time
154    times_2nd[0] = 0;
155    for(i=1; i<=N_SAMPLES; i++) {
156        dt_in_ticks = ticks_2nd[i] - ticks_2nd[i-1];
157        dt_in_us = (float64)dt_in_ticks/(float64)ticks_per_us;
158        times_2nd[i] = dt_in_us + times_2nd[i-1];
159    }
160    // Write the buffer on a file
161    fp_samples = fopen(samples_path_2nd,"a");
162    for(i=0; i<N_SAMPLES; i++) {
163        fprintf(fp_samples,"%d\n", (int)buffer_2nd[i]);
164    }
165    fclose(fp_samples);
166    // Write the base time on a file
167    fp_times = fopen(times_path_2nd,"a");
```

```
168     for(i=0; i<N_SAMPLES; i++) {
169         fprintf(fp_times,"%f\n", times_2nd[i]);
170     }
171     fclose(fp_times);
172 }
173 
174 // Function to delete old buffers on PC
175 void delete_files(void){
176     remove(samples_path_1st);
177     remove(times_path_1st);
178     remove(samples_path_2nd);
179     remove(times_path_2nd);
180 }
```

Appendice C

Applicativo Matlab

C.1 *post_processing_analysis.m*

```
close all
clear all
clc
% Reads ADC samples and computes FFT

%% DATA ACQUISITION
% 12 bit result registers
% samples range [0 ; 2^n_bit-1] --> [0 ; 4095]

% Signal samples from primary group
file = fopen('..\SC_TC397_TOM_ADC\samples_1st\samples_1st.txt');
x1 = fscanf(file,'%d')';
fclose(file);

% Time instants of primary group acquisitions (us)
file = fopen('..\SC_TC397_TOM_ADC\samples_1st\times_1st.txt');
t1 = fscanf(file,'%f')';
fclose(file);

% Signal samples from secondary group
file = fopen('..\SC_TC397_TOM_ADC\samples_2nd\samples_2nd.txt');
x2 = fscanf(file,'%d')';
fclose(file);

% Time instants of secondary group acquisitions (us)
file = fopen('..\SC_TC397_TOM_ADC\samples_2nd\times_2nd.txt');
t2 = fscanf(file,'%f')';
fclose(file);

%% GENERAL PARAMETERS
fs = 500e3; % ADC sampling frequency
```

```
32  V_ADC = 5; %[V] ADC supply voltage
33  V_Rx  = 3.3; %[V] Rx supply voltage
34  full_scale = 4096*V_Rx/V_ADC;
35
36  %% PRIMARY CONVERTER
37  x1 = x1-(full_scale/2); % removing BIAS
38  x1 = x1/full_scale; % normalizing wave amplitude
39  N1 = length(x1); % number of samples
40
41  % Signal plotting
42  subplot(2,2,1)
43  plot(t1,x1,'.-')
44  grid on
45  title('Acquisizione onda quadra (CANALE PRIMARIO)');
46  xlabel('tempo [us]')
47  ylabel('ampiezza normalizzata')
48
49  % FFT computing
50  X1 = fft(x1)/N1; % computing the FFT
51  X1 = 2*X1(1:N1/2); % truncating due to symmetry
52  Mag1 = abs(X1); % FFT amplitude
53  f1 = (0:N1/2-1)/N1*fs; % from bins to frequency
54  f1 = f1./1000; % frequency from Hz to kHz
55
56  % FFT plotting
57  subplot(2,2,2)
58  plot(f1,Mag1)
59  grid on
60  title('FFT onda quadra (CANALE PRIMARIO)');
61  xlabel('frequenze [kHz]')
62  ylabel('modulo')
63
64  %% SECONDARY CONVERTER
65
66  x2 = x2-(full_scale/2); % removing BIAS
67  x2 = x2/full_scale; % normalizing wave amplitude
68  N2 = length(x2); % number of samples
69
70  % Signal plotting
71  subplot(2,2,3)
72  plot(t2,x2,'.-')
73  grid on
74  title('Acquisizione onda quadra (CANALE SECONDARIO)');
75  xlabel('tempo [us]')
76  ylabel('ampiezza normalizzata')
77
78  % FFT computing
79  X2 = fft(x2)/N2; % computing the FFT
80  X2 = 2*X2(1:N2/2);  % truncating due to symmetry
81  Mag2 = abs(X2); % FFT amplitude
```

```
82 f2 = (0:N2/2-1)/N1*fs; % from bins to frequency
83 f2 = f2./1000; % frequency from Hz to kHz
84
85 % FFT plotting
86 subplot(2,2,4)
87 plot(f2,Mag2)
88 grid on
89 title('FFT onda quadra (CANALE SECONDARIO)');
90 xlabel('frequenze [kHz]')
91 ylabel('modulo')
```

Elenco delle figure

Elenco delle tabelle

Elenco dei registri