UNIVERSITÀ
DEGLI STUDI
DI PALERMO

Valutazione energetica di modelli di esecuzione simbolica su dispositivi embedded

Tesi di Laurea Magistrale in Ingegneria Informatica

Antonio Bordonaro

Relatore: Prof. Daniele Peri
Correlatore: Ing. Gloria Martorella

UNIVERSITÀ DEGLI STUDI DI PALERMO

SCUOLA POLITECNICA

LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Valutazione energetica di modelli di esecuzione simbolica su dispositivi embedded

Tesi di Laurea di

Dott. Antonio Bordonaro

Relatore:

Prof. Daniele Peri

Controrelatore:

Ch.mo Prof. Salvatore Gaglio

Correlatore:

Ing. Gloria Martorella

Anno Accademico 2017/2018

Valutazione energetica di modelli di esecuzione simbolica su dispositivi embedded

Tesi di Laurea di

Dott. Antonio Bordonaro

Relatore:

Prof. Daniele Peri

Controrelatore:

Ch.mo Prof. Salvatore Gaglio

Correlatore:

Ing. Gloria Martorella

Sommario

I *Sistemi Embedded* sono dispositivi elettronici digitali, tipicamente basati su microcontrollori che vengono progettati e sviluppati per essere impiegati in ben definiti ambiti applicativi in cui è spesso essenziale sviluppare applicazioni orientate all'uso efficiente delle risorse al fine di garantire bassi consumi, costi di mantenimento inferiori ed eventualmente durata maggiore delle batterie. L'adozione di opportune suite di benchmark permette di ottenere una stima di quelle che sono le richieste in termini di energia e dunque, dimensionare opportunamente le fonti di alimentazione, individuare possibili miglioramenti ed effettuare delle ottimizzazioni mirate. Accanto ai tradizionali ambienti operativi, sistemi basati su modelli di esecuzione simbolica non sono stati ampiamente valutati da un punto di vista energetico. Per tale ragione, in questo lavoro di tesi è stata condotta una valutazione energetica di un ambiente simbolico interpretato basato sull'ambiente Mecrisp-Stellaris a bordo della piattaforma *Nucleo-Board-F446RE* ed il relativo microcontrollore STM32F446RE. In questo lavoro sono state pertanto definite opportune suite di benchmark che permettono di ottenere dati ed informazioni relativamente al consumo energetico al variare di differenti configurazioni, sia hardware che software. In particolare, gli esperimenti condotti sono in linea con i principi alla base *dell'energy aware computing*. I risultati sono stati confrontati con quelli ottenuti da una implementazione basata su C. Le misurazioni rilevate, mostrano che la presenza dell'interprete comporta un maggiore consumo energetico causato

dalla continua esecuzione dell'interprete che necessariamente richiede l'abilitazione di molte periferiche non strettamente necessarie per l'applicazione sviluppata.
Di contro, la presenza dell'interprete rende altamente interattivo il sistema e permette di ottenere tempi di sviluppo, generalmente caratterizzati da cicli di codifica-test, molto più rapidi rispetto alle classiche operazioni di compilazione-flash.
Possibili sviluppi futuri riguardano lo sviluppo di sistemi, anche distribuiti, energy aware basati su dispositivi che, sfruttando ambienti di programmazione simbolica, implementino algoritmi, tecniche e metodi per un uso efficiente delle risorse dell'intero sistema.

Indice

Introduzione **1**

1. Capitolo - Concetti preliminari **7**

1.1 Sistemi embedded e IoT 7

1.2 Piattaforme per lo sviluppo di IoT e Smart System su architetture embedded a risorse limitate 12

1.3 Fattori del consumo energetico 13

2. Capitolo 2 - STM32F446RE e Nucleo-board F446RE **16**

2.1 Introduzione e architettura 16

2.2 Programming 19

2.2.1 MBED 19

2.2.2 C – Crosscompilazione con ARM Toolchain 21

2.2.3 Mecrisp-Stellaris 23

3. Capitolo 3 – Benchmark **30**

3.1 Suite di base 30

3.1.1 Blink idle-loop 31

3.1.2 Blink timer-IRQ 32

3.2 Suite Energy Aware 33

3.3 Definizione parametri di benchmark 34

3.3.1 LOC – Lines of Code 35

3.3.2 Consumo di memoria 36

4. Capitolo 4 – Setup sperimentale **37**

4.1 Implementazione driver I2C 41

4.2 Implementazione del protocollo di comunicazione microcontrollore-sistema *host* 50

5. Capitolo 5 – Valutazione sperimentale suite di base 58
5.1 Esperimento 0 – Baseline 59
5.2 Esperimento 1 – Periferiche disabilitate 63
5.3 Esperimento 2 – Timer 68
5.4 Esperimento 3 – Timer 72
5.5 Esperimento 4 – Timer, GPIO 75
5.6 Esperimento 5 – Timer, GPIO, I2C 79
5.7 Esperimento 6 – Timer, GPIO, I2C, USART 82
5.8 Esperimento 7 – Timer, GPIO, I2C, USART, SPI 86
5.9 Benchmark 1 – Blink idle-loop 89
5.9.1 Mecrisp-stellaris 89
5.9.2 Linguaggio C 91
5.9.3 Confronto e valutazioni 93
5.10 Benchmark 2 – Blink (timer e IRQ) 95
5.10.1 Mecrisp-stellaris 95
5.10.2 Linguaggio C 96
5.10.3 Confronto e valutazioni 98
6. Capitolo 6 – Valutazione sperimentale Suite Energy Aware 100
6.1 Benchmark 1 – Blink idle-loop 101
6.1.1 Mecrisp-stellaris 101
6.2 Benchmark 2 – Blink (timer e IRQ) 103
6.2.1 Mecrisp-stellaris 103
6.2.2 Confronto e valutazioni 104
7. Conclusioni 107
Indice delle figure 110
Indice delle tabelle 112
Appendice 114
Bibliografia 153

Introduzione

I *Sistemi Embedded* sono dispositivi elettronici digitali che, a differenza di un computer *general purpose*, utilizzabile cioè per affrontare e risolvere problemi di qualsiasi tipo, vengono progettati e sviluppati per essere impiegati in un ben definito ambito applicativo nel quale svolgerà sempre e solo la stessa funzione.
Differentemente da quanto accade in altri contesti in cui, in genere, si cerca di utilizzare dispositivi con elevate prestazioni e performance, nell'ambito dei sistemi embedded succede spesso che uno dei principali fattori vincolanti sia il consumo energetico.
Infatti, se da un lato è vero che processori con prestazioni maggiori permettono sicuramente un'esecuzione più rapida del software, dall'altro è anche vero che, il fatto di avere maggiori performance, si ripercuote in maniera quasi diretta sui consumi energetici.
Sviluppare e progettare un sistema embedded, in molti casi, si traduce nel dover sviluppare applicazioni per microcontrollori, circuiti integrati che hanno a bordo tutti i dispositivi per il funzionamento e l'esecuzione di un programma (processore, memoria, dispositivi hardware e interfacce di comunicazione).
L'enorme diffusione di microcontrollori è stata in parte favorita dalla crescita sempre maggiore dell'*Internet of Things* (IoT), paradigma secondo il quale gli oggetti di uso quotidiano vengono collegati alla rete e acquisiscono la capacità di comunicare fra di loro. Tali dispositivi, chiamati *nodi*, nella maggior parte dei casi includono microcontrollori, vengono collegati in maniera wireless e sono alimentati da batterie.
In tali contesti è dunque essenziale sviluppare applicazioni orientate all'uso efficiente delle risorse per poter garantire bassi consumi e, di conseguenza, costi di mantenimento inferiori e durata maggiore delle batterie.

In letteratura esistono numerosi approcci, tecniche e metodologie che mirano alla realizzazione di applicazioni per sistemi embedded orientate all'uso efficiente delle risorse per garantire bassi consumi energetici.

Alcuni approcci agiscono a livello di sistema proponendo algoritmi per ridurre il consumo energetico in una rete Wireless Sensors Area Network (WSAN) [1] o definendo nuovi protocolli di comunicazione che siano più efficienti e tali da poter essere impiegati in scenari applicativi che richiedano la presenza di nodi distribuiti ed interconnessi [2].

Mentre tali approcci intervengono esclusivamente sulla comunicazione, l'efficienza dell'applicazione ed i consumi del singolo nodo rimangono un aspetto non trattato. Alternativamente, in letteratura sono stati analizzati e presentati metodi per ridurre i consumi energetici che intervengono sia a livello di rete che di singola unità. In particolare, per quanto riguarda la riduzione dei consumi a bordo dei singoli nodi, questi utilizzano solo le periferiche necessarie per il completamento dei *task* o tecniche per la gestione dell'alimentazione guidata da eventi [3,4].

Sfruttando l'esecuzione simbolica, è anche possibile agire a livello di codice, fornendo al programmatore degli strumenti che gli consentano di determinare le richieste, in termini di energia, del codice applicativo [5] al fine di supportare lo sviluppo di applicazioni embedded *energy aware* già nelle prime fasi della codifica.

In ogni caso, la riduzione dei consumi non può prescindere da una valutazione energetica relativa alle piattaforme e alle applicazioni. In tal senso, l'utilizzo di benchmark appropriati risulta essere fondamentale.

Alcuni lavori propongono infatti l'analisi e valutazione dei consumi energetici mediante opportune suite di benchmark [9, 10]. A tal proposito, il confronto tra diversi setup sperimentali di benchmark consente di valutarne differenze e determinare quanto i risultati ottenuti in ambiente sperimentale siano coerenti con i consumi in un'ambiente reale [11].

Risulta chiaro che l'uso di benchmark permette, innanzitutto di avere una stima di quelle che sono le richieste in termini di energia e dunque, dimensionare opportunamente le fonti di alimentazione. Inoltre, ottenuti quelli che sono gli

andamenti relativi ai consumi, è possibile individuare possibili punti di miglioramenti ed effettuare delle ottimizzazioni mirate.

In tal senso, questo lavoro di tesi si occupa di fornire una valutazione energetica di un modello di esecuzione simbolica basato sull'ambiente Mecrisp-Stellaris.

L'ambiente è interpretato rispetto ai tradizionali ambienti di programmazione basati su cross-compilazione. Pertanto consente lo sviluppo interattivo di applicazioni anche su dipositivi embedded. L'architettura target utilizzata è rappresentata dalla Nucleo-board STM32F446RE [13].

Per la valutazione energetica dell'ambiente simbolico, è stato sviluppato un setup sperimentale e definiti benchmark che permettono di ottenere dati ed informazioni relativamente al consumo energetico al variare di differenti configurazioni, sia hardware che software. In questo lavoro, la valutazione energetica si focalizza sul singolo nodo, piuttosto che all'intero sistema composto da nodi.

Per il raggiungimento di tale obiettivo si è realizzato un setup sperimentale atto alla raccolta dati costruito mediante sensore INA219 [12]. Il setup consiste di un microcontrollore, del sensore INA219 e di un calcolatore utilizzato per l'analisi dei dati rilevati e per il calcolo di parametri statistici.

Al fine di ottenere dati che possano essere considerati validi e realistici sono stati valutati i consumi del microcontrollore al variare di differenti configurazioni hardware e software. Per quanto riguarda la componente hardware si sono analizzati consumi in relazione alle periferiche utilizzate come GPIO, timer, USART e UART, interfacce I2C ed interfacce SPI.

Si è resa necessaria la realizzazione di driver appositi per pilotare le periferiche hardware di cui è dotato il microcontrollore.

Relativamente allo strato software, l'ambiente simbolico, che utilizza il linguaggio Forth è stato confrontato con la rispettiva implementazione C al fine di effettuare una analisi comparativa dei consumi energetici. Tale processo viene formalizzato con la definizione di suite di benchmark e di metriche di valutazione, quali *Lines of Code* (LOC) e occupazione di memoria.

Ottenute le valutazioni energetiche relative a diverse configurazioni, la fase successiva potrebbe essere lo sviluppo di sistemi software per microcontrollori in grado di rilevare il consumo istantaneo e sulla base dei task da eseguire, attuare strategie di risparmio energetico adattive.

L'obiettivo finale è dunque la realizzazione di metodi, tecniche e approcci per lo sviluppo di applicazioni orientate all'uso efficiente delle risorse per sistemi embedded con risorse hardware limitate, con particolare riferimento ad architettura ARM ed ai microcontrollori della famiglia STM32.

La valutazione energetica effettuata può essere infatti impiegata per lo sviluppo di *Battery management System* (BMS) e *Energy Management System* (EMS) o essere utilizzati in architetture di modelli distribuiti/cooperativi.

Infatti, i dati ottenuti possono essere utilizzati come base per la progettazione e lo sviluppo di tecniche e algoritmi di *energy aware computing* che consentano al microcontrollore di acquisire conoscenza, dunque, dei suoi consumi e sulla base di questi, e della tipologia di task che sta eseguendo, attuare delle strategie di esecuzione opportune, ad esempio in termini di *deadline*, velocità di esecuzione, caratteristiche real-time, etc.

Nel primo capitolo vengono fornite le conoscenze di base sui sistemi embedded e i relativi microcontrollori, dispositivi principali di un sistema embedded.

Dopo un'introduzione di ampia visione sui sistemi embedded, vengono presentati quelli che sono oggi i principali *tool*, tecnologie, sistemi e approcci presenti in letteratura per lo sviluppo di applicazioni embedded, con focus sui sistemi orientati all'uso efficiente delle risorse (mirate alla realizzazione di software per dispositivi con limitate risorse). Saranno introdotte le principali problematiche inerenti il consumo energetico e quelli che sono le principali motivazioni che hanno spinto ad uno studio continuo e sempre più specializzato nell'energy aware computing.

Il secondo capitolo descrive nel dettaglio le principali caratteristiche della famiglia dei microcontrollori STM32. Viene analizzato nel dettaglio il microcontrollore STM32F446RE, basato su processore ARM Cortex-M4.

Inoltre, vengono presentati ambienti di sviluppo di applicazioni per STM32. Il primo utilizza il linguaggio C e, partendo da un codice sorgente verranno illustrati tutti i passi necessari per ottenere l'eseguibile (o comunque il binario).
L'altra metodologia è basata su Mecrisp-Stellaris, ambiente simbolico basato su Forth per la famiglia dei microcontrollori STM32.
Infine, viene presentato l'ambiente di sviluppo MBED, piattaforma online che offre servizi, *tool* e strumenti per lo sviluppo di applicazioni per microcontrollori della famiglia STM32.

Il capitolo successivo descrive il setup sperimentale realizzato per l'acquisizione delle misurazioni energetiche. I componenti coinvolti sono rappresentati dal sensore INA219, dalla Nucleo Board, da un Arduino Uno e da un calcolatore che ha l'obiettivo di ricevere le misurazioni, di elaborarli e di calcolare dei parametri statistici. Il numero di microcontrollori varia a seconda del tipo di misurazione che si intende effettuare. Le misurazioni sono distinte in *on-board*, cioè effettuate dalla stessa scheda che esegue il codice di benchmark e *off-board*, cioè misurate con un microcontrollore esterno.
Il capitolo si conclude con la descrizione della suite di software di benchmark. In tal senso, viene definita una suite di benchmark "tradizionale" ed una suite orientata all'acquisizione di informazioni relative all' *energy aware*. Per quantificare in maniera obiettiva i risultati dei test di benchmark, sono stati definiti criteri di confronto, quali *Lines of Code* (LOC) e memoria occupata.

Il quarto capitolo descrive nel dettaglio i protocolli di comunicazione che sono coinvolti in fase di acquisizione delle misurazioni energetiche.
Poiché le comunicazioni fra microcontrollori e INA219 avvengono mediante protocollo I2C, il capitolo dettaglia la realizzazione di un driver I2C per il microcontrollore STM32.
Differentemente, la comunicazione fra il microcontrollore che sta eseguendo le misurazioni e calcolatore, denominato *sistema host*, avviene tramite connessione

seriale. Pertanto, il capitolo si conclude con la descrizione del protocollo sviluppato per consentire la comunicazione tra il microcontrollore e il sistema host.

Nel capitolo 5 vengono presentati i benchmark della suite di base, costituita dai programmi in cui le misurazioni vengono effettuate *off-board*.
Oltre alla descrizione dei benchmark vengono anche presentati i risultati sperimentali relativi alla valutazione energetica dell'ambiente Mecrisp-Stellaris. I risultati sono presentati mediante grafici e diagrammi e confrontati con le rispettive implementazioni in linguaggio C.

Nel capitolo 6, invece, vengono descritti e presentati i risultati sperimentali relativi alla suite *energy aware*. In questo caso, le misurazioni, coerentemente con i principi alla base dell'energy aware computing, vengono effettuate *on-board*. La valutazione energetica dell'ambiente Mecrisp-Stellaris è presentato mediante grafici e diagrammi.

Il capitolo 7 riporta le conclusioni, mentre l'Appendice incorpora il codice relativo alle suite di benchmark, al driver I2C e al protocollo di comunicazione tra il microcontrollore e il sistema host.

1. Capitolo - Concetti preliminari

In questo capitolo sono fornite conoscenze di base relativamente agli argomenti principali del lavoro di tesi.
In particolar modo, vengono date nozioni e concetti relativi ai microcontrollori e a quelli che sono le differenze con sistemi di calcolo tradizionali, chiamati in genere *sistemi general purpose*.
Vengono anche presentati i principali ambiti applicativi dei microcontrollori, come ad esempio l' IoT, e quali sono gli strumenti per lo sviluppo di applicazioni e sistemi per microcontrollori.
Parlando di microcontrollori non si può non discutere dell'importanza e dell'impatto che hanno i consumi energetici nelle applicazioni per sistemi embedded.
Per tale motivo vengono descritti i principali fattori relativi ai consumi energetici e quali sono le strategie che possono essere utilizzate per ridurli.

1.1 Sistemi embedded e IoT

Con il termine "sistema embedded" ci si riferisce ad un sistema elettronico-digitale progettato e realizzato per essere impiegato in una specifica applicazione.
Differentemente ad un sistema *general purpose*, ai quali si contrappongono, un sistema embedded dovrà eseguire solo un particolare compito per tutto il tempo del suo utilizzo.
Le caratteristiche e le specifiche che devono possedere variano in funzione al compito che devono eseguire. Per esempio, in alcuni casi potrebbe essere necessario che il sistema mantenga una velocità minima nell'eseguire alcune operazioni, in altri casi

si potrebbe richiedere funzionalità real-time mentre in altre applicazioni potrebbe diventare un vincolo determinante la dimensione del sistema. In altri contesti potrebbe essere un requisito fondamentale un basso consumo energetico.
Le specifiche di un sistema, dunque, variano in funzione al contesto applicativo nel quale sarà impiegato.
Per questo motivo, e per il grande numero di requisiti che potrebbero essere definiti per un sistema embedded, non esiste "il sistema perfetto" ma, caso per caso, deve essere effettuata una progettazione ad-hoc al fine di soddisfare al pieno i vincoli richiesti.
Un sistema embedded è formato da diversi componenti controllati da una CPU, che è il cuore dei microprocessori e dei microcontrollori.
Un microcontrollore integra, su un singolo chip, la CPU, la memoria (RAM e ROM), le porte di I/O e altri componenti.
La CPU è composta dall'ALU (Arithmetic Logic Unit), dalla CU (Control Unit) e da un insieme, più o meno numeroso, di registri.
L'ALU si occupa d tutte le operazioni matematiche elementari (somma, differenza) e delle operazioni logiche (AND, OR).
La CU si occupa della sincronizzazione e del controllo di tutte le operazioni della CPU.
Questa è responsabile nel dirigere il flusso di istruzioni e dati all'interno della CPU ed esegue, step by step, le istruzioni del programma.
La CPU esegue continuamente istruzioni, non fermandosi mai.
La scelta della CPU, in fase di progettazione di un Sistema Embedded, è influenzata da diversi fattori, come la grandezza massima, in bit, di un singolo operando dell'ALU o la frequenza del clock.
Un altro aspetto delle CPU sono gli *interrupt*, ovvero il verificarsi di un evento che richiede l'intervento del processore. Al verificarsi di un evento, la CPU interrompe il programma in esecuzione ed esegue un sottoprogramma, chiamato *Interrupt Service Routine* (ISR). Terminata l'esecuzione dell'ISR, la CPU riprende il programma

interrotto. Le richieste di interrupt vengono effettuate su particolari pin della CPU, chiamati *Interrupt Request* (IRQ).

Il microcontrollore è in grado di interagire con il mondo esterno grazie a delle porte di I/O.

La più generica porta di I/O, la *General Purpose I/O* (GPIO), è un'interfaccia a cui vengono associati determinati pin del microcontrollore, ognuno dei quali è configurabile in maniera individuale.

La GPIO può essere usata sia in input (lettura di segnali elettrici) o in output (impostare il valore di tensione).

Un microcontrollore è, generalmente, dotato di più porte GPIO.

A partire dall'indirizzo in cui viene mappata la GPIO, all'interno dello spazio di indirizzamento del microcontrollore, vi sono i registri di configurazione che permettono di avere un controllo completo sui pin associati alla GPIO.

Tramite le GPIO, dunque, il microcontrollore può interagire con il mondo che lo circonda e, a seconda degli stimoli che riceve, configurare adeguatamente lo stato delle GPIO stesse.

La componente *software* di un sistema embedded è rappresentato dall'insieme di linee di codice che vengono eseguite, in maniera sequenziale, dalla CPU.

Il software sviluppato è strettamente dipendente dai requisiti e dalle componenti hardware di cui dispone.

Nella progettazione di software per sistemi embedded si deve prestare particolarmente attenzione all'uso di risorse, e dunque, all'efficienza; questo è tanto più vero quanto più limitate sono le risorse hardware.

Il software risiede in una memoria di archiviazione, in modo da essere permanente in caso di riavvio del sistema, ed è, come detto prima, costitituito da una serie di istruzioni che vengono prelevate, sequenzialmente, dalla CPU ed eseguite iterativamente seguendo quello che viene definito ciclo *fetch-decode-execute*.

È possibile, in alcuni sistemi, utilizzare un *sistema operativo* che ha la funzione principale di offrire un'interfaccia fra l'hardware su cui gira ed il software, fornendo

così un livello di astrazione che nasconde al livello applicativo le caratteristiche ed i dettagli del livello hardware.

Tali sistemi operativi sono compatti e orientati all'efficienza dell'uso delle risorse, essendo queste molto limitate nella maggior parte dei sistemi embedded.

I sistemi operativi con funzionalità *real-time* sono indispensabili in tutti quei casi in cui si esige che il sistema esegua un determinato task entro un certo limite, chiamato *deadline*.

Un sistema operativo *real-time*, contrariamente a quanto si potrebbe pensare, non deve necessariamente essere veloce nell'esecuzione delle istruzioni; è importante, piuttosto, che il sistema risponda non oltre un determinato intervallo temporale, oltre il quale la risposta non è più utile, o può, nel peggiore dei casi, causare danni all'intero sistema.

In altre parole, il sistema operativo deve essere prevedibile o, quanto meno, deterministico nelle stime di quelli che vengono definiti *best case* e *worst case*, rispettivamente il miglior e il peggior tempo nel quale terminare l'elaborazione corrente.

Per garantire questo, è richiesto che la schedulazione delle operazioni sia fattibile. Il concetto di *fattibilità di schedulazione* è alla base della teoria dei sistemi real-time ed è quello che permette di stabilire se un insieme di task sia eseguibile o meno, in funzione dei vincoli temporali dati.

In un sistema real-time, un task può essere *periodico* (quando consiste in una sequenza di attività attivate con cadenza regolare), *aperiodico* (quando consiste in una sequenza di attività attivate ad intervalli irregolari) o *sporadici* (quando consiste in una sequenza di attività attivate in modo impredicibile, come a degli input o richieste da parte dell'utente).

Inoltre, un'altra classificazione può essere fatta in relazione alle conseguenze di un non rispetto della deadline.

Un *task soft real-time* che non rispetta la sua scadenza provoca un danno non irreparabile al sistema. Il superamento della deadline produce un degrado delle prestazioni proporzionale al tempo di superamento della deadline.

Un *task hard real-time* che non rispetta la scadenza provoca un danno irreparabile al sistema. Infine, un *task anytime* elabora iterativamente gli stessi dati e li raffina progressivamente. I dati elaborati dai *task* anytime rispondono a requisiti di qualità minima e qualità massima. Dunque, tali t*ask*, sono considerati *hard* fino a che i dati non raggiungono la qualità minima, diventano soft prima di raggiungere la qualità massima, dopodiché non vengono più eseguiti. Introdotti tali concetti, un sistema *real-time* viene definito *hard* se può garantire la fattibilità di un insieme di schedulazione di hard e soft task. Analogamente, un sistema real-time si dice *soft* se può garantire la fattibilità di schedulazione di un insieme di soli task soft real-time.

Gli algoritmi di schedulazione maggiormente utilizzati in sistemi real-time sono l'EDD (*Earliest Due Date*), l'EDF (*Earliset Deadline First*) e RM (*Rate Monotonic*). Ogni algoritmo rispetta determinati vincoli e riesce a garantire particolari condizioni. Esistono, ovviamente, anche dei SO che non offrono funzionalità real-time e sono indicati in tutti quei casi in cui non è strettamente necessario conoscere con precisione i tempi di risposta del sistema.

Ad esempio in MantisOS [6] vengono attuate tecniche per ridurre tempi di latenza nella schedulazione dei processi e per permettere un uso efficiente delle risorse.

Come detto, in genere i sistemi embedded sono dotati di piccole quantità di memorie ed è per questo che *MantisOS* funziona con soltanto 500 Byte di RAM.

Esistono altri software che si interpongono fra il livello hardware ed il livello applicativo. È il caso, per esempio, di software che vengono caricati nel sistema e che, una volta eseguiti, espongono un interprete, rendendo di fatto il sistema interattivo.

Si possono citare, per esempio, implementazioni di Forth progettate e sviluppate per essere impiegate in sistemi embedded. Ad esempio, *Mecrisp-Stellaris* è un interprete Forth per la famiglia dei microcontrollori STM32 che ha dimensioni molto ridotte (circa 16 Kb).

Più in generale le piattaforme embedded sia hardware che software vengono oggi utilizzate in moltissimi e svariati contesti. Uno di questi, che negli ultimi anni si è molto diffuso, è l'*Internet of Things* (IoT).

Quando si parla di IoT ci si riferisce ad una rete di dispositivi connessi, mediante varie tecnologie di rete, in grado di acquisire informazioni dall'ambiente e di attuare opportuni effetti.

Una caratteristica delle *things*, ovvero dei *nodi* della rete, è quella di poter comunicare, direttamente o indirettamente, in Internet.

Si cerca, quindi, di attribuire un'identità agli oggetti reali di uso comune, che acquisiscono quindi la possibilità di poter scambiare dati e informazioni con l'ambiente e con altri nodi connessi alla rete.

I campi di applicazioni in cui le *things* diventano *smart* sono numerosi e questa vastità è dovuta anche dalle potenzialità che oggetti di uso quotidiano possono offrirci se connessi ad Internet o interconnessi fra di loro.

L'IoT riscuote sempre più consenso e rappresenta una occasione di sviluppo anche in ambito professionale. Si investe in ambito medico, nell'avionica, nell'industria automobilistica ed in altri numerosi settori, come mostrato in [14, 15, 16].

Molte società di ricerca sostengono che il numero di apparati IoT arriverà o supererà i 25 miliardi entro il 2020. In realtà, molti operatori del settore ritengono che tale stima sarà abbondantemente superata e già questo, da solo, rappresenta una straordinaria opportunità di business per tutte le figure coinvolte in tale ambito.

E come cresce la diffusione di apparati e sensori, tanto velocemente cresce la quantità di dati che dovranno essere gestiti e, di conseguenza, cresce il numero di applicazioni che dovranno essere sviluppate. Sotto questo profilo è prevedibile una notevole diffusione di piattaforme di sviluppo specifiche ed ottimizzate.

1.2 Piattaforme per lo sviluppo di IoT e Smart System su architetture embedded a risorse limitate

Sviluppare un'*applicazione IoT* significa scrivere un programma per uno specifico dispositivo che verrà impiegato in un contesto IoT.

Tutto si riduce, in molti casi, allo sviluppo di applicazioni per determinati microcontrollori.
In letteratura sono presenti numerosi articoli e studi riguardo software, applicazioni e framework per lo sviluppo di applicazioni embedded, con particolar attenzione alle architetture di sistemi IoT e al consumo energetico ridotto dei nodi, caratteristica sempre più di grande interesse.
Fra le più grandi aziende che offrono soluzioni altamente innovative per lo sviluppo dedicato di applicazioni in contesti IoT figura Microsoft con un sistema dedicato integrato nell'ecosistema di Azure, chiamato *Azure IoT Hub [17]*.
MBED invece è una piattaforma online che offre innumerevoli servizi, tool e strumenti per lo sviluppo di applicazioni embedded. Tale piattaforma supporta un grande numero di microcontrollori fra i quali figurano anche quelli della famiglia STM32.
MBED supporta lo sviluppo di applicazioni in linguaggio C/C++ ed offre una libreria che permette l'accesso completo alle periferiche hardware di cui sono dotati i microcontrollori.
Negli ultimi anni si è assistito ad un vertiginoso aumento del numero di piattaforme che permettono, a livelli più o meno professionali, la progettazione e lo sviluppo di applicazioni per sistemi embedded.
Tale tendenza è stata enormemente spinta dalla diffusione che ha avuto l'IoT negli ultimi anni.
Per tali motivi, tale settore ha rappresentato, rappresenta e sicuramente continuerà a rappresentare forti opportunità di business.

1.3 Fattori del consumo energetico

Date le risorse limitate di cui è dotato, in genere, un sistema embedded, è fondamentale sviluppare, indipendentemente dall'ambiente che viene utilizzato,

software orientati all'efficienza dell'uso delle risorse (in termini di memoria, processore, consumo energetico).

E' bene ricordare che il consumo energetico rappresenta un aspetto fondamentale perché, nella maggior parte dei casi e per permettere una topologia della rete più dinamica, i nodi non vengono alimentati collegandoli alla rete elettrica bensì tramite batterie. Questo si traduce nel fatto che, maggiore sarà il consumo energetico, maggiore sarà il numero di interventi di sostituzione delle batterie che si dovrà effettuare, con conseguente aumento dei costi di manutenzione dell'intero sistema.

In questo paragrafo verrà definito cosa si intende per *consumo energetico* e verranno esaminate quali sono le problematiche ad esso legate.

Si presenteranno quali sono, o potrebbero essere, le tecniche e le strategie oggi maggiormente utilizzate per gestire tale aspetto.

Per consumo energetico si intende la quantità di energia elettrica che viene utilizzata da un dispositivo affinché possa svolgere i compiti per il quale è stato progettato e realizzato.

Nel caso di un processore il consumo energetico viene formalizzato con la seguente equazione:

$$E = C * V^2 * f$$

essendo C la *capacità*, V la *tensione* di alimentazione ed f la *frequenza*.

Il consumo energetico può essere influenzato utilizzando strategie e tecniche che riguardano principalmente:

- Hardware (HW) (materiali, organizzazione e numero dei componenti)
- Software (SW) (algoritmi, protocolli, compilatori)
- Architettura

Per quanto riguarda le tecniche e le strategie software vanno sicuramente menzionate le ottimizzazioni che può effettuare un compilatore, come riordinare le istruzioni per

ridurre il numero di transizioni di bit, che sono le operazioni che maggiormente richiedono energia, o limitare l'accesso ai registri.

E' ovvio che tali ottimizzazioni influenzano in maniera più o meno pesante il consumo energetico e pertanto dipende molto dal programma in esecuzione.

Molti studi sono incentrati sulle interazioni e la comunicazione fra più nodi all'interno di una rete. Usare protocolli di comunicazione efficienti e attivare solo i nodi strettamente necessari di certo può migliorare e diminuire il consumo energetico [1, 7].

Spesso è comunque necessario introdurre dell'*overhead* nella comunicazione per trovare il giusto *tradeoff* fra il risparmio energetico e la latenza della comunicazione, aspetti in contrasto fra loro.

Per quanto riguarda tecniche a livello HW, dalla sola analisi della formula per il calcolo del consumo energetico, si intuisce che è possibile diminuire tale quantità cercando di diminuire i fattori che la compongono.

Questo è possibile fino a che non si raggiungono valori di V e di f tali da non permettere il normale funzionamento dei processori. Infatti, se da un lato è vero che diminuendo i valori di alimentazione si vedranno diminuire anche i consumi, è anche vero che tanto minore è la tensione di alimentazione tanto minori saranno i valori di tensione utilizzati per lo scambio di segnali elettrici, introducendo difficoltà nel discriminare valori o, nel caso peggiore, si è costretti ad utilizzare degli amplificatori dei segnali, annullando di fatto i vantaggi ottenuti dall'abbassamento del valore di tensione di alimentazione.

Si potrebbe pensare di dotare il sistema in esame di ulteriore memoria, anche locale. Tenere traccia di un'informazione piuttosto che calcolarla ogni volta, potrebbe aumentare l'efficienza del sistema a fronte del costo della memoria stessa.

Strategie, tecniche e metodiche per l'abbattimento di consumo energetico si concretizzano in piattaforme e ambienti di sviluppo software dedicati alla produzione di programmi da usare in sistemi embedded IoT orientati all'efficienza dell'uso delle risorse e che supportano diversi tipi di linguaggi, per esempio in C, Python o altri [8].

2. Capitolo 2 - STM32F446RE e Nucleo-board F446RE

In questo capitolo viene presentata la scheda Nucleo-Board F446RE, la piattaforma di test adottata in questo lavoro di tesi, e il relativo microcontrollore STM32F446RE. Vengono descritte le periferiche di cui è dotato il microcontrollore nonché l'organizzazione della memoria ed infine i principali *controller* che vengono utilizzati nei capitoli successivi.

Si procede descrivendo gli approcci oggi comunemente utilizzati per la programmazione del microcontrollore, presentandone nel dettaglio tre.

Il primo è rappresentato dalla piattaforma online di sviluppo MBED.

La seconda metodologia prevede l'utilizzo solamente del linguaggio C, non appoggiandosi a nessuna libreria esterna.

Il terzo metodo è basato su *Mecrisp-Stellaris,* implementazione dell'interprete Forth per la famiglia dei microcontrollori STM32.

2.1 Introduzione e architettura

STM32 è una famiglia di microcontrollori a 32-bit prodotta dalla STMicroelectronics.

I chip STM32 sono suddivisi in 10 distinte serie, ma tutte integrano processori ARM. Ogni serie si distingue dalle altre per potenza di calcolo, quantità di memoria, performance, consumo energetico, dotazioni hardware e altre caratteristiche.

I microcontrollori che offrono maggiore potenza di calcolo e maggiori quantità di memoria sono senz'altro quelli appartenenti alle serie F2, F4, F7, H7. Questi microcontrollori integrano processori ARM (Cortex-M3, Cortex-M4 o Cortex-M7), garantendo frequenze molto elevate, dai 120MHz della serie F2 ai 400MHz della

serie H7. Inoltre sono dotati di una grande quantità di memoria (raggiungendo anche i 1024kB per la serie H7).
Se da un lato aumentano le performance in termini di potenza, dall'altro non possono che diminuire quelle relative ai consumi energetici.
Per tale ragione, STM offre anche soluzioni orientate a bassi consumi energetici, come le serie L0, L1, L4. Microcontrollori appartenenti a questa serie hanno potenza di calcolo ridotte (infatti alcune integrano il più semplice Cortex-M0, con frequenze che scendono anche fino ai 32MHz) e quantità di memoria molto ridotte.
La grande varietà di microcontrollori prodotti dalla STM è prova del fatto che, a seconda dell'ambito applicativo in cui il microcontrollore sarà utilizzato, possono essere definiti vincoli differenti e non sempre il microcontrollore più *potente* è la scelta migliore.
Le Nucleo-Boards sono delle schede di prototipazione basate sui microcontrollori STM32.
In questo lavoro di tesi è stata utilizzata la Nucleo-F446RE, basata sull'omonimo microcontrollore.
L'STM32F446RE integra il processore ARM Cortex-M4. Il sistema principale è formato da un bus a matrice multilivello a 32 bit che permette la connessione fra:

- Sette Masters, fra cui:
  - Cortex-M4 con FPU core I-Bus, D-Bus and S-Bus
  - DMA1 Memory bus
  - DMA2 Memory bus
  - DMA2 peripheral bus
  - USB OTG HS DMA Bus

- Sette Slaves:
  - Memoria Flash Interna, I-Code bus
  - Memoria Flash Interna, D-Code bus
  - SRAM1 interna principale (112KB)
  - Auxiliary internal SRAM interna ausiliaria (16KB)

- AHB1 peripherals including AHB-to-APB bridge and APB peripherals
- AHB2 peripherals
- FMC/QUADSP

La matrice bus permette ai master di interagire con gli slaves, garantendo accessi concorrenti e arbitrando le richieste dei master, secondo algoritmo Round-Robin.

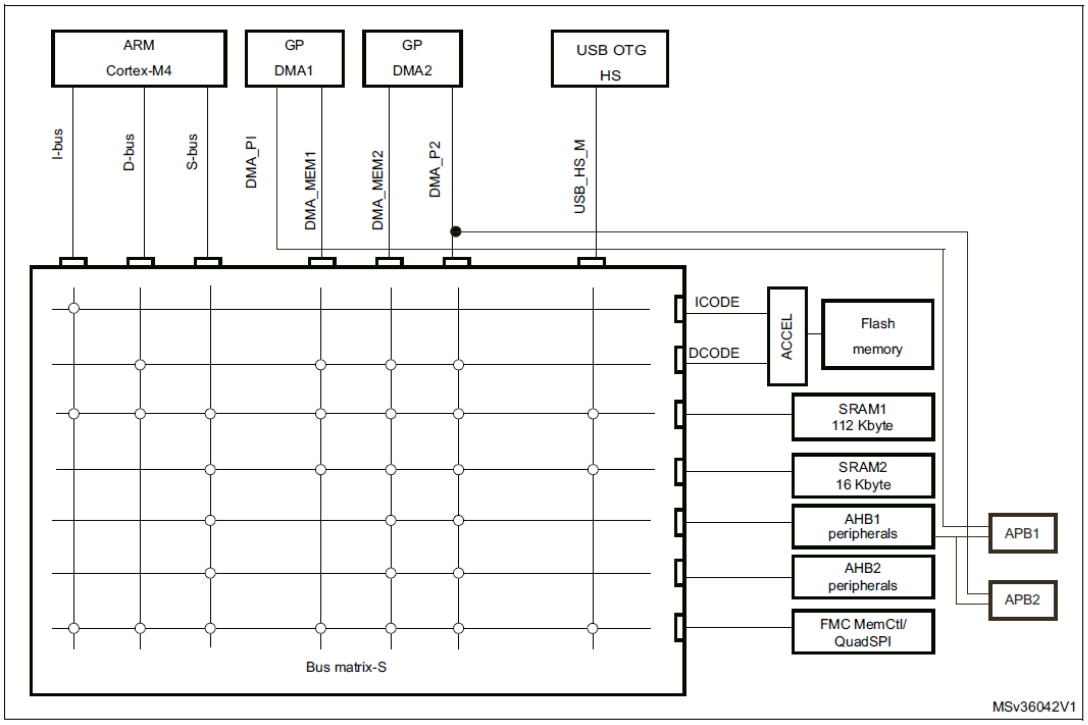

Figura 1 - Bus Matrix STM32F446RE [13]

Diversi tipi di bus interconnettono differenti parti del sistema. L'I-Bus connette il bus delle istruzioni del processore Cortex al bus matrice. Il "target" di tale bus è la memoria nel quale risiedono le istruzioni (Flash, SRAM o memoria esterna).
Il D-Bus connette il bus dati del Cortex al bus matrice. Il "target" di questo bus è la memoria che contiene i dati che saranno usati dal programma in esecuzione (Flash, SRAM o memoria esterna).
L'S-Bus connette il bus di sistema del Cortex al bus matrice. Questo bus è usato per accedere a dati che risiedono in periferiche o nella SRAM. I "target" di tal bus sono

la memoria interna SRAM, SRAM2, il bridge AHB1 (incluse le periferiche APB) e le periferiche collegate al AHB2.
Il DMA bus (Direct Memory Access) connette il bus dell'interfaccia del master DMA bus al bus matrice. E' usato dal DMA per effettuare trasferimenti da/verso le memorie. Il "target" di tale bus sono le memorie dati: memoria flash interna, memoria SRAM interne (SRAM1 e SRAM2) e memorie esterne.
I due AHB/APB *bridges*, APB1 e APB2, consentono una connessione sincrona fra AHB e i due bus APB.
Dopo ogni riavvio, il clock che arriva in ogni periferica è disabilitato eccetto ovviamente, per la SRAM e per l'interfaccia della Flash. Prima di usare le periferiche, dunque, è necessario abilitare il clock per la relativa periferica. Ciò viene eseguito tramite RCC AHBxENR.
L'STM32F446RE è dotato di un grande numero di periferiche e controller.
Timer, *General Purpose Input Output* (GPIO), interfacce di comunicazione I2C, USART e SPI, *Reset and Clock Controller* (RCC), *Nested Vector Interrupt Controller* (NVIC) sono periferiche e controller che sono state utilizzate in questo lavoro.

2.2 Programming

Vengono adesso presentate le principali metodologie che possono essere usate per lo sviluppo di applicazioni per tale microcontrollore.
Vengono descritti ed analizzati sia paradigmi di programmazione convenzionali, rappresentati dallo sviluppo con linguaggi quali C/C++, che un paradigma di programmazione simbolica basato sulla programmazione con il linguaggio Forth.

2.2.1 MBED

MBED è una piattaforma web che offre innumerevoli servizi e *tool* per la progettazione, lo sviluppo ed il *deployment* di applicazioni per microcontrollori.

Supporta un vastissimo numero di schede di sviluppo e sistemi ARM-*based*, tra cui le Nucleo Board della STM.
Fra i tanti strumenti che, dopo aver creato un account, vengono messi a disposizione, quello che si è utilizzato maggiormente è l'IDE di sviluppo.
Definito il target dell'applicazione che si intende sviluppare, si viene indirizzati in un ambiente intuitivo che garantisce all'utente un'esperienza molto vicina ai tradizionali IDE desktop.
L'ambiente permette, con riferimento alle Nucleo Board, lo sviluppo di applicazioni in linguaggi di alto livello quali C e C++.
Viene utilizzata una libreria che permette un accesso e configurazione a tutte le periferiche di cui è dotato il microcontrollore della scheda in maniera immediata. Purtroppo, risulta anche molto corposa, infatti ha dimensioni di circa 16Kb.
Una volta scritto il codice sorgente, e dopo essere stato compilato per il target selezionato, viene prodotto il file oggetto. Tale file è il risultato della compilazione e sarà caricato nella memoria del microcontrollore.
L'operazione di caricamento è resa estremamente semplice grazie alla presenza del programmatore STLink nella Nucleo-Board. Tale dispositivo espone una memoria di massa virtuale USB.
Ciò significa che, una volta ottenuto il file oggetto, basterà collegare la Nucleo-Board tramite USB al PC e copiare il file oggetto nella memoria virtuale che l'STLink espone.
Il programmatore avvia una comunicazione con il microcontrollore Cortex-M4 e provvede in maniera del tutto automatica, all'operazione di *flash* del file oggetto.
Anche se l'utilizzo MBED fornisce supporto adeguato allo sviluppo di applicazioni, la dimensione della libreria stessa ed eventuali *overhead* che vengono introdotti fanno sì che il file oggetto abbia dimensioni notevoli. Il ciclo di sviluppo è basato su cross-compilazione, pertanto in genere, si richiedono molte iterazioni del processo compilazione-flash-test, anche per piccolissime variazioni al codice. Questo è senz'altro un processo che introduce tempi morti nello sviluppo di applicazioni e che accomuna tutte le tecniche di programmazione in cui è prevista, a partire dal codice

sorgente, la generazione di un file oggetto sull'*host* e il caricamento in memoria del microcontrollore.
Altri sistemi, che sono presentati nel dettaglio nei paragrafi successivi, permettono di dotare il *target* di un ambiente interattivo e che risponda ai comandi inviati dall'utente.
Il risultato è un microcontrollore interattivo e la possibilità di testare il codice in maniera immediata, introducendo notevoli vantaggi nelle fasi di codifica e test.

2.2.2 C – Crosscompilazione con ARM Toolchain

Lo sviluppo di applicazioni embedded utilizzando la piattaforma MBED offre innumerevoli vantaggi.
Il primo fra tutti è la rapidità di sviluppo grazie alla libreria che viene messa a disposizione e che permette di pilotare tutte le periferiche hardware del microcontrollore senza avere un'approfondita conoscenza di come queste funzionino a basso livello.
Di fatto, la presenza della libreria sgrava lo sviluppatore dall'onere di dover studiare ed approfondire, tramite documentazione e *datasheets*, il funzionamento a basso livello del microcontrollore e di tutto l'hardware di cui è dotato.
Questo approccio che prevede il caricamento della piattaforma e dell'applicazione insieme a tutte le librerie, ha degli effetti sulle dimensioni finali del file binario che, anche per programmi molto semplici risulta essere notevole. Ad esempio per un *main* vuoto, privo di qualsiasi istruzione, il file oggetto ha la dimensione di circa 25kb.
Per evitare l'*overhead* di un'intera libreria che incide notevolmente sulle dimensioni finale dell'eseguibile e che, spesso espone funzionalità che non verranno mai richiamate dall'applicazione si può utilizzare il solo linguaggio C, senza l'ausilio di altri moduli.

L'idea di tale approccio è molto semplice e basilare: produrre codice sorgente in C e compilarlo con *target* architettura ARM Cortex-M4 ed eseguire il *flashing* sul microcontrollore.
Se da un lato tale approccio richiede la scrittura di tutto il software necessario per pilotare periferiche hardware e per l'accesso a basso livello, dall'altro consente di ottenere programmi eseguibili e sorgenti di dimensioni molto contenute.
Quest'approccio presuppone una profonda conoscenza dell'hardware del microcontrollore ed è stato adottato per confrontare programmi sviluppati in C con programmi basati su piattaforma simbolica come Mecrisp.
Il processo di sviluppo richiede una configurazione iniziale dell'ambiente che consiste di tutti i componenti software necessari per il processo di compilazione.
Il processo di compilazione consiste di due fasi:

- Generazione dei file oggetto, partendo dai rispettivi file sorgente
- *Linking* dei file oggetto

L'intero processo di compilazione viene eseguito su un sistema *host* il quale genera un eseguibile per il microcontrollore STM32F446RE.
In linea generale, si parla di *cross-compilazione* quando l'architettura dell'*host* che effettua la compilazione è diversa dall'architettura del *target*.
Il sistema *host* utilizzato è un calcolatore con sistema operativo *Linux Mint*.
Di seguito i passi necessari per la realizzazione di un'applicazione sfruttando la catena di compilazione per architettura ARM.
Per prima cosa è necessario installare il *cross-compilatore*.
Tale software dipenderà dal sistema operativo utilizzato e, per ambiente Linux, si installa con il seguente comando:

sudo apt-get install gcc-arm-none-eabi

In alcuni sistemi potrebbe essere necessario installare altri pacchetti (come *libnewlib-arm-none-eabi*).

Per automatizzare il processo di compilazione si è scritto un *Makefile* il quale definisce le dipendenze, in funzione del *goal*, fra i vari file. Analizzando il Makefile emerge maggiormente la suddivisione del processo di compilazione in *generazione file oggetti* e *linking*.
Relativamente al processo di *linking* è stato necessario realizzare uno script di configurazione che indica dove mappare in memoria le varie sezioni del codice.
Per lanciare la compilazione, e generare l'eseguibile, basta eseguire:

make nomePrg.bin

Il file binario generato viene copiato, a questo punto, nell'unità virtuale USB che espone il programmatore della Nucleo-Board, effettuando così il caricamento del programma. Il codice relativo al *linker* e al *Makefile* sono visualizzabili in Appendice.

2.2.3 Mecrisp-Stellaris

Mecrisp-Stellaris è un'implementazione del linguaggio Forth per microcontrollori ARM-based.
Prima di descrivere nel dettaglio l'ambiente Mecrisp-Stellaris è utile introdurre il linguaggio Forth.
Forth è un linguaggio di programmazione sviluppato da Chuck Moore nei primi anni '60 ma che venne formalizzato soltanto nel 1977. Il nome deriva dal fatto che l'inventore, Moore, riteneva dovesse essere un *linguaggio di quarta generazione (fourth generation language)* ma il computer sul quale il sistema fu sviluppato ammetteva solo nomi di cinque lettere.
Il linguaggio è interpretato che implementa un paradigma di programmazione simbolica basato sui concetti di *stack* e *dizionario*.
Lo *stack* è un'area attiva della memoria nella quale il Forth mantiene la rappresentazione dei numeri e dei dati utilizzati nei processi di elaborazione.

lo stack implementa una struttura dati di tipo LIFO, pertanto solo l'ultimo valore aggiunto è direttamente disponibile. L'inserimento nello stack avviene in maniera trasparente all'utente, infatti basta digitare un valore, ed esso sarà scritto direttamente sullo stack. Dualmente, vi è la possibilità di estrarre il valore in testa allo stack.
L'insieme dei simboli che possono essere eseguiti vengono chiamate *word* e sono memorizzate in un dizionario di simboli. Un insieme di base di word (un *core)* sono già definite e disponibili ma è possibile definirne di proprie senza alcuna distinzione tra le parole già definite e quelle definite dall'utente.
Tale linguaggio, poiché consente di scrivere applicazioni mediante sequenza di simboli di alto livello, si presta bene alla creazione di Domain Specific Languages (DSLs) mediante la definizione di tutte quelle parole specifiche del dominio applicativo.
Il processo di inserimento di una word nel dizionario viene chiamato *compilazione* e ha la seguente sintassi:

```
: Word_name   List_of_words ;
```

List_of_words rappresenta il corpo della definizione della *word* e forma il codice che verrà eseguito quando si utilizzerà la word *Word_name*.
Inoltre l'operatore **:** (due punti), anche se apparentemente potrebbe sembrare un comando specifico di Forth o un solo elemento sintattico, in realtà è anch'esso una *word* che consente all'inetrprete di entrare in modalità di compilazione.
L'interprete scansiona lo *stream* d'input, individuando stringhe separate da spazio. Tale stringa, se presente nel dizionario, viene interpretata ed eseguita. Se non è presente nel dizionario, viene interpretato come numero; in quel caso, viene inserito sulla cima dello stack. Se nessuno dei passi precedenti va a buon fine, viene generato e segnalato all'utente un errore.
Concetto fondamentale di Forth, come si è detto, è l'accesso diretto allo stack. Vengono utilizzati due tipi di stack. Lo *stack dei parametri* (o semplicemente *stack*) viene utilizzato per il passaggio di parametri fra le chiamate di diverse *word*. Infatti,

l'uso tipico è quello di *lasciare* sullo stack dei valori (i parametri) che saranno poi eventualmente consumati dalla *word* che si sta invocando.
Lo *stack di ritorno* viene utilizzato per conservare i puntatori necessari a tenere traccia del flusso di esecuzione delle word. Tale *stack*, nonostante sia utilizzato per gestire il flusso di esecuzione fra le word, può anche essere utilizzato dallo sviluppatore per conservare valori temporanei.
Altra caratteristica è l'uso di una notazione *fixed point* (piuttosto che *floating* point) per la rappresentazione di numeri razionali.
Un numero razionale ha una parte intera (prima della virgola) ed una decimale (dopo la virgola). Nella notazione a virgola fissa, supponendo di avere una codifica a N bit, vengono utilizzati M bit per la parte intera ed (N-M) bit per la parte decimale.
Invece, nella notazione a virgola mobile (che sarebbe l'equivalente dell'usuale notazione scientifica) la sequenza di bit utilizzata per rappresentare il numero reale viene suddiviso in due parti: la *mantissa*, M, e l'*esponente*, E.
Un generico numero reale, *n,* in base *b*, utilizzando una notazione floating point, può essere rappresentato come:

$$n = M \cdot b^{E}$$

Il numero di bit, *p,* riservato per la codifica della mantissa determina la *precisione* del valore *a*. Tale notazione permette di rappresentare un amplissimo insieme numerico, molto più grande di quello rappresentabile con la notazione a virgola fissa. Infatti qualsiasi numero viene definito dalla sola conoscenza del valore della mantissa e dell'esponente, rendendo possibile, di fatto, la rappresentazione di numeri estremamente piccoli (atomici) o estremamente grandi (astronomici).
La ragione circa l'uso della notazione in virgola fissa è legata questioni di efficienza. Molti sistemi presenti durante lo sviluppo di Forth non erano dotati di *Floating Point Unit (FPU),* unità di calcolo specializzata nell'aritmetica di numeri razionali con notazione a virgola mobile) e quindi, linguaggi che, invece, prevedevano l'uso di numeri razionali con notazione a virgola mobile utilizzavano delle librerie o moduli

software che simulavano l'aritmetica *floating point*. Tale livello introduce una perdita di efficienza nei programmi che a volte, soprattutto in ambito di sistemi embedded con risorse limitate, non sono ammissibili. L'uso di un'aritmetica a virgola fissa evita questi possibili problemi rendendo, di fatto, il linguaggio molto più efficiente sia dal punto di vista della memoria che dalla velocità di esecuzione. In realtà, non è neanche necessario gestire i numeri con una notazione a virgola fissa; si può far uso solo di interi senza perdere di espressività. Tutto ciò che può essere realizzato utilizzando un'aritmetica a virgola fissa, può essere implementato anche usando soltanto aritmetica intera. Basta portare tutte le grandezze alla stessa unità di misura tramite operazione di *scaling* e definendo la precisione della grandezza. Se la grandezza in questione è relativa ad una tensione, espressa in Volt [V] con parte decimale, e la più piccola variazione a cui si è interessati è il millivolt, si può esprimere la stessa grandezza con una notazione intera in cui viene utilizzata l'unità di misura del millivolt [mV]. In questo modo è possibile esprimere qualsiasi grandezza (intrinsecamente rappresentabile mediante notazione a virgola fissa/mobile) utilizzando soltanto numeri interi. I vantaggi si riscontrano i termini di efficienza ed in termini di comunicazione con altri sistemi. Infatti, gestire numeri interi richiede molti meno calcoli rispetto al gestire numeri in altre notazioni così come inviare grandezze rappresentate tramite interi richiede meno conversioni ed evita eventuali errori di interpretazione.

Mecrisp è un'implementazione *stand-alone* nativo per controllori della famiglia MSP430. E' estremamente compatto e richiede soltanto 512 byte di RAM. Offre la possibilità di compilare su flash o RAM.

Mecrisp-Stellaris, sviluppato da Matthias Koch, può essere considerato "il fratello minore" di Mecrisp e gira su un grande numero di microcontrollori ARM basati su processore Cortex-M.

Fra i microcontrollori ARM-based supportati da Mecrisp-Stellaris, rientrano anche i microcontrollori appartenenti alla famiglia STM32, incluse le Nucleo-Board.

Come accennato precedentemente, tale famiglia è formata da un grande numero di microcontrollori accomunati da un'architettura comune, ma con caratteristiche che

possono variare anche notevolmente in base al tipo di microcontrollore, ad esempio in termini di periferiche, consumi, potenza di calcolo.
Sono state sviluppate tante versioni di Mecrisp-Stellaris, al fine di poter supportare un grande numero di Nucleo-Board. Tutte le implementazioni utilizzano un *core* di componenti software. Tale scelta è resa possibile grazie al fatto che microcontrollori della STM32 sono accomunati da un'architettura comune.
Oltre ai binari, sono disponibili anche i sorgenti di Mecrisp-Stellaris, scelta che favorisce lo sviluppo di implementazioni per microcontrollori non attualmente supportati partendo da un'implementazione già presente e modificando solo le parti necessarie.
Si procede adesso descrivendo la configurazione dell'ambiente utilizzato e analizzando tutti i passi necessari per effettuare il flash del binario di Mecrisp-Stellaris nel microcontrollore STM32F446RE.
Il sistema operativo del sistema *host* è Linux Mint.
Il *flashing* dell'eseguibile viene effettuato utilizzando il *tool* a linea di comando *st-flash*. Questo non comunica direttamente con il processore, il Cortex-M4, ma comunica con il programmatore ST-LINK della Nucleo-Board. E' infatti il programmatore ad effettuare il flash nel processore.
Prima di installare st-flash, bisogna installare alcuni pacchetti necessari per soddisfare le dipendenze.
La prima libreria da installare è *libusb-1.0-0-dev,* installabile tramite il gestore dei pacchetti presente nel sistema (*apt)* lanciando il seguente comando

apt-get install libusb-1.0-0-dev

A questo punto si può procedere con il download dei sorgenti del pacchetto st-flash dalla piattaforma *github*. Dopo il download, posizionarsi nella directory contenente il sorgente appena scaricato (*cd stlink.git)* e procedere con la compilazione tramite *make*.
I comandi da eseguire sono i seguenti:

git clone https://github.com/texane/stlink stlink.git

make

Nella directory *./flash*, se la compilazione è andata a buon fine e se non si sono verificati errori, sarà presente l'eseguibile *st-flash*. Copiare tale eseguibile in una directory presente nel PATH (si consiglia /usr/bin) o aggiungere la directory corrente alla variabile PATH (per poter essere lanciato da qualsiasi directory).
A questo punto la configurazione dell'ambiente è completa.
Il flash viene effettuato lanciando il seguente comando:

st-flash write nomeBinario.bin address

I nomi sono molto intuitivi: *nomeBinario.bin* è il binario di Mecrisp-Stellaris, mentre *address* rappresenta l'indirizzo dal quale iniziare a scrivere il binario. Nelle Nucleo Board F446RE tale valore sarà 0x8000000 che è l'inizio della memoria flash.
St-flash permette di effettuare anche letture dalla memoria, utile in fase di debug, lanciando il comando.

st-flash read output.bin address length

Anche in questo caso i nomi sono auto-esplicativi: la lettura inizierà dall'indirizzo *address* e leggerà esattamente *length* byte, salvando il tutto nel file *output.bin*.
A questo punto si può proseguire avviando *Mecrisp*. In realtà, Mecrisp-Stellaris sarà in esecuzione sul processore subito dopo averlo caricato il binario.
Bisogna avviare una comunicazione seriale per poter interagire con l'interprete Forth. Per far ciò si è utilizzato il terminale seriale *minicom*. Per avviare la comunicazione bisogna lanciare il comando

minicom –D /dev/ttyACMx –b 115200

Il parametro *–D* permette di specificare il *device* con il quale avviare la comunicazione, mentre con *–b* si specifica il *baud rate* della comunicazione

3. Capitolo 3 – Benchmark

In questo capitolo si analizzeranno alcune delle caratteristiche, relativamente alle metodologie e ai linguaggi di programmazione visti ed analizzati nel Capitolo 2, mediante la definizione di software di benchmark. Per ogni benchmark viene fornita un'implementazione sia in linguaggio C che in Forth.
Si è ritenuto opportuno e proficuo suddividere i benchmark in più *suite*. La prima si pone l'obiettivo di evidenziare delle caratteristiche generali dei linguaggi (LOC, memoria) insieme all'analisi dei consumi energetici mediante misurazioni *off-board*, mentre la seconda, denominata *suite energy aware* mira all'analisi dei consumi energetici mediante misurazioni effettuate *on-board*.

3.1 Suite di base

La suite di base contiene software di benchmark che vengono valutati in modalità *off-board*.
Le misurazioni *off-board* vengono effettuate utilizzando un microcontrollore di supporto che, comunicando con il sensore INA219, rileva i consumi energetici relativi al *target* di misurazione e li invia al sistema *host*.
Lo schema dei collegamenti per le misurazioni *off-board* è mostrato in figura 2

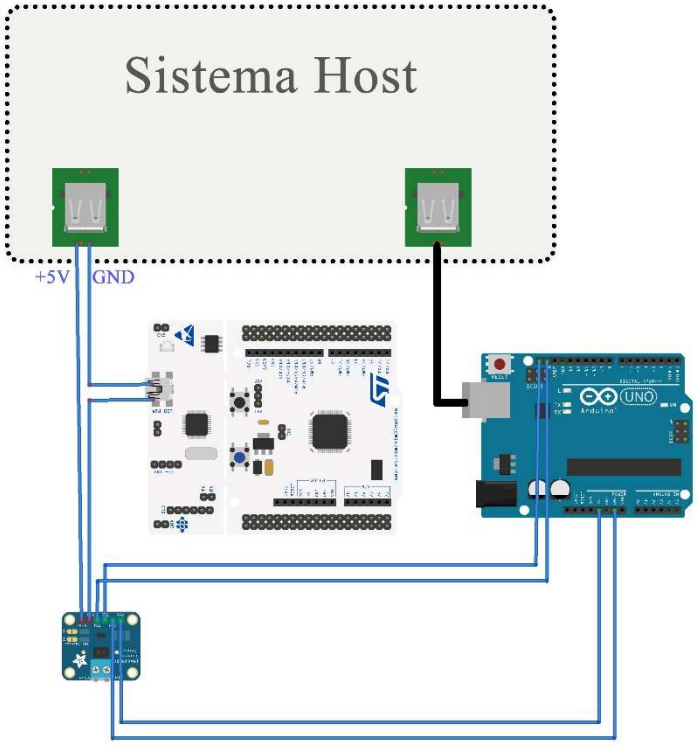

Figura 2 - Schema dei collegamenti off-board

La Nucleo-Board, il target di misurazione, viene alimentata dal sistema *host* mediante un cavo USB al quale viene collegato in serie il sensore INA219, che comunica con il microcontrollore Arduino. L'Arduino acquisisce periodicamente campioni energetici e li invia al sistema *host*.

Secondo questa architettura il target delle misurazioni è completamente indipendente dal resto del sistema e per questo non viene influenzato in nessun modo.

I campioni rilevati, dunque, sono relativi esclusivamente al consumo energetico della Nucleo-Board.

I benchmark coinvolgono alcune periferiche di cui è dotato il microcontrollore STM32, fra cui GPIO, timer e led.

3.1.1 Blink idle-loop

Il primo benchmark esegue il *blink* del led utente collegato al pin 5 della *GPIO_A*.

Per far ciò è necessario abilitare il clock alla *GPIO_A* e configurarne opportunamente i registri.
I passi da seguire, sostanzialmente, sono:

- Abilitare il clock alla GPIO_A;
- Settare opportunamente i registri della GPIO_A (*MODER* E *PUPDR*) come mostrato in Appendice;
- Effettuare lo *switch* dello stato del led modificando i bit relativi al pin 5 del registro ODR.

Il blink viene effettuato utilizzando un delay *busy wait*, cioè bloccante. Inoltre, viene calibrato per far cambiare stato al led una volta ogni 3 secondi.

3.1.2 Blink timer-IRQ

Analogamente al benchmark precedente, obiettivo di tale software è quello di effettuare il *blink* del led.
In questo benchmark, però, a differenza del precedente, avviene utilizzando e configurando opportunamente un timer (in questo caso, tim2) il quale genera un interrupt. A tale interrupt segue l'esecuzione dell'ISR associata all'interrupt del timer, la quale non fa altro che cambiare stato al led.
Le operazioni necessarie per l'implementazione di tale benchmark sono le seguenti:

- Abilitare il clock alla GPIO A
- Configurare opportunamente i registri MODER e PUPDR della GPIO
- Abilitare il clock al timer (tim2)
- Abilitare la gestione degli interrupt nella periferica NVIC (relativo, ovviamente, al timer 2)
- Configurare opportunamente i registri del timer 2 (prescaler, autoreload, generazione degli interrupt)
- Definire l'ISR relativa all'interrupt del timer 2. Tale ISR non deve far nient'altro che cambiare stato al led
- Avviare il timer

Per i dettagli di tali operazioni consultare il codice disponibile in appendice.

3.2 Suite Energy Aware

La suite di Energy Aware consiste degli stessi benchmark della suite base.
Le valutazioni vengono eseguite però in un'ottica di *Energy Aware Computing*.
Ciò si traduce nel fatto che, a differenza di quanto accade negli esperimenti relativi alla suite di base, in questo caso le misurazioni vengono effettuate *on-board*.
In questo modello di acquisizione dati, il microcontrollore target della misurazione ed il microcontrollore che esegue la lettura dal sensore, coincidono.
Si ha, in fine, un microcontrollore che, oltre ad eseguire i test di benchmark, nello stesso tempo ed a intervalli regolari, comunica col sensore di cui è dotato e acquisisce i dati relativi al proprio consumo energetico.
Tali dati vengono poi inviati al *sistema host,* il calcolatore che esegue analisi sperimentali sui dati ricevuti.
Lo schema dei collegamenti è mostrato in figura 3.

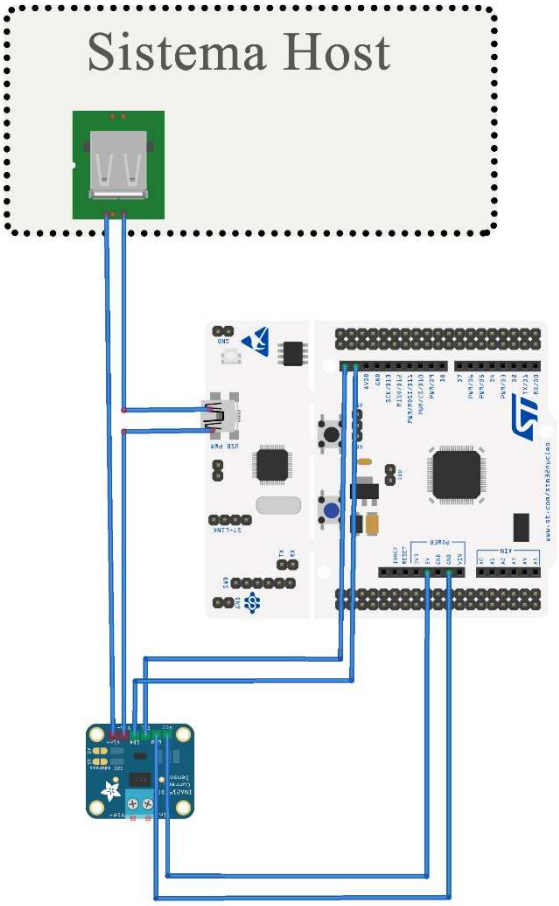

Figura 3 - Schema dei collegamenti on-board

In un'implementazione reale o in fase di produzione, tali dati potrebbero essere gestiti dallo stesso microcontrollore il quale, avendo coscienza dei suoi stessi consumi, può decidere di attuare tecniche di risparmio energetico dinamiche in funzione a quelli che sono i task da eseguire.

3.3 Definizione parametri di benchmark

Per poter effettuare valutazioni relativamente ai benchmark è necessario l'utilizzo di parametri che siano facilmente quantificabili.

Per gli scopi di questo lavoro si è deciso di adottare delle grandezze che misurino, sotto diversi punti di vista, delle caratteristiche del codice legate all'efficienza.
Di seguito la descrizione dettagliata di tali parametri con alcune motivazioni che hanno portato al loro utilizzo.

3.3.1 LOC – Lines of Code

LOC, *Lines Of Code,* misura la dimensione di un codice sorgente analizzando, appunto, il numero di linee di codice sorgente.
Il conteggio delle LOC nasce con i linguaggi *line-oriented* (C, Assembly e FORTRAN). In questi casi il numero delle linee di codice forniva effettivamente una misura della complessità del software. Oggi, nei linguaggi con altri paradigmi di programmazione (ad oggetti, per esempio) questo non è sempre vero.
Nonostante ciò, la valutazione basata sulle LOC è ancora oggi ampiamente utilizzata, sia perché in molti contesti riesce a dare una stima molto vicina alla realtà della complessità del software, sia perché il calcolo è molto semplice e viene effettuato in modo automatizzato.
In questo lavoro, il calcolo delle LOC viene impiegato per ottenere delle valutazioni sulla compattezza e su quanto sia complesso, in termini di quantità di codice, scrivere lo stesso software implementato nei diversi linguaggi che si stanno analizzando.

Nel calcolo della metrica LOC è bene porre particolare ad alcuni aspetti tipici del mondo della programmazione.
Si osservino i due codici sorgenti seguenti.

```
for(i=0; i<100; ++i) printf("hello");
```

```
for(i=0; i<100; ++i){
     printf("hello");
 }
```

Tali codici, in linguaggio C, eseguono esattamente la stessa funzione ma, come si può vedere, sembra che abbiano differenti valori di LOC.
A tal proposito, è utile distinguere le *Physical LOC* dalle *Logical LOC*.
Le *Physical LOC* misurano semplicemente il numero di linee di codice così per come si presenta (includendo nel conteggio, dunque, commenti e linee vuote).
Le *Logical LOC*, invece, misurano il numero di *statements,* ovvero le effettive istruzioni (nel caso del linguaggio C, uno statements è rappresentato da ogni istruzione che termina con il *;*)
Nel seguito di questa trattazione, si adotterà il criterio delle Physical LOC, conteggiando dunque tutto il codice sorgente nella sua interezza.

3.3.2 Consumo di memoria

E' giusto precisare che le LOC non fanno riferimento a nessuna informazione circa il consumo di memoria. Quando ci si riferisce alla quantità di memoria occupata, si fa riferimento, in linea generale, alle dimensioni del codice eseguibile.
In contesti di sviluppo di applicazioni per sistemi embedded tale metrica riveste un ruolo fondamentale e, potrebbe essere determinante nella scelta di quale linguaggio utilizzare.
Infatti, molte volte si ha a che fare con dispositivi e microcontrollori con quantità di memoria molto esigue e, l'occupazione di memoria, potrebbe essere un requisito fondamentale per lo sviluppo e l'implementazione dell'applicazione. Tale parametro indica la dimensione esatta del file eseguibile.

4. Capitolo 4 – Setup sperimentale

Questo capitolo descrive il setup sperimentale utilizzato, dettagliandone componenti e architettura, insieme ad alcuni concetti preliminari relativi al protocollo di comunicazione I2C.
Il capitolo descrive inoltre l'implementazione dei protocolli di comunicazione che sono coinvolti in fase di acquisizione delle misurazioni energetiche.
Poiché le comunicazioni fra il microcontrollore e il sensore INA219 avvengono mediante protocollo I2C, è stato necessario sviluppare il software di un driver I2C per il microcontrollore STM32 che verrà descritto nel presente capitolo.
Differentemente, la comunicazione fra il microcontrollore che effettua le misurazioni energetiche e il sistema *host*, a cui le misurazioni vengono inviate, avviene tramite connessione seriale. Pertanto, il capitolo include la descrizione del protocollo sviluppato per consentire la comunicazione tra il microcontrollore e il sistema *host*.
Tali software sono parte integrante e fondamentale del setup sperimentale ed hanno permesso, in maniera agevole, l'esecuzione degli esperimenti e di tutti i processi relativi all'acquisizione dei dati relativi ai consumi energetici.

Il setup sperimentale consiste, oltre alle già descritte *Nucleo Board* , di un sensore di corrente e tensione.
L'INA219 è un sensore che permette il rilevamento di grandezze di corrente, tensione di shunt e tensione del bus di alimentazione comunicando con altri dispositivi utilizzando il protocollo I2C.
Il sensore va alimentato con una tensione di 3.3/5.0V e funziona in un intervallo di temperatura compreso fra -40° e +125°.
Il diagramma logico è il seguente:

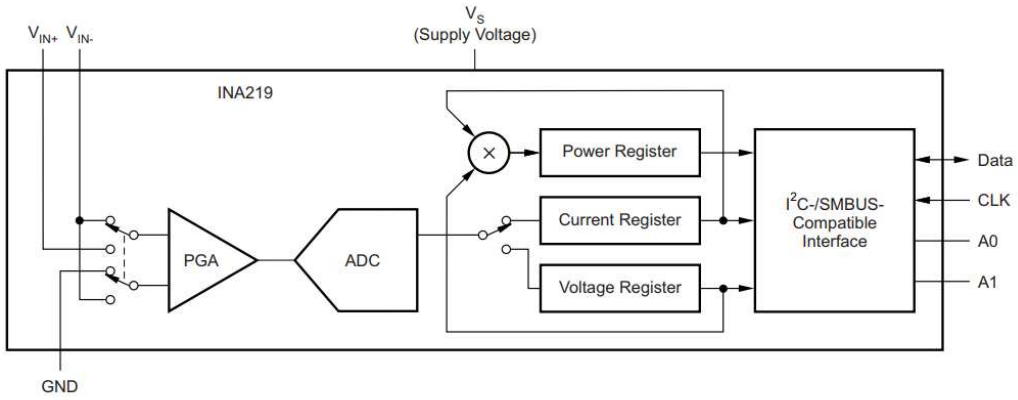

Figura 4 - Schema logico INA219 [12]

Mentre di seguito una rappresentazione del package, nelle due versioni disponibili:

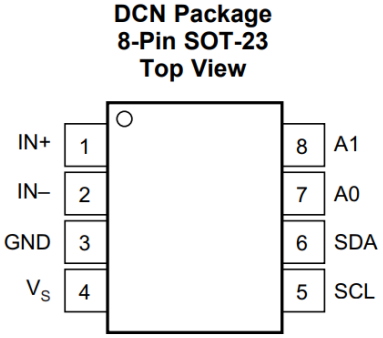

Figura 5 - Package e pin [12]

L'INA219 comunica con altri dispositivi tramite protocollo I2C.

In questo lavoro di tesi si è utilizzata una breakout board prodotta dalla *Adafruit* che utilizza il chip integrato INA219. Sul modulo sono già presenti tutti i componenti necessari per il funzionamento del chip integrato INA219. Inoltre, è disponibile un grande numero di librerie disponibili che permettono un semplice uso del chip.

La breakout è quella mostrata nella figura seguente:

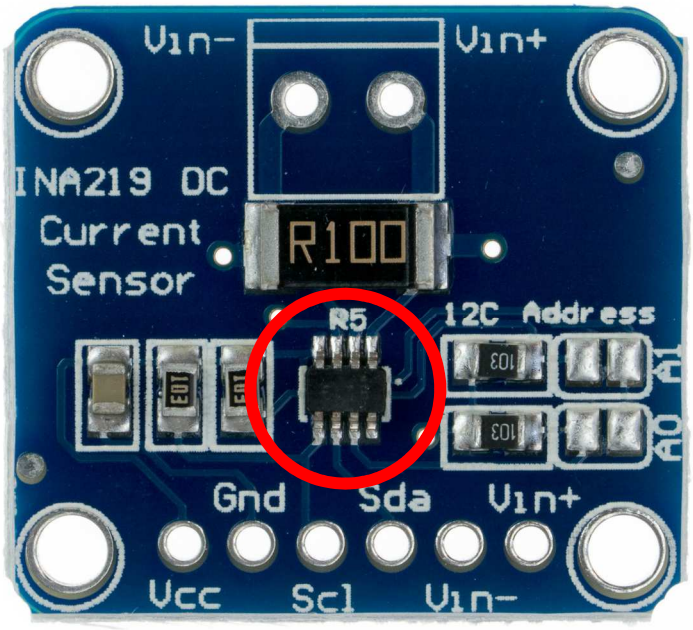

Figura 6 - INA219 all'interno della breakout board

Come illustrato in Figura 6, i pin sono esattamente 6 (considerando che i pin Vin+ e Vin- sono duplicati). Il chip centrale (cerchiato in rosso) è esattamente l'INA219 e si possono vedere i suoi 8 pin.

I 2 pin che non vengono esposti sono A0 ed A1 e sono quelli che vengono utilizzati per l'assegnazione dell'indirizzo I2C del dispositivo. Ciò significa che l'interfaccia I2C di cui è dotato l'INA219 avrà un indirizzo diverso a seconda di come vengono cortocircuitati i due pin A0 ed A1 al momento dell'alimentazione. Le combinazioni possibili sono quelle mostrate in Figura 5.

A1	A0	SLAVE ADDRESS
GND	GND	1000000
GND	V_{S+}	1000001
GND	SDA	1000010
GND	SCL	1000011
V_{S+}	GND	1000100
V_{S+}	V_{S+}	1000101
V_{S+}	SDA	1000110
V_{S+}	SCL	1000111
SDA	GND	1001000
SDA	V_{S+}	1001001
SDA	SDA	1001010
SDA	SCL	1001011
SCL	GND	1001100
SCL	V_{S+}	1001101
SCL	SDA	1001110
SCL	SCL	1001111

Figura 7 - Indirizzi I2C INA219 [12]

Nonostante l'INA219 permetta grande flessibilità nella scelta dell'indirizzo I2C da utilizzare, nella breakout tali pin sono saldati e dunque l'indirizzo è definito e non è possibile modificarlo (a meno di non dissaldare i contatti e cambiare la configurazione dei collegamenti).

Nella configurazione di fabbrica i pin A0 ed A1 sono collegati con GND; per tale motivo, l'indirizzo I2C della breakout è 0x40 (in esadecimale).

L'INA219 è dotato di alcuni registri utilizzati per effettuare le letture ed altri per la configurazione.La comunicazione I2C prevede che, prima di effettuare la lettura di un registro, debba essere avviata una sessione di comunicazione in modalità scrittura ed indicato il registro che si intende leggere, rappresentato da un numero intero.

Tale valore sarà memorizzato dall'INA21 che, nella successiva sessione di comunicazione I2C in modalità lettura, invierà i dati contenuti nel registro indicato al passo precedente.

Tale contatore viene incrementato in maniera automatica per permettere di eseguire delle letture sequenziali di tutti i valori presenti nei registri.

Poiché l'INA219 comunica con gli altri dispositivi tramite protocollo I2C, si è deciso, di effettuare le analisi sui campioni non direttamente sul microcontrollore.

Dopo aver progettato e sviluppato i protocolli di comunicazione necessari, è stato possibile effettuare gli esperimenti con estrema semplicità, anche in caso di modifiche dei parametri inziali (come frequenza di campionamento, numero di campioni o dispositivo con il quale interfacciarsi).

4.1 Implementazione driver I2C

Inter Integrated Circuit (I2C), come accennato precedentemente, è un protocollo di comunicazione seriale utilizzato fra circuiti integrati.
Ha un'architettura Master-Slave e, nella maggior parte dei casi, si vedono coinvolti un Master ed uno (o più) slave.
Il Master inizia la comunicazione contattando lo slave il quale, dopo aver ricevuto la richiesta, risponde con l'invio dei dati richiesti.
I2C è stato originariamente sviluppato da Philips nel 1982. La specifica originale prevedeva solo comunicazioni a 100kHz e indirizzi a 7 bit, limitando il numero di dispositivi presenti sullo stesso bus a 112 (infatti, 16 indirizzi sono riservati e non possono essere liberamente utilizzati). Nel 1992 fu pubblicata la prima specifica pubblica, aggiungendo una modalità *veloce* di comunicazione (400kHz) e estendendo lo spazio di indirizzamento portando il numero di bit dedicati all'indirizzo a 10. Successivamente sono state pubblicate altre versioni I2c che aumentavano le velocità di trasmissione (arrivando anche a supportare comunicazioni a 5MhZ).
Nello stesso periodo, Intel ha introdotto una variante di I2C, chiamata SMBus (System Management Bus). A differenze di I2C che permette velocità che vanno fino ai 5MhZ, SMBus permette velocità comprese nel range 10-100KhZ.
Trattandosi di un protocollo di comunicazione seriale il vantaggio principale è quello di impegnare solo due linee di comunicazione. Viene utilizzato da un grande numero di dispositivi come, appunto, microcontrollori, memorie, driver hardware.
Le due linee richieste dal protocollo sono *SDA (Serial Data)* utilizzata per i dati e *SCL* (*Serial Clock),* per la sincronizzazione della comunicazione. Anche se è il master

che ha il controllo del clock, alcuni dispositivi possono parzialmente controllarlo (tramite quello che viene chiamato meccanismo di *stretching del clock)* per rallentare la comunicazione nel caso in cui i dati da inviare non siano ancora disponibili.

Per la presenza della liea di clock (che viene pilotta dal master) il protocollo è detto sincrono.

Ogni dispositivo I2C ha un indirizzo che, nelle prime versioni di I2C, aveva dimensioni pari a 7 bit, definendo quindi uno spazio di indirizzamento con 128 possibili indirizzi diversi. In realtà, sedici indirizzi sono riservati e non possono essere liberamente utilizzati, portando il numero di dispositivi contemporaneamente collegabili in uno stesso bus I2C a 112. Nelle versioni successive di I2C, il numero di bit dedicato all'indirizzo venne esteso permettendo l'uso di indirizzi a 10 bit, aumentando di gran lunga il numero di dispositivi potenzialmente collegati allo stesso bus.

Anche se i dispositivi sono dotati di interfacce dedicate alla comunicazione I2C, le quali permettono di gestire la connessione con maggiore semplicità, la comunicazione I2C è certamente più complessa da implementare rispetto altri tipi di protocolli o sistemi (UART o SPI).

I messaggi sono divisi in due tipi di frame: un *frame indirizzo,* in cui il master indica lo slave con il quale intende iniziare la comunicazione ed uno o più *frame dati,* sequenze di dati formati da 8 bit inviati dal master allo slave o viceversa.

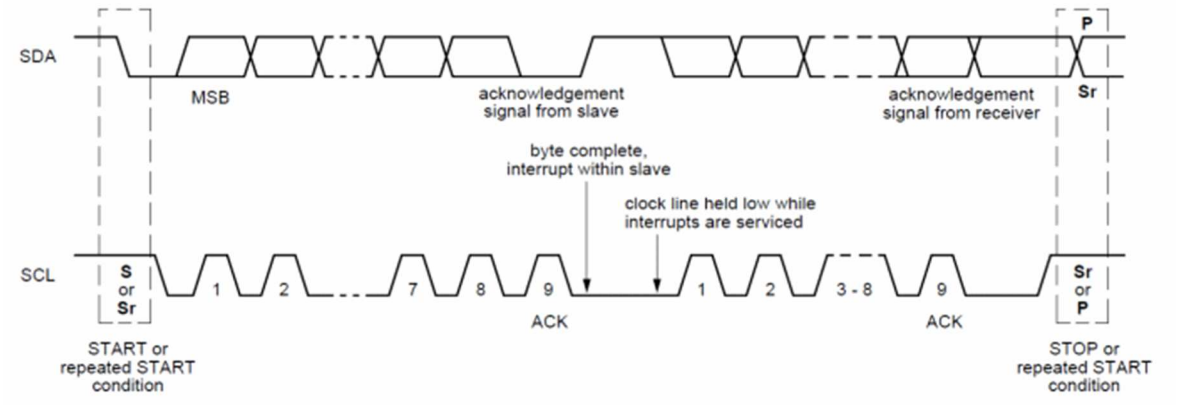

Figura 8 - Protocollo di comunicazione I2C [18]

Per inizializzare la comunicazione, e dunque inviare il frame indirizzo, deve inviare un segnale di start. Tale segnale viene definito *Start Condition* ed è definito dalla transizione di SDA da alto a basso mentre SCL resta alto, come mostrato in figura. La *Start Condition* notifica a tutti gli *slave* che un *master* intende iniziare la comunicazione.

Inviato il segnale di start, il master procede all'invio del frame indirizzo. Per indirizzi a 7 bit, il primo bit ad essere inviato è il bit più significativo (MSB), seguito da un bit (chiamato bit R/W) che indica se l'operazione che intende eseguire il master è una lettura (valore del bit 1) o di scrittura (valore del bit a 0). Un nono bit (bit di ACK) viene utilizzato per determinare se lo slave, individuato dall'indirizzo inviato precedentemente, ha ricevuto il messaggio.

Dopo che l'invio dell'indirizzo è stato completato, si procede con l'invio dei dati. I dati vengono inviati dal master allo slave o viceversa a seconda del bit R/W inviato nel frame indirizzo. I dati vengono così inviati come impulsi ad intervalli regolari in funzione del segnale di clock. In questa fase possono essere inviati più frame dati.

Una volta che tutti i frame dati sono stati inviati, la comunicazione può terminare. Per far ciò, il master invia (analogamente a quanto succede per inizializzare la comunicazione) un segnale di stop (*Stop Condition*). Tale condizione viene definita con la transizione da basso ad alto del segnale SDA mentre SCL è alto, come mostrato in figura.

Spesso capita che la velocità del clock pilotato dal master è troppo elevata ed è superiore alla velocità con cui lo slave può inviare i dati. Questo, per esempio, può capitare quando lo slave è rappresentato non può procedere all'invio dei dati richiesti dal master perché è in attesa di un'operazione che non si è ancora conclusa (per esempio, una lettura dalla memoria). In questo caso, lo slave può eseguire quello che viene chiamato *clock stretching*. Anche se è il master a pilotare il segnale di clock, lo slave può mantenere basso il segnale SCL dopo che il master lo ha rilasciato. Il master non può procedere all'invio di altri segnali di clock fino a quando non viene rilasciato

dallo slave. Questo è un meccanismo che permette e garantisce un alto grado di flessibilità in quanto il master può comunicare con dispositivi che hanno velocità di trasferimento inferiore rispetto alla propria.

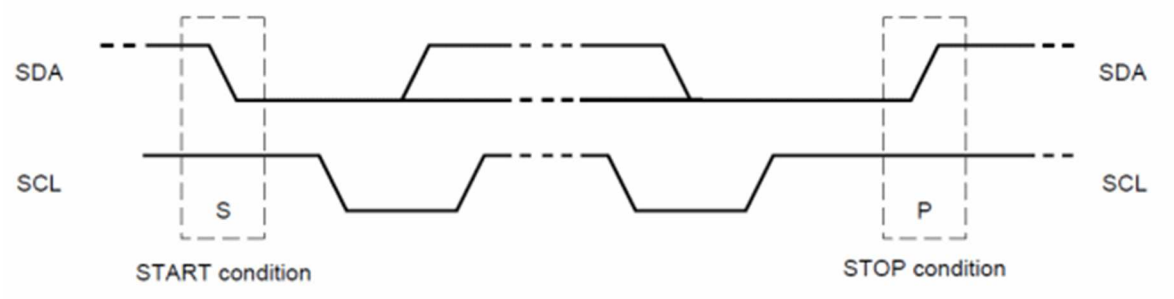

Figura 9 - Start condition e Stop condition [18]

Questi appena discussi sono aspetti del tutto generali di I2C ed indipendenti da qualsiasi dispositivo.

Ogni dispositivo, poi, sarà dotato di tutta la circuiteria necessaria per l'implementazione di tale protocollo.

Si descriverà adesso l'insieme dei dispositivi hardware di cui è dotato l'STM32 che permettono la realizzazione della comunicazione tramite I2C.

L'STM32 è dotato di più interfacce I2C (denominate con *I2C_1*, *I2C_2, I2C_3*), permettendo così una flessibilità maggiore.

L'interfaccia è dotata di un protocollo che converte i dati che devono essere trasmessi da un formato parallelo ,che, in genere, sono contenuti nella memoria o nei registri, in un formato seriale e viceversa.

Una caratteristica molto interessante e che permette un alto grado di operabilità è che l'interfaccia può ricoprire il ruolo sia di Master che di Slave.

Quando viene utilizzata come Master, allora può, in accordo con le specifiche del protocollo I2C, generare il segnale di clock ed inviare i segnali di Start e Stop.

Nella modalità Slave, invece, permette di definire l'indirizzo (al quale risponde in caso di richiesta dal master) ed ha la funzionalità di rilevamento del bit di stop.

L'interfaccia I2C supporta sia l'indirizzamento a 7 bit che quello a 10, oltre ad implementare anche i meccanismi per il supporto delle *General Call* (l'equivalente di richieste effettuate in broadcast).

Possono essere utilizzate differenti velocità di comunicazioni: la prima, che viene definita *standard speed*, arriva fino a 100kHz mentre, la *fast speed,* può arrivare fino a 400kHz.

L'interfaccia è dotata anche di funzionalità avanzate quali la presenza dei filtri di rumore e l'individuazione di condizioni particolari di errori quali quelli inerenti i segnali di start/stop, quelli che riguardano gli ACK che vengono trasmessi o quelli relativi allo *stretching* del clock.

Per quanto riguarda la generazione degli interrupt, l'interfaccia può generarne di due tipi: il primo in caso di successo nella trasmissione di indirizzo/data; il secondo in caso di errore.

Il flusso della comunicazione va distinto nei casi in cui l'interfaccia operi come master o come slave.

Nel caso in cui venga utilizzata come master, ha il compito di iniziare la comunicazione e di generare il clock. La comunicazione avviene sempre inviando il segnale di start e termina sempre con l'invio della condizione di stop. Tali condizioni vengono generate agendo opportunamente su particolari registri, che verranno descritti nel seguito, di cui è dotata l'interfaccia.

Nella modalità slave, l'interfaccia è in grado di riconoscere il suo *indirizzo* (quando questo viene inviato dal master) e risponde anche alle general call (anche se può, opzionalmente, essere disabilitato).

I dati scambiati, detti anche *payload* della comunicazione, vengono inviati in sequenze di 8 bit. I primi dati seguono la start condition rappresentano l'indirizzo (1 Byte in caso di indirizzamento a 7 bit, 2 Byte nel caso di indirizzamento a 10 bit). L'indirizzo viene sempre trasmesso dal master dopo che invia la start condition.

L'immagine seguente rappresenta la struttura dell'interfaccia I2C ed i componenti di cui è formata, descrivendo come questi interagiscono.

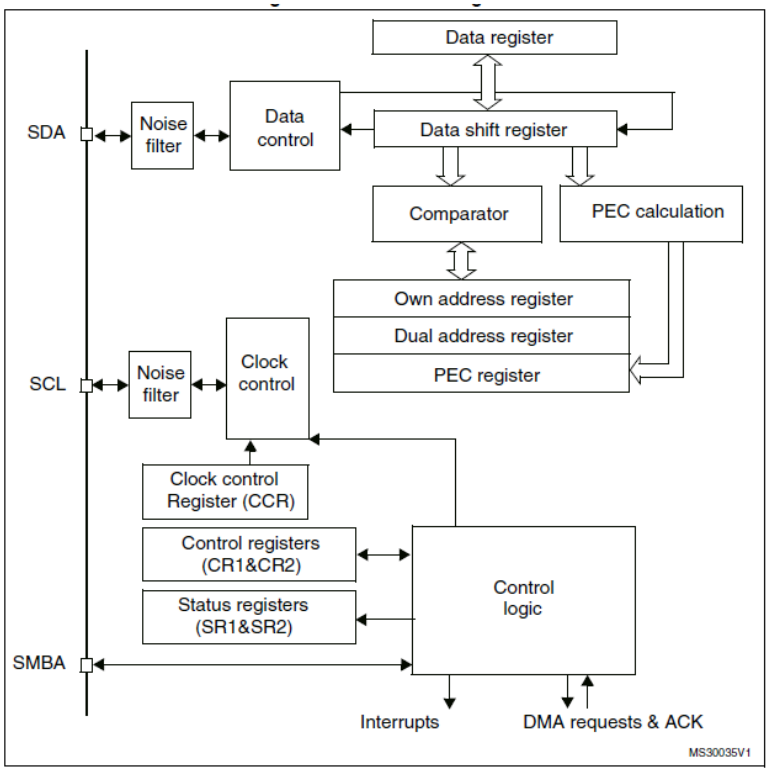

Figura 10 - Schema logico interfaccia I2C [13]

L'interfaccia, dal momento che può essere utilizzata sia in modalità master che slave, può trovarsi, in qualsiasi momento, in una di queste 4 configurazioni:

- Master transmitter
- Maste receiver
- Slave transmitter
- Slave receiver

Poiché l'obiettivo è permettere lo scambio di dati e informazioni fra il microcontrollore STM32 ed il sensore INA219, l'interfaccia si troverà, in questo caso, sempre in modalità master. Per tale motivo, si devono solo considerare le prime due configurazioni (Master transmitter e master receiver).

L'STM32 che inizierà la comunicazione e che genererà il clock necessario per la comunicazione. Il numero di registro viene comunicato all'INA219 nel byte successivo a quello dell'indirizzo (che viene inviato dal master).

Ciò significa che, se per esempio, l'STM32 volesse leggere il valore della tensione, deve prima avviare una comunicazione I2C in modalità scrittura con il sensore e, dopo aver inviato l'indirizzo I2C dell'INA, deve procedere all'invio del numero di registro che intende leggere (valore chiamato *count*).
Il microcontrollore STM deve iniziare poi una seconda sessione di comunicazione (stavolta in modalità lettura) con il sensore il quale invia i dati presenti nel registro individuato dal valore *count*.
Per una migliore comprensione della stessa, si procede con una descrizione del flusso che dovrà avvenire fra il microcontrollore STM32 e il sensore INA219:

1. L'STM32 inizia la comunicazione in modalità scrittura. Per far ciò genera il segnale di clock, invia la start condition e procede con la trasmissione del primo byte (formato da 7 bit che rappresentano l'indirizzo dell'INA219 e dall'ottavo bit che avrà valore 0, ad indicare la modalità scrittura).
2. L'INA riconosce il suo indirizzo nella linea dati e, dato che il bit successivo ai 7 dell'indirizzo ha valore 0, si mette in attesa di ricevere *count*, il numero del registro che, successivamente, dovrà inviare al master.
3. Il master (STM32) riceve l'ACK e procede con l'avvio di un'altra sessione di comunicazione (stavolta in modalità lettura). Invia dunque start condition e l'indirizzo dell'INA, seguito sta volta dall'ultimo bit con valore 1, il quale specifica che la comunicazione è da intendersi in lettura.
4. L'INA219 riconosce il suo indirizzo sulla linea dati e, dal valore dell'ultimo bit, "capisce" che la comunicazione è di lettura. Invia così i valori contenuti nel registro identificato da *count* il quale viene incrementato per un'eventuale comunicazione successiva (in modo che il master riceva il valore del registro successivo). Ricordandosi che i registri dell'INA sono a 16 bit, il valore contenuto nel registro viene inviato 1 byte alla volta.
5. L'STM32 riceve così il valore del registro desiderato (formato da 2 byte) e invia la stop condition.

Con riferimento ad una singola sessione di comunicazione I2C, in cui il master richiede dati e lo slave risponde, si può rappresentare il flusso dei messaggi scambiati con una notazione *ad eventi*, che è quella che il datasheet dell'STM32 adotta.
Ogni azione che viene effettuata durante la comunicazione viene associata ad un evento il quale rappresenta ed identifica "lo stato corrente" della trasmissione.
La figura seguente sintetizza e rappresenta i possibili eventi che vengono generati durante una sessione di comunicazione *master-transmitter*.

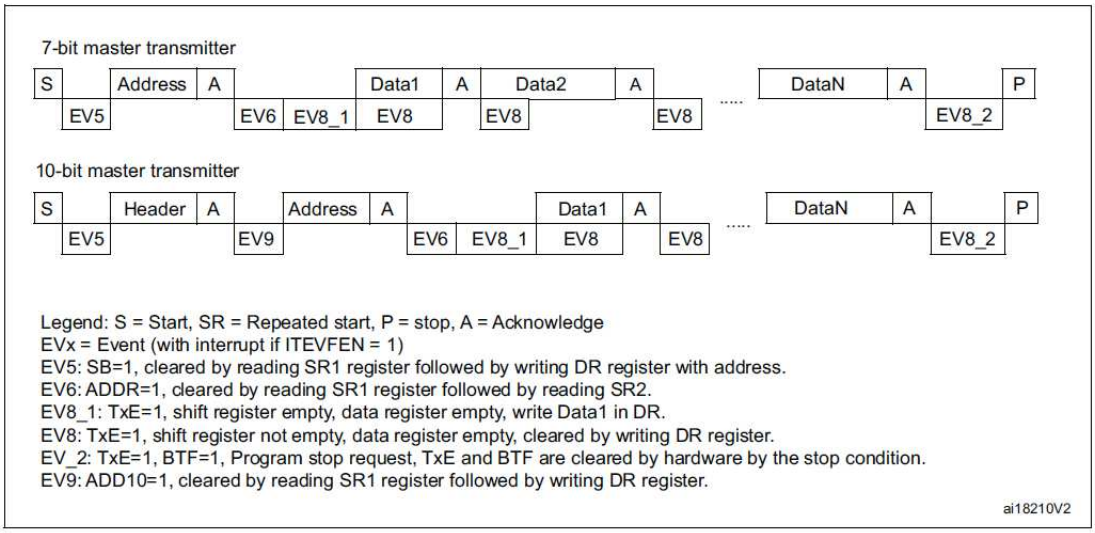

Figura 11 - Eventi I2C Master Transmitter [13]

Nella figura 9, il primo caso fa riferimento ad una comunicazione utilizzando un indirizzamento a 7 bit, mentre il secondo caso fa riferimento ad una comunicazione con indirizzamento a 10 bit.
Gli eventi rappresentati nella figura precedente vengono utilizzati per determinare lo stato della comunicazione e, di conseguenza, pilotare l'interfaccia I2C per eseguire le giuste operazioni (quelle previste dal protocollo).
Ogni evento modifica lo stato di alcuni valori dei registri dell'interfaccia. Alcune variazioni vengono effettuate dall'hardware direttamente mentre altre, si richiede che vengano eseguite da software.

L'evento *S* rappresenta la *start condition* e viene inviato dal master. Viene generato settando a 1 il bit 8 (bit *start*) del registro *CR1* (*Control Register 1*).
L'evento *EV5* viene generato effettuando una lettura del registro SR1 seguita da una scrittura del registro DR (data register, il registro che contiene i dati da inviare).
A questo punto l'interfaccia I2C procede inviando i dati presenti in DR, che in questa fase dovrà essere l'indirizzo della periferica I2C con la quale si intende iniziare la comunicazione (per esempio, verrà specificato l'indirizzo I2C dell'INA219).
Terminata la trasmissione dell'indirizzo e dopo aver processato l'ACK, viene generato l'evento *EV6* (individuato dalla condizione *ADDR = 1,* bit 1 del registro *SR1 – Status Register 1)*. Per individuare tale evento, in un approccio molto semplicistico e, probabilmente, poco efficiente, si potrebbe effettuare una *busy-wait* sul bit ADDR.
Leggendo i registri *SR1 e SR2,* viene generato dunque l'evento *EV8_1*. Tale evento viene individuato da *TxE = 1* (bit 7 di *SR1)*. Questo evento indica che la fase di trasmissione dell'indirizzo è terminata e bisogna procedere con la trasmissione dei dati. Per far ciò bisogna inserire in *DR* i dati che si intende inviare.
Non appena viene effettuata la scrittura in *DR,* viene generato l'evento *EV8* (che indica che ci sono dati da inviare) e i dati presenti in *DR* vengono inviati.
Si può, in questa fase, monitorare i bit coinvolti negli eventi *EV8* e *EV8_1* per inviare più byte.
Tutte le trasmissioni avvengono secondo il medesimo meccanismo ad eccezione dell'ultimo byte.
Nella trasmissione dell'ultimo byte deve essere indicato che si tratta dell'ultima trasmissione che si intende eseguire. Per far ciò, nella trasmissione dell'ultimo byte va settato il bit *BTF = 1* (bit 2 di *SR1)*. In questo modo l'interfaccia I2C invierà, dopo la trasmissione degli ultimi 8 bit, la *stop condition*, terminando di fatto la comunicazione.
Nella figura Figura 12 sono mostrate le parole che, in accordo alla documentazione ufficiale e con quanto descritto, implementano ed identificano gli eventi necessari per la comunicazione I2C.

```
: i2c-DR! ( value -- )    I2C1 DR c! ;
: i2c-DR@ ( -- value )    I2C1 DR c@ ;
: i2c-start! ( -- )             8 1BIT MASK I2C1 CR1 hbis! ;
: i2c-stop! ( -- )        9 1BIT MASK I2C1 CR1 hbis! ;
: i2c-clear-AF              10 1BIT MASK I2C1 SR1 hbic! ;
: i2c-set-ACK               10 1BIT MASK I2C1 CR1 hbis! ;
: i2c-SR2-flag? ( u -- ) I2C1 SR2 hbit@ ;
$ffff variable i2c.timeout
: i2c-MSL?   ( -- b )     0 1BIT MASK I2C1 SR2 hbit@ ;
: i2c-SR1-flag? ( u -- b ) I2C1 SR1 hbit@ ;  ( lascia sullo stack un Bool,
corrispondente alla maschera u )
: i2c-SR1-wait   ( u -- )  i2c.timeout @ begin 1- 2dup 0= swap i2c-SR1-flag?
or until 2drop ;
: i2c-SR2-!wait  ( u -- )  i2c.timeout @ begin 1- 2dup 0= swap i2c-SR2-flag?
0= or until 2drop ;
: i2c-nak?          ( -- b ) 10 1BIT MASK i2c-SR1-flag? ;
: i2c-START  ( -- ) i2c-start! ;
: i2c-EV5 i2c-SR1-SB i2c-SR1-wait ;
: i2c-EV6a i2c-SR1-ADDR i2c-SR1-AF or i2c-SR1-wait ;
: i2c-EV6b I2C1 SR1 h@ DROP  I2C1 SR2 h@ DROP ;
: i2c-EV6  i2c-EV6a  i2c-EV6b ;
: i2c-EV7  i2c-SR1-RxNE i2c-SR1-wait ;
: i2c-STOP       (  --  )  i2c-stop! i2c-SR2-MSL i2c-SR2-!wait ;
: I2C-PROBE ( c -- nak )  I2C-START i2c-EV5 shl i2c-DR!  i2c-EV6 i2c-nak?
                          i2c-clear-AF  I2C-STOP ;
: i2c-EV8_1  i2c-SR1-TxE  i2c-SR1-wait ;
: i2c-EV8   ;
: i2c-EV8_2  ( i2c-EV8_1 ) i2c-SR1-BTF i2c-SR1-TxE OR  i2c-SR1-wait ;
```

Figura 12 – Codice Forth per gli eventi I2C in Mecrisp-Stellaris

In Appendice è allegato l'intero codice Forth del driver che implementa la comunicazione I2C.

4.2 Implementazione del protocollo di comunicazione microcontrollore-sistema *host*

Nel processo di acquisizione dei dati relativi ai consumi energetici, si è reso necessario progettare e sviluppare un'infrastruttura che permettesse, in modo agevole e stabile, la comunicazione fra i microcontrollori e la macchina che esegue l'analisi sui dati stessi.

A tal proposito distinguiamo il microcontrollore *target* della misurazione ed il microcontrollore *host* del programma di misurazione.

Il target è il microcontrollore di cui si intende acquisire i dati relativi ai consumi energetici. L'host del programma di acquisizione è rappresentato dal microcontrollore su cui viene caricato ed eseguito il programma che, comunicando con il sensore INA219, acquisisce i dati relativi ai consumi energetici.

E' bene sottolineare il fatto che i due microcontrollori (target di misurazione e host) potrebbero coincidere; è il caso delle misurazioni effettuate *on-board*, in cui lo stesso microcontrollore che si intende analizzare, esegue e comunica con l'INA219 per l'acquisizione dei dati.

Qualsiasi sia la configurazione usata, non si può prescindere dal calcolatore che effettua analisi e calcolo di parametri statistici dei campioni ricevuti.

Il microcontrollore su cui è caricato il programma che effettua le misurazioni può essere rappresentato da una Nucleo-board STM32F446RE o da un Arduino Uno (ATMega 328P).

Il calcolatore, dunque, deve essere in grado di comunicare con entrambi i microcontrollori.

L'idea alla base di tale sistema di comunicazione è che il microcontrollore che deve effettuare ed inviare le misurazioni, si mette "in ascolto" ed in attesa di ricevere, da parte del calcolatore, i parametri della comunicazione.

Il calcolatore, dunque, invia i parametri, inizializzando così il microcontrollore il quale procede con l'acquisizione dei dati e con l'invio dei dati.

La comunicazione avviene tramite seriale 8N1 (lunghezza dati 8 bit, senza bit di parità e con 1 bit di stop).

Il programma che implementa la comunicazione (lato calcolatore) è scritto in C ed è rappresentata dalla configurazione della comunicazione, che avviene sfruttando la struttura dati *struct termios*.

La struttura termios è così definita:

```
struct termios {
     tcflag_t c_iflag;
     tcflag_t c_oflag;
     tcflag_t c_cflag;
     tcflag_t c_lflag;
     cc_t c_cc[NCCS];
     speed_t c_ispeed;
     speed_t c_ospeed;
};
```

Una variabile di tipo termios astrae un'interfaccia seriale (ed i suoi registri) e ne permette, settando opportunamente i valori degli attributi, la configurazione.
L'interfaccia seriale prevede due modalità di funzionamento: *canonica* o *non canonica*.Nella modalità canonica l'input viene processato a *linee*. Una linea è delimitata da un carattere \n o di fine file (*EOF*).
Una richiesta di lettura non viene completata fino a quando non viene letta un'intera riga o non viene ricevuto un segnale. Inoltre, indipendentemente dal numero di byte che sono stati richiesti in lettura, verrà sempre letta, al più, una riga.
Dunque, in modalità canonica, tutto l'input/output che si trova a gestire l'interfaccia di comunicazione viene elaborato in linee. Inoltre, in genere, vengono eseguite conversioni di caratteri del tutto trasparenti allo sviluppatore (ciò significa che i byte letti potrebbero essere diversi da quelli che sono stati effettivamente inviati).
Tale modalità è indicata in contesti *line-oriented* (si pensi alla manipolazione di stringhe, testo, file testuali…) ma non possono essere usati in contesti byte-oriented (in cui si richiedono vincoli sul numero di byte da leggere/inviare).
Nella modalità canonica l'elaborazione dell'input/output è orientata al byte ed è possibile specificare come, tali byte, verranno elaborati.
Per l'implementazione di questo programma si è utilizzata la modalità non canonica, con una configurazione tale da ricevere 4 byte per volta (in accordo con le dimensioni utilizzate per la rappresentazione dei valori sia in ambiente mecrisp-stellaris sia nel microcontrollore Arduino.

Oltre la configurazione propria della comunicazione (come baud rate, modalità, echo, bit di parità) il programma permette anche la definizione di alcuni parametri relativi all'acquisizione dei dati e dunque agli esperimenti.
In particolar modo, è possibile specificare:

- Dispositivo che esegue l'acquisizione dei dati
- Tipologia di dispositivo che esegua l'acquisizione
- Numero di campioni da acquisire
- Intervallo temporale fra un campione ed il successivo
- Nome dei file di output

Il *dispositivo che esegue l'acquisizione dei dati* è individuato dal path con cui il microcontrollore è individuato dal sistema operativo Linux. Tale path è in genere qualcosa che assomiglia a */dev/ttyUSBx* o */dev/ttyACMx*.
L'introduzione di tale parametro è stata necessaria a causa del fatto che, ad ogni collegamento della periferica (che sia la Nucleo o Arduino) il sistema operativo assegna un identificativo che può essere differente. Dunque, per evitare di modificare il valore dell'identificativo hard-coded e di rieseguire la compilazione prima dell'esecuzione di ogni processo di acquisizione dei dati, si è preferito parametrizzare tale valore.
La *tipologia* relativa al dispositivo fa riferimento al tipo di microcontrollore che eseguirà l'acquisizione dei dati.
Infatti, come accennato precedentemente, l'acquisizione dati può essere eseguita dalla Nucleo Board con ambiente Mecrisp-Stellaris o dalla scheda Arduino, in cui è presente un programma sviluppato con il suo ambiente ufficiale.
Anche se, concettualmente e funzionalmente, il protocollo di comunicazione che deve essere utilizzato fra il microcontrollore ed il calcolatore è lo stesso, devono essere previsti alcuni accorgimenti realizzati ad-hoc nel caso in cui il microcontrollore che esegue l'acquisizione è la Nucleo Board.
Infatti, mentre nel caso di Arduino basta inizializzare la comunicazione inviando alcun parametri, come numero di campioni e intervallo fra un campione e l'altro, nel

caso di Nucleo e Mecrisp-stellaris, la comunicazione viene inizializzata con l'invio di una stringa, anche in questo caso contenente numero di campioni e ritardo fra un campione e l'altro, che rappresenta una word di Mecrisp.
Per evitare di dover realizzare due programmi che implementassero lo stesso protocollo con qualche lieve differenza, si è preferito introdurre tale parametro e distinguere, applicativamente, tali differenze.
Numero campioni e *intervallo* fra un campione e l'altro definiscono, invece, i parametri propri dell'esperimento. L'intervallo temporale fra un campione e l'altro è in stretta relazione con la frequenza di campionamento. Tale parametro viene specificato in millisecondi.
Il programma, terminato l'invio dei campioni, procede effettuando un salvataggio dei dati ottenuti.In particolar modo, vengono memorizzati (su file) i campioni relativi alla corrente, alla tensione di shunt ed alla tensione del bus di alimentazione.
Ogni tipologia di campioni viene memorizzata in un file separato contenente, oltre ai valori dei campioni, anche altre informazioni come media ed altri parametri statistici.
Tali parametri vengono specificati durante l'avvio (che avviene da terminale) del programma con la classica notazione linux "*-flag valueFlag"*.
Di seguito è riportato un esempio di avvio del programma.

./serial.out -t arduino -D /dev/ttyACM0 -n 1000 -d 700 -o test01.txt

Questa configurazione prevede la comunicazione con il dispositivo (un arduino) individuato dal sistema operativo con la stringa /dev/ttyACM0.
L'esperimento consiste di 1000 campioni, uno ogni 700 millisecondi.
I risultati sono salvati in 3 file (uno per ogni tipologia di misurazione) con prefisso test01.
Spiegato nel dettaglio quali sono le funzionalità del programma, si procede adesso ad una descrizione del protocollo di comunicazione.

Il protocollo prevede che il calcolatore inizi la comunicazione inviando, sequenzialmente, prima il numero di campioni e dopo il delay fra un campione e l'altro. Tali valori sono identificati da interi senza segno a 32 bit.
Nel caso in cui il microcontrollore sia un Arduino, il calcolatore invia semplicemente questi due valori. Invece, se il microcontrollore è l'STM32 con Mecrisp-Stellaris, il calcolatore invia la seguente stringa:

Numero_campioni *Delay_campioni* `GO`

Questo presuppone che nel microcontrollore con ambiente Mecrisp-Stellaris esista una word `GO`.
Successivamente verrà descritta nel dettaglio l'implementazione di tale word.
Inviati i primi dati al microcontrollore (due semplici valori nel caso di Arduino o la stringa `GO` nel caso di Nucleo con Mecrisp-Stellaris), il calcolatore resta in attesa di ricevere i campioni.
Per ogni *campionamento* vengono acquisiti tre tipi di grandezze:

- Tensione di Shunt
- Tensione del bus di alimentazione
- Corrente

Particolare attenzione va riposta alla rappresentazione, in termini di bit, con cui vengono gestiti tali valori.
Nel caso di microcontrollore Arduino, tali valori vengono codificati come float a 32 bit; nel caso di Mecrisp-Stellaris, invece, vengono codificati, come si può vedere dal sorgente, come interi con segno a 32 bit.
Terminata la fase di acquisizione e popolati gli array con i campioni (di tensione shunt/bus e corrente), si procede con delle analisi. In particolar modo, vengono calcolati, per ogni tipologia di campioni, media e deviazione.
La deviazione standard, viene calcolata come:

$$\sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}}$$

essendo μ la media aritmetica dei campioni.

La deviazione da un'idea di quanto i campioni distino dalla media. Tanto più tale valore risulta essere basso, tanto più i campioni risultano essere *condensati* attorno la media.

Le informazioni così ottenute (valori dei singoli campioni, media e deviazione), vengono salvate in file.

L'interfaccia del programma permette di tenere sotto controllo, durante tutta la durata dell'acquisizione, informazioni utili relative all'esperimento quali l'ultimo campione rilevato, la percentuale di completamento ed il tempo residuo.

Di seguito è mostrata l'interfaccia durante l'esecuzione di un esperimento

Figura 13 - Software per la comunicazione microcontrollore-sistema host

Mentre, terminato l'esperimento, la schermata mostrata è la seguente:

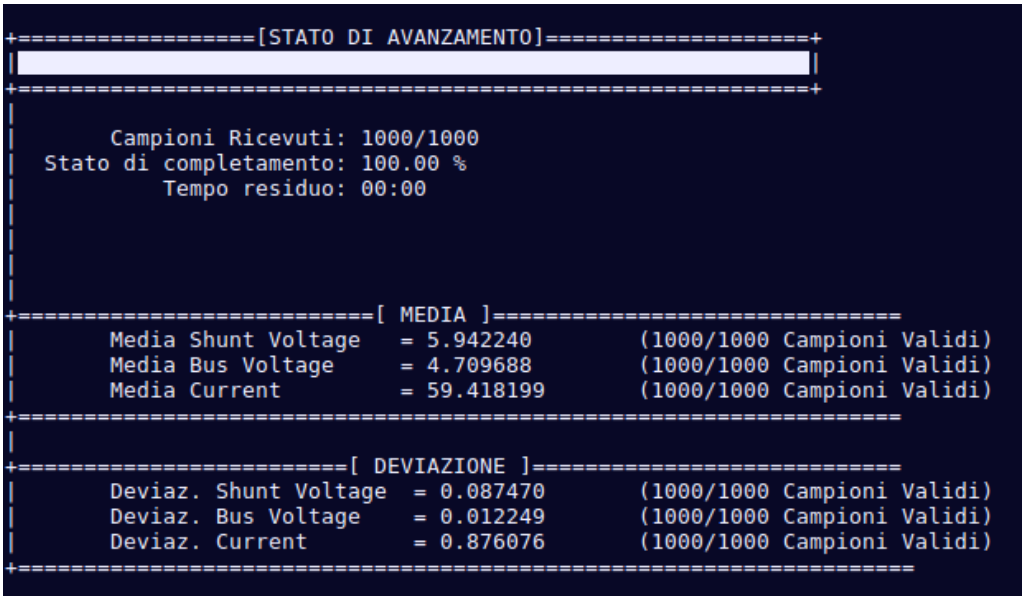

Figura 14 - Schermata finale esperimento

5. Capitolo 5 – Valutazione sperimentale suite di base

Nel corso di questo capitolo vengono presentati i risultati ottenuti mediante test di benchmark insieme alle differenti configurazioni hardware che sono state considerate.

La valutazione energetica dell'ambiente simbolico Mecrisp-Stellaris è stata confrontata con i risultati ottenuti mediante l'implementazione in C. Il confronto è stato effettuato anche alla luce dei parametri di benchmark, definiti nel dettaglio nel Capitolo 3.

Alla fine di ogni esperimento, vengono poi realizzati dei grafici utili per il confronto dei risultati ottenuti nelle due implementazioni.

Le misurazioni vengono effettuate in modalità *off-board*, come in Figura 2.

Come si nota osservando la Figura 2, il target di misurazione (Nucleo-board F446RE) esegue soltanto il codice di benchmark, mentre le misurazioni vengono effettuate da un microcontrollore esterno, AVR ATMEGA324P a bordo dell'Arduino Uno.

In tale architettura, il sensore è collegato al microcontrollore dell'Arduino Uno il quale, comunicando tramite I2C con l'INA219 e ottenendo i campioni delle misurazioni energetiche, procede con l'invio dei campioni al sistema *host*, secondo l'architettura ed il sistema descritto nel Capitolo 4.

Gli esperimenti da "Esperimento 0" a "Esperimento 7" rappresentano differenti configurazioni hardware del microcontrollore in cui un numero differente di periferiche sono abilitate.

Le periferiche vengono abilitate incrementalmente, partendo da una configurazione in cui sono tutte disabilitate, fino ad arrivare ad una configurazione con un alto numero di periferiche/interfacce abilitate.

A seguire sono presentate le caratteristiche comuni a tutti gli esperimenti della Suite di Base.

Per ciascun esperimento sono stati rilevati 1000 campioni con periodo di campionamento pari a 700 millisecondi.
I grafici riportati, senza perdere di generalità, mostrano soltanto i risultati relativi a 250 campioni. Ciò è stato possibile in quanto i campioni presentano un andamento periodico ed in particolar modo, a *lisca di pesce*. Ad ogni modo, le valutazioni statistiche relative a media e varianza vengono comunque calcolate sulla base dei 1000 campioni.
Nella descrizione della configurazione hardware, si assume che tutto quello non esplicitamente menzionato sia disabilitato.

5.1 Esperimento 0 – Baseline

Poiché la Nucleo-Board è dotata, fra le altre cose, di un programmatore, questo esperimento consente di analizzare e raccogliere grandezze energetiche del solo programmatore. L'obiettivo è appunto valutarne il consumo energetico per poi, eventualmente, sottrarre i risultati ottenuti alle valutazioni energetiche degli altri esperimenti, così da escludere l'apporto dovuto al programmatore stesso.
Per disabilitare il microcontrollore e alimentare solo il programmatore, basta rimuovere il jumper evidenziato in Figura 14. In figura 15 è mostrato l'andamento della tensione di Shunt man mano che i campioni vengono raccolti nel tempo, mentre la Figura 16 illustra l'andamento della corrente in relazione ai campioni raccolti. L'andamento della tensione sul bus è invece riportata in Figura 17.

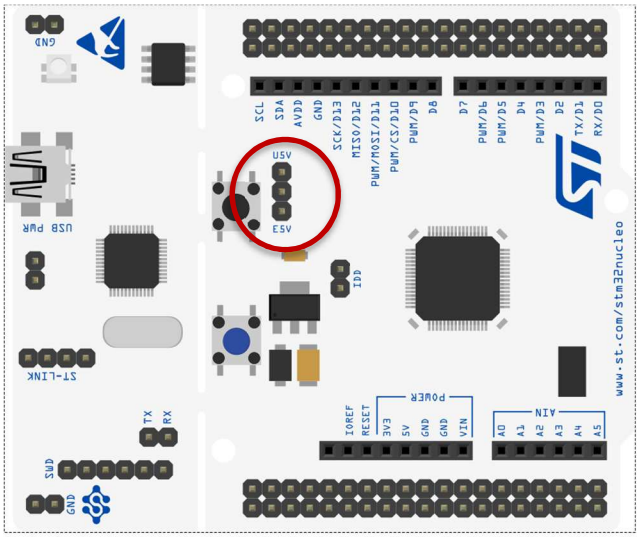

Figura 15 - Jumper Alimentazione nucleo-programmatore

La media è calcolata, come descritto nell'introduzione di questo capitolo, su 1000 campioni ottenuti campionando con periodo di campionamento pari a 700 millisecondi.

In generale, i risultati ottenuti sono stati riportati in figura considerando soltanto 250 campioni, che sono sufficienti a mostrare l'andamento delle grandezze fisiche misurate dato che l'andamento è periodico (a *lisca di pesce*).

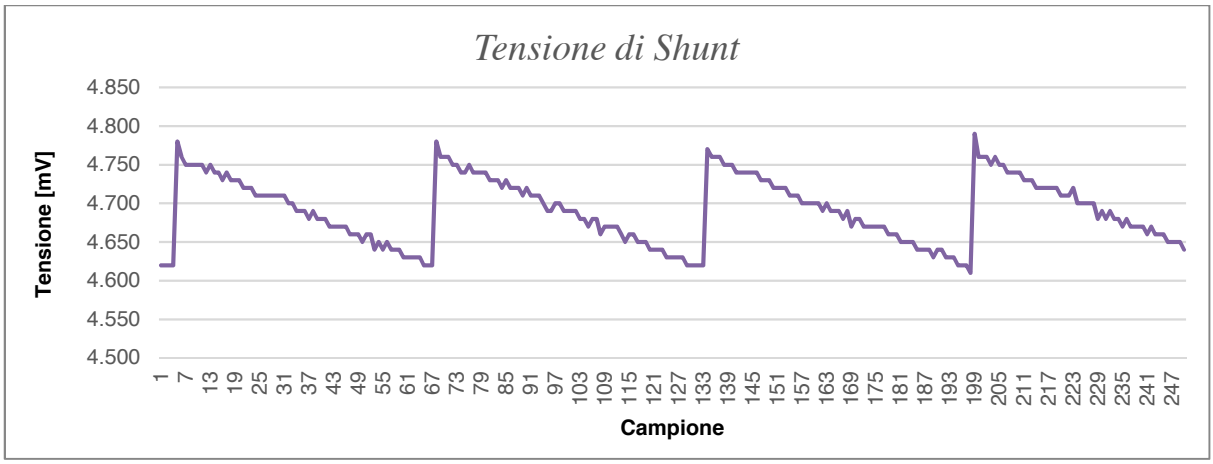

Figura 15 – Tensione di Shunt

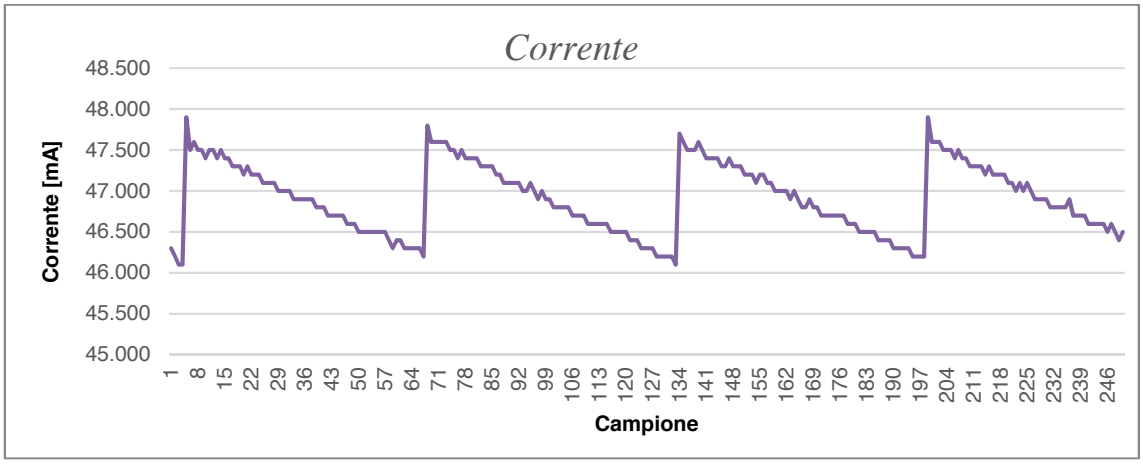

Figura 16 – Andamento della corrente per 250 campioni

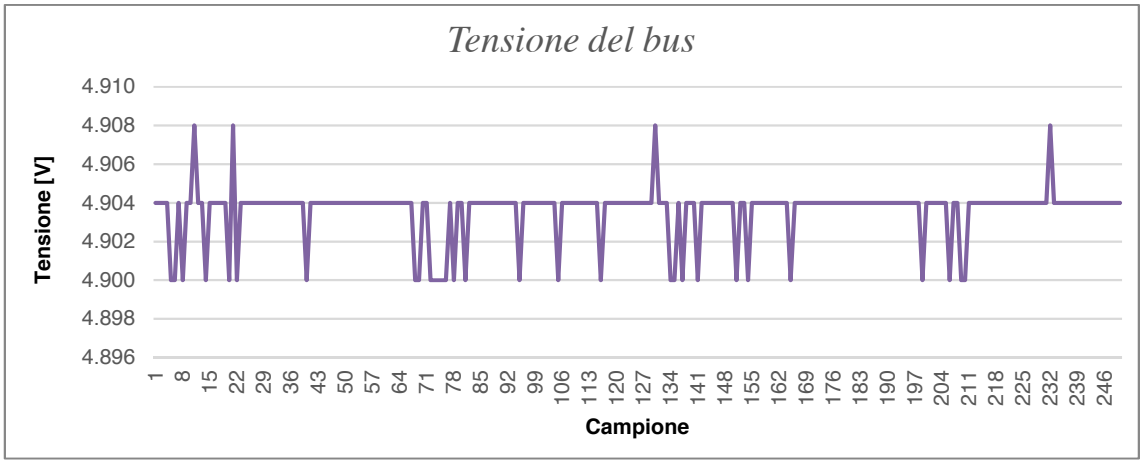

Figura 17 – Andamento della tensione sul bus per 250 campioni

In Tabella 1 invece, sono riportati i valori relativi a media e deviazione.

Tali calcoli sono stati effettuati considerando 1000 campioni.

Per il calcolo della media è stata utilizzata la media aritmetica, mentre per la deviazione si è usata la seguente formula:

$$\sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}}$$

essendo x_i il campione i-esimo e μ la media aritmetica calcolata sulla base dei 1000 campioni.

	Media	Deviazione
Tensione di Shunt (mV)	4.691020	0.043753
Corrente (mA)	46.910500	0.436783
Tensione del Bus (V)	4.903804	0.001191

Tabella 1 - Valutazioni energetiche baseline

5.2 Esperimento 1 – Periferiche disabilitate

Come configurazione successiva si considera quella in cui tutte le periferiche sono disabilitate.

Ovviamente, con *tutte* le periferiche ci si riferisce a quelle oggetto di valutazioni (Timer, GPIO, interfacce di comunicazione I2C, USART e SPI). Tutte le altre periferiche vengono considerate abilitate o disabilitate secondo i valori di default presenti e consultabili nel *datasheet*.

Di default, *tutte* le periferiche sono disabilitate ad eccezione delle GPIO. Per tale motivo, le uniche istruzioni che sono presenti nelle implementazioni di tale configurazione sono quelle relative alla disabilitazione delle porte GPIO.

Per far ciò, basta settare opportunamente i valori dei registri *RCC* (*Reset and Clock Controller)* relativi al bus *AHB1*. Dalla Figura 16, risulta chiaro che bisogna solamente settare a 0 i valori dei bit [0-7] del registro *RCC_AHB1NER*.

Tale operazione ha l'effetto di disabilitare il clock alle GPIO, le quali risulteranno disabilitate.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	OTGHS ULPIEN	OTGHS EN	Res.	Res.	Res.	Res.	Res.	Res.	DMA2 EN	DMA1 EN	Res.	Res.	BKP SRAMEN	Res.	Res.
	rw	rw							rw	rw			rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CRC EN	Res.	Res.	Res.	Res.	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw					rw	rw	rw	rw	rw	rw	rw	rw

Figura 16 - Registro RCC_AHB1ENR

Per quanto concerne la valutazione energetica relativa all'ambiente simbolico Mecrisp-Stellaris, i risultati ottenuti sono riportati in Figura 17.

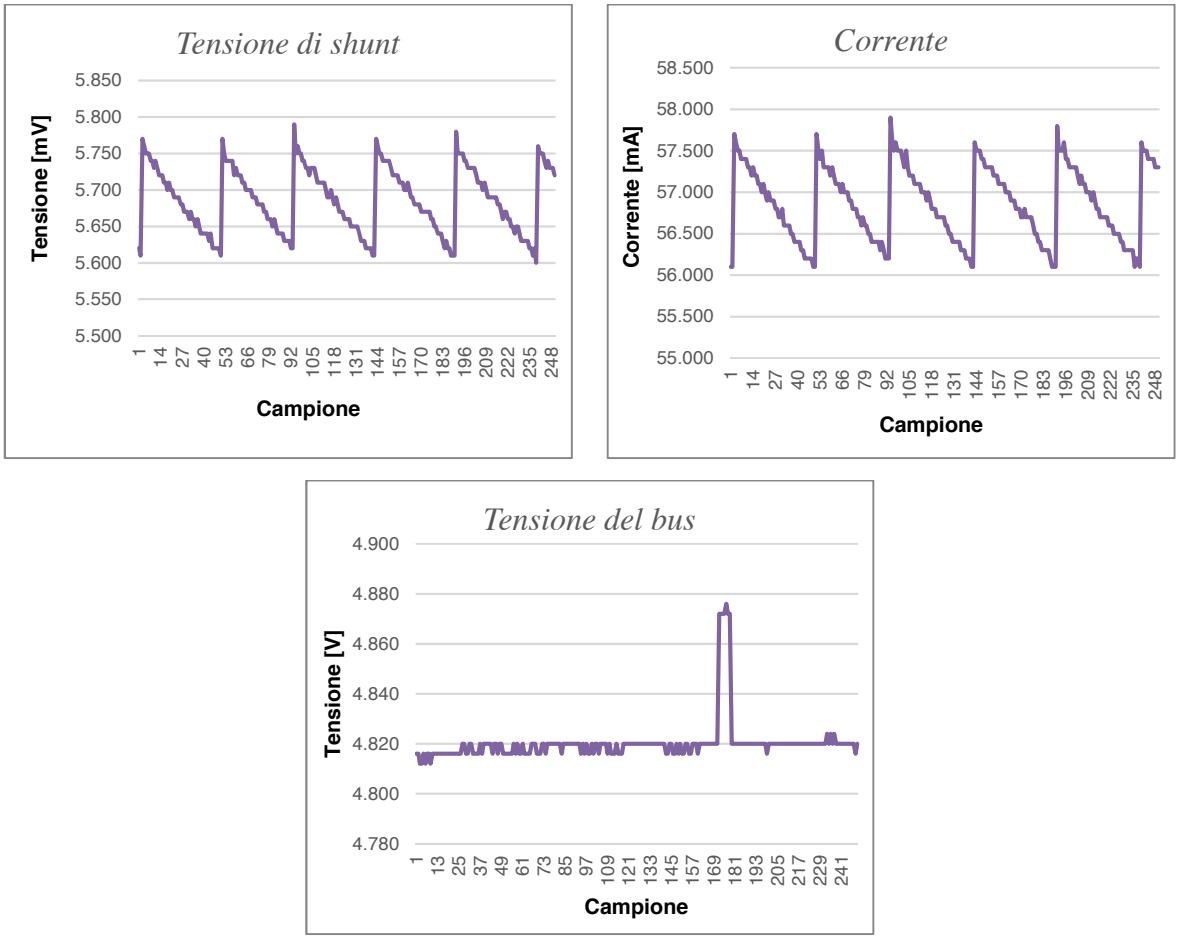

Figura 17 – Grafici valutazioni energetiche esperimento 1 – Mecrisp-stellaris

In Tabella 2 sono riportati i valori di media e deviazione.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.68423	0.043946
Corrente (mA)	56.842301	0.439592
Tensione del Bus (V)	4.821044	0.007817

Tabella 2 - Media e deviazione esperimento 1 - Mecrisp-stellaris

Per confronto, le valutazioni relative ai consumi energetici sono state effettuate anche mediante l'approccio tradizionale che usa il linguaggio C.

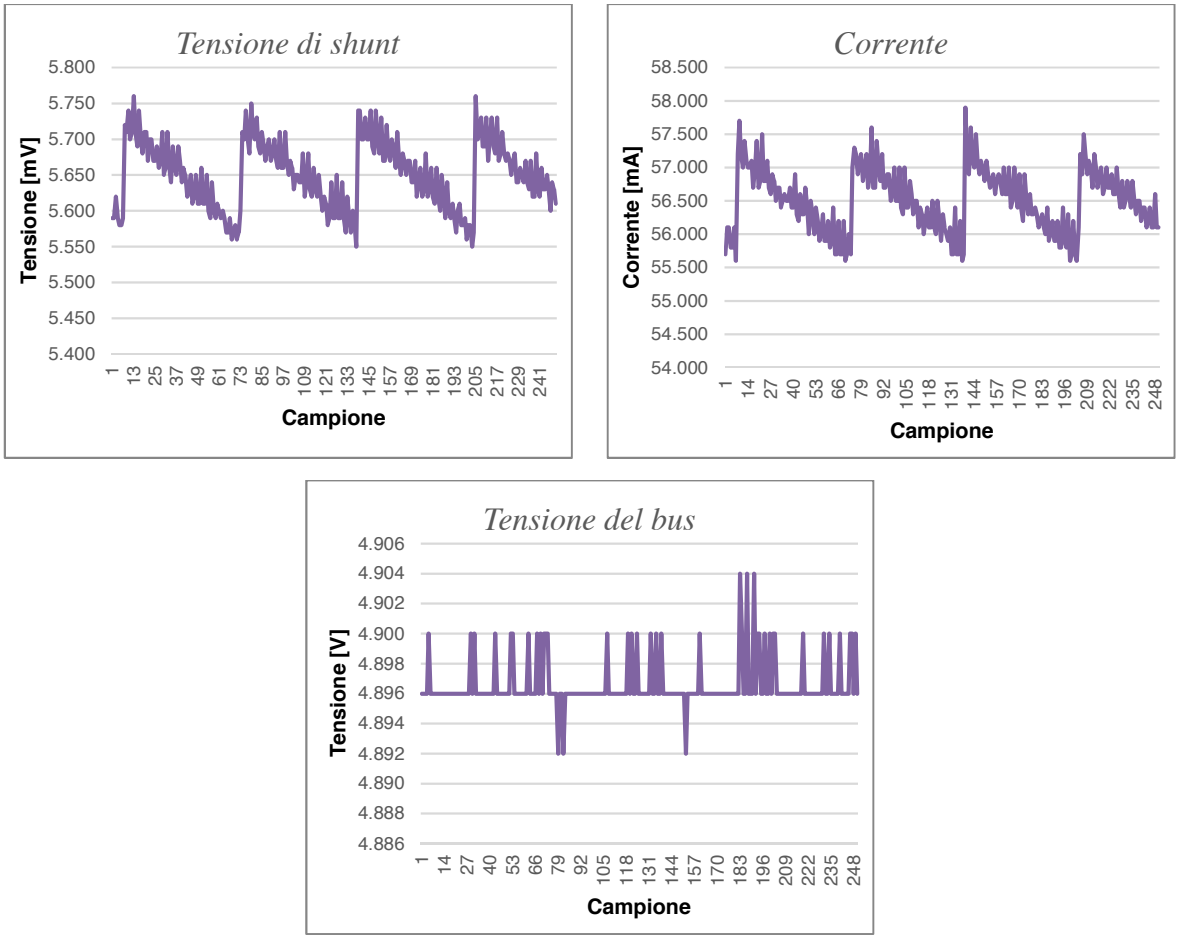

Figura 18 – Grafici valutazioni energetiche esperimento 1 – C

I valori di media e deviazione sono riportati in Tabella 3.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.64551	0.04748
Corrente (mA)	56.451398	0.476296
Tensione del Bus (V)	4.896656	0.001635

Tabella 3 - Media e deviazione esperimento 1 - C

Per confrontare i consumi relativi alle due implementazioni, la Figura 19 confronta i risultati ottenuti in termini della sola corrente, in quanto essa è in diretta relazione con i consumi in termini di potenza. Come si evince, i valori ottenuti in entrambi gli ambienti di sviluppo sono pressocchè identici. L'utilizzo dell'interprete e dunque del modello simbolico non influisce sui consumi i quali si mantengono quasi identici a quelli relativi all'ambiente di programmazione C. Dunque, anche se a differenza dell'implementazione C, in cui il software sostanzialmente consiste di un *while* che viene ripetuto indefinitamente, nell'implementazione dell'ambiente simbolico si ha l'esecuzione costante di codice (basti pensare all'interprete stesso), i consumi si mantengono comunque coerenti ed in linea con quelli relativi ad approcci classici o comunque, le differenze sono limitate.

L'andamento a *lisca di pesce* che caratterizza gli esperimenti è determinato dalle caratteristiche dello strumento.

A causa della periodicità dei campioni è possibile omettere e non mostrare gran parte dei campioni nei grafici.

Infatti vengono mostrati solo 250 dei 1000 campioni, sebbene le quantità di media e deviazione sono calcolate sulla base di tutti e 1000 i campioni

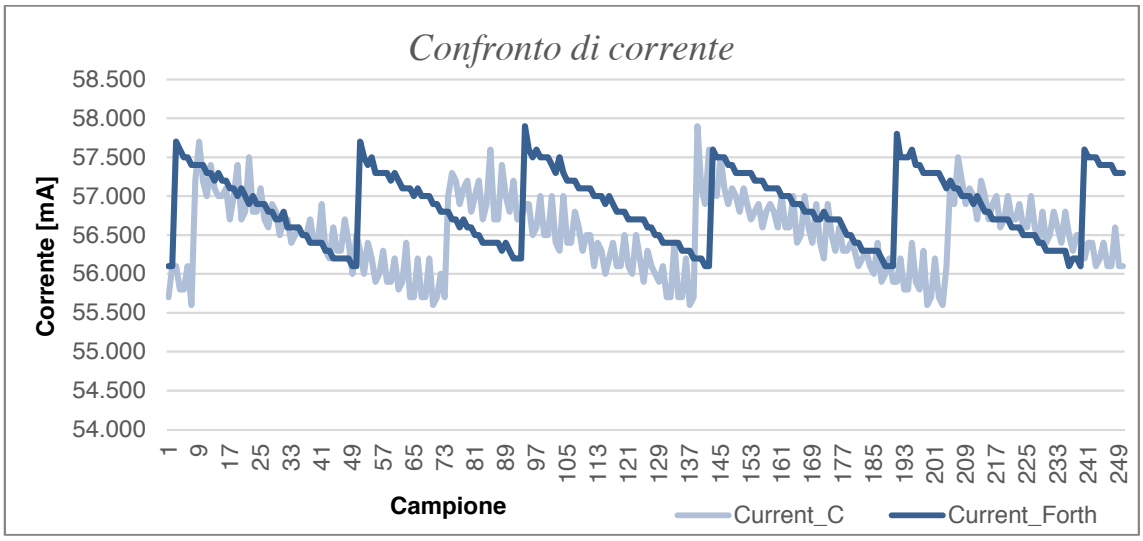

Figura 19 - Confronto valutazioni energetiche esperimento 1

Nella Tabella 4 sono confrontati i valori di media e deviazione relativi al primo esperimento nelle due implementazioni.

Fra le due implementazioni non sono presenti sostanziali differenze in termini di consumo energetico e dunque, l'interprete Forth non incide, almeno in questo caso, negativamente sui consumi.

	Mecrisp-stellaris	**C**
Tensione di Shunt (mV)	5.68423	5.64551
Corrente (mA)	56.842301	56.451398
Tensione del Bus (V)	4.821044	4.896656

Tabella 4 - Confronto media e deviazione esperimento 1

5.3 Esperimento 2 – Timer

La configurazione hardware relativa a questo esperimento prevede l'abilitazione del Timer 1 e del Timer 2.
Per ogni timer viene anche configurato il corrispondente IRQ (impostando opportunamente le ISR nella *vector table*) che elimina la richiesta di interrupt settando opportunamente il bit UIF (bit 0) all'interno del registro *TIMx_SR*.
I registri coinvolti sono quelli relativi alle periferiche dei timer.
In particolar modo, per abilitare i timer e configurarli opportunamente è necessario:

- Abilitare il clock usando i registri RCC (relativi ai timer che si intende abilitare)
- Abilitare gli interrupt nel controller *NVIC* (Nested Vector Interrupt Controller) relativi ai timer, per far in modo che il processore gestisca gli interrupt provenienti dai timer.
- Configurare opportunamente i registri relativi alla periferica del timer in base alla modalità desiderata.

Per quest'esperimento, i due timer abilitati sono stati settati con un *autoreload_value* pari a 3000 ed un *prescaler_value* pari a 16000 (per far in modo di *contare* millisecondi, essendo impostato il clock del microcontrollore a 16 MHz.
Il codice sorgente è allegato in Appendice.

Per quanto riguarda la valutazione energetica del modello simbolico di Mecrisp-Stellaris la Figura 20 riporta i valori ottenuti per la tensione di Shunt, la corrente e la tensione al bus. I valori riportati sono relativi a 250 campioni per via dell'andamento periodico dei segnali campionati.

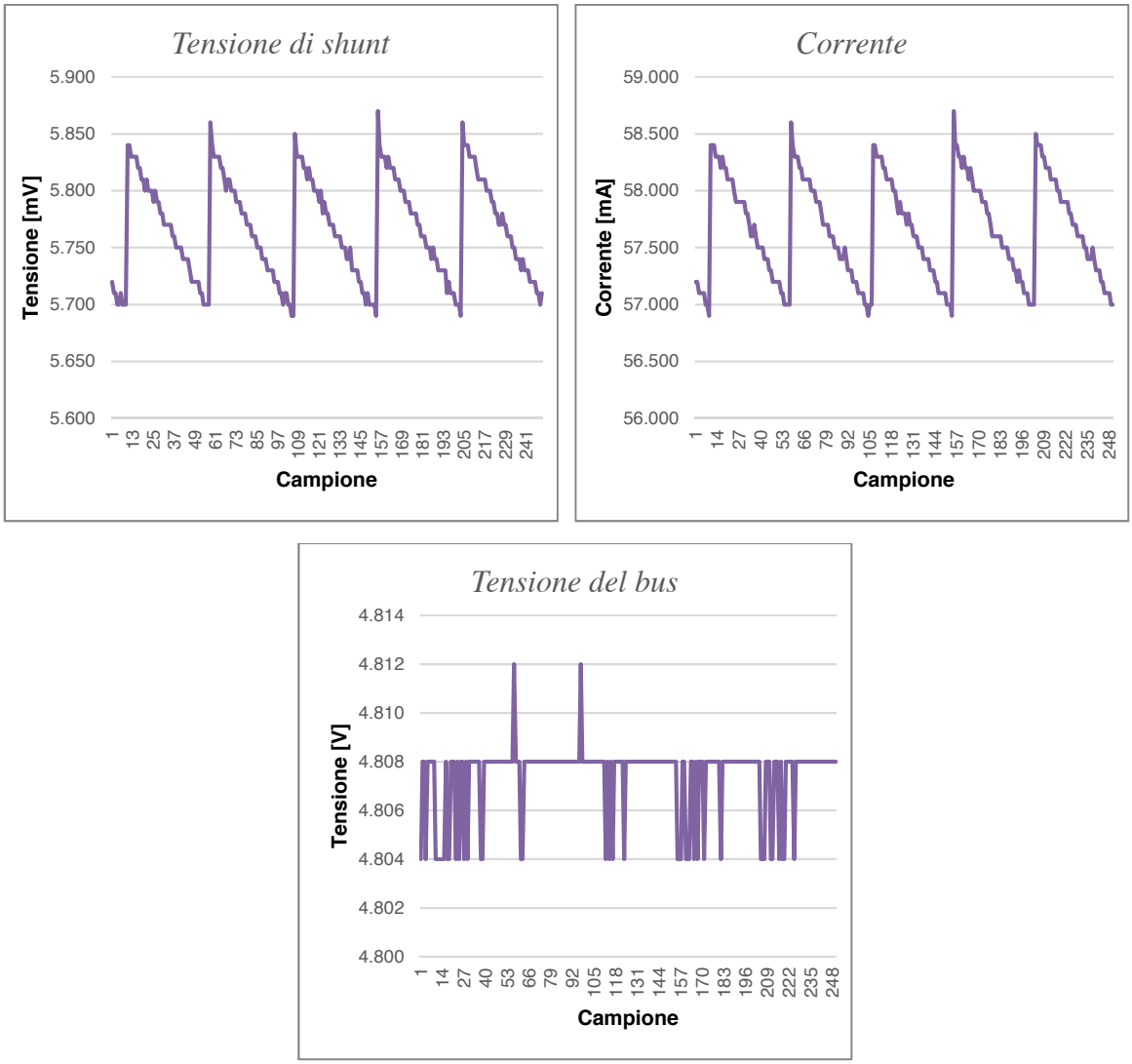

Figura 20 - Grafici valutazioni energetiche esperimento 2 - Mecrisp-Stellaris

I valori di media e deviazione sono mostrati nella Tabella 5.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.76583	0.04453
Corrente (mA)	57.659	0.442989
Tensione del Bus (V)	4.807744	0.001743

Tabella 5 - Media e deviazione esperimento 2 - Mecrisp-Stellaris

I dati relativi ai consumi energetici del sistema che esegue i banchmark in C sono mostrati in Figura 21.

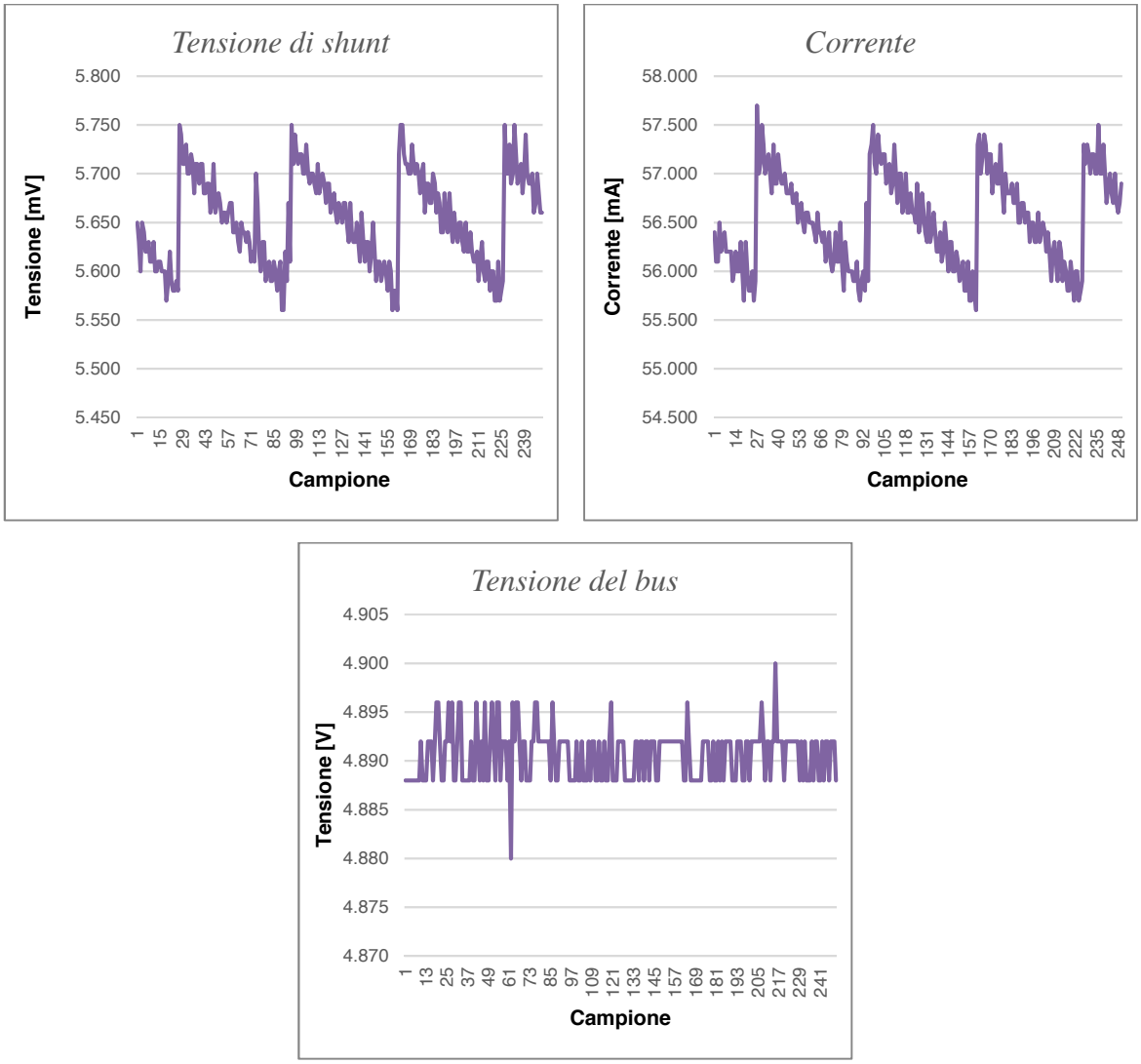

Figura 21 - Grafici valutazioni energetiche esperimento 2 - C

Sulla base dei 1000 campioni relativi a tale esperimento, vengono riportati i valori di media e deviazione nella Tabella 6.

	Media	Deviazione
Tensione di Shunt (mV)	5.65297	0.04682
Corrente (mA)	56.53	0.469596
Tensione del Bus (V)	4.89112	0.002484

Tabella 6 - Media e deviazione esperimento 2 - C

La Figura 22 sovrappone i due consumi per i valori di corrente.

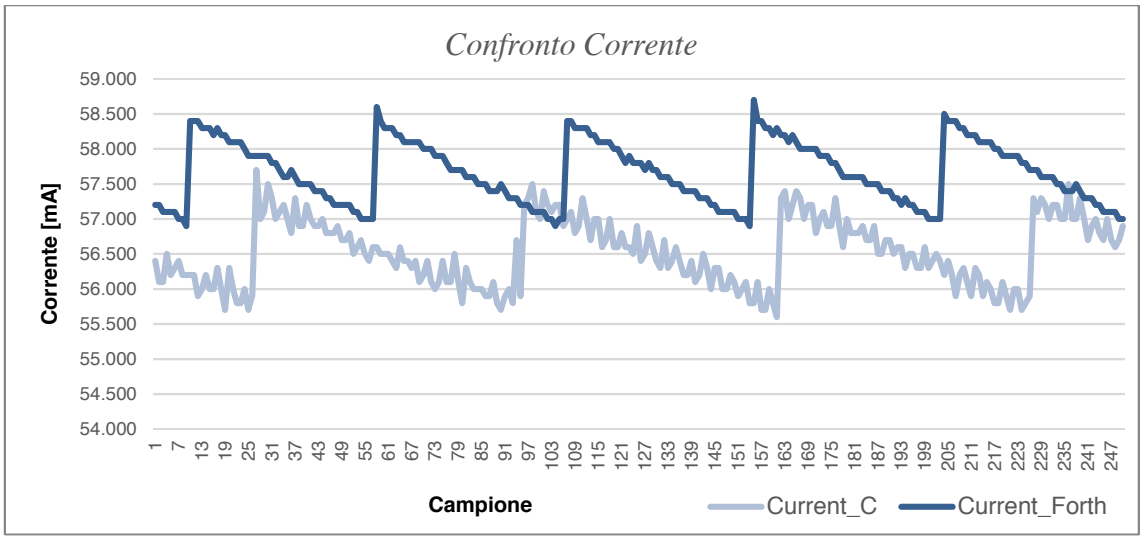

Figura 22 - Confronto valutazioni energetiche esperimento 2

Dalla Figura 22 emerge che i consumi relativi all'ambiente simbolico sono leggermente superiori rispetto a quelli relativi all'implementazione C.

L'incremento è di circa il 1,75% rispetto all'implementazione C, percentuale comunque bassa che può essere imputata al fatto che, mentre in C il software consiste di un *while* che non esegue codice, nell'ambiente simbolico invece si ha sempre l'esecuzione di codice (per eseguire l'interprete) e quindi richiede circuiteria hardware maggiore (banalmente la seriale per la comunicazione), incrementando lievemente i consumi.

Nella Tabella 7 sono messi a confronto i valori di media e deviazione, relative all'esperimento 2, delle due implementazioni.

	Mecrisp-stellaris	C
Tensione di Shunt (mV)	5.76583	5.65297
Corrente (mA)	57.659	56.53
Tensione del Bus (V)	4.807744	4.89112

Tabella 7 - Confronto media e deviazione esperimento 2

5.4 Esperimento 3 – Timer

Il presente esperimento prevede l'abilitazione dei seguenti timer:
Timer_2, Timer_3, Timer_4, Timer_5, Timer_6
Rispetto all'esperimento precedente, vengono abilitati altri 3 timer.

In Figura 23 sono mostrate le valutazioni energetiche relative all'ambiente Mecrisp-Stellaris.

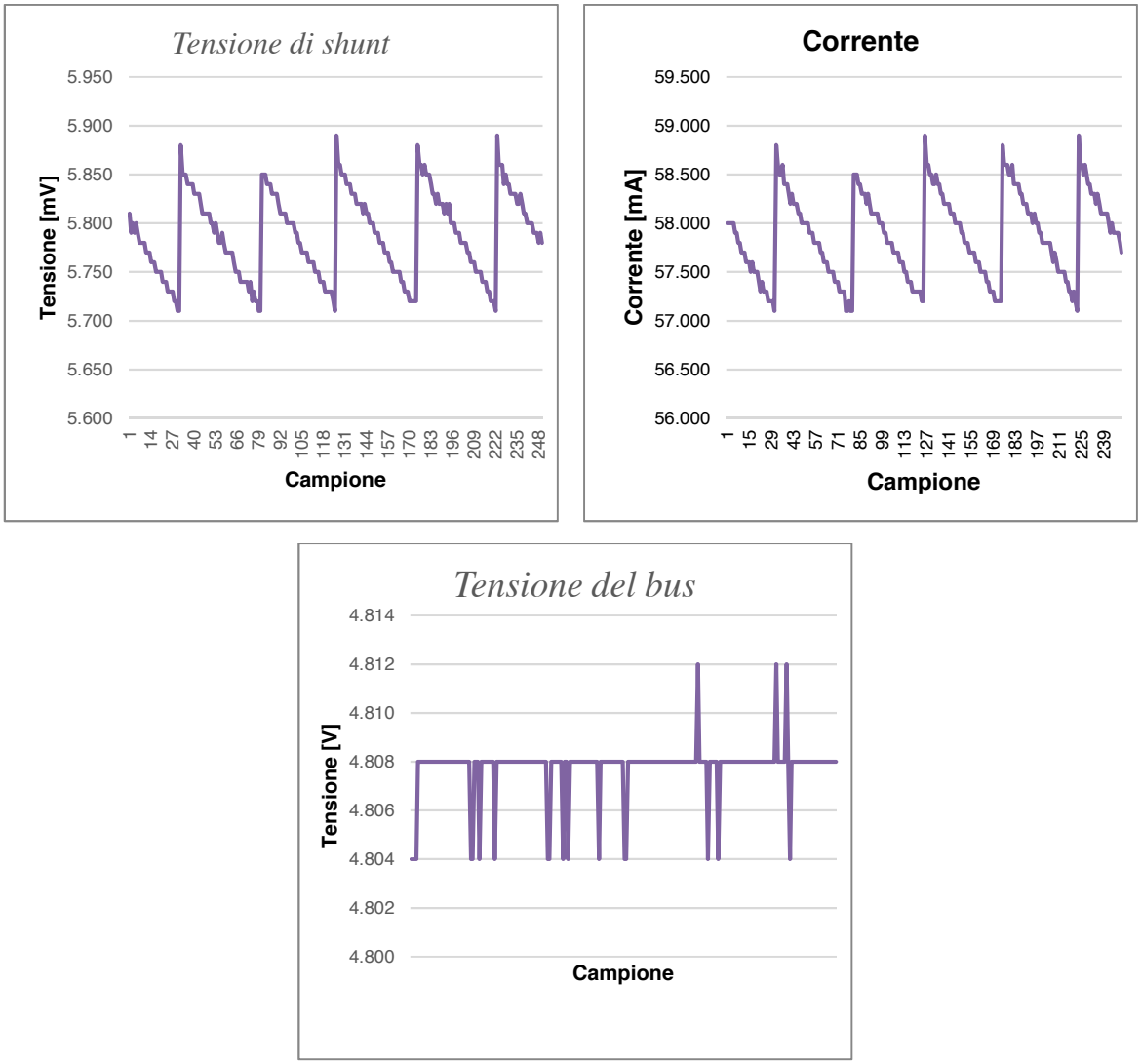

Figura 23 - Grafici valutazioni energetiche esperimento 3 - Mecrisp-Stellaris

I valori di media e deviazione sono riportati in Tabella 8.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.78599	0.044168
Corrente (mA)	57.857898	0.442219
Tensione del Bus (V)	4.807924	0.001243

Tabella 8 - Media e deviazione esperimento 3 - Mecrisp-Stellaris

Le valutazioni energetiche dell'implemntazione in C sono mostrate in Figura 24

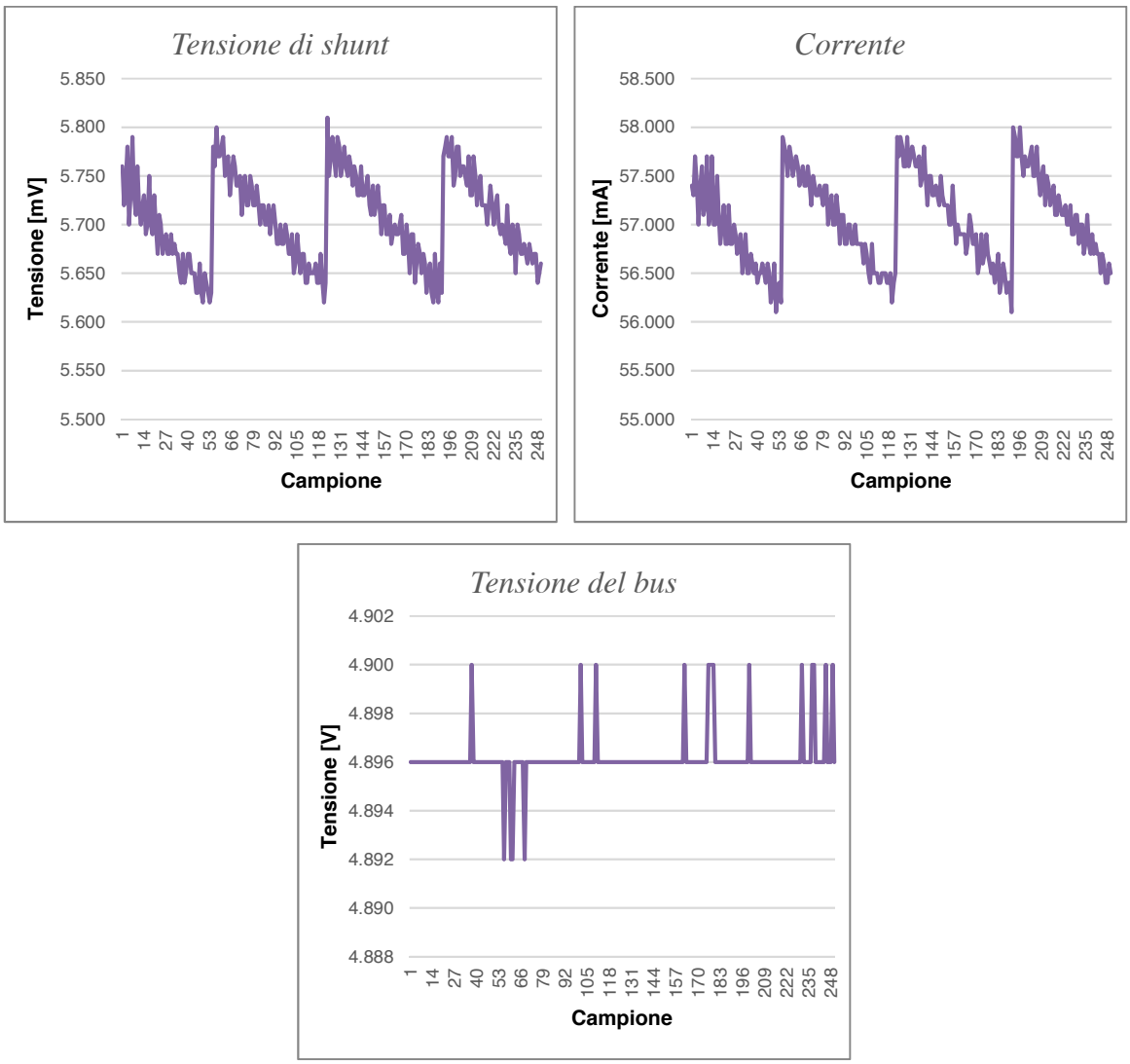

Figura 24 - Grafici valutazioni energetiche esperimento 3 - Mecrisp-Stellaris

I valori di media e deviazione sono riportati in Tabella 9

	Media	**Deviazione**
Tensione di Shunt (mV)	5.71257	0.050824
Corrente (mA)	57.119102	0.496614
Tensione del Bus (V)	4.896432	0.001969

Tabella 9 - Media e deviazione esperimento 3 - Mecrisp-Stellaris

I grafici relativi ai consumi di corrente vengono sovrapposti in un unico grafico per effettuare il confronto.

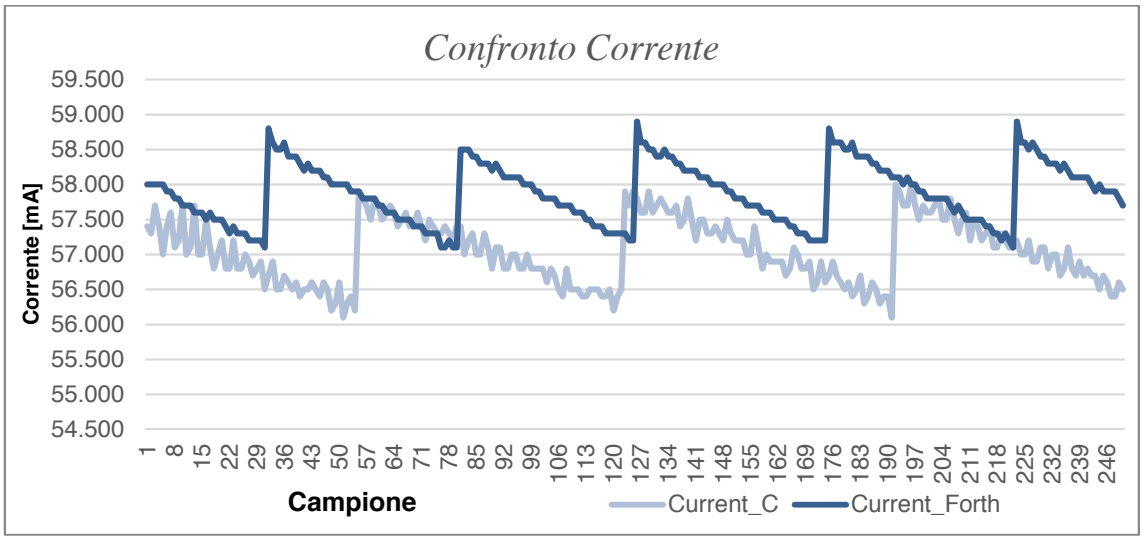

Figura 25 - Confronto valutazioni energetiche esperimento 3

Anche in questo caso, la presenza dell'interprete Forth e dunque di codice che viene continuamente eseguito, comporta un incremento dei valori di corrente di circa il 1,25%.

I valori di media a confronto sono riportati in Tabella 10

	Mecrisp-stellaris	**C**
Tensione di Shunt (mV)	5.78599	5.71257
Corrente (mA)	57.857898	57.119102
Tensione del Bus (V)	4.807924	4.896432

Tabella 10 - Confronto media e deviazione esperimento 3

5.5 Esperimento 4 – Timer, GPIO

In questo esperimento, vengono abilitate le seguenti periferiche:

- *Timer_1, Timer_2, Timer_3, Timer_4, Timer_5,*
- *GPIO_A, GPIO_B, GPIO_C, GPIO_D, GPIO_E, GPIO_F, GPIO_H*

Pertanto, rispetto all'esperimento precedente, vengono abilitate tutte le porte GPIO. Le GPIO vengono abilitate con configurazione di default come riportato in [13].
Le valutazioni energetiche sono mostrate in Figura 26

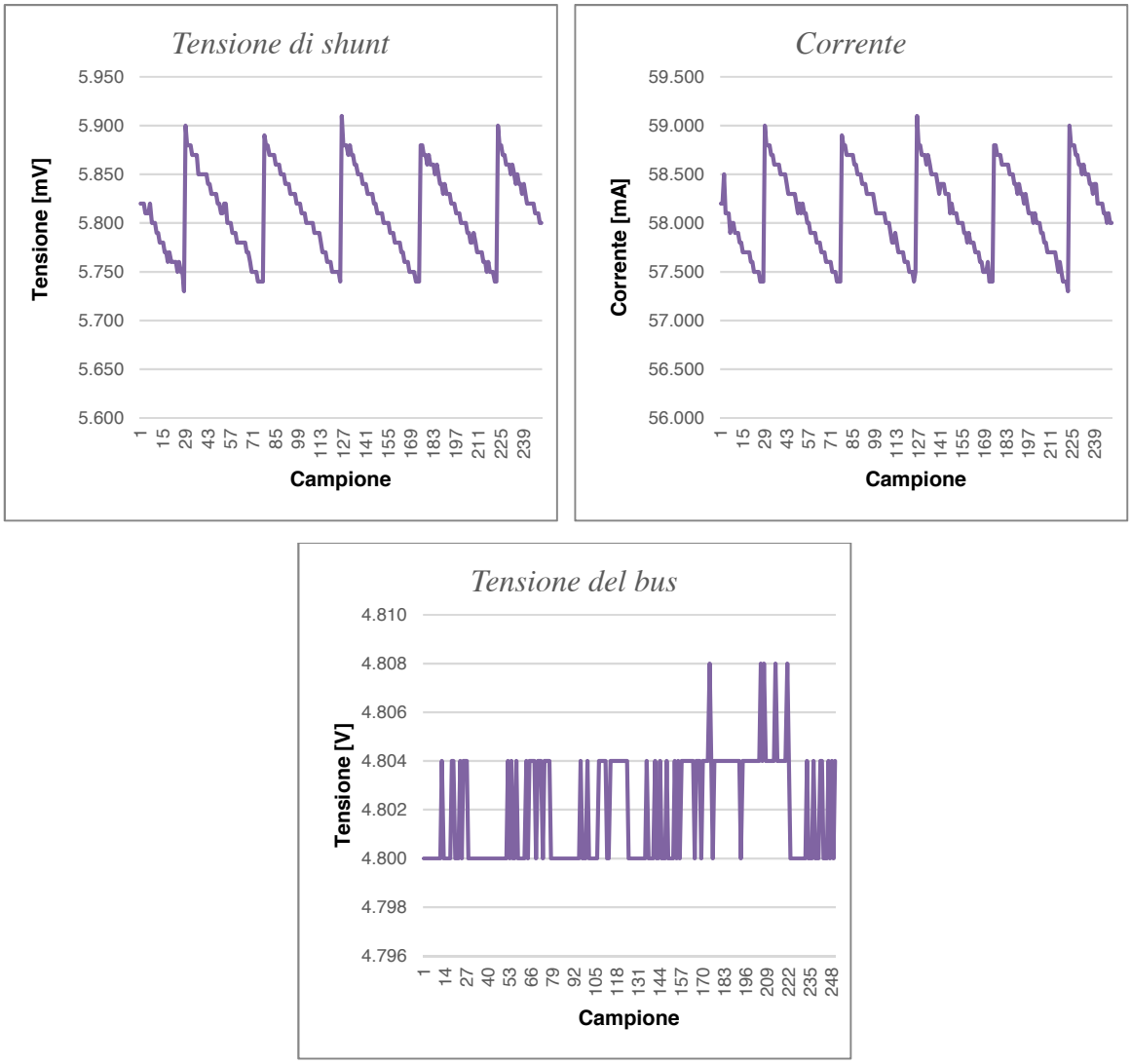

Figura 26 - Grafici valutazioni energetiche esperimento 4 - Mecrisp-Stellaris

I valori di media e varianza sono mostrati nella Tabella 11

	Media	Deviazione
Tensione di Shunt (mV)	5.80937	0.042927
Corrente (mA)	58.092898	0.430859
Tensione del Bus (V)	4.80246	0.002066

Tabella 11 - Media e deviazione esperimento 4 - Mecrisp-Stellaris

Le valutazioni energetiche relative all'implementazione C dell'esperimento 4 sono mostrate in

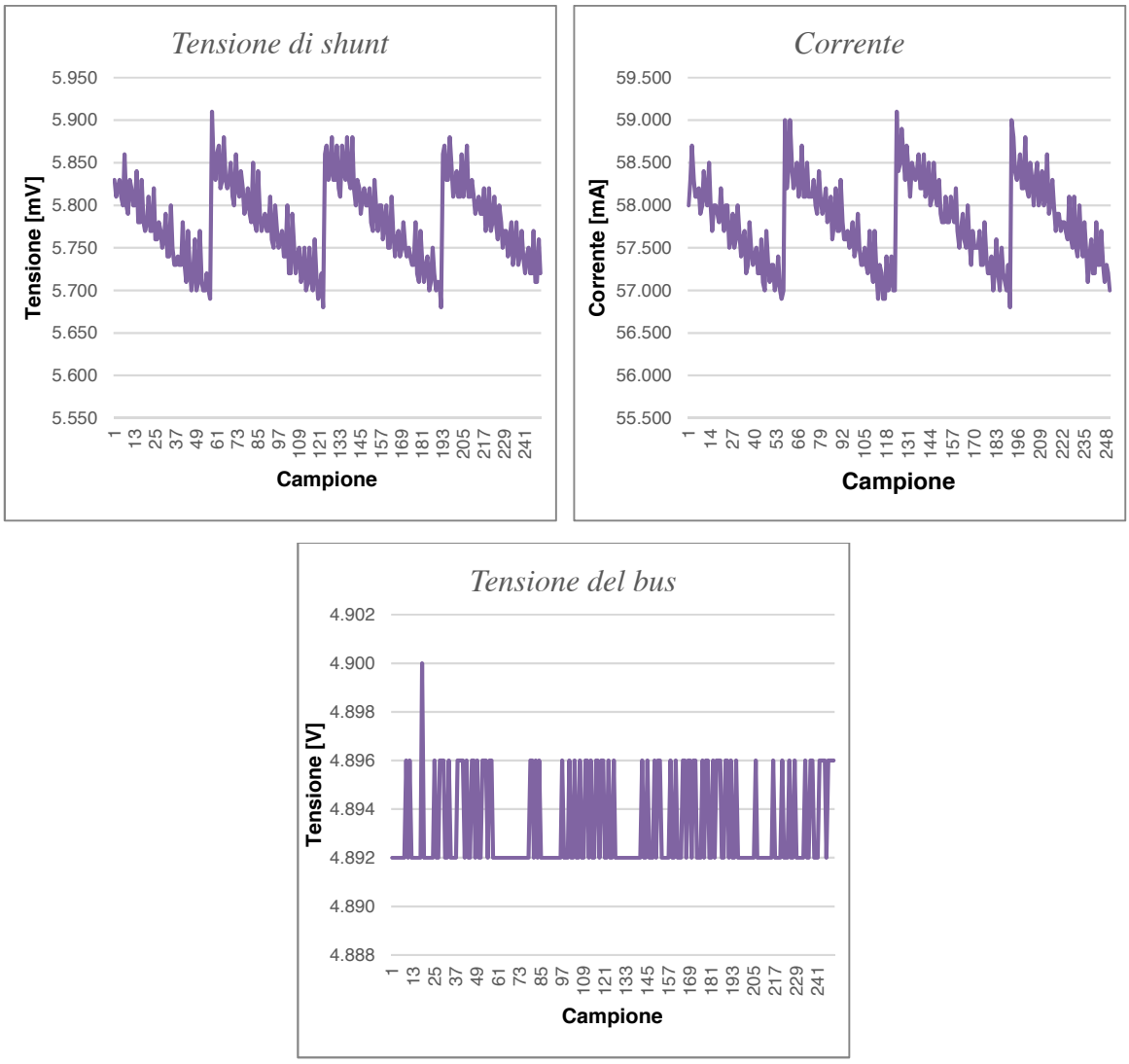

Figura 27 - Grafici valutazioni energetiche esperimento 4 - C

Media e deviazione sono indicati nella Tabella 12

	Media	Deviazione
Tensione di Shunt (mV)	5.76093	0.049846
Corrente (mA)	57.613102	0.499118
Tensione del Bus (V)	4.893844	0.00225

Tabella 12 - Media e deviazione esperimento 4 - C

In Figura 28 sono sovrapposti i consumi di corrente dell'esperimento 4 nelle due implementazioni.

In questo caso i valori sono molto vicini e quasi uguali.

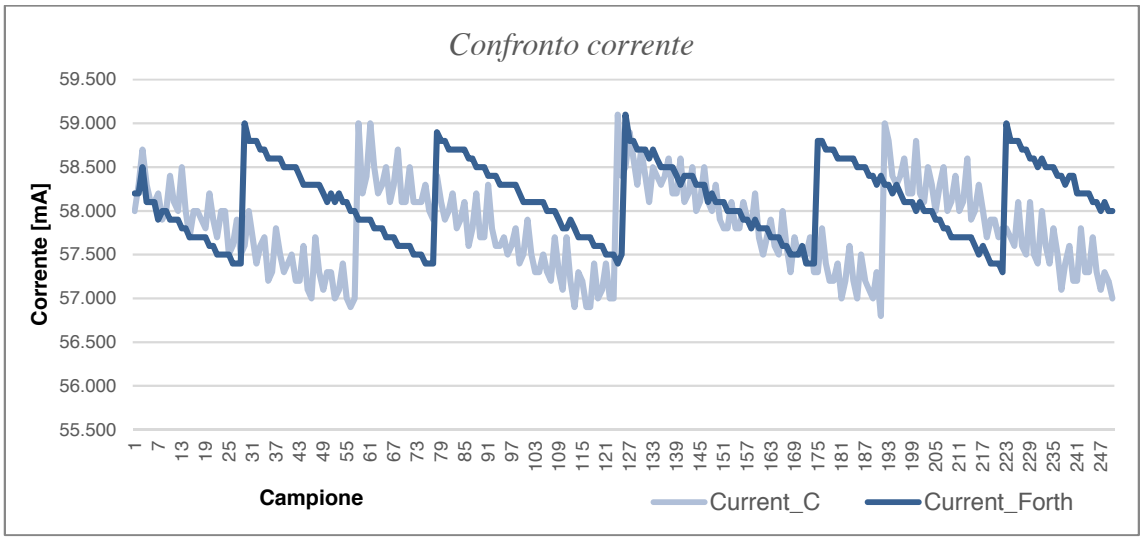

Figura 28 - Confronto valutazioni energetiche esperimento 4

I confronti dei valori medi sono riportati nella Tabella 13

	Mecrisp-stellaris	C
Tensione di Shunt (mV)	5.80937	5.76093
Corrente (mA)	58.092898	57.613102
Tensione del Bus (V)	4.80246	4.893844

Tabella 13 - Confronto media e deviazione esperimento 4

5.6 Esperimento 5 – Timer, GPIO, I2C

In questo esperimento, la valutazione energetica viene effettuata abilitando le seguenti periferiche:

- *Timer_1, Timer_2, Timer_3, Timer_4, Timer_5,*
- *GPIO_A, GPIO_B, GPIO_C, GPIO_D, GPIO_E, GPIO_F, GPIO_H*
- *I2C_1, I2C_2, I2C_3*

Le valutazioni energetiche relative all'ambiente Mecrisp-Stellaris sono mostrate in Figura 29.

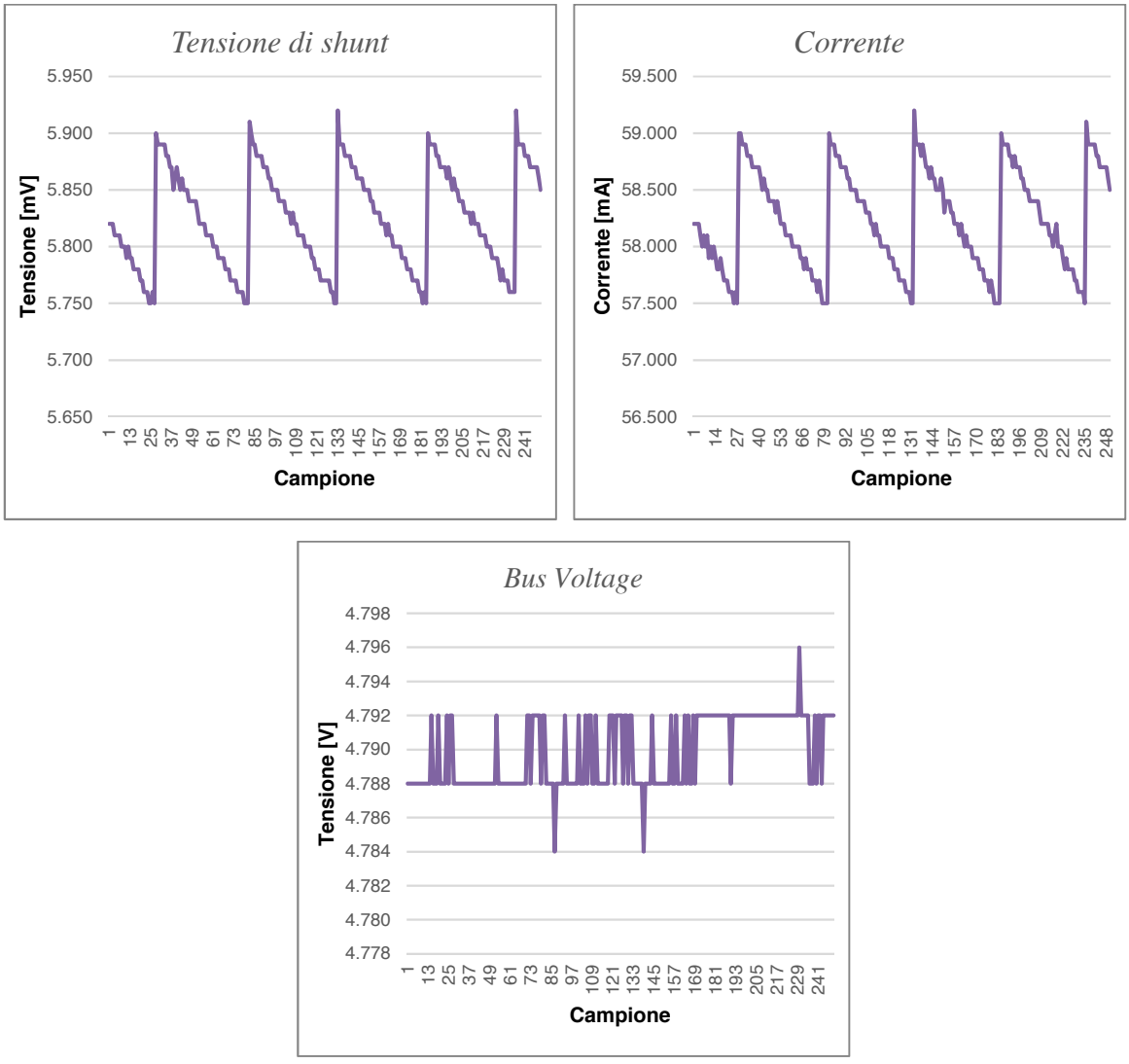

Figura 29 - Grafici valutazioni energetiche esperimento 5 - Mecrisp-Stellaris

Media e varianza sono mostrati nella Tabella 14.

	Media	Deviazione
Tensione di Shunt (mV)	5.82214	0.044245
Corrente (mA)	58.22	0.44308
Tensione del Bus (V)	4.78898	0.002533

Tabella 14 - Media e deviazione esperimento 5 - Mecrisp-Stellaris

In Figura 30 le valutazioni energetiche relative all'implementazione in C.

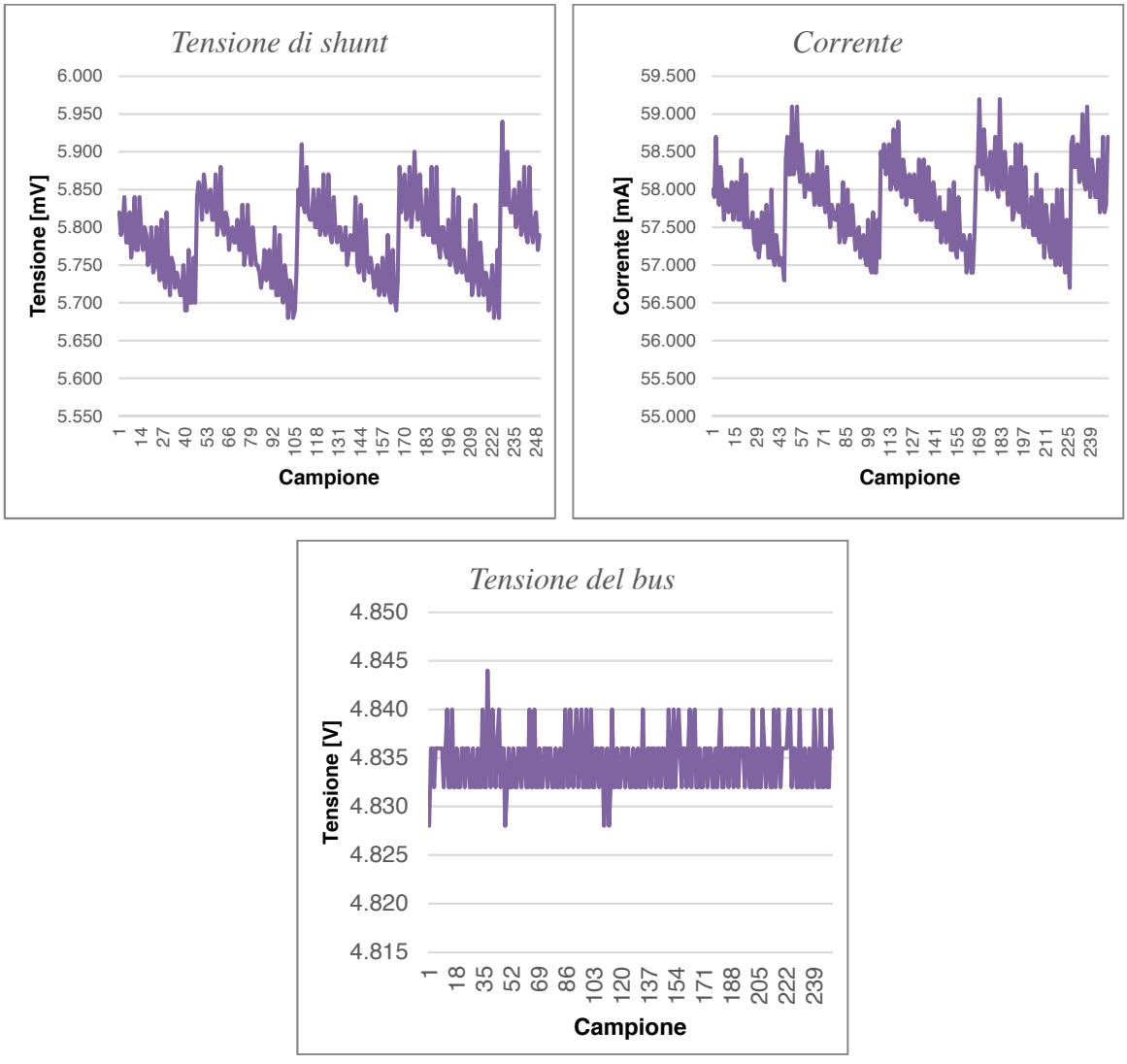

Figura 30 - Grafici valutazioni energetiche esperimento 5 - C

Nella Tabella 15 sono mostrati i risultati di media e deviazione

	Media	Deviazione
Tensione di Shunt (mV)	5.75942	0.054597
Corrente (mA)	57.590801	0.534879
Tensione del Bus (V)	4.89536	0.00192

Tabella 15 - Media e deviazione esperimento 5 - Mecrisp-Stellaris

I grafici dei campioni della corrente sono mostrati in Figura 31. Si evince che, anche in questo caso, i valori di corrente dell'implementazione Forth, se pur di poco, sovrastano sempre quelli dell'implementazione C, a causa della presenza dell'interprete che comunque comporta continuamente l'esecuzione di codice.

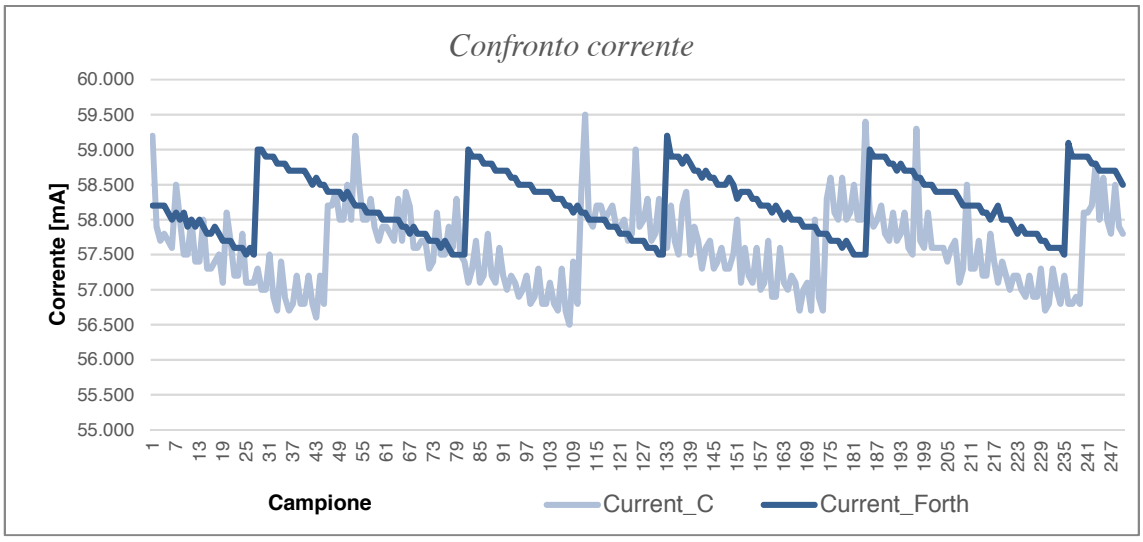

Figura 31 - Confronto valutazioni energetiche esperimento 5

Mentre Tabella 16 sono riportati i confronti dei valori medi.

	Mecrisp-stellaris	C
Tensione di Shunt (mV)	5.82214	5.75942
Corrente (mA)	58.22	57.590801
Tensione del Bus (V)	4.78898	4.89536

Tabella 16 - Confronto media e valutazione esperimento 5

5.7 Esperimento 6 – Timer, GPIO, I2C, USART

La configurazione hardware relativa al ptesente esperimento prevede l'abilitazione delle seguenti periferiche hardware:

- *Timer_1, Timer_2, Timer_3, Timer_4, Timer_5,*
- *GPIO_A, GPIO_B, GPIO_C, GPIO_D, GPIO_E, GPIO_F, GPIO_H*
- *I2C_1, I2C_2, I2C_3*
- *USART_1, USART_2, USART_3, USART_4, USART_5, USART_6*

Le valutazioni energetiche relative a tale configurazione nell'implementazione Mecrisp-stellaris sono mostrate in Figura 32.

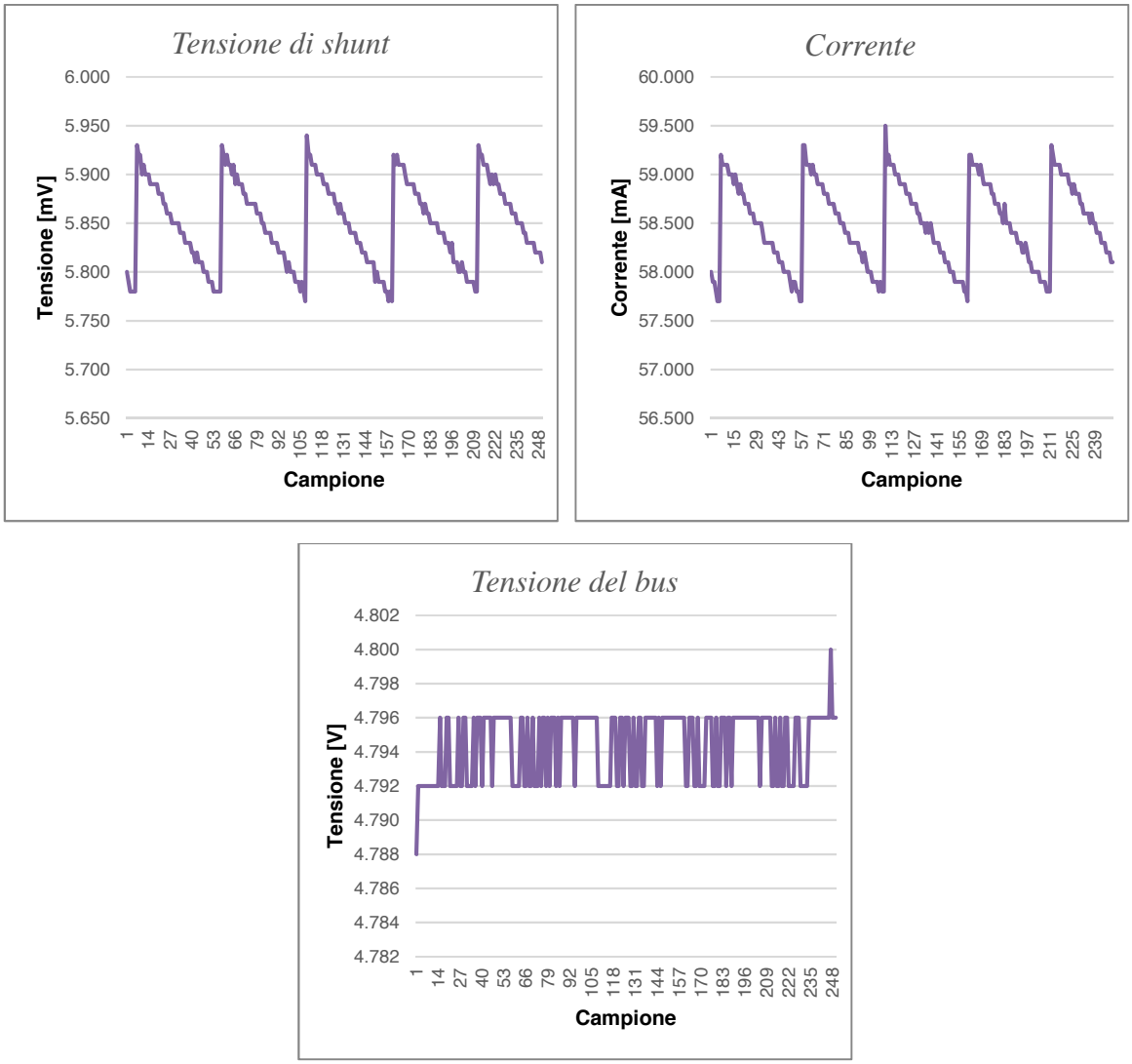

Figura 32 - Grafici valutazioni energetiche esperimento 6 - Mecrisp-Stellaris

Nella Tabella 17 sono mostrati i valori di media e deviazione relativi all'esperimento.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.84887	0.044105
Corrente (mA)	58.486199	0.438326
Tensione del Bus (V)	4.794216	0.00262

Tabella 17 - Media e deviazione esperimento 6 - Mecrisp-Stellaris

Le valutazioni energetiche nell'implementazione C sono mostrate in Figura 33

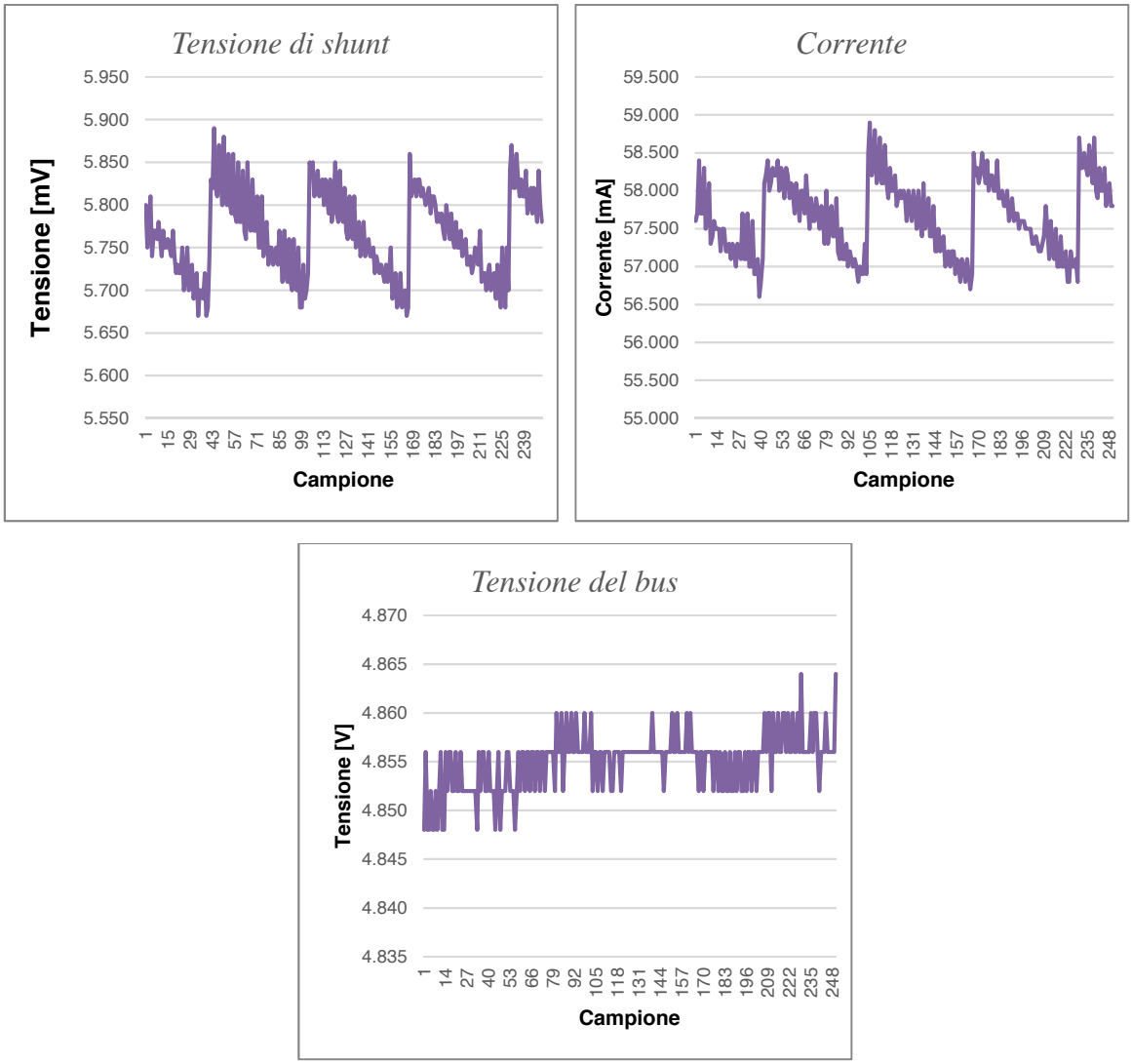

Figura 33 - Grafici valutazioni energetiche esperimento 6 - C

Valori di media e deviazione sono mostrati nella Tabella 18

	Media	**Deviazione**
Tensione di Shunt (mV)	5.76727	0.048837
Corrente (mA)	57.655	0.495454
Tensione del Bus (V)	4.8539	0.017437

Tabella 18 - Media e deviazione esperimento 6 - C

Di seguito, nella Tabella 19 e in Figura 34 sono mostrati i confronti relativi ai consumi energetici dell'esperimento 6 nelle due implementazioni.

L'interprete Forth, anche in questo caso, introduce un incremento relativo ai valori di corrente, incidendo dunque, se pur in minima parte (intorno all'1.50%), sui consumi energetici.

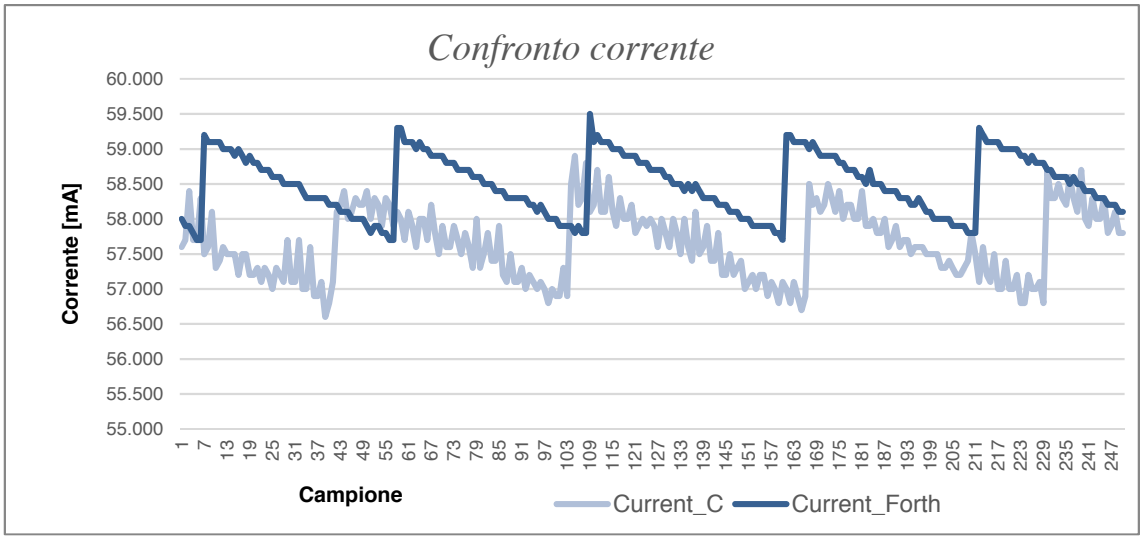

Figura 34 - Confronto corrente esperimento 6

Nella Tabella 19, invece, i valori medi delle misurazioni nelle due implementazioni.

	Mecrisp-stellaris	**C**
Tensione di Shunt (mV)	5.84887	5.76727
Corrente (mA)	58.486199	57.655
Tensione del Bus (V)	4.794216	4.8539

Tabella 19 – Confronto media e deviazione esperimento 6

5.8 Esperimento 7 – Timer, GPIO, I2C, USART, SPI

La configurazione hardware è la seguente:

- *Timer_1, Timer_2, Timer_3, Timer_4, Timer_5,*
- *GPIO_A, GPIO_B, GPIO_C, GPIO_D, GPIO_E, GPIO_F, GPIO_H*
- *I2C_1, I2C_2, I2C_3*
- *USART_1, USART_2, USART_3, USART_4, USART_5, USART_6*
- *SPI_1, SPI_2, SPI_3*

Le valutazioni energetiche sono mostrate in Figura 35

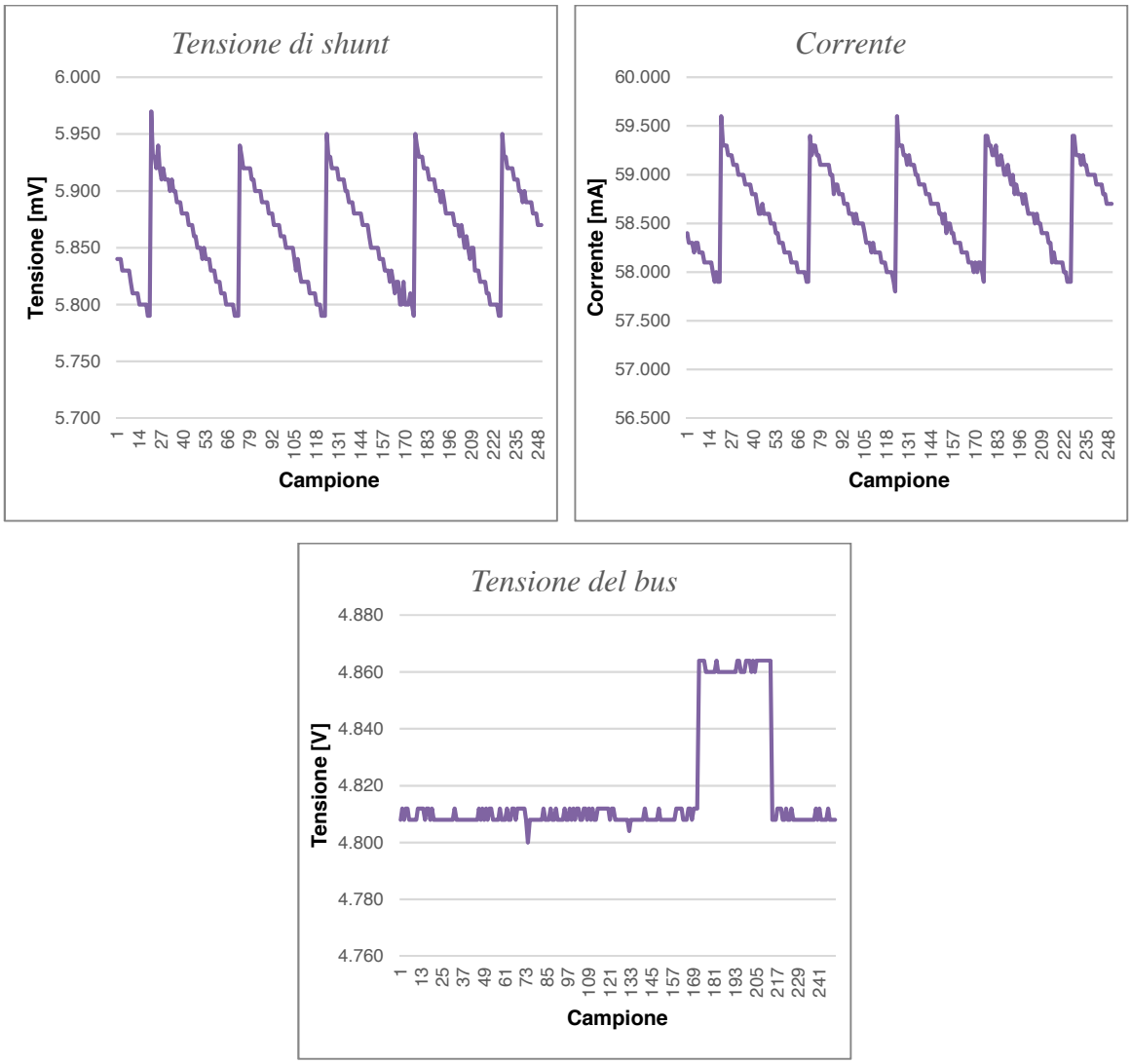

Figura 35 - Valutazioni energetiche esperimento 7 - Mecrisp-Stellaris

I valori di media e deviazione sono mostrati nella Tabella 20

	Media	Deviazione
Tensione di Shunt (mV)	5.85969	0.043623
Corrente (mA)	58.595699	0.438191
Tensione del Bus (V)	4.812072	0.012317

Tabella 20 - Media e deviazione esperimento 7 - Mecrisp-Stellaris

Le valutazioni energetiche dell'implementazione C dell'esperimento 7 sono mostrate in Figura 36

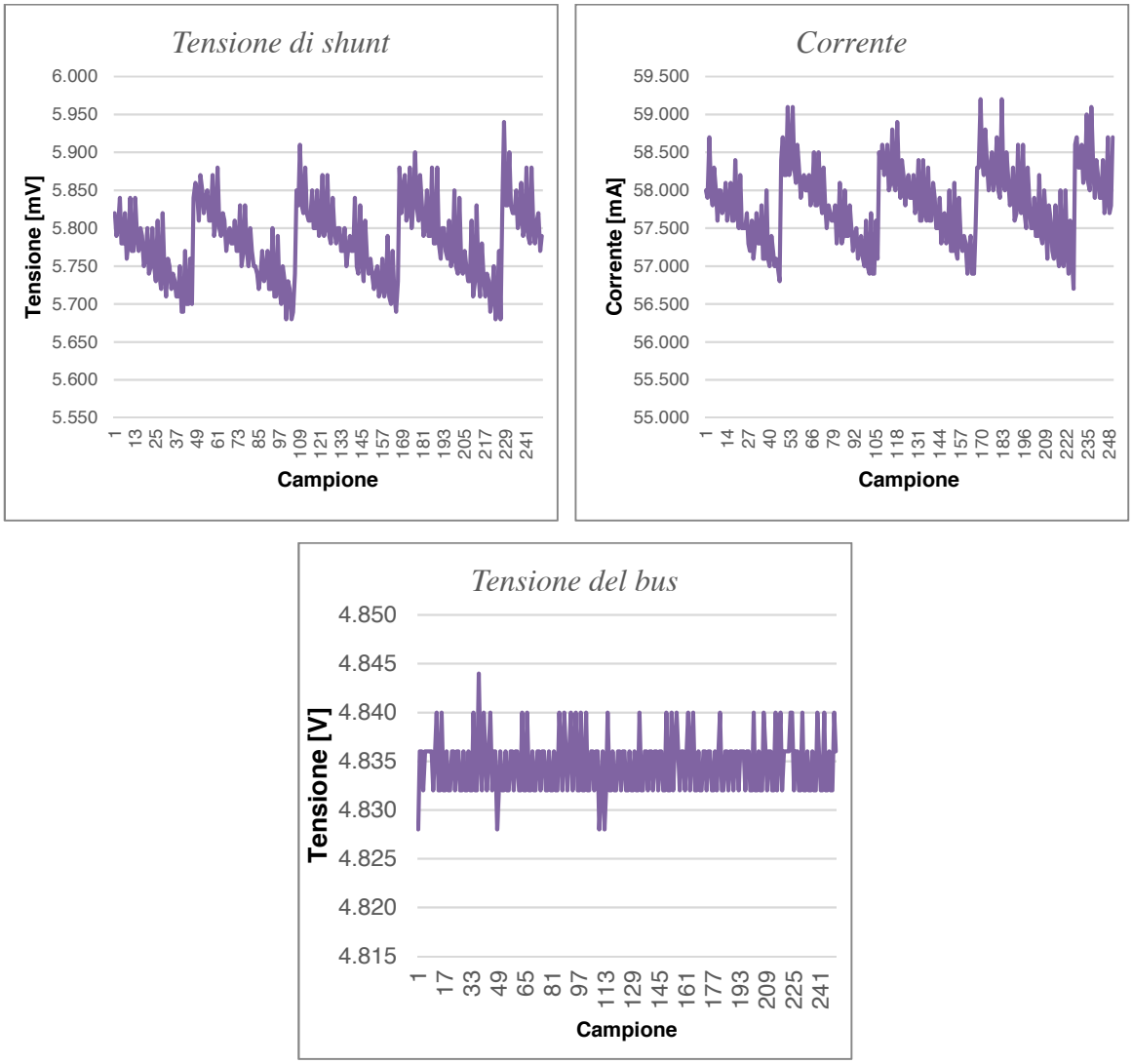

Figura 36 - Grafici valutazioni energetiche esperimento 7 - C

I valori di media e deviazione sono mostrati nella Tabella 21

	Media	Deviazione
Tensione di Shunt (mV)	5.77876	0.052586
Corrente (mA)	57.788	0.528428
Tensione del Bus (V)	4.83484	0.003459

Tabella 21 - Media e deviazione esperimento 7 - C

In Figura 37 è mostrato il grafico che mette a confronto i valori di corrente relativi alle due implementazioni, continuando a mostrare consumi lievemente superiori nell'implementazione dell'ambiente simbolico (di circa il i,35%).
L'interprete, anche in questo caso, fa sì che i valori di corrente ottenuti nell'implementazione Forth siano superiori.

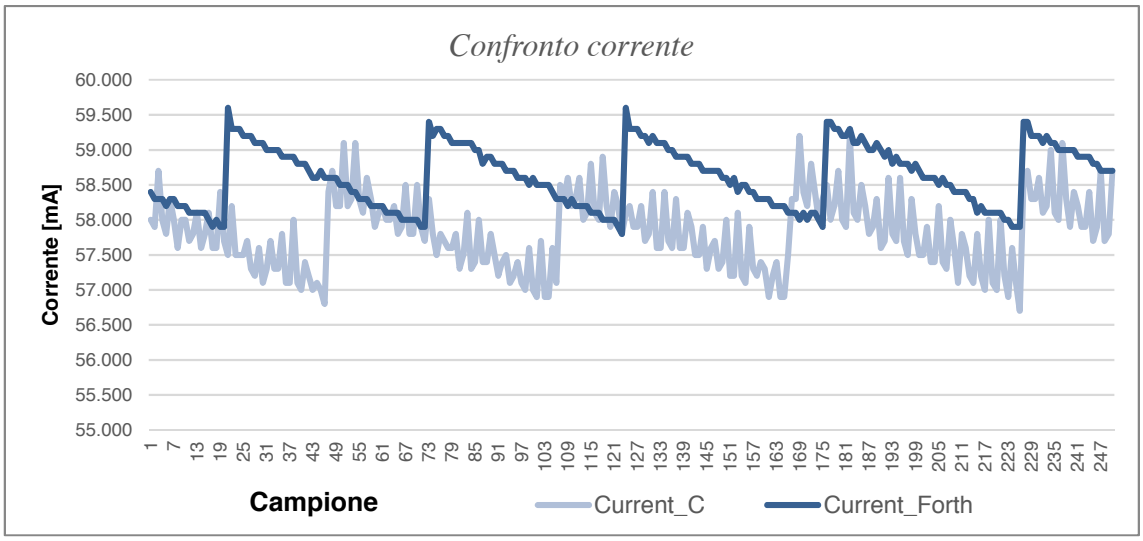

Figura 37 - Confronto valutazioni energetiche esperimento 7

Mentre nella Tabella 22 sono messi a confronto i valori di tutte le grandezze relative ai consumi energetici delle implementazioni in C ed in Mecrisp-Stellaris.

	Mecrisp-stellaris	C
Tensione di Shunt (mV)	5.85969	5.77876
Corrente (mA)	58.595699	57.788
Tensione del Bus (V)	4.812072	4.83484

Tabella 22 - Confronto media e deviazione esperimento 7

5.9 Benchmark 1 – Blink idle-loop

Come descritto nel dettaglio nel capitolo 3, il primo benchmark è relativo al lampeggio del led presente nella nucleo-board, tramite *busy-wait*.
Ciò significa che viene richiamata una funzionalità bloccante e che il microcontrollore resta in attesa finche una certa condizione non sia soddisfatta.
Il ritardo del lampeggio è stato calibrato per far cambiare stato al led (acceso-spento) ogni 3 secondi.
Di seguito verranno mostrate le implementazioni nei due linguaggi (Forth e C).
Il codice che viene mostrato nelle due implementazioni non richiede l'inclusione di codice esterno (header file o altri sorgenti) ma è completo. Tale scelta è stata fatta per ottenere risultati più coerenti e rappresentativi nel calcolo dei parametri di benchmark (soprattutto i calcoli relativi alle LOC).
Dopo aver descritto le due implementazioni, seguirà un paragrafo riepilogativo in cui verranno messi a confronto i risultati ottenuti.

5.9.1 Mecrisp-stellaris

Le valutazioni energetiche relative al primo benchmark, in linguaggio Forth, sono le seguenti:

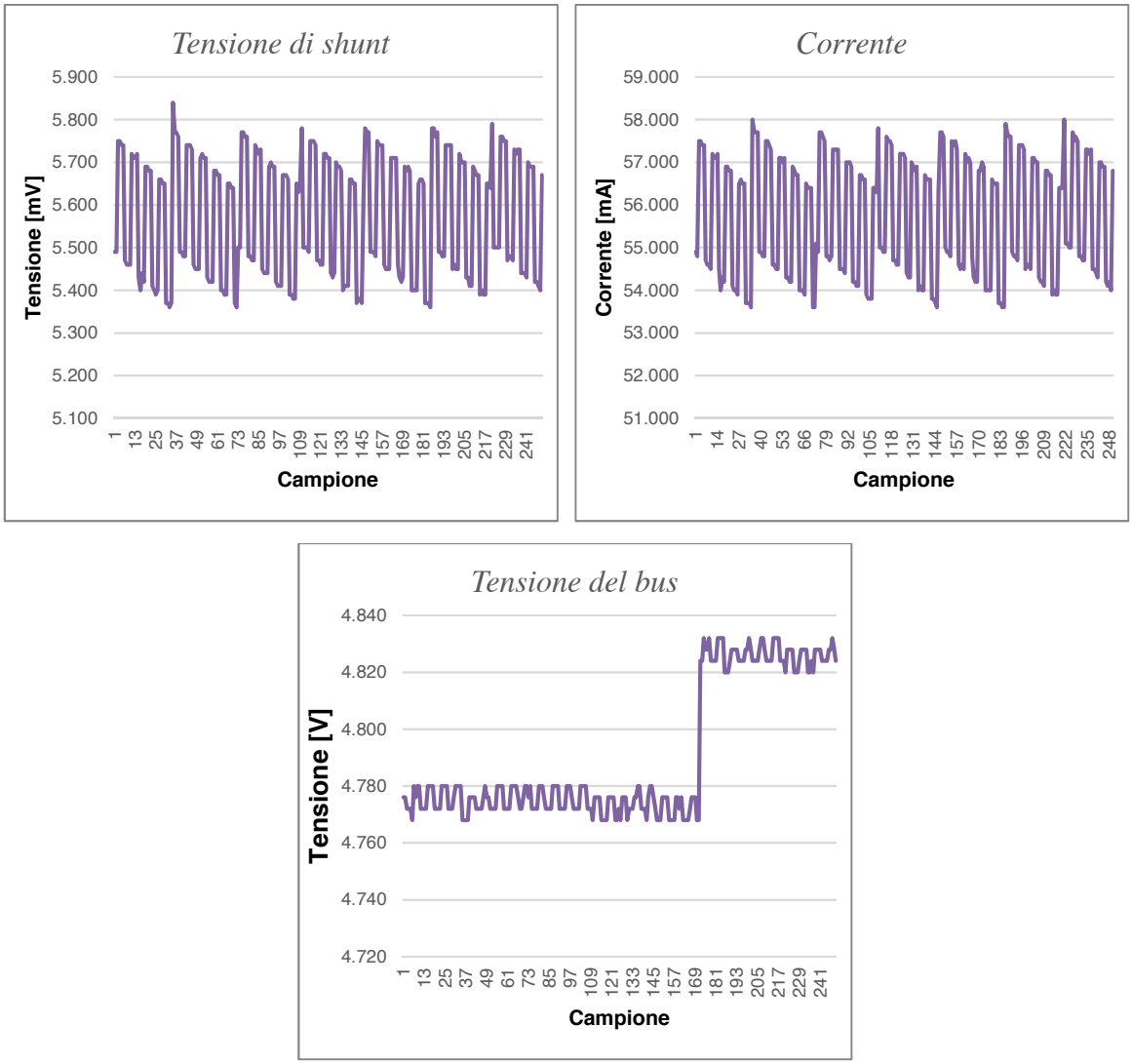

Figura 38 - Grafici valutazioni energetiche benchmark 1 - Mecrisp-Stellaris

Come si può notare dalla Figura 38 ,l'andamento della corrente, rispetto ai precedenti esperimenti, è molto più oscillante.

Questo è causato proprio dal blinking del led.

In Tabella 23 sono riportati i dati relativi ai valori medi e di deviazione.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.5662	0.142208
Corrente (mA)	55.657898	1.420288
Tensione del Bus (V)	4.749476	0.046646

Tabella 23 - Media e deviazione benchmark 1 - Mecrisp-Stellaris

Di seguito, invece, sono mostrati i valori ottenuti dei parametri di benchmark.
Il codice non viene inserito in questo paragrafo per non appesantire graficamente il tutto, ma rimane comunque visualizzabile in appendice.
I valori ottenuti sono mostrati in Tabella 24.

Metrica	Valore
LOC	21
LOC (compreso inclusioni)	21
Memoria Sorgente	681 Byte
Memoria Binario	681 Byte

Tabella 24 - Parametri benchmark 1 - Mecrisp-Stellaris

Si faccia attenzione al fatto che nel calcolo delle LOC, il benchmark 1 viene considerato l'insieme dei file *common.f* e *benchmark_1.f* (che in appendice vengono divisi per questioni di praticità).

5.9.2 Linguaggio C

Le valutazioni energetiche relative al primo benchmark, in linguaggio C, sono mostrate in Figura 39.

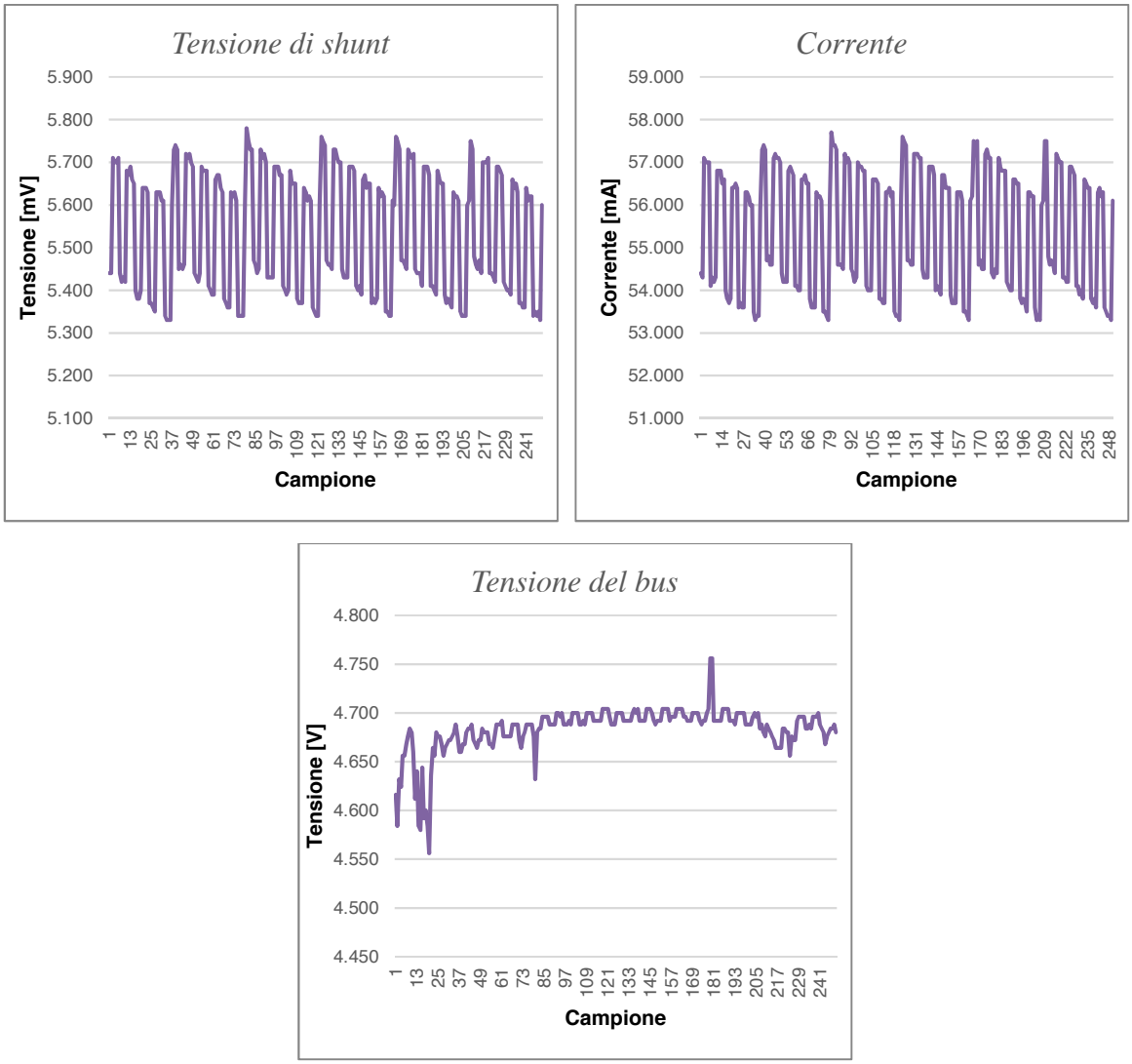

Figura 39 - Valutazioni energetiche benchmark 1 - C

I valori di media e deviazione sono mostrati in Tabella 25

	Media	**Deviazione**
Tensione di Shunt (mV)	5.5377	0.140904
Corrente (mA)	55.385199	1.412367
Tensione del Bus (V)	4.69268	0.017961

Tabella 25 - Media e deviazione benchmark 1 – C

Di seguito sono riportati invece i valori dei parametri di benchmark.

Parametro	Valore
LOC	36
LOC (compreso inclusioni)	384
Memoria Sorgente	10.8kB
Memoria Binario	4.6kB

Tabella 26 - Parametri benchmark 1 - C

Notare che la dimensione del sorgente (ma anche le LOC) include anche i moduli importati (e quindi, per esempio, tutte le funzionalità per la gestione delle GPIO ed il file in cui viene definito la vector table). Infatti, le dimensioni risultanti del binario sono notevolmente inferiori.

5.9.3 Confronto e valutazioni

La Tabella 27 mette a confronto i valori di media relativi ai consumi energetici.

	Mecrisp-stellaris	C
Tensione di Shunt (mV)	5.5662	5.5377
Corrente (mA)	55.657898	55.385199
Tensione del Bus (V)	4.749476	4.69268

Tabella 27 - Cofronto media e deviazione benchmark 1

Nella tabella Tabella 28 invece, il confronto dei valori dei parametri di benchmark delle due implementazioni.

Parametro	Mecrisp-stellaris	C
LOC	21	36
LOC (compreso inclusioni)	21	384
Memoria Sorgente	681 Byte	10.8kB
Memoria Binario	681 Byte	4.6kB

Tabella 28 - Confronto parametri benchmark 1

Anche se il file sorgente/binario dell'implementazione C risulta di dimensioni maggiori, bisogna considerare che, per esempio, in tale dimensione è incluso anche il file relativo alla vector table. In forth, invece, tale "file" è incluso, ovviamente, nel sorgente di mecrisp-stellaris e pertanto non compare in tali valori.

Nell'implementazione C, si sono considerati soltanto i moduli relativi alle gpio (*gpio.h* e *gpio.c*) ignorando del tutto le componenti non utilizzate (timer, usart, nvic…).

Quello che si evince dalla Tabella 28 è che il codice Forth risulta molto più compatto, sia in termini di LOC che di dimensioni del sorgente, rispetto all'implementazione C. Dall'altro lato ci si aspetta che Forth, essendo interpretato, sia, se pur in maniera trascurabile, più lento nell'esecuzione rispetto all'implementazione C che, invece, è compilata.

5.10 Benchmark 2 – Blink (timer e IRQ)

In questo caso, viene programmato un timer che, ogni 3 secondi genera un interrupt il quale esegue lo switch del led.

Il vantaggio è che l'interprete Forth rimane comunque interattivo, e non rimane bloccato, come nel caso precedente, permettendo un normale uso della Nucleo.

5.10.1 Mecrisp-stellaris

Le valutazioni energetiche del benchmark 2, in Forth, sono mostrate in Figura 40.

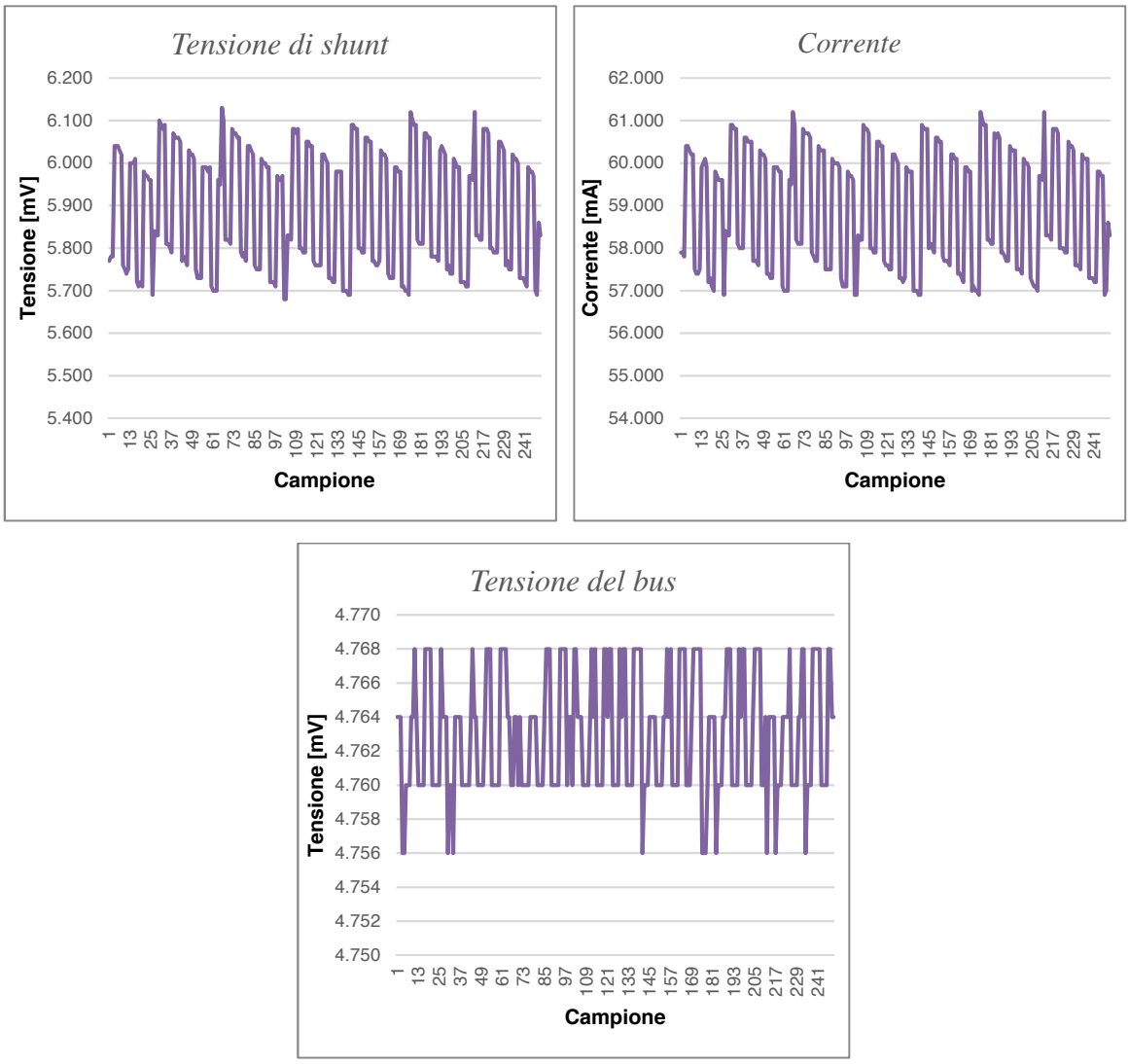

Figura 40 - Grafici valutazioni energetiche benchmark 2 - Mecrisp-Stellaris

Mentre i valori di media e deviazione sono mostrati in Tabella 29.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.88692	0.140704
Corrente (mA)	58.868801	1.406388
Tensione del Bus (V)	4.747108	0.011389

Tabella 29 - Media e deviazione benchmark 2 - Mecrisp-Stellaris

I parametri dei benchmark relativi al benchmark 2, nell'implementazione Forth, sono mostrati nella Tabella 30

Parametro	**Valore**
LOC	109
LOC (compreso inclusioni)	109
Memoria Sorgente	4,2kB
Memoria Binario	4,2kB

Tabella 30 - Parametri benchmark 2 - Mecrisp-Stellaris

5.10.2 Linguaggio C

Le valutazioni energetiche del secondo benchmark, in linguaggio C, sono mostrate in Figura 41.

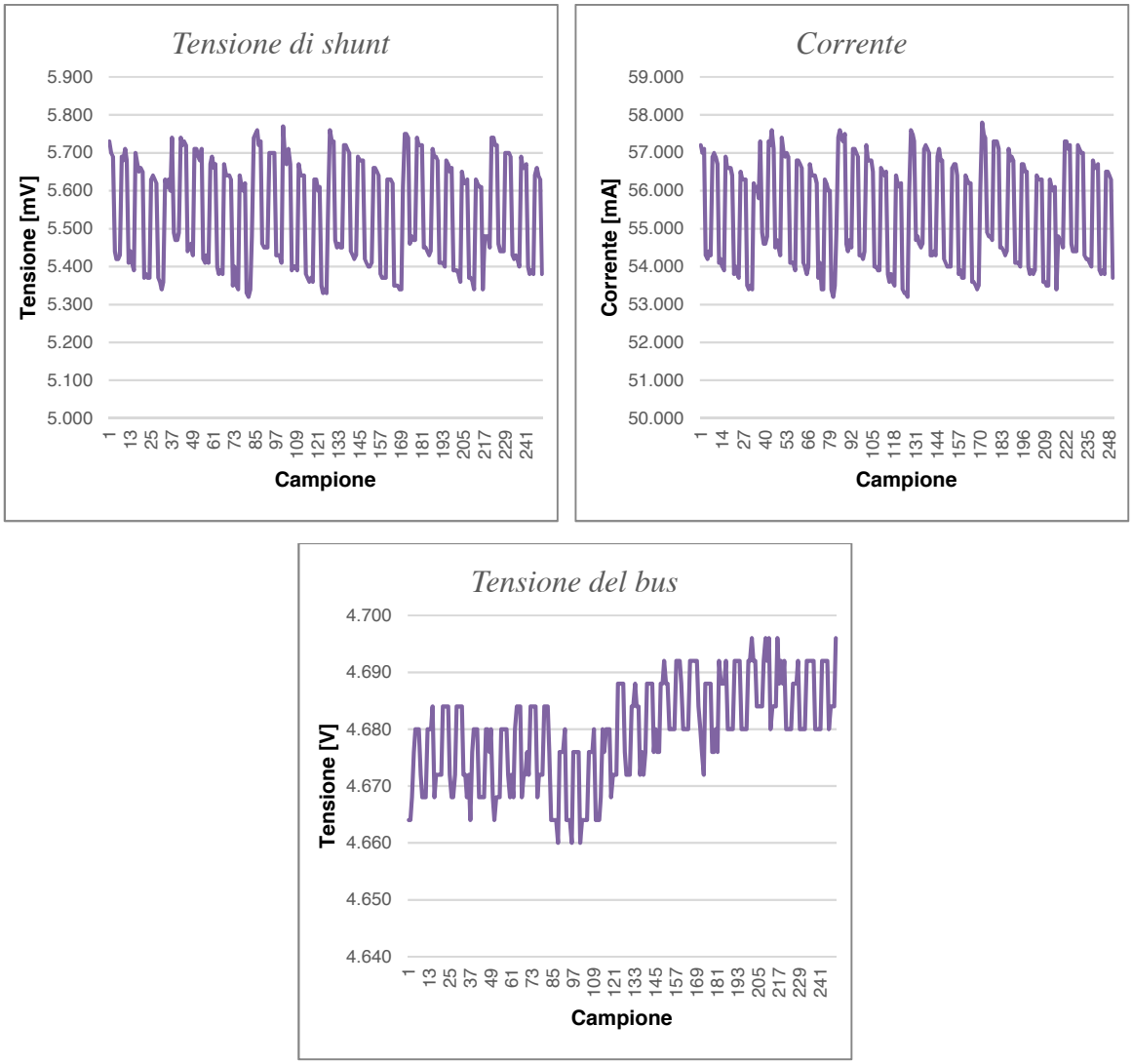

Figura 41 - Grafici valutazioni energetiche benchmark 2 - C

I valori di media e deviazione sono mostrati nella Tabella 31.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.54156	0.141823
Corrente (mA)	55.412	1.418046
Tensione del Bus (V)	4.686024	0.013126

Tabella 31 - Media e deviazione benchmark 2 - C

I parametri del benchmark, nell'implementazione C, sono mostrati nella Tabella 32.

Parametro	Valore
LOC	52
LOC (compreso inclusioni)	704
Memoria Sorgente	29.3kB
Memoria Binario	4.6kB

Tabella 32 - Parametri benchmark 2 - C

5.10.3 Confronto e valutazioni

La Tabella 33 mette a confronto i valori di media relativi ai consumi energetici.

	Mecrisp-stellaris	C
Tensione di Shunt (mV)	5.88692	5.54156
Corrente (mA)	58.868801	55.412
Tensione del Bus (V)	4.747108	4.686024

Tabella 33 - Confronto media e deviazione benchmark 2

In Tabella 34 invece, il confronto dei valori dei parametri di benchmark delle due implementazioni.

Il codice Forth risulta sempre molto più compatto e anche le dimensioni del sorgente risultano essere notevolmente minori.

Nonostante ciò, le dimensioni degli eseguibili, risultano simili. Questo è dovuto al fatto che, durante il processo di compilazione che viene effettuato nell'ambiente C, viene inserito soltanto il necessario.

Parametro	Mecrisp-stellaris	C
LOC	109	52
LOC (compreso inclusioni)	109	704
Memoria Sorgente	4,2kB	29.3kB
Memoria Binario	4,2kB	4.6kB

Tabella 34 - Confronto parametri benchmark 2

Anche in questo caso, nel sorgente C sono inclusi tutti i file come *timer.h* e *timer.c,* quelli relativi alla vector table e tutto ciò che si è reso necessario sviluppare per pilotare le periferiche. Dunque, la dimensione de sorgente potrebbe includere anche porzioni di codice non direttamente essenziali per l'implementazione del benchmark. Molte di queste cose, in ambiente forth, sono incluse nel sorgente di mecrisp-stellaris. I risultati mostrano chiaramente che, in genere, il codice sorgente di un benchmark Forth risulta essere molto più compatto rispetto il corrispettivo in C anche se, essendo un linguaggio interpretato, ci si aspetta che sia più lento nell'esecuzione del software. Inoltre, Forth presenta il grande vantaggio di esporre un'interprete, rendendo di fatto il ciclo di codifica-test molto più rapido del linguaggio C, approccio nel quale, per ogni iterazione di codifica-test, è necessario eseguire interamente la compilazione ed il flash, rallentando il processo di sviluppo.

6. Capitolo 6 – Valutazione sperimentale Suite Energy Aware

In questo capitolo, gli esperimenti vengono effettuati, a differenza di quanto fatto nel capitolo 5, con modalità on-board.

In questa modalità, il microcontrollore target di misurazione è dotato di sensore con il quale comunica e dal quale ottiene dati relativi ai propri consumi energetici.

Ciò si traduce nel fatto che il microcontrollore, in perfetta linea con i principi dell'energy aware computing, acquisisce conoscenza dei propri consumi energetici e può sfruttarli per modificare dinamicamente il proprio comportamento.

Negli esperimenti, nonostante ciò, il microcontrollore invia i dati ottenuti dal sensore al sistema host, il quale procede con l'esecuzione delle analisi.

Ovviamente, in un caso reale o in ambiente di produzione, il microcontrollore può gestire a bordo i dati sensoriali ricevuti e sulla base di questi attuare strategie di risparmio energetico.

Lo schema dei collegamenti è mostrato in figura 3.

Sono stati ripresi solamente gli esperimenti in ambiente mecrisp-stellaris, in quanto la modalità di misurazione on-board richiede che il microcontrollore comunichi con il sensore ed il driver i2c è stato implementato in linguaggio forth.

Il sensore INA219 comunica tramite I2C e per questo motivo, si è utilizzato il driver I2C realizzato per ambiente Forth.

I dati ottenuti in questi esperimenti saranno messi a confronto con gli stessi esperimenti effettuati in modalità off-board.

Quello che ci si aspetta, ancor prima di effettuare calcoli, è che i consumi nella modalità *on-board* siano maggiori in quanto il sensore è alimentato dalla stessa scheda target di misurazione.

6.1 Benchmark 1 – Blink idle-loop

Il primo benchmark esegue il blink del led in modalità *busy-wait*. Il benchmark è calibrato per far cambiare stato al led una volta ogni 3 secondi.

6.1.1 Mecrisp-stellaris

Le valutazioni energetiche, relative al primo benchmark, effettuate in modalità on board, sono riportate in Figura 42.

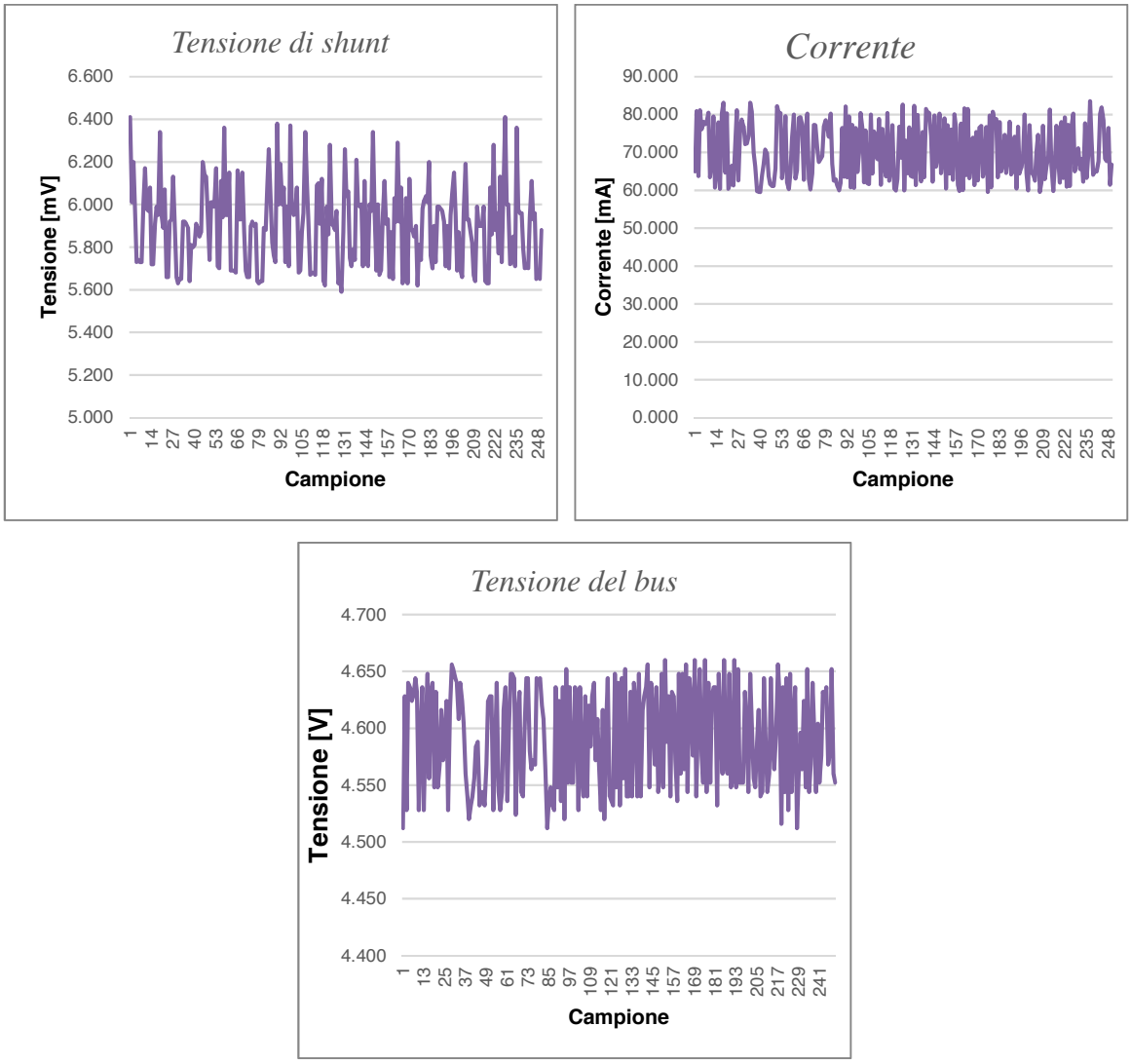

Figura 42 - Grafici valutazioni energetiche benchmark 1 on-board - Mecrisp-Stellaris

Come si può notare dalla Figura 42, il sensore introduce una deviazione maggiore che fa perdere del tutto l'andamento periodico ottenuto nel corso degli esperimenti effettuati in modalità *off-board*.
Altra cosa da notare è che i valori di corrente sono aumentati, rispetto gli esperimenti precedenti, di circa 12-15 mA.
Nella Tabella 35 sono riportati i valori relativi a media e deviazione.

	Media	**Deviazione**
Tensione di Shunt (mV)	5.89853	0.192864
Corrente (mA)	70.030703	7.431566
Tensione del Bus (V)	4.58796	0.045334

Tabella 35 - Media e deviazione benchmark 1 on-board - Mecrisp-Stellaris

La media di corrente raggiunge uno dei valori più alti rispetto tutti gli esperimenti, misurando circa 70mA. Tale valore deve tener conto, perché potrebbe alterare i risultati e le valutazioni effettuate, dell'enorme valore di deviazione (circa 7.4mA), valore molto elevato.
Nella Tabella 36 vengono messi a confronto i risultati relativi al benchmark 1 ottenuti nelle due modalità di misurazione (off-board ed on-board).

	Off-board		**On-board**	
	Media	**Deviazione**	**Media**	**Deviazione**
Tensione di Shunt (mV)	5.5662	0.142208	5.89853	0.192864
Corrente (mA)	55.657898	1.420288	70.030703	7.431566
Tensione del Bus (V)	4.749476	0.046646	4.58796	0.045334

Tabella 36 - Confronto on-board e off-board benchmark 1

Come si può notare, i consumi della misurazione *on-board,* ovviamente, sono nettamente superiori.

Questo è da imputarsi alla presenza del sensore il quale, venendo alimentato dal target di misurazione, incide notevolmente (di circa il 30%) sui consumi.

6.2 Benchmark 2 – Blink (timer e IRQ)

A differenza del primo benchmark, e come discusso nei capitoli precedenti, il *blink* viene gestito da interrupt e ISR del timer.

6.2.1 Mecrisp-stellaris

Nella figura Figura 43 le valutazioni energetiche relative al secondo benchmark, effettuate in modalità on-board.

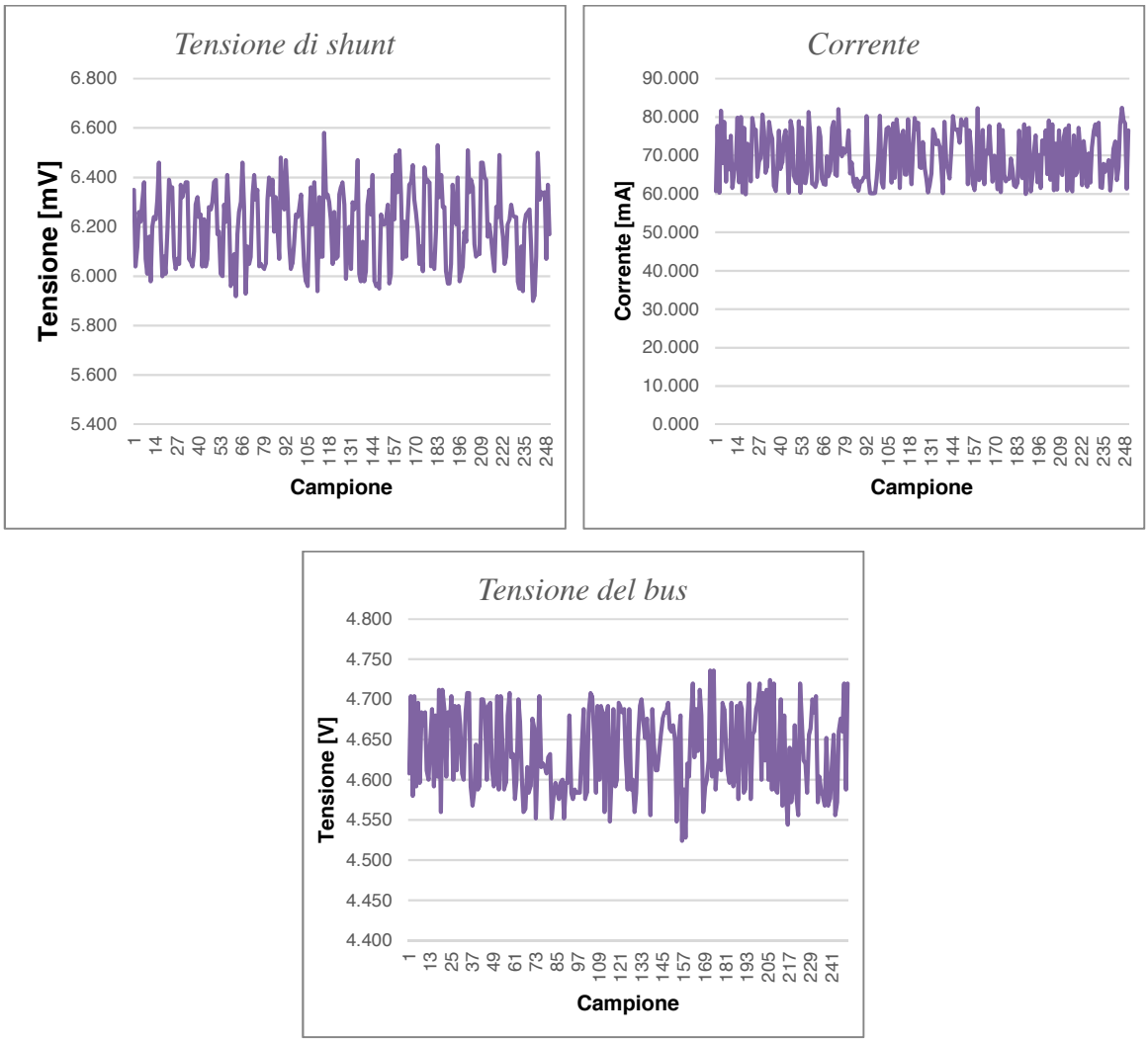

Figura 43 - Grafici valutazioni energetiche benchmark 2 on-board - Mecrisp-Stellaris

I valori di media e deviazione sono riportati nella Tabella 37

	Media	**Deviazione**
Tensione di Shunt (mV)	6.2117	0.159599
Corrente (mA)	69.940297	6.698514
Tensione del Bus (V)	4.644168	0.052424

Tabella 37 - Media e deviazione benchmark 2 on-board - Mecrisp-Stellaris

6.2.2 Confronto e valutazioni

In questo paragrafo vengono messi a confronto i risultati, relativi al secondo benchmark, nelle due modalità di misurazione (off-board e on-board).

	Off-board		**On-board**	
	Media	**Deviazione**	**Media**	**Deviazione**
Tensione di Shunt (mV)	5.88692	0.140704	6.2117	0.159599
Corrente (mA)	58.868801	1.406388	69.940297	6.698514
Tensione del Bus (V)	4.747108	0.011389	4.644168	0.052424

Tabella 38 - Confronto on-board e off-board benchmark 2 - Mecrisp-Stellaris

Anche in questo caso, ciò che si nota è un notevole aumento dei consumi.
In genere, negli esperimenti effettuati on board, la presenza del sensore, il quale viene alimentato dal target stesso di misurazione, introduce non soltanto aumenti notevoli dei consumi, ma anche una grande deviazione dei dati.
Infatti, come si può osservare dai risultati, la deviazione si assesta intorno ai (nel caso della corrente) 6.7mA.

In tutti gli esperimenti effettuati i valori dei campioni hanno un andamento periodico a *lisca di pesce*. In tutti gli esperimenti viene calcolata la deviazione dei campioni che può risultare più o meno elevata.

I primi sette esperimenti riguardano le configurazioni hardware del microcontrollore. In particolar modo, si vanno abilitando, in maniera incrementale, le periferiche e si valuta quanto queste incidono sul consumo. Ovviamente i consumi aumentano, in maniera più o meno significativa, fra un esperimento e l'altro.

I risultati ottenuti dai 7 esperimenti mostrano che i due approcci (C e Forth) presentano un andamento molto simile con una piccola differenza che, nel caso della corrente, si mantiene fra l'1% ed il 2% in più nel caso dell'ambiente forth.

Tale valore risulta comunque di poca utilità in quanto differenze minime, considerando anche, soltanto il programmatore della Nucleo-Board, incide per circa l'80% del consumo energetico (basta confrontare l'esperimento in cui viene alimentato solo il programmatore con l'esperimento in cui il microcontrollore viene alimentato ma tutte le periferiche sono disabilitate).

Per quanto riguarda i benchmark, invece, possono essere fatte anche considerazioni relativamente ai parametri di benchmark.

Nei benchmark, i sorgenti in forth presentano in genere dimensioni molto inferiori rispetto a quelli delle implementazioni del linguaggio C.

Se pur tale risultato sembra rappresentare bene la realtà dei fatti, bisogna però non dimenticare che, mentre nell'implementazione C vengono considerati, ai fini della valutazione della dimensione del sorgente, tutti i file necessari per il funzionamento del microcontrollore (considerando anche file e moduli che non sono strettamente necessari per il benchmark in quanto tale, come la vector table o altri moduli), in Forth molti di questi "moduli" per il funzionamento di tutto l'ambiente sono incorporati nel sorgente di mecrisp-stellaris e quindi non vengono considerati nel calcolo dei valori dei parametri di benchmarking.

Le misurazioni relative alla suite energy aware, effettuate in modalità on-board, presentano innnanzitutto altissimi valori di deviazione.

Tale caratteristica è da attribuirsi quasi completamente alla presenza del sensore (che, in questa modalità, viene alimentato dal target stesso di misurazione) e dal processo di campionamento periodico.

I consumi, in tale caso, sono mediamente superiori (rispetto agli analoghi esperimenti effettuati in modalità off-board) e, confrontando i risultati nelle due modalità, si ottiene che il sensore, oltre ad introdurre altissimi livelli di deviazione, incide per circa il 27-30% sui consumi energetici dell'intero nodo.

Questo risultato, ovviamente, dipende dal tipo di sensore utilizzato e dal microcontrollore ma mediamente, il sensore incide notevolmente sui consumi.

Nonostante il consumo maggiore, innegabili sono i vantaggi che si possono trarre da un'architettura in cui il nodo è dotato di un sensore dal quale può ottenere informazioni relative ai propri consumi.

7. Conclusioni

Gli obiettivi della tesi riguardano valutazioni energetiche di un modello di esecuzione simbolica nell'ambito dello sviluppo di applicazioni per sistemi embedded che, esponendo un interprete e rendendo interattivo il sistema, rende più efficiente il processo di compilazione evitando, di fatto, le operazioni tipiche di approcci basati su linguaggi compilati quali la generazione del binario e *flash* in memoria. In particolare, come architettura target è stata utilizzata la scheda Nucleo-Board basata sul microcontrollore STM32F446RE e si è analizzato Mecrisp-Stellaris, implementazione Forth per microcontrollori della famiglia STM32.

Per la realizzazione di tali obiettivi si è reso necessario, in prima istanza, lo studio dello stato dell'arte per valutare quali sono, oggi, le principali tecniche e metodologie che mirano allo sviluppo di applicazioni per sistemi resource constrained orientate all'*energy aware computing*.

Si è osservato che molti lavori sono incentrati su tecniche che operano, in contesti di WSAN, a livello dell'intero sistema.

Questo lavoro di tesi, differentemente dagli altri, si focalizza sui consumi del singolo nodo (che rappresenta, sostanzialmente, un microcontrollore) e su quelli che sono i principali fattori del consumo energetico, puntando alla fine allo sviluppo di metodologie che operano a livello del singolo nodo.

Tali metodologie sono da intendersi non mutuamente esclusive con tutte le altre che operano a livello di sistema.

Infatti, i risultati ottenuti sono perfettamente utilizzabili ed impiegabili in molti dei contesti in cui vengono utilizzate delle tecniche che operano a livello di sistema.

Per far ciò, si è preso in esame la scheda di sviluppo *Nucleo Board F446RE,* basata sul microcontrollore STM32F446RE e si sono analizzati i principali approcci per la programmazione dello stesso.

Si sono portati avanti l'approccio basato sul linguaggio C puro e l'approccio basato su Mecrisp-stellaris, implementazione dell'interprete Forth per la famiglia di microcontrollori STM32.
Si sono definiti dei software di benchmark che sono stati oggetto di valutazione sia in termini di consumi energetici, sia in termini di alcuni parametri di benchmark definiti appositamente (LOC, memoria).
Per l'esecuzione degli esperimenti si è sviluppato un setup sperimentale consistente del microcontrollore target di misurazione, del sensore INA219 e del sistema host (macchina alla quale vengono inviati i campioni delle misurazioni).
I risultati ottenuti, consistenti delle valutazioni energetiche e dei valori dei parametri di benchmarking, sono stati messi a confronto al fine di valutare vantaggi, svantaggi e quanto, le metodologie di programmazione, incidono sui consumi energetici.
Inoltre, in un'ottica di energy aware computing, si è ritenuto utile distinguere due schemi di misurazione; nelle misurazioni *off-board*, le misurazioni vengono effettuate da un microcontrollore esterno al target di misurazione (il quale sconosce di essere oggetto di rilevamenti) mentre nelle misurazioni *on-board* le misurazioni vengono effettuate dal target di misurazione; cioè, mentre esegue il suo software, periodicamente comunica con il sensore ed ottiene i dati relativi al proprio consumo energetico.
Dunque, il singolo nodo può essere reso consapevole dei suoi consumi e sulla base di questi, può attuare strategie opportune che non sono incompatibili con altri approcci di ottimizzazione energetica che operano a livello di sistema.
I risultati ottenuti, possono essere usati con altre tecniche e metodi che mirano all'efficienza dei consumi energetici.
E' stato sviluppato un setup sperimentale per l'esecuzione di esperimenti relativi ai consumi energetici.
Negli esperimenti condotti è emerso che, in genere, le implementazioni dei benchmark in Forth sono più compatte del relativo in C.
Questo è dovuto al fatto che, essendo un linguaggio interpretato ed avendo a bordo un sistema già completo, l'implementazione di un'applicazione può richiedere solo

la codifica di poche word. In C, viceversa, ogni applicazione deve essere consistente di tutte le componenti, anche non necessariamente utili per il codice applicativo, atte al funzionamento del sistema (basti pensare alla *vector table,* presente in ogni applicazione).

Di contro, la presenza dell'interprete (che comporta la continua esecuzione di codice anche quando non sono in esecuzione task) si ripercuote sui consumi energetici.

Questi incrementi di consumi energetici, che si assestano, mediamente, fra l'1% ed il 2%, comunque risultano in genere trascurabili se rapportati ai grandi benefici che un interprete può comportare in fase di sviluppo.

Infatti, la presenza dell'interprete, velocizza notevolmente il processo di codifcia-test in quanto, ad ogni iterazione di tale processo, non è richiesta la generazione dell'eseguibile ed il flash (operazioni tipiche degli approcci basati su linguaggi compilati).

Tale caratteristica, congiunta agli enormi vantaggi propri dell'ambiente Forth, getta le basi per innumerevoli sviluppi futuri.

Come primo approccio, in un'ottica *energy aware*, il microcontrollore (dotato di sensore) potrebbe sfruttare i dati che ottiene periodicamente per attuare delle strategie di consumo energetico adattive che modificano lo stato del microcontrollore (inteso come configurazione hardware) in funzione a quelli che sono i task da eseguire.

Alternativamente, potrebbero essere previste architetture ben più complesse in cui si vedono coinvolti reti di nodi che, per esempio, cooperano per il raggiungimento di un obiettivo comune sfruttando i dati relativi ai consumi energetici e attuando delle modifiche dinamiche all'intero sistema. Questo può coinvolgere anche strategie che operano a livello dell'intero sistema.

Altre applicazioni di questi risultati, alla luce degli studi effettuati, potrebbero essere trovate nello sviluppo di BMS (Battery management System) o EMS (Energy Management System).

Indice delle figure

Figura 1 - Bus Matrix STM32F446RE [13] 18
Figura 2 - Schema dei collegamenti off-board 31
Figura 3 - Schema dei collegamenti on-board 34
Figura 4 - Schema logico INA219 [12] 38
Figura 5 - Package e pin [12] 38
Figura 6 - INA219 all'interno della breakout board 39
Figura 7 - Indirizzi I2C INA219 [12] 40
Figura 8 - Protocollo di comunicazione I2C [18] 42
Figura 9 - Start condition e Stop condition [18] 44
Figura 10 - Schema logico interfaccia I2C [13] 46
Figura 11 - Eventi I2C Master Transmitter [13] 48
Figura 12 – Codice Forth per gli eventi I2C in Mecrisp-Stellaris 50
Figura 13 - Software per la comunicazione microcontrollore-sistema host 56
Figura 14 - Schermata finale esperimento 57
Figura 15 - Jumper Alimentazione nucleo-programmatore 60
Figura 16 - Registro RCC_AHB1ENR 63
Figura 17 – Grafici valutazioni energetiche esperimento 1 – Mecrisp-stellaris 64
Figura 18 – Grafici valutazioni energetiche esperimento 1 – C 65
Figura 19 - Confronto valutazioni energetiche esperimento 1 66
Figura 20 - Grafici valutazioni energetiche esperimento 2 - Mecrisp-Stellaris 69
Figura 21 - Grafici valutazioni energetiche esperimento 2 - C 70
Figura 22 - Confronto valutazioni energetiche esperimento 2 71
Figura 23 - Grafici valutazioni energetiche esperimento 3 - Mecrisp-Stellaris 73
Figura 24 - Grafici valutazioni energetiche esperimento 3 - Mecrisp-Stellaris 74
Figura 25 - Confronto valutazioni energetiche esperimento 3 75
Figura 26 - Grafici valutazioni energetiche esperimento 4 - Mecrisp-Stellaris 76

Figura 27 - Grafici valutazioni energetiche esperimento 4 - C 77
Figura 28 - Confronto valutazioni energetiche esperimento 4 78
Figura 29 - Grafici valutazioni energetiche esperimento 5 - Mecrisp-Stellaris 79
Figura 30 - Grafici valutazioni energetiche esperimento 5 - C 80
Figura 31 - Confronto valutazioni energetiche esperimento 5 81
Figura 32 - Grafici valutazioni energetiche esperimento 6 - Mecrisp-Stellaris 83
Figura 33 - Grafici valutazioni energetiche esperimento 6 - C 84
Figura 34 - Confronto corrente esperimento 6 85
Figura 35 - Valutazioni energetiche esperimento 7 - Mecrisp-Stellaris 86
Figura 36 - Grafici valutazioni energetiche esperimento 7 - C 87
Figura 37 - Confronto valutazioni energetiche esperimento 7 88
Figura 38 - Grafici valutazioni energetiche benchmark 1 - Mecrisp-Stellaris 90
Figura 39 - Valutazioni energetiche benchmark 1 - C 92
Figura 40 - Grafici valutazioni energetiche benchmark 2 - Mecrisp-Stellaris 95
Figura 41 - Grafici valutazioni energetiche benchmark 2 - C 97
Figura 42 - Grafici valutazioni energetiche benchmark 1 on-board - Mecrisp-Stellaris 101
Figura 43 - Grafici valutazioni energetiche benchmark 2 on-board - Mecrisp-Stellaris 103

Indice delle tabelle

Tabella 1 - Valutazioni energetiche baseline 62
Tabella 2 - Media e deviazione esperimento 1 - Mecrisp-stellaris 64
Tabella 3 - Media e deviazione esperimento 1 - C 65
Tabella 4 - Confronto media e deviazione esperimento 1 67
Tabella 5 - Media e deviazione esperimento 2 - Mecrisp-Stellaris 69
Tabella 6 - Media e deviazione esperimento 2 - C 71
Tabella 7 - Confronto media e deviazione esperimento 2 72
Tabella 8 - Media e deviazione esperimento 3 - Mecrisp-Stellaris 73
Tabella 9 - Media e deviazione esperimento 3 - Mecrisp-Stellaris 74
Tabella 10 - Confronto media e deviazione esperimento 3 75
Tabella 11 - Media e deviazione esperimento 4 - Mecrisp-Stellaris 77
Tabella 12 - Media e deviazione esperimento 4 - C 78
Tabella 13 - Confronto media e deviazione esperimento 4 78
Tabella 14 - Media e deviazione esperimento 5 - Mecrisp-Stellaris 80
Tabella 15 - Media e deviazione esperimento 5 - Mecrisp-Stellaris 81
Tabella 16 - Confronto media e valutazione esperimento 5 82
Tabella 17 - Media e deviazione esperimento 6 - Mecrisp-Stellaris 83
Tabella 18 - Media e deviazione esperimento 6 - C 84
Tabella 19 – Confronto media e deviazione esperimento 6.....85
Tabella 20 - Media e deviazione esperimento 7 - Mecrisp-Stellaris 87
Tabella 21 - Media e deviazione esperimento 7 - C 88
Tabella 22 - Confronto media e deviazione esperimento 7 89
Tabella 23 - Media e deviazione benchmark 1 - Mecrisp-Stellaris 90
Tabella 24 - Parametri benchmark 1 - Mecrisp-Stellaris 91
Tabella 25 - Media e deviazione benchmark 1 – C.....92
Tabella 26 - Parametri benchmark 1 - C 93
Tabella 27 - Cofronto media e deviazione benchmark 1 93
Tabella 28 - Confronto parametri benchmark 1 94

Tabella 29 - Media e deviazione benchmark 2 - Mecrisp-Stellaris 96
Tabella 30 - Parametri benchmark 2 - Mecrisp-Stellaris 96
Tabella 31 - Media e deviazione benchmark 2 - C 97
Tabella 32 - Parametri benchmark 2 - C 98
Tabella 33 - Confronto media e deviazione benchmark 2 98
Tabella 34 - Confronto parametri benchmark 2 99
Tabella 35 - Media e deviazione benchmark 1 on-board - Mecrisp-Stellaris 102
Tabella 36 - Confronto on-board e off-board benchmark 1 102
Tabella 37 - Media e deviazione benchmark 2 on-board - Mecrisp-Stellaris 104
Tabella 38 - Confronto on-board e off-board benchmark 2 - Mecrisp-Stellaris 104

Appendice

Forth – common.f

```
$40020000     CONSTANT     GPIOA
$00           CONSTANT     GPIO_MODER
$0C           CONSTANT     GPIO_PUPDR
$14           CONSTANT     GPIO_ODR
: DELAY       1000 * 0 DO LOOP ;
: BIT         ( mask addr -- addr value mask )  DUP @ ROT ;
: SET         ( addr value mask -- )  OR SWAP ! ;
: CLEAR       ( addr value mask -- )  NOT AND SWAP ! ;
: TRUTH       ( addr value mask -- value )  AND 0<> NIP ;
: LED         $20 GPIOA GPIO_ODR + ;      \ Same as 5 GPIOA ODR PIN
: ON          ( mask addr -- ) BIT SET ;
: OFF         ( mask addr -- ) BIT CLEAR ;
: ON?         BIT TRUTH ;
: SWITCH      LED ON?  IF LED OFF ELSE LED ON THEN ;
```

Forth – timer.f

```
$40023800    CONSTANT RCC        ( BASE of Reset and Clock Control Register)
$40          CONSTANT RCC_APB1
$00001             CONSTANT MASK_ENABLE_TIM2
$00002             CONSTANT MASK_ENABLE_TIM3
$00004             CONSTANT MASK_ENABLE_TIM4
$00008             CONSTANT MASK_ENABLE_TIM5
$00010             CONSTANT MASK_ENABLE_TIM6
$40000000 CONSTANT TIM2
$40000400 CONSTANT TIM3
$40000C00 CONSTANT TIM5
$40000800 CONSTANT TIM4
$40001000 CONSTANT TIM6
$40001400 CONSTANT TIM7

$E000E100  CONSTANT NVIC   ( Base Address of NVIC )
$00000000  CONSTANT NVIC_ISER0  ( Interrupt Set-Enable R 0, IRQ  0 - 31 )
$00000004  CONSTANT NVIC_ISER1  ( Interrupt Set-Enable R 1, IRQ 32 - 63 )
$00000008  CONSTANT NVIC_ISER2  ( Interrupt Set-Enable R 2, IRQ 64 - 80? )

$00  CONSTANT TIM_CR1     ( Controller R 1 )
$04  CONSTANT TIM_CR2     ( Controller R 2 )
$08  CONSTANT TIM_SMCR  ( Slave Mode Control R )
$0C  CONSTANT TIM_DIER    ( DMA Interrupt Enable  R )
$10  CONSTANT TIM_SR      ( Status R )
$14  CONSTANT TIM_EGR     ( Event Generation R )
$18  CONSTANT TIM_CCMR1 ( Capture/Compare Mode R1 )
$1C  CONSTANT TIM_CCMR2 ( Capture/Compare Mode R2 )
$20  CONSTANT TIM_CCER  ( Capture/Compare Enable R )
$24  CONSTANT TIM_CNT     ( Counter )
$28  CONSTANT TIM_PSC     ( Prescaler )
$2C  CONSTANT TIM_ARR     ( Auto Reload R)


: VALUE  ( addr valToSet -- addr valToSet maskZero  )  $0 ;

: ENABLE-REG_MASK   ( Tim -- RccReg  Mask )
                    DUP TIM2 = IF  RCC_APB1  MASK_ENABLE_TIM2  ELSE
                    DUP TIM3 = IF  RCC_APB1  MASK_ENABLE_TIM3  ELSE
                    DUP TIM4 = IF  RCC_APB1  MASK_ENABLE_TIM4  ELSE
                    DUP TIM5 = IF  RCC_APB1  MASK_ENABLE_TIM5  ELSE
                    DUP TIM6 = IF  RCC_APB1  MASK_ENABLE_TIM6  ELSE
                    THEN THEN THEN THEN THEN
                          ROT DROP ;
: ENABLE   ( Tim -- ) ENABLE-REG_MASK SWAP RCC + BIT SET ;
: DISABLE  ( Tim -- ) ENABLE-REG_MASK SWAP RCC + BIT CLEAR ;
: STOP     ( tim -- ) TIM_CR1 + $1 SWAP BIT CLEAR ;
: START    ( tim -- ) TIM_CR1 + $1 SWAP BIT SET ;
: START?   ( tim -- ) TIM_CR1 + $1 SWAP BIT TRUTH ;
: INIT   ( tim -- ) DUP  ENABLE DUP TIM_DIER + $1 SWAP BIT SET    DROP ;

: LIMIT      TIM_ARR + ;         ( LIMIT is the auto-reload Value )
: LIMIT?           LIMIT @ . ;
: PRESCALER  TIM_PSC + ;
: PRESCALER? PRESCALER @ . ;
: CNT                  TIM_CNT + ;
: CNT?             CNT @ . ;
: +INTERRUPT   TIM_DIER + $1 SWAP BIT SET ;
: -INTERRUPT   TIM_DIER + $1 SWAP BIT CLEAR ;
: UPDATE               TIM_EGR + $1 SWAP BIT SET ; ( Re-initialize the
counter and update the registers )
: -UIF     ( Tim -- )   TIM_SR +  $1 SWAP BIT CLEAR ;   ( Clear the signal of
interrupt request )
```

Forth – benchmark_1.f

```
\ include common.f
$A80204A0 GPIOA GPIO_MODER + !  $64000050 GPIOA GPIO_PUPDR + !

 : BENCHMARK_1   ( n -- )  0 DO  SWITCH  7500 DELAY  LOOP  ;
```

Forth – benchmark_2.f

```
\ include common.f
\ include timer.f

( Enable the interrupt line NVIC for Timer 2 )
 $10000000 NVIC NVIC_ISER0 + BIT SET

: ISR-TIM2    SWITCH TIM2 -UIF ;

TIM2 ENABLE
TIM2 PRESCALER      16000  VALUE SET
TIM2 LIMIT          3000   VALUE SET

' ISR-TIM2 IRQ-TIM2 !
TIM2         +INTERRUPT
TIM2         START
```

Forth — benchmark_1_onboard.f

```
\ include common.f
\ include timer.f
\ include driver_i2c.f
\ include driver_ina219.f

( ISR FOR TIM 2 )
0 VARIABLE FLAG-DELAY
1 VARIABLE N        ( Creo la variabile N, del numero dei campioni)
: ISR-TIM2


             INA_READ-SHUNT-VOLTAGE-microV         ( shunt-voltage )
             DUP    $000000FF AND   EMIT
             DUP    $0000FF00 AND   256 /             EMIT
             DUP    $00FF0000 AND   256 256 * /       EMIT
                    $FF000000 AND   256 256 256 * * / EMIT

            INA_READ-BUS-VOLTAGE-mV         ( bus-voltage )
            DUP    $000000FF AND   EMIT
            DUP    $0000FF00 AND   256 /             EMIT
            DUP    $00FF0000 AND   256 256 * /       EMIT
                   $FF000000 AND   256 256 256 * * / EMIT

            INA_READ-CURRENT-microA         ( current )
            DUP    $000000FF AND   EMIT
            DUP    $0000FF00 AND   256 /             EMIT
            DUP    $00FF0000 AND   256 256 * /       EMIT
                   $FF000000 AND   256 256 256 * * / EMIT

              TIM2 -UIF ;


TIM2 ENABLE
TIM2 PRESCALER      16000  VALUE SET
TIM2 LIMIT          700    VALUE SET

' ISR-TIM2 IRQ-TIM2 !
TIM2         +INTERRUPT
\ TIM2              START

 : BLINK_IDLE_LOOP   ( n -- )  0 DO    SWITCH  7500 DELAY  LOOP  ;

\ 1000 BLINK_IDLE_LOOP
```

Forth — benchmark_2_onboard.f

```
\ include common.f
\ include timer.f
\ include driver_i2c.f
\ include driver_ina219.f

0 VARIABLE FLAG-DELAY
1 VARIABLE N        ( Creo la variabile N, del numero dei campioni)
: ISR-TIM2


             INA_READ-SHUNT-VOLTAGE-microV         ( shunt-voltage )
             DUP    $000000FF AND   EMIT
             DUP    $0000FF00 AND   256 /             EMIT
             DUP    $00FF0000 AND   256 256 * /       EMIT
                    $FF000000 AND   256 256 256 * * / EMIT

            INA_READ-BUS-VOLTAGE-mV         ( bus-voltage )
            DUP    $000000FF AND   EMIT
            DUP    $0000FF00 AND   256 /             EMIT
            DUP    $00FF0000 AND   256 256 * /       EMIT
                   $FF000000 AND   256 256 256 * * / EMIT

            INA_READ-CURRENT-microA         ( current )
            DUP    $000000FF AND   EMIT
            DUP    $0000FF00 AND   256 /             EMIT
            DUP    $00FF0000 AND   256 256 * /       EMIT
                   $FF000000 AND   256 256 256 * * / EMIT

              TIM2 -UIF ;

: ISR-TIM3          SWITCH  TIM3 -UIF ;

TIM2 ENABLE
TIM2 PRESCALER      16000  VALUE SET
TIM2 LIMIT          700    VALUE SET

' ISR-TIM2 IRQ-TIM2 !
TIM2         +INTERRUPT
\ TIM2              START


TIM3 ENABLE
TIM3 PRESCALER 16000 VALUE SET
TIM3 LIMIT  3000 VALUE SET
' ISR-TIM3 IRQ-TIM3 !
TIM3 +INTERRUPT
TIM3 START
```

Forth – i2c_driver_part1.f

```
: GPIO{  ( addr -- )  ;
: PORT   ( addr -- addr )  DUP CONSTANT $0400 + ;
: }GPIO  ( addr -- )  DROP ;

$40020000
GPIO{  PORT GPIOA  PORT GPIOB  PORT GPIOC ( ... ) }GPIO
( More GPIO port could be added... )

: REGS{   0  ;
: OFFSET>REGS  ( u -- ) NIP ;
: REG  CREATE  DUP , OVER + DOES> @ + ;
: }REGS   2DROP ;

$04 REGS{
      REG MODER  REG OTYPER  REG OSPEEDR
      REG PUPDR  REG IDR     REG ODR
      REG BSSR   REG LCKR    REG AFRL
      REG AFRH
}REGS

$40023800 CONSTANT RCC
$04 REGS{
   $20 OFFSET>REGS     \ Define offset
   REG APB1RSTR  REG APB2RSTR
   $40 OFFSET>REGS
   REG APB1ENR   REG APB2ENR
}REGS

: 1BIT   $1 1 ;
: 2BIT   $3 2 ;
: 4BIT   $F 4 ;
: MASK  ( index mask width -- offset_mask ) ROT * LSHIFT ;
: PIN   SWAP 1BIT MASK SWAP ;
: (MODE)   MODER OVER 2BIT MASK SWAP ;
: MODE@   (MODE) @ AND SWAP 2 * RSHIFT ;
: MODE!   >R R@ (MODE) @ SWAP NOT AND ROT $3 AND ROT 2 * LSHIFT OR R> ! ;
: (AF)  OVER 8 < IF AFRL ELSE AFRH SWAP 8 - SWAP THEN  OVER 4BIT MASK SWAP ;
: AF@   (AF) @ AND SWAP 4 * RSHIFT ;
: AF!   (AF) >R R@ @ SWAP NOT AND ROT $F AND ROT 4 * LSHIFT OR R> ! ;
: OUT@   ODR PIN BIT TRUTH ;
: OUT!   ODR PIN BIT SWAP OVER NOT AND >R ROT AND R> OR SWAP ! ;


$40005400 constant I2C1
$04 REGS{
    REG CR1  REG CR2  REG OAR1  REG OAR2  REG DR
    REG SR1  REG SR2  REG CCR   REG TRISE
}REGS

21 1BIT MASK CONSTANT I2C1EN
21 1BIT MASK CONSTANT I2C1RST

: SCL   8 GPIOB ;
: SDA   9 GPIOB ;
```

Forth – i2c_driver_part2.f

```
: i2c-init
   $4 SCL AF!    $4 SDA AF!    \ Set Alternate Function 4 (I2C1) for pins
   $2 SCL MODE!  $2 SDA MODE!  \ Select Alternate Function mode for pins
   I2C1EN  RCC APB1ENR bis!  \ Enable I2C1
   I2C1RST RCC APB1RSTR bis!  \ Reset I2C1
   I2C1RST RCC APB1RSTR bic!  \
   16                        \ The APB frequency is f=16 MHz  (1/f)=62.5
ns=62500 ps
   DUP
   $3F I2C1 CR2 hbic!        \ Set Freq
       I2C1 CR2 hbis!        \
   DUP
   1000000 SWAP /            \ T=(1/(f*10^6))*10^12 = (10^6)/f, f is expressed
in MHz, T in picoseconds
                             \ We are doing all the math in picoseconds to
avoid fractions
   5000 1000 * SWAP /        \ (5000 ns = 5000000 ps) 5000000 ps / T= CCR
(Clock Control Register)
                             \ eg. FREQ = 16MHZ, 62.5 ns; DUTY=0 62.5ns * 80 =
5000ns -> 200KHz
   DUP 15 1BIT MASK OR  I2C1 CCR h! \ SCL CCR
   1 +  I2C1 TRISE h!         \ SCL Rise Time must be +1000ns:
(1000ns/(1/f))+1=f+1
10 1BIT MASK I2C1 CR1 hbis! \ Set ACK (Acknowledge enable, CR1: bit 10)
   0 1BIT MASK I2C1 CR1 hbis! ; \ Set Periphal Enable PE (CR1: bit 0)
0  1BIT MASK CONSTANT I2C-SR1-SB              (    1  )
1  1BIT MASK CONSTANT I2C-SR1-ADDR            (    2  )
2  1BIT MASK CONSTANT I2C-SR1-BTF             (    4  )
6  1BIT MASK CONSTANT I2C-SR1-RxNE            (   64  )
7  1BIT MASK CONSTANT I2C-SR1-TxE             (  128  )
10 1BIT MASK CONSTANT I2C-SR1-AF              ( 1024  )
 0 1BIT MASK CONSTANT I2C-SR2-MSL             (    1  )

: i2c-DR! ( value -- )    I2C1 DR c! ;
: i2c-DR@ ( -- value )    I2C1 DR c@ ;
: i2c-start! ( -- )             8 1BIT MASK I2C1 CR1 hbis! ;
: i2c-stop! ( -- )        9 1BIT MASK I2C1 CR1 hbis! ;
: i2c-clear-AF              10 1BIT MASK I2C1 SR1 hbic! ;
: i2c-set-ACK               10 1BIT MASK I2C1 CR1 hbis! ;
: i2c-SR2-flag? ( u -- ) I2C1 SR2 hbit@ ;
$ffff variable i2c.timeout
: i2c-MSL?   ( -- b )     0 1BIT MASK I2C1 SR2 hbit@ ;
: i2c-SR1-flag? ( u -- b ) I2C1 SR1 hbit@ ;  ( lascia sullo stack un Bool,
corrispondente alla maschera u )
: i2c-SR1-wait   ( u -- )  i2c.timeout @ begin 1- 2dup 0= swap i2c-SR1-flag?
or until 2drop ;
: i2c-SR2-!wait  ( u -- )  i2c.timeout @ begin 1- 2dup 0= swap i2c-SR2-flag?
0= or until 2drop ;
: i2c-nak?          ( -- b ) 10 1BIT MASK i2c-SR1-flag? ;
: i2c-START  ( -- ) i2c-start! ;
: i2c-EV5 i2c-SR1-SB i2c-SR1-wait ;
: i2c-EV6a i2c-SR1-ADDR i2c-SR1-AF or i2c-SR1-wait ;
: i2c-EV6b I2C1 SR1 h@ DROP  I2C1 SR2 h@ DROP ;
: i2c-EV6  i2c-EV6a  i2c-EV6b ;
: i2c-EV7  i2c-SR1-RxNE i2c-SR1-wait ;
: i2c-STOP       (  --  )  i2c-stop! i2c-SR2-MSL i2c-SR2-!wait ;
: I2C-PROBE ( c -- nak )  I2C-START i2c-EV5 shl i2c-DR!  i2c-EV6 i2c-nak?
                          i2c-clear-AF  I2C-STOP ;
: i2c-EV8_1  i2c-SR1-TxE  i2c-SR1-wait ;
: i2c-EV8   ;
: i2c-EV8_2  ( i2c-EV8_1 ) i2c-SR1-BTF i2c-SR1-TxE OR  i2c-SR1-wait ;
: I2C-CLOSE ( -- nak )  i2c-EV8_2  i2c-nak? i2c-clear-AF i2c-stop ;
```

Forth – i2c_driver_part3.f

```
: I2C-ADDR-W ( u --) \ Start a new transaction and send address in write mode
  I2C-START
  shl
  i2c-EV5
  i2c-DR!                   \ Sends address (write mode)
  i2c-EV6a                  \ wait for completion of addressing or AF
;

: I2C-ADDR-R ( u --) \ Start a new transaction and send address in read mode
  I2C-START
  shl  1 or
  i2c-EV5
  i2c-DR!                   \ Sends address (read mode)
  i2c-EV6a                  \ wait for completion of addressing or AF
;

: >I2C  ( u -- ) \ Sends a byte over i2c. Use after i2c-addr
  i2c-EV6b       \ Clears AF
  i2c-EV8_1
  i2c-DR! ;


: I2C>
      i2c-EV6b            ( Clear il Bit AF )
      i2c-EV7             ( wait a byte )
      i2c-DR@             ( fetch DR )

      i2c-set-ACK

;
: I2C2>      I2C> I2C> ;
```

Forth – driver_ina219.f

```
\ include driver_i2c.f

$40 CONSTANT INA219

$00 CONSTANT INA_CONFIGURATION
$01 CONSTANT INA_SHUNT-VOLTAGE
$02 CONSTANT INA_BUS-VOLTAGE
$03 CONSTANT INA_POWER
$04 CONSTANT INA_CURRENT
$05 CONSTANT INA_CALIBRATION

I2C-INIT

: INA_SET-REGISTER-POINTER    ( addr -- ) INA219 I2C-ADDR-w  >I2C ;
: INA_READ1 ( -- value )      INA219  I2C-ADDR-R  I2C>  I2C-STOP ;
: INA_READ2 ( -- val1 val2)   INA219  I2C-ADDR-R  I2C2> I2C-STOP ;

: INA_READ-BUS-VOLTAGE-REG  ( -- val1  val2 )  INA_BUS-VOLTAGE
INA_SET-REGISTER-POINTER  INA_READ2 ;
: INA_READ-BUS-VOLTAGE-mV ( -- voltage_mV )    INA_READ-BUS-VOLTAGE-
REG SWAP 256 * or  shr shr shr 4 * ;

: INA_READ-SHUNT-VOLTAGE-REG  ( -- val1 val2 )  INA_SHUNT-VOLTAGE
INA_SET-REGISTER-POINTER  INA_READ2 ;
: INA_READ-SHUNT-VOLTAGE-microV   ( -- voltage_microV ) INA_READ-
SHUNT-VOLTAGE-REG       ( -- val1  val2 )
                  SWAP  256 * or 10 *;


: INA_READ-SHUNT-VOLTAGE-milliV  ( -- voltage_milliV )
                  INA_READ-SHUNT-VOLTAGE-microV      1000 / ;
: INA_READ-CONFIGURATION-REG  ( -- val1 val2 ) INA_CONFIGURATION
INA_SET-REGISTER-POINTER INA_READ2 ;

: INA_READ-CURRENT-microA     ( -- current_val )
                        INA_READ-SHUNT-VOLTAGE-microV 10 * ;
```

C – gpio.h

```
#ifndef GPIO_HEADER
#define GPIO_HEADER
#include "utils.h"
typedef void * gpio_t;
#define GPIO_A          (void *) 0x40020000
#define GPIO_B          (void *) 0x40020400
#define GPIO_C          (void *) 0x40020800
#define GPIO_D          (void *) 0x40020C00
#define GPIO_E          (void *) 0x40021000
#define GPIO_F          (void *) 0x40021400
#define GPIO_G          (void *) 0x40021800
#define GPIO_H          (void *) 0x40021C00
#define GPIO_PUPDR      (int) 0x0C
#define GPIO_A_BIT_RCC        0
#define GPIO_B_BIT_RCC        (int) 1
#define GPIO_C_BIT_RCC        (int) 2
#define GPIO_D_BIT_RCC        (int) 3
#define GPIO_E_BIT_RCC        (int) 4
#define GPIO_F_BIT_RCC        (int) 5
#define GPIO_G_BIT_RCC        (int) 6
#define GPIO_H_BIT_RCC        (int) 7

void gpio_enable_rcc(gpio_t gpio);
void gpio_disable_rcc(gpio_t gpio);
void gpio_set_pullup_all(gpio_t gpio);
#endif
```

C – gpio_part_1.c

```
#include "gpio.h"
#include "utils.h"

void gpio_enable_rcc(gpio_t gpio){
      if(gpio == GPIO_A){
            RCC_AHB1EN |= bit(GPIO_A_BIT_RCC);
      } else if(gpio == GPIO_B){
            RCC_AHB1EN |= bit(GPIO_B_BIT_RCC);
      } else if(gpio == GPIO_C){
            RCC_AHB1EN |= bit(GPIO_C_BIT_RCC);
      } else if(gpio == GPIO_D){
            RCC_AHB1EN |= bit(GPIO_D_BIT_RCC);
      } else if(gpio == GPIO_E){
            RCC_AHB1EN |= bit(GPIO_E_BIT_RCC);
      } else if(gpio == GPIO_F){
            RCC_AHB1EN |= bit(GPIO_F_BIT_RCC);
      } else if(gpio == GPIO_G){
            RCC_AHB1EN |= bit(GPIO_G_BIT_RCC);
      } else if(gpio == GPIO_H){
            RCC_AHB1EN |= bit(GPIO_H_BIT_RCC);
      } else {
            /* ERROR */
      }
}
```

C — gpio_part_2.c

```
void gpio_disable_rcc(gpio_t gpio){
      if(gpio == GPIO_A){
            RCC_AHB1EN &= ~bit(GPIO_A_BIT_RCC);
      } else if(gpio == GPIO_B){
            RCC_AHB1EN &= ~bit(GPIO_B_BIT_RCC);
      } else if(gpio == GPIO_C){
            RCC_AHB1EN &= ~bit(GPIO_C_BIT_RCC);
      } else if(gpio == GPIO_D){
            RCC_AHB1EN &= ~bit(GPIO_D_BIT_RCC);
      } else if(gpio == GPIO_E){
            RCC_AHB1EN &= ~bit(GPIO_E_BIT_RCC);
      } else if(gpio == GPIO_F){
            RCC_AHB1EN &= ~bit(GPIO_F_BIT_RCC);
      } else if(gpio == GPIO_G){
            RCC_AHB1EN &= ~bit(GPIO_G_BIT_RCC);
      } else if(gpio == GPIO_H){
            RCC_AHB1EN &= ~bit(GPIO_H_BIT_RCC);
      } else {
            /* ERROR */
      }
}
void gpio_set_pullup_all(gpio_t gpio){
      *((uint32_t*)(gpio + GPIO_PUPDR)) = 0x55555555;
}
```

C — linker.ld

```
MEMORY {

      FLASH (rx) : ORIGIN = 0X08000000, LENGTH = 0x4000
      SRAM (xrw) : ORIGIN = 0X20000000, LENGTH = 0x4000
}

SECTIONS {

      .text : {
            *(.vector_table)
            *.(.text)
      } > FLASH

      .data : {
            *(.data)
      } > SRAM
}
```

C — Makefile

```
ARM=arm-none-eabi

LNK_SCR = linker.ld

CFLAGS = -c -fno-common -mcpu=cortex-m4 -mthumb
LFLAGS =  -T ${LNK_SCR}
CPFLAGS = -O Binary

OBJECTS = nvic.o vector_table.o timer.o serial.o gpio.o i2c.o utils.o

all: main.bin blink.bin blink_busy_wait.bin

%.o : %.c
      ${ARM}-gcc ${CFLAGS} $< -o $@

%.elf : %.o ${OBJECTS}
      ${ARM}-ld ${LFLAGS} $+ -o $@

%.bin : %.elf
      ${ARM}-objcopy $< $@ -O binary

clean:
      rm -rf *.bin
      rm -rf *.o
      rm -rf *.elf
```

C – nvic.h

```
#ifndef NVIC_H
#define NVIC_H

/* NVIC REGISTERS*/
#define NVIC_BASE       (void*)     0xE000E100  // Nested Vector
Interrupt Controller
#define NVIC_ISR0 (int) 0x00              // Interrupt Set-enable
Register 1 - Periferiche [0  - 31]
#define NVIC_ISR1 (int) 0x04              // Interrupt Set-enable
Register 2 - Periferiche [32 - 63]
#define NVIC_ISR2 (int) 0x08              // Interrupt Set-enable
Register 3 - Periferiche [64 - 95]

/* Ricordare che:
 *
 * TIM2 NVIC Position 28  -> ISER0    -> MASK = $10000000 )
 * TIM3 NVIC Position 29  -> ISER0    -> MASK = $20000000 )
 * TIM4 NVIC Position 30  -> ISER0    -> MASK = $40000000 )
 * TIM5 NVIC Position 50  -> ISER1 50-32=18 -> MASK = $00040000 )
 * TIM6 NVIC Position 54  -> ISER1 54-32=22 -> MASK = $00400000 )
 */

#define NVIC_TIMER_1    (int) 25
#define NVIC_TIMER_2    (int) 28
#define NVIC_TIMER_3    (int) 29
#define NVIC_TIMER_4    (int) 30    // Valori Vector Table
#define NVIC_TIMER_5    (int) 50
#define NVIC_TIMER_6    (int) 54

#define NVIC_I2C_1_EV   (int) 31
#define NVIC_I2C_1_ERR  (int) 32
#define NVIC_I2C_2_EV   (int) 33
#define NVIC_I2C_2_ERR  (int) 34
#define NVIC_I2C_3_EV   (int) 72
#define NVIC_I2C_3_ERR  (int) 73

int NVIC_enableIRQ(int position);

#endif
```

C – nvic.c

```
#include "nvic.h"

#include <stdint.h>

int NVIC_enableIRQ(int position){
      unsigned long mask = 0x01;

      /* Posizione NON VALIDA  */
      if(position < 0 || position > 95){
            // Out of vector table
            return 0;
      }
      if(position >=  0 && position <= 31){
            // ISR 0
            mask = bit(position);
            *((uint32_t*)(NVIC_BASE + NVIC_ISR0)) |= mask;
      } else if(position >= 32 && position <= 63){
            // ISR 1
            mask = bit(position - 32);
            *((uint32_t*)(NVIC_BASE + NVIC_ISR1)) |= mask;
      } else {
            // ISR 2
            mask = bit(position - 64);
            *((uint32_t*)(NVIC_BASE + NVIC_ISR2)) |= mask;
      }

      return 1;
}
```

C – serial.h

```
#ifndef SERIAL_H
#define SERIAL_H

#include "utils.h"

struct usart_t{
      void *base;
};
typedef struct usart_t usart_t;

#define USART_SR        (int) 0x00        // Status Register
#define USART_DR        (int) 0x04        // Data Register
#define USART_BRR       (int) 0x08        // BaudRate Register
#define USART_CR1       (int) 0x0C        // Control Register 1 //
BitEnable, BitM (WordLength), Pce (Parity),
                                                        // TE,RE
(Transmitter/Recv Enable), BitBreak
#define USART_CR2       (int) 0x10        // BitStops, ClockEnable,
#define USART_CR3       (int) 0x14        //
#define USART_GTPR      (int) 0x18        // Guard Time And Prescaler
Register // Prescaler value,  RCC APB1ENR 0x40 bit 17

/* Defines USART */
#define USART_1         (void*) 0x40011000      // APB2
#define USART_2         (void*) 0x40004400      // APB1
#define USART_3         (void*) 0x40004800      // APB1
#define USART_4         (void*) 0x40004C00      // APB1
#define USART_5   (void*) 0x40005000      // APB1
#define USART_6         (void*) 0x40011400  // APB2

#define USART_8N1 (int) 1                 // 8Bit, NoParity, 1bitStop

int usart_enable(usart_t *usart);
int usart_set_baudrate(usart_t *usart, uint32_t baudrate);
int usart_send_char(usart_t *usart, char c);
int usart_send_string(usart_t *usart, char *str);

void serial_begin(int conf, int baudrate);
void serial_write_char(char c);
void serial_write_string(char *str);
void serial_write_int(uint32_t val);


#endif
```

C — serial_part_1.c

```
#include "serial.h"
#include "utils.h"

int usart_enable(usart_t *usart){

      // GPIO - Mecrisp
      RCC_AHB1EN |= 0x9F;
      RCC_APB1EN |= bit(17);
      GPIOA_MODE |= 0xA0;

      GPIOA_AFRL &= ~(0x0000FF00);
      GPIOA_AFRL |= 0x7700;

      *((uint32_t*)(usart->base + USART_CR1)) |= (
bit(13)|bit(3)|bit(2) );

      *((uint32_t*)(usart->base + USART_CR2)) |= bit(11);

      return 1;
}

int usart_set_baudrate(usart_t *usart, uint32_t baudrate){

      if(baudrate == 115200){
            *((uint32_t*)(usart->base + USART_BRR)) = 0x8B;
      } else if(baudrate == 9600){
            // Value..
      } else {
            // Default Value = 115200
            *((uint32_t*)(usart->base + USART_BRR)) = 0x8B;
      }
      return 1;
}

int usart_send_char(usart_t *usart, char c){

      while(!( *((uint32_t*) (usart->base + USART_SR)) & bit(7)));
      * ((uint32_t *)(usart->base + USART_DR)) =  c;

      while(!( *((uint32_t*) (usart->base + USART_SR)) & bit(6)));

}
```

C — serial_part_1.c

C – serial_part_2.c

```
int usart_send_string(usart_t *usart, char *str){
	int i=0;
	char current = str[0];

	while(current != '\0'){
		usart_send_char(usart, current);
		i++;
		current = str[i];
	}

	return 1;

}
int usart_send_int(usart_t *usart, uint32_t val){
	char const digit[] = "0123456789";
	char buffer[16];
	int i=0;
	char *p = buffer;
	int shifter = val;

	for(i=0; i<16; ++i){
		buffer[i] = '\0';
	}

	if(val < 0){
		*p++ = '-';
		val *= -1;
	}
	do{
		++p;
		shifter = shifter / 10;
	}while(shifter);

	*p = '\0';
	do{
		*--p = digit[val%10];
		val = val/10;
	}while(val);

	shifter = 0;
	while(buffer[shifter] != '\0'){
		usart_send_char(usart, buffer[shifter]);
		shifter++;
	}
}
```

C – serial_part_3.c

```
void serial_begin(int format, int baudrate){
	usart_t usart_2;
	usart_2.base = USART_2;

	usart_enable(&usart_2); // Set baudrate to 115200

	usart_set_baudrate(&usart_2, 115200);
}

void serial_write_char(char c){
	usart_t usart_2;
	usart_2.base = USART_2;

	// Send Char
	usart_send_char(&usart_2, c);
}
void serial_write_string(char *str){
	int i=0;
	char current = str[0];

	while(current != '\0'){
		serial_write_char(current);
		i++;
		current = str[i];
	}
}

void serial_write_int(uint32_t val){
	usart_t usart_2;
	usart_2.base = USART_2;

	usart_send_int(&usart_2, val);
}
```

C – timer_part_1.h

```
#ifndef TIMER_H
#define TIMER_H

#include "nvic.h"
#include "vector_table.h"
#include "utils.h"

struct timer_t{
      pointer base;
};

typedef struct timer_t timer_t;

/* TIMER peripheral addresses */
#define TIMER_1         (void *) 0x40010000
#define TIMER_2         (void *) 0x40000000
#define TIMER_3         (void *) 0x40000400
#define TIMER_4         (void *) 0x40000800
#define TIMER_5         (void *) 0x40000C00

#define TIMER_6         (void *) 0x40001000
#define TIMER_7         (void *) 0x40001400
#define TIMER_8         (void *) 0x40010400
#define TIMER_9         (void *) 0x40014000
#define TIMER_10  (void *) 0x40014400
#define TIMER_11  (void *) 0x40014800
#define TIMER_12  (void *) 0x40001800
#define TIMER_13  (void *) 0x40001C00
#define TIMER_14  (void *) 0x40002000
```

C – timer_part_2.h

```
#define TIM_CR1               (int) 0x00        // Control Register 1
#define TIM_CR2               (int) 0X04        // Control Register 2
#define TIM_SMCR        (int) 0x08        //
#define TIM_DIER        (int) 0x0C        //
#define TIM_SR                (int) 0x10        //
#define TIM_EGR               (int) 0x14        //
#define TIM_CCMR1       (int) 0x18        //
#define TIM_CCMR2       (int) 0x1C        //
#define TIM_CCER        (int) 0x20        //
#define TIM_CNT               (int) 0x24        //
#define TIM_PSC               (int) 0x28        //
#define TIM_ARR               (int) 0x2C        //
#define TIM_CCR1        (int) 0x34
#define TIM_CCR2        (int) 0x38
#define TIM_CCR3        (int) 0x3C
#define TIM_DCR               (int) 0x48
#define TIM_DMAR        (int) 0x4C

#define TIMER_UP_MODE         1
#define TIMER_DOWN_MODE       2

extern int timer_init(timer_t *timer, unsigned long * base);
extern int timer_enable_nvic(timer_t *timer);
extern int timer_enable_rcc(timer_t *timer);
extern int timer_disable_rcc(timer_t *timer);
extern int timer_enable_interrupt(timer_t *timer);
extern int timer_start(timer_t *timer);
extern int timer_stop(timer_t *timer);
extern int timer_set_autoreload(timer_t *timer, int autoreload_value);
extern uint32_t timer_get_autoreload(timer_t *timer);
extern int timer_set_prescaler(timer_t *timer, int prescaler_value);
extern uint32_t timer_get_prescaler(timer_t *timer);
extern uint32_t timer_get_autoreload(timer_t *timer);
extern int timer_set_counter_mode(timer_t *timer, int counter_mode);
extern int timer_get_count(timer_t *timer);
extern int timer_clear_IRQ(timer_t *timer);
extern int timer_set_ISR(timer_t *timer, void (*isr)(void));

#endif
```

C – timer_part_1.c

```
#include "timer.h"
#include "nvic.h"
#include "vector_table.h"
int timer_enable_nvic(timer_t *timer){
      int position = 0;
      pointer base =  timer->base;
      if(base == TIMER_1){
            position = NVIC_TIMER_1;      // TIMER_1
      } else if(base == TIMER_2){
            position = NVIC_TIMER_2;      // Timer_2
      } else if(base == TIMER_3){
            position = NVIC_TIMER_3;      // Timer_3
      } else if(base == TIMER_4){
            position = NVIC_TIMER_4;      // Timer_4
      } else if(base == TIMER_5){
            position = NVIC_TIMER_5;      // Timer_5
      }
      NVIC_enableIRQ(position);
      return 1;
}
int timer_enable_rcc(timer_t *timer){
      int mask = 0;
      int APB1_register = 0, APB2_register = 0;
      pointer base = timer->base;
      if(base == TIMER_1){
            mask = bit(0);          // Timer 1 - APB2
            APB2_register = 1;
      } else if(base == TIMER_2){
            mask = bit(0);          // Timer 2 - APB1
            APB1_register = 1;
      } else if(base == TIMER_3){
            mask = bit(1);          // Timer 3 - APB1
            APB1_register = 1;
      } else if(base == TIMER_4){
            mask = bit(2);          // Timer 4 - APB1
            APB1_register = 1;
      } else if(base == TIMER_5){
            mask = bit(3);          // Timer 5 - APB1
            APB1_register = 1;
      }

      if(APB1_register){
            RCC_APB1EN |= mask;
      } else if(APB2_register){
            RCC_APB2EN |= mask;
      }

      return 1;

}
```

C – timer_part_2.c

```
int timer_disable_rcc(timer_t *timer){
      int mask = 0;
      int APB1_register = 0, APB2_register = 0;
      pointer base = timer->base;

      // Select Mask and APBx Register
      if(base == TIMER_1){
            mask = bit(0);          // Timer 1 - APB2
            APB2_register = 1;
      } else if(base == TIMER_2){
            mask = bit(0);          // Timer 2 - APB1
            APB1_register = 1;
            serial_write_string("Selezionato il timer 2\n\r");

      } else if(base == TIMER_3){
            mask = bit(1);          // Timer 3 - APB1
            APB1_register = 1;
      } else if(base == TIMER_4){
            mask = bit(2);          // Timer 4 - APB1
            APB1_register = 1;
      } else if(base == TIMER_5){
            mask = bit(3);          // Timer 5 - APB1
            APB1_register = 1;
            serial_write_string("Selezionato il timer 5\n\r");
      }

      // Applicare la maschera in base al registro selezionato
      if(APB1_register){
            RCC_APB1EN &= ~mask;
      } else if(APB2_register){
            RCC_APB2EN &= ~mask;
      }

      //* ( (unsigned long *) (RCC_BASE + RCC_APB1EN) ) |= 0x01;
      return 1;

}

int timer_enable_interrupt(timer_t *timer){
      *((pointer_32_bit) (timer->base + TIM_DIER)) |= 0x01;
}

int timer_start(timer_t *timer){
      *((pointer_32_bit) (timer->base + TIM_CR1)) |= 0x01;
      return 1;
}
```

C – timer_part_3.c

```
int timer_set_autoreload(timer_t *timer, int autoreload_value){
      *((pointer_32_bit) (timer->base + TIM_ARR)) = autoreload_value;
      return 1;
}

uint32_t timer_get_autoreload(timer_t *timer){
      return *((pointer_32_bit) (timer->base + TIM_ARR));
}

int timer_set_prescaler(timer_t *timer, int prescaler_value){
      *((pointer_32_bit) (timer->base + TIM_PSC)) = prescaler_value;
      //16000
      return 1;
}

uint32_t timer_get_prescaler(timer_t *timer){
      return *((pointer_32_bit) (timer->base + TIM_PSC));
}

int timer_set_counter_mode(timer_t *timer, int counter_mode){
      int bit = 0x10;   // Quarto bit

      if(counter_mode == TIMER_UP_MODE){
            // 4 bit a 0
            *((pointer_32_bit) (timer->base + TIM_CR1)) &= ~bit;
      } else if(counter_mode == TIMER_DOWN_MODE){
            // 4 bit a 1
            *((pointer_32_bit) (timer->base + TIM_CR1)) |= bit;
      }
      return 1;
}

int timer_get_count(timer_t *timer){

      return *((pointer_32_bit) (timer->base + TIM_CNT));

}

int timer_clear_IRQ(timer_t *timer){
      *((pointer_32_bit) (timer->base + TIM_SR)) &= ~(0x01);
}

int timer_set_ISR(timer_t *timer, void (*isr)(void)){
      if(timer->base ==  TIMER_2){
            vector_table[44] = (unsigned long *) isr;
      }
      return 1;
}
```

C — utils_part_1.h

```
#ifndef UTILS_H
#define UTILS_H

#include <stdint.h>
#include "serial.h"
typedef void * pointer;
typedef unsigned long * pointer_32_bit;
#define PERIPH_BASE          0x40000000
#define OUTPUT_MODE     (0x10|0x03) // output mode: push-pull + 50MHz

#define GPIOA_BASE      (PERIPH_BASE + 0x20000)
#define GPIOB_BASE     (PERIPH_BASE + 0x20400)
#define GPIOA_MODE     (*(volatile unsigned long*)(GPIOA_BASE + (int)
0x00))
#define GPIOA_TYPE      (*(volatile unsigned long*)(GPIOA_BASE + (int)
0x04))
#define GPIOA_SPEED     (*(volatile unsigned long*)(GPIOA_BASE +
0x08))
#define GPIOA_PUPD      (*(volatile unsigned long*)(GPIOA_BASE +
0x0C))
#define GPIOA_IDR       (*(volatile unsigned long*)(GPIOA_BASE +
0x10))
#define GPIOA_ODR       (*(volatile unsigned long*)(GPIOA_BASE +
0x14))
#define GPIOA_AFRL      (*(volatile unsigned long*)(GPIOA_BASE +
0x20))
#define GPIOA_AFRH      (*(volatile unsigned long*)(GPIOA_BASE +
0x24))

#define GPIOB_MODE      (*(volatile unsigned long*)(GPIOB_BASE + (int)
0x00))
#define GPIOB_TYPE      (*(volatile unsigned long*)(GPIOB_BASE + (int)
0x04))
#define GPIOB_SPEED     (*(volatile unsigned long*)(GPIOB_BASE +
0x08))
#define GPIOB_PUPD      (*(volatile unsigned long*)(GPIOB_BASE +
0x0C))
#define GPIOB_IDR       (*(volatile unsigned long*)(GPIOB_BASE +
0x10))
#define GPIOB_ODR       (*(volatile unsigned long*)(GPIOB_BASE +
0x14))
#define GPIOB_AFRL      (*(volatile unsigned long*)(GPIOB_BASE +
0x20))
#define GPIOB_AFRH      (*(volatile unsigned long*)(GPIOB_BASE +
0x24))
```

C — utils_part_2.h

```
/* RCC - Reset And Clock Control - Per abilitare il clock nelle
periferiche */
#define PERIPH_BASE          0x40000000
#define RCC_BASE        (PERIPH_BASE + 0x23800)
#define RCC_AHB1EN      (*(volatile unsigned long*)(RCC_BASE + 0x30))
#define RCC_APB1EN           (*(volatile unsigned long*)(RCC_BASE +
0x40))
#define RCC_APB2EN           (*(volatile unsigned long*)(RCC_BASE +
0x44))

#define SPI_1           (void*) 0x40013000
#define SPI_2           (void*) 0x40003800
#define SPI_3           (void*) 0x40003C00
#define SPI_4           (void*) 0x40013400
#define SPI_CR1         (int) 0x00

void delay(long ms);
void led_on();
void led_off();
void led_switch();
uint32_t bit(int pos);


/* SPI */
void enable_SPI_1();
void enable_SPI_2();
void enable_SPI_3();
void enable_SPI_4();

/* USART */
void enable_USART_1();
void enable_USART_2();
void enable_USART_3();
void enable_USART_4();
void enable_USART_5();
void enable_USART_6();

/* I2C */
void enable_I2C_1();
void enable_I2C_2();
void enable_I2C_3();

#endif
```

C — utils_part_1.c

```c
#include "utils.h"
#include <stdint.h>
#include "i2c.h"

void delay(long ms){
	while(ms){
		ms--;
	}
}
void led_on(){
	GPIOA_ODR |= 0x20;
}
void led_off(){
	GPIOA_ODR &= ~(0X20);
}
void led_switch(){

	if(GPIOA_ODR & 0x20){
		// LED ON, lo spengo
		GPIOA_ODR &= ~(0X20);  // LED OFF
	} else {
		GPIOA_ODR |= 0x20;  // LED ON
	}

}
uint32_t bit(int pos){
	uint32_t mask = 1;
	if(pos >= 0 && pos < 32){
		mask = mask << pos;
	} else {
		mask = 0;
	}
	return mask;
}

/* SPI */
void enable_SPI_1(){
	/* Enable RCC */
	RCC_APB2EN |= bit(12);

	/* Enable periferica */
	*((uint32_t*)(SPI_1 + SPI_CR1)) |= bit(6);
}
void enable_SPI_2(){
	/* Enable RCC */
	RCC_APB1EN |= bit(14);

	/* Enable periferica */
	*((uint32_t*)(SPI_2 + SPI_CR1)) |= bit(6);
}
```

C — utils_part_2.c

```
void enable_SPI_3(){
      /* Enable RCC */
      RCC_APB1EN |= bit(15);

      /* Enable periferica */
      *((uint32_t*)(SPI_3 + SPI_CR1)) |= bit(6);
}
void enable_SPI_4(){
      /* Enable RCC */
      RCC_APB2EN |= bit(13);

      /* Enable periferica */
      *((uint32_t*)(SPI_4 + SPI_CR1)) |= bit(6);
}


/* USART 1 */
void enable_USART_1(){
      /* Enable RCC */
      RCC_APB2EN |= bit(4);

      /* Enable Periferica */
      *( (uint32_t*)(USART_1 + USART_CR1)) |= bit(13);
}

/* USART 2*/
void enable_USART_2(){
      /* Enable RCC */
      RCC_APB1EN |= bit(17);

      /* Enable Periferica */
      *( (uint32_t*)(USART_2 + USART_CR1)) |= bit(13);
}

/* USART 3 */
void enable_USART_3(){
      /* Enable RCC */
      RCC_APB1EN |= bit(18);

      /* Enable Periferica */
      *( (uint32_t*)(USART_3 + USART_CR1)) |= bit(13);
}

/* USART 4 */
void enable_USART_4(){
      /* Enable RCC */
      RCC_APB1EN |= bit(19);

      /* Enable Periferica */
      *( (uint32_t*)(USART_4 + USART_CR1)) |= bit(13);
}
```

C — utils_part_3.c

```
/* USART 5 */
void enable_USART_5(){
      /* Enable RCC */
      RCC_APB1EN |= bit(20);

      /* Enable Periferica */
      *( (uint32_t*)(USART_5 + USART_CR1)) |= bit(13);
}
/* USART 6 */
void enable_USART_6(){
      /* Enable RCC */
      RCC_APB2EN |= bit(5);

      /* Enable Periferica */
      *( (uint32_t*)(USART_6 + USART_CR1)) |= bit(13);
}
/* I2C 1 */
void enable_I2C_1(){
      // I2C_SCL in PB6 AF4
      // I2C_SDA in PB7 AF4
      GPIOB_MODE |= 0xA000;   // Mode AF in PB6 e PB7
      GPIOB_AFRL |= 0x44000000;

      /* Enable RCC */
      RCC_APB1EN |= bit(21);
      /* Periferica */
      *((uint32_t*)(I2C_1 + I2C_CR1)) |= bit(0);
}
/* I2C 2 */
void enable_I2C_2(){
      // I2C_SCL in PB10 AF4
      // I2C_SDA in PB11 AF4
      GPIOB_MODE |= 0x00A00000;
      GPIOB_AFRH |= 0x00004400;

      /* Enable RCC */
      RCC_APB1EN |= bit(22);

      /* Periferica */
      *((uint32_t*)(I2C_2 + I2C_CR1)) |= bit(0);
}
/* I2C 3 */
void enable_I2C_3(){
      // I2C_SCL in PA8 AF4
      // I2C_SDA in PB4 AF4
      GPIOA_MODE |= 0x00020000;
      GPIOB_MODE |= 0x00000200;
      GPIOA_AFRH |= 0x00000004;
      GPIOB_AFRL |= 0x00040000;
      /* Enable RCC */
      RCC_APB1EN |= bit(23);

      /* Periferica */
      *((uint32_t*)(I2C_3 + I2C_CR1)) |= bit(0);
}
```

C – comunicazione_micro_sistema_host_part_1.c

```
#include <stdio.h>
#include <fcntl.h>            //File control Definitions
#include <termios.h>
#include <unistd.h>
#include <errno.h>
#include <stdint.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <stdint.h>



#define START_CONDITION       (uint32_t) 0xC1A0

#define CURRENT_MIN                 (int) 2
#define CURRENT_MAX                 (int) 200

#define SHUNT_VOLTAGE_MIN     (float) 0.5
#define SHUNT_VOLTAGE_MAX     (float)     20.0

#define BUS_VOLTAGE_MIN       (float) 0.5
#define BUS_VOLTAGE_MAX       (float) 15.0

#define DEVICE_NOT_FOUND            1
#define TYPE_DEVICE_NOT_FOUND 2

struct options{
      uint32_t numero_campioni;
      uint32_t delay;
      char device[32];  // /dev/ttyACMx
      char type_device[16];   // "arduino" oppure "forth"
      char suffix_output[64];
};
typedef struct options options_t;

void init_port(struct termios *serial_port_settings);
int init_options(int argc, char **argv, options_t *opt);
int filter_current(float *values, int size);
int filter_shunt_voltage(float *values, int size);
int filter_bus_voltage(float *values, int size);
float media(float *values, int size);
float deviazione(float *values, int size);
void create_file_samples(char *file_name, float *values, int
number_samples, char *test_name, char *type_samples);
void create_file_report(void);
void show_screen(float *shunt, float *bus, float* current_buffer,
options_t *opt, int current);
void post_process_forth(float * shunt_voltage_buffer, float *
bus_voltage_buffer, float * current_buffer, int number_samples);
void print_results(float *shunt, float *bus, float *current, int
number);
```

C — comunicazione_micro_sistema_host_part_2.c

```
int main(int argc, char **argv){
      int port, i, j, load_bar;
      float percentuale = 0;
      struct termios serial_port_settings;
      int minuti, secondi;

      /* Flag per le varie decisioni */
      int forth = 0;
      int arduino = 0;

      /* File */
      char file_name[64];

      /* Variabili per la comunicazione */
      uint32_t startCondition = 0;
      uint32_t numero_campioni = 0;
      int campioni_validi_shunt = 0;
      int campioni_validi_bus = 0;
      int campioni_validi_current = 0;
      int retval = 0;

      /* Buffer per ricevere i campioni */
      /* Usati per la comunicazione con ARDUINO */
      float *shunt_voltage_buffer;
      float *bus_voltage_buffer;
      float *current_buffer;

      /* Usati per la comunicazione con FORTH */
      int32_t *shunt_voltage_buffer_int;
      int32_t *bus_voltage_buffer_int;
      int32_t *current_buffer_int;

      options_t options;
      char wordMecrisp[32];

      /* INIT PARAMETRI */
      retval = init_options(argc, argv, &options);


      /* CHECK PARAMETRI OBBLIGATORI */
      if(retval == DEVICE_NOT_FOUND){
            printf("Required option -D DEVICE\n");
            return(1);
      }
      if(retval == TYPE_DEVICE_NOT_FOUND){
            printf("Required option -t TYPE_DEVICE\n");
            printf("(TYPE_DEVICE == ['arduino', 'forth'])\n");
            return(1);
      }
      /* CHECK VALIDITA TYPE_DEVICE */
      if(strcmp(options.type_device, "arduino") &&
strcmp(options.type_device, "forth")){
            printf("TYPE_DEVICE non valido. Inserire una valore fra
'arduino' o 'forth' (senza apici)\n");
            return 1;
      }
```

C — comunicazione_micro_sistema_host_part_3.c

```
      if(!strcmp(options.type_device, "forth")){
            forth = 1;
      }
      if(!strcmp(options.type_device, "arduino")){
            arduino = 1;
      }
      printf("DEVICE: %s\n", options.device);

      /* OPEN PORT */
      port = open(options.device, O_RDWR | O_NOCTTY | O_NDELAY);
      if(port < 0){
            // ERROR
            printf("Errore nell'apertura della porta\n");
            return 0;
      }
      fcntl(port, F_SETFL, 0);
      init_port(&serial_port_settings);

      tcflush(port, TCIOFLUSH);
    tcsetattr(port, TCSAFLUSH, &serial_port_settings);     // Flush
and apply config
      tcflush(port, TCIOFLUSH);
      system("clear");
      printf("\n\n");
      printf("\t+---------------------------------------------\n");
      printf("\t|         Parametri Inseriti:\n");
      printf("\t+---------------------------------------------\n");
      printf("\t|            Porta:  %s\n", options.device);
      printf("\t|             Tipo:  %s\n", options.type_device);
      printf("\t|  Numero Campioni:  %d\n", options.numero_campioni);
      printf("\t|   Delay Campioni:  %d\n", options.delay);
      printf("\t+---------------------------------------------\n");
      if(forth){
            sprintf(wordMecrisp, "%d %d GO\n",
options.delay,options.numero_campioni);
            printf("COMANDO: %s\n\n", wordMecrisp);
            printf("> Premere Enter per iniziare...\n");
            getchar();
            write(port, (void*) wordMecrisp, strlen(wordMecrisp));
            sleep(1);
            /* Elimino l'echo proveniente dall'interprete forth */
            tcflush(port, TCIOFLUSH);     // Elimina l'echo
      } else if(arduino){
            /* CASO DI ARDUINO */
            printf("Riavviare l'arduino, attendere qualche secondo e
premere ENTER per iniziare...\n");
            getchar();
            write(port, (void*) &options.numero_campioni,
sizeof(uint32_t));

            /* Invio Delay */
            write(port, (void*) &options.delay, sizeof(uint32_t));
      }
```

C – comunicazione_micro_sistema_host_part_4.c

```
      shunt_voltage_buffer =  malloc(options.numero_campioni *
sizeof(float));
      bus_voltage_buffer =  malloc(options.numero_campioni *
sizeof(float));
      current_buffer =  malloc(options.numero_campioni *
sizeof(float));

      if(!shunt_voltage_buffer || !bus_voltage_buffer ||
!current_buffer){
            printf("Errore nell'allocazione dei Buffer. Exit\n");
            return 0;
      }

      /* Valori Usati per la lettura da Forth, interi */
      int int_shunt = 0;
      int int_bus = 0;
      int int_current = 0;

      tcflush(port, TCIOFLUSH);
      for(i=0; i<options.numero_campioni; ++i){

            /* Leggo Shunt_voltage */
            if(forth){
                  /* Allore leggo int e li copio nei buffer appositi */
                  read(port, &int_shunt, 4);
                  read(port, &int_bus, 4);
                  read(port, &int_current, 4);

                  /* Copio i dati nei buffer float */
                  shunt_voltage_buffer[i] = (float) int_shunt / 1000;
                  bus_voltage_buffer[i] = (float) int_bus / 1000;
                  current_buffer[i] = (float) int_current / 1000;
            } else {
                  retval = read(port, &(shunt_voltage_buffer[i]), 4);
                  retval = read(port, &(bus_voltage_buffer[i]), 4);
                  retval = read(port, &(current_buffer[i]), 4) ;
            }
            if(options.delay < 50){
                  if(i%3 == 0){
                        show_screen(shunt_voltage_buffer,
bus_voltage_buffer, current_buffer, &options, i);
                  }
            } else {
                  show_screen(shunt_voltage_buffer, bus_voltage_buffer,
current_buffer, &options, i);
            }

      if(forth){
            post_process_forth(shunt_voltage_buffer,
bus_voltage_buffer, current_buffer, options.numero_campioni);
      }
      print_results(shunt_voltage_buffer, bus_voltage_buffer,
current_buffer, options.numero_campioni);
```

C – comunicazione_micro_sistema_host_part_5.c

```
      strcpy(file_name, options.suffix_output);
      strcat(file_name, "_shunt_voltage.txt");
      create_file_samples(file_name, shunt_voltage_buffer,
options.numero_campioni, argv[2], "SHUNT_VOLTAGE");

      /* Creo il File Di Bus Voltage */
      strcpy(file_name, options.suffix_output);
      strcat(file_name, "_bus_voltage.txt");
      create_file_samples(file_name, bus_voltage_buffer,
options.numero_campioni, argv[2], "BUS_VOLTAGE");

      /* Creo il File Di Current Voltage */
      strcpy(file_name, options.suffix_output);
      strcat(file_name, "_current.txt");
      create_file_samples(file_name, current_buffer,
options.numero_campioni, argv[2], "CORRENTE");
      close(port);
      free(shunt_voltage_buffer);
      free(bus_voltage_buffer);
      free(current_buffer);
      return 0;
}

void init_port(struct termios *serial_port_settings){
      memset(serial_port_settings, 0, sizeof(struct termios));
      cfsetispeed(serial_port_settings, B115200);
      cfsetospeed(serial_port_settings, B115200);
      serial_port_settings->c_cflag &= ~PARENB; // No Parity Bit
      serial_port_settings->c_cflag &= ~CSTOPB; // Stop Bit = 1
      serial_port_settings->c_cflag &= ~CSIZE;
      serial_port_settings->c_cflag |= CS8;          // 8 Bit length


      serial_port_settings->c_cflag |= CREAD;

      /* Set Flow Control */
      serial_port_settings->c_cflag &= ~CRTSCTS;
      serial_port_settings->c_iflag &= ~(IXON | IXOFF | IXANY);
      /* Set in RAW MODE (not canonical) */
      serial_port_settings->c_lflag &= ~(ICANON | ECHO | ECHOE |
ISIG);
      /* Altre opzioni di INPUT */
      serial_port_settings->c_iflag &= ~IGNPAR; // Ignore parity check
      serial_port_settings->c_iflag |= IGNBRK;       // Ignore break
signal
      serial_port_settings->c_iflag &= ~ INLCR; // Not Map NL to CR
      serial_port_settings->c_iflag &= ~ ICRNL; // Not Map CR to NL
      serial_port_settings->c_iflag &= ~ IUCLC;
      /* Set VMIN (number of char to wait) */
      serial_port_settings->c_cc[VMIN] = sizeof(float);
      serial_port_settings->c_cc[VTIME] = 20;
}
```

C – comunicazione_micro_sistema_host_part_6.c

```
int init_options(int argc, char **argv, options_t *opt){
      int i=0;
      int find_device = 0;
      int find_type = 0;
      /* Init Valori di Default */
      opt->numero_campioni = 1000;
      opt->delay = 700;
      strcpy(opt->suffix_output, "temp.txt");
      for(i=1; i<argc; ++i){
            /* DEVICE */
            if(strcmp(argv[i], "-D") == 0){
                  /* Set Device */
                  if(i+1 < argc){
                        find_device = 1;
                        strcpy(opt->device, argv[i+1]);
                  }
            }
            /* SUFFIX FILE */
            if(strcmp(argv[i], "-o") == 0){
                  /* Set Device */
                  if(i+1 < argc){
                        strcpy(opt->suffix_output, argv[i+1]);
                  }
            }
            /* NUMERO CAMPIONI */
            if(strcmp(argv[i], "-n") == 0){
                  /* Set Device */
                  if(i+1 < argc){
                        opt->numero_campioni = atoi(argv[i+1]);
                  }
            }
            /* DELAY */
            if(strcmp(argv[i], "-d") == 0){
                  /* Set Device */
                  if(i+1 < argc){
                        opt->delay = atoi(argv[i+1]);
                  }
            }
            /* TIPO DEVICE */
            if(strcmp(argv[i], "-t") == 0){
                  /* Set Device */
                  if(i+1 < argc){
                        find_type = 1;
                        strcpy(opt->type_device, argv[i+1]);
                  }
            }
      }
      if(!find_device){
            return DEVICE_NOT_FOUND;
      }
      if(!find_type){
            return TYPE_DEVICE_NOT_FOUND;
      }

      return 0;
}
```

C – comunicazione_micro_sistema_host_part_7.c

```
float media(float *values, int size){
	double acc = 0;
	int i = 0;
	int campioni_validi = 0;

	for(i=0; i<size; ++i){
		if(values[i] != -1){
			acc += values[i];
			campioni_validi++;
		}
	}
	return ( (float) acc/campioni_validi);
}
float deviazione(float *values, int size){
	float media_value = 0;
	double acc = 0;
	int i=0;
	int campioni_validi = 0;

	media_value = media(values, size);

	for(i=0; i<size; ++i){
		if(values[i] != -1){
			acc += (values[i] - media_value) * (values[i] -
media_value); // (x-xi)^2
			campioni_validi++;
		}
	}
	acc = acc / campioni_validi;  //  ( (x-xi)^2 )/N
	return ( (float) sqrt(acc)); //  sqrt( (x-xi)^2) /N )
}
int filter_current(float *values, int size){
	int campioni_validi = 0;
	int i=0;

	for(i=0; i<size; ++i){
		if(values[i] < CURRENT_MIN || values[i] > CURRENT_MAX){
			values[i] = -1;
		} else {
			campioni_validi++;
		}
	}
	return campioni_validi;
}
int filter_shunt_voltage(float *values, int size){
	int campioni_validi = 0;
	int i=0;

	for(i=0; i<size; ++i){
		if(values[i] < SHUNT_VOLTAGE_MIN || values[i] >
SHUNT_VOLTAGE_MAX){
			values[i] = -1;
		} else {
			campioni_validi++;
		}
	}
	return campioni_validi;
}
```

C – comunicazione_micro_sistema_host_part_8.c

```
int filter_bus_voltage(float *values, int size){
      int campioni_validi = 0;
      int i=0;
      for(i=0; i<size; ++i){
            if(values[i] < BUS_VOLTAGE_MIN || values[i] >
BUS_VOLTAGE_MAX){
                  values[i] = -1;
            } else {
                  campioni_validi++;
            }
      }
      return campioni_validi;
}
void create_file_samples(char *file_name, float *values, int
number_samples, char *test_name, char *type_samples){
      FILE *file_ptr;
      int i=0;

      file_ptr = fopen(file_name, "a+");
      fprintf(file_ptr, "Esperimento:   \t%s  [%s] \n", test_name,
type_samples);
      fprintf(file_ptr, "Num. Campioni: \t%d\n", number_samples);
      // Calcolarla una sola volta e riutilizzarla, invece di
ricalcolarla
      fprintf(file_ptr, " Media  Campioni:\t%f\n", media(values,
number_samples));
      fprintf(file_ptr, "Deviaz. Campioni:\t%f\n", deviazione(values,
number_samples));

      for(i=0; i<number_samples; ++i){
            fprintf(file_ptr, "%f\n", values[i]);
      }

      fclose(file_ptr);
}
void show_screen(float *shunt, float *bus, float* current_buffer,
options_t *opt, int current){
      int i=0, j=0;
      int load_bar = 0;
      float percentuale = 0;
      int minuti = 0, secondi = 0;
      system("clear");
      load_bar = (current+1)*60/opt->numero_campioni;
      percentuale = ((current+1)/(float)opt->numero_campioni) * 100;
      printf("\n\n+=================[STATO DI
AVANZAMENTO]==================+\n");
      printf("|");
      for(j=0; j< ceil(load_bar); ++j)    printf("\u2587");
      for(j=0; j<60-load_bar; ++j)        printf(" ");
      printf("|\n");
      printf("+=========================================================
=====+\n");
      minuti = ((opt->numero_campioni-current) * opt->delay) / 60000;
      secondi = (((opt->numero_campioni-current)*opt->delay)/1000) -
minuti * 60;
```

C – comunicazione_micro_sistema_host_part_9.c

```
      printf("|\n");
      printf("|       Campioni Ricevuti: %d/%d\n", current+1, opt-
>numero_campioni);
      printf("|  Stato di completamento: %.2f %%\n", percentuale);
      printf("|           Tempo residuo: %02d:%02d\n", minuti, secondi
);
      printf("|\n");
      printf("|\n");
      printf("|\n");
      printf("+-------------------------------------------------------
-\n");
      printf("|              Ultimo Campione Ricevuto    \n");
      printf("+====================== [ %d/%d ]
========================\n", current+1, opt->numero_campioni);
      printf("|\tSHUNT_VOLTAGE\t= %.6f\n", shunt[current]);
      printf("|\tBUS_VOLTAGE  \t= %.6f\n", bus[current]);
      printf("|\tCURRENT      \t= %.6f\n", current_buffer[current]);

      printf("+=======================================================
=====\n\n\n\n");
}
void post_process_forth(float * shunt_voltage_buffer, float *
bus_voltage_buffer, float * current_buffer, int number_samples){
      int i=0;

      for(i=0; i<number_samples; ++i){
            shunt_voltage_buffer[i] = shunt_voltage_buffer[i] * 1;
            bus_voltage_buffer[i] = bus_voltage_buffer[i] * 1;
            current_buffer[i] = current_buffer[i] * 1;
      }
}
void print_results(float *shunt, float *bus, float *current, int
number){
      int campioni_validi_shunt = 0;
      int campioni_validi_bus = 0;
      int campioni_validi_current = 0;
      int j = 0;
      campioni_validi_shunt = filter_shunt_voltage(shunt, number);
      campioni_validi_bus = filter_bus_voltage(bus, number);
      campioni_validi_current = filter_current(current, number);
      printf("\n\n+==================[STATO DI
AVANZAMENTO]====================+\n");
      printf("|");
      for(j=0; j< 60; ++j)    printf("\u2587"); printf("|\n");
      printf("+=======================================================
=====+\n");
```

C – comunicazione_micro_sistema_host_part_10.c

```
      printf("|\n");
      printf("|       Campioni Ricevuti: %d/%d\n", number, number);
      printf("|  Stato di completamento: 100.00 %%\n");
      printf("|           Tempo residuo: 00:00\n");
      printf("|\n");
      printf("|\n");
      printf("|\n");
      printf("|\n");
      printf("+=========================[ MEDIA
]=============================\n");
      printf("|\tMedia Shunt Voltage   = %.6f\t(%d/%d Campioni
Validi)\n", media(shunt, number), campioni_validi_shunt, number);
      printf("|\tMedia Bus Voltage     = %.6f\t(%d/%d Campioni
Validi)\n", media(bus, number), campioni_validi_bus, number);
      printf("|\tMedia Current         = %.6f\t(%d/%d Campioni
Validi)\n", media(current, number), campioni_validi_current, number);
      printf("+=========================================================
==========\n");
      printf("|\n");
      printf("+=========================[ DEVIAZIONE
]============================\n");
      printf("|\tDeviaz. Shunt Voltage  = %.6f\t(%d/%d Campioni
Validi)\n", deviazione(shunt, number), campioni_validi_shunt, number);
      printf("|\tDeviaz. Bus Voltage    = %.6f\t(%d/%d Campioni
Validi)\n", deviazione(bus, number), campioni_validi_bus, number);
      printf("|\tDeviaz. Current        = %.6f\t(%d/%d Campioni
Validi)\n", deviazione(current, number), campioni_validi_current,
number);
      printf("+=========================================================
===========\n\n\n");

}
```

C — benchmark_1.c

```
 #include "vector_table.h"
#include "timer.h"
#include "nvic.h"
#include "serial.h"
#include "gpio.h"
#include <stdint.h>
#include <float.h>
#include "utils.h"

int main(){
      int d = 4000000;
      RCC_AHB1EN |= 0x01;

      GPIOA_MODE |= 0x400;
    while(1){
            led_off();
            delay(d);
            led_on();
            delay(d);
    }
      return 0;
}
```

C — benchmark_2.c

```
#include "vector_table.h"
#include "timer.h"
#include "nvic.h"
#include "serial.h"
#include "gpio.h"
#include <stdint.h>
#include <float.h>
#include "utils.h"
int main(){
      timer_t t2;
      t2.base = TIMER_2
      RCC_AHB1EN |= 0x01;
      GPIOA_MODE |= 0x400;

      timer_enable_rcc(&t2);
      timer_enable_nvic(&t2);
      timer_set_prescaler(&t2, 16000);
      timer_set_autoreload(&t2, 3000);
      timer_enable_interrupt(&t2);
      timer_start(&t2);

    while(1){
            delay(4000000);

    }


      return 0;
}
```

Bibliografia

[1] Binary Algorithm for Energy Saving in Low Power Wireless Sensor Networks – M. Hodzic, I. Muhic

[2] Energy-Aware Wireless Sensor Network with Ambient Intelligence for Smart LED Lighting System Control - T. P. Huynh, Y. K. Tan, K. J. Tseng

[3] Energy-Aware Wireless Microsensor Networks – V. Raghunathan, C. Schurgers, S. Park, M. B. Srivastava

[4] Energy-Aware Sensor Node Design With Its Application in Wireless Sensor Networks – R. Yan, H. Sun, Y. Qian

[5] SEEP: exploiting symbolic execution for energy-aware programming – T. Hönig, C. Eibel, R. Kapitza, W. Schröder-Preikschat

[6] MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms – S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, R. Han

[7] Power Saving Algorithms for Wireless Sensor Networks on IEEE 802.15.4 – T. R. Park, M. J. Lee

[8] Calvin Constrained - A Framework for IoT Applications in Heterogeneous Environments – A. Mehta, R. Baddour, F. Svensson, H. Gustafsson, E. Elmroth

[9] Systematic Energy Characterization of CMP/SMT Processor Systems via Automated Micro-Benchmarks – R. Bertran, A. Buyuktosunoglu, M. S. Gupta, M. Gonzalez, P. Bose

[10] An evaluation of energy efficient microcontrollers - I. Tsekoura, G. Rebel, P. Glösekötter, M. Berekovic

[11] Realistic Simulation of Energy Consumption in Wireless Sensor Networks – C. Haas, J. Wilke, V. Stöhr

[12] Datasheet INA219 - http://www.ti.com/lit/ds/symlink/ina219.pdf

[13] Datasheet STM32F446RE

[14] The Internet of Things (IoT): Applications, investments, and challenges for enterprises – I. Lee, K. Lee

[15] Economics of Internet of Things (IoT): An Information Market Approach – D. Niyato, X. Lu, P. Wang, D. I. Kim, Z. Han

[16] Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions – J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami

[17] Hub IoT Azure - https://azure.microsoft.com/it-it/services/iot-hub/

[18] I2C Bus, Interface and Protocol - http://i2c.info/i2c-bus-specification