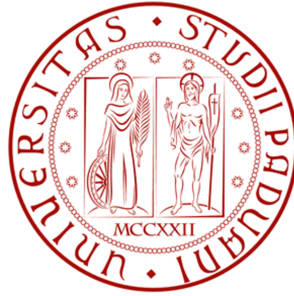


Università degli studi di Padova

Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Magistrale in

Ingegneria delle Telecomunicazioni

Sviluppo e implementazione di uno stack protocollare per smart microgrid

Relatore

Prof. Tomaso Erseghe

Laureando

Paolo Conte

Anno Accademico 2011/2012

Abstract

Nel presente lavoro di tesi è stato progettato e implementato uno stack di protocolli, volto alla realizzazione di una semplice Smart Microgrid.

Ciascuno dei protocolli sviluppati ha lo scopo di risolvere un particolare problema e nel loro insieme costituiscono una *proof of concept* per applicazioni Smart Grid.

Si parte dai livelli inferiori, ovvero fisico e MAC, che permettono di avere una comunicazione affidabile tra più dispositivi, sfruttando lo stesso mezzo fisico impiegato per il trasporto di energia elettrica.

I livelli superiori di registrazione e controllo, invece, forniscono le funzionalità di base per monitorare il funzionamento della rete e controllare le operazioni da svolgere. Inoltre, è stata sviluppata la funzionalità di sincronizzazione del tempo, grazie alla quale i nodi possono coordinarsi per eseguire comandi a precisi istanti.

Per poter osservare dall'esterno quanto accade nella Smart Grid, si può fare uso del software, dotato di interfaccia grafica, realizzato appositamente per il monitoraggio della rete.

Infine, è stato implementato un protocollo per l'aggiornamento remoto del firmware dei nodi, per poter modificare a distanza il software di un dispositivo senza averne fisicamente l'accesso.

Indice

1 Introduzione **1**
1.1 Smart Microgrid . . . 2
1.2 Comunicazioni Powerline . . . 3

2 Architettura del progetto **5**
2.1 Descrizione . . . 5
2.2 Nodi . . . 6
2.3 Hardware . . . 7
2.4 Software . . . 9
2.5 Setup . . . 10

3 Stack **11**
3.1 Introduzione . . . 11
3.2 Struttura dello stack . . . 13
3.3 Implementazione . . . 14

4 Livello fisico **15**
4.1 Librerie PRIME . . . 15
4.2 Livello fisico PRIME . . . 15
4.3 Implementazione . . . 18

5 Livello MAC **21**
5.1 Introduzione . . . 21
5.2 Struttura del protocollo . . . 21
5.3 Implementazione . . . 23

6 Sincronizzazione **25**

6.1 Introduzione 25

6.2 Principio di funzionamento 26

6.3 Algoritmo e implementazione 29

6.4 Risultati sperimentali 32

7 Registrazione **37**

7.1 Introduzione 37

7.2 Struttura del protocollo 37

7.3 Implementazione 38

8 Controllo **41**

8.1 Introduzione 41

8.2 Struttura del protocollo 42

8.3 Base Node 43

8.4 Service Node 43

9 Aggiornamento firmware **45**

9.1 Introduzione 45

9.2 Struttura del firmware 45

9.3 Procedura di aggiornamento 46

9.4 Protocollo 47

9.5 Implementazione 50

9.6 Prestazioni 52

9.7 Possibili sviluppi 53

10 Software PC **55**

10.1 Introduzione 55

10.2 Interfaccia web 56

10.3 Connessione USB 56

10.4 Monitoraggio 57

10.5 Aggiornamento firmware 57

11 Ambiente di sviluppo **59**
11.1 Introduzione 59
11.2 Ambiente CCS 59
11.3 Ambiente Eclipse CDT 63

12 Conclusioni **65**

A Protocollo seriale **69**

B Overflow mode **71**

C Librerie, linker e map file **73**

D Quick start **79**

E Documentazione **85**

Capitolo 1

Introduzione

La rete tradizionale di distribuzione dell'energia elettrica è concepita come un sistema centralizzato, unidirezionale, controllato in base alla quantità di corrente richiesta dalle utenze. Questa architettura, però, ha dei limiti che ne fanno innalzare i costi e ridurre l'affidabilità.

Il concetto di Smart Grid (Figura 1.1) rappresenta l'evoluzione tecnologica della rete elettrica, volta a superare i limiti dell'attuale sistema, sostituendolo con una struttura distribuita, intelligente e dinamica.

Fig. 1.1: Rappresentazione di una Smart Grid.

Una Smart Grid permette ai consumatori di immettere nella rete l'energia prodotta localmente, ad esempio con impianti fotovoltaici, e utilizzarla per

la fornitura di elettricità ad altre utenze. Questa energia può anche essere immagazzinata per rispondere a necessità future. Inoltre, grazie all'impiego di dispositivi intelligenti, la rete può essere costantemente monitorata e possono essere effettuate modifiche in tempo reale sulla stessa.

Insieme a queste importanti funzionalità, tutte le caratteristiche della Smart Grid hanno come obiettivo la riduzione dei costi, l'aumento di efficienza e di sicurezza e anche la riduzione dell'impatto ambientale.

Vediamo alcune delle principali proprietà:

- *Affidabilità*: la rete utilizza tecnologie per rilevare malfunzionamenti e autoripararsi, garantendo la continuità del servizio.
- *Efficienza*: un utilizzo intelligente delle risorse e dei dispositivi allacciati alla rete, coordinati tra loro, permette di aumentare l'efficienza di consumo di energia elettrica.
- *Economicità*: la riduzione delle perdite e le agevolazioni alla produzione locale, portano a una riduzione dei costi, sia per i consumatori che per i fornitori del servizio.
- *Sostenibilità*: la smart grid facilità l'introduzione di fonti rinnovabili, affiancate da sistemi di stoccaggio dell'energia, mentre la rete tradizionale non è adatta a questo tipo di sorgenti distribuite.

1.1 Smart Microgrid

Una Microgrid è una versione su piccola scala del sistema centralizzato di distribuzione dell'energia. La Smart Microgrid ha scopi simili a quelli di un Smart Grid e, allo stesso modo, genera, distribuisce e regola il flusso di energia elettrica, ma agisce soltanto a livello locale. Per questo, costituisce il sistema ideale per l'integrazione delle fonti rinnovabili, dove i singoli cittadini sono coinvolti nella produzione di energia.

La Microgrid può essere connessa direttamente alla griglia principale, oppure può operare in modo autonomo, così da garantire la fornitura di energia anche in caso di malfunzionamento della rete.

Gli elementi che costituiscono la Smart Microgrid sono molteplici, ad esempio:

- Centrali elettriche in grado di soddisfare la richiesta interna ed immettere nella griglia principale l'energia in eccesso.
- Mezzi per l'immagazzinamento dell'energia, utili per sfruttare al meglio le fonti rinnovabili.
- Un'infrastruttura di telecomunicazioni che permetta alle componenti della griglia di scambiarsi informazioni.
- Dispositivi intelligenti in grado di adattarsi ai bisogni della rete.
- Un cuore centrale in grado di gestire la rete in base alle istruzioni fornite.

1.2 Comunicazioni Powerline

Come già accennato, le Smart Grid necessitano di un sistema di telecomunicazioni per coordinare il funzionamento di tutti i dispositivi. Le tecnologie disponibili sono molte, tra cui fibra ottica, wireless e coassiale, ma quelle più interessanti sono certamente le *comunicazioni powerline* (PLC). Queste ultime, infatti, sfruttano come mezzo fisico gli stessi cavi utilizzati per la distribuzione dell'energia elettrica, eliminando il bisogno di realizzare una nuova infrastruttura.

Le PLC sono già largamente impiegate nelle griglie esistenti per lo scambio di dati a bassa velocità, quindi rappresentano una tecnica consolidata e adatta ad essere integrata nelle Smart Grid e Microgrid.

Esistono diversi schemi di modulazione e interi standard di protocolli per comunicazioni powerline, che si distinguono per varie caratteristiche come velocità di trasmissione, banda occupata, robustezza al rumore, distanze raggiungibili e servizi offerti ad alto livello. La scelta della tecnica più adatta per una Smart Grid, deve rispondere a molti requisiti, prima di tutto tecnici, ma anche normativi ed economici. Tra i più promettenti, vi sono gli standard di tipo *narrowband* (banda stretta) *PRIME* e *G3-PLC*, che sono stati oggetto di studio per questo progetto, ma utilizzati solo in parte, come si vedrà più avanti.

Per una panoramica più ampia dei protocolli di comunicazione powerline si rimanda a [14].

PRIME

PRIME (PoweRline Intelligent Metering Evolution) rappresenta una architettura di telecomunicazioni pubblica, aperta e non proprietaria, che supporta funzionalità di monitoraggio automatico e costituisce un punto di partenza per la realizzazione degli edifici e smart grid del futuro.

L'obiettivo di PRIME è di stabilire un insieme di standard internazionali che permettono l'interoperabilità tra diversi dispositivi e la coesistenza di più fornitori di servizi.

PRIME definisce protocolli per i livelli inferiori del modello OSI, progettati per essere efficienti, ma implementatibili a basso costo. Per la trasmissione attraverso la rete elettrica, utilizza la tecnica OFDM nella banda CENELEC-A (3-148KHz), mentre il livello superiore (MAC) si occupa di gestire lo scambio di dati, in modo affidabile, tra i nodi della rete.

Capitolo 2

Architettura del progetto

2.1 Descrizione

Il progetto presentato in questa tesi ha come scopo lo sviluppo del software per i nodi di una Smart Microgrid, dove l'implementazione non è stata realizzata a puro scopo simulativo, ma per una specifica piattaforma hardware. In pratica, è stata creata una piccola rete di dispositivi, connessi tra loro grazie alla comunicazione powerline.

L'obiettivo è di monitorare e coordinare il funzionamento degli apparati e delle risorse collegate alla rete elettrica, grazie alla collaborazione di tutti i nodi.

Il controllo è delegato a un nodo centrale, che può disporre di una connessione ad altre grid o offrire un'interfaccia verso internet; sotto il suo comando, la rete può modificare le proprie caratteristiche in tempo reale, per raggiungere uno scopo, definito dal sistema di controllo.

In Figura 2.1 è rappresentato uno schema di Smart Microgrid, dove ad ogni nodo periferico sono collegati carichi o fonti di energia elettrica.

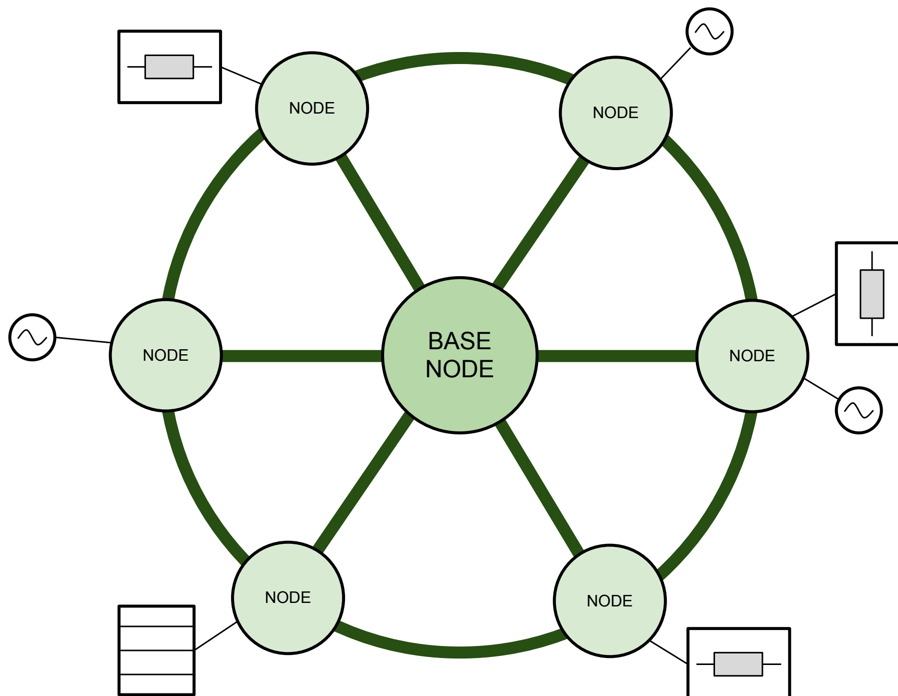


Fig. 2.1: Smart Microgrid composta da un nodo centrale e più nodi periferici, ad ognuno dei quali sono connessi carichi, sorgenti elettriche o altri apparati.

2.2 Nodi

All'interno della rete sono presenti due tipologie di nodi: un singolo Base Node e numerosi Service Node.

Il Base Node è il nodo centrale della rete, che si occupa della gestione di tutti i nodi e dell'eventuale comunicazione con l'ambiente esterno. Inizialmente, la rete è costituita soltanto da questo nodo, poi altri nodi si connettono, espandendo la griglia.

Tutti i restanti nodi sono Service Node e ciascuno di essi fornisce un punto di contatto con vari apparati, che possono essere dei carichi o fonti energetiche. Attraverso opportuni protocolli, il Base Node impartisce comandi ai Service Node e riceve informazioni di varia natura.

2.3 Hardware

Per lo sviluppo del software, è stata realizzata una semplice rete di nodi, costituiti da modem PLC programmabili, isolata dall'esterno, per non interferire alle comunicazioni già presenti nella rete elettrica. Inoltre, si è utilizzato un misuratore multifunzione NEMO, per il monitoraggio di un carico.

Kit di sviluppo

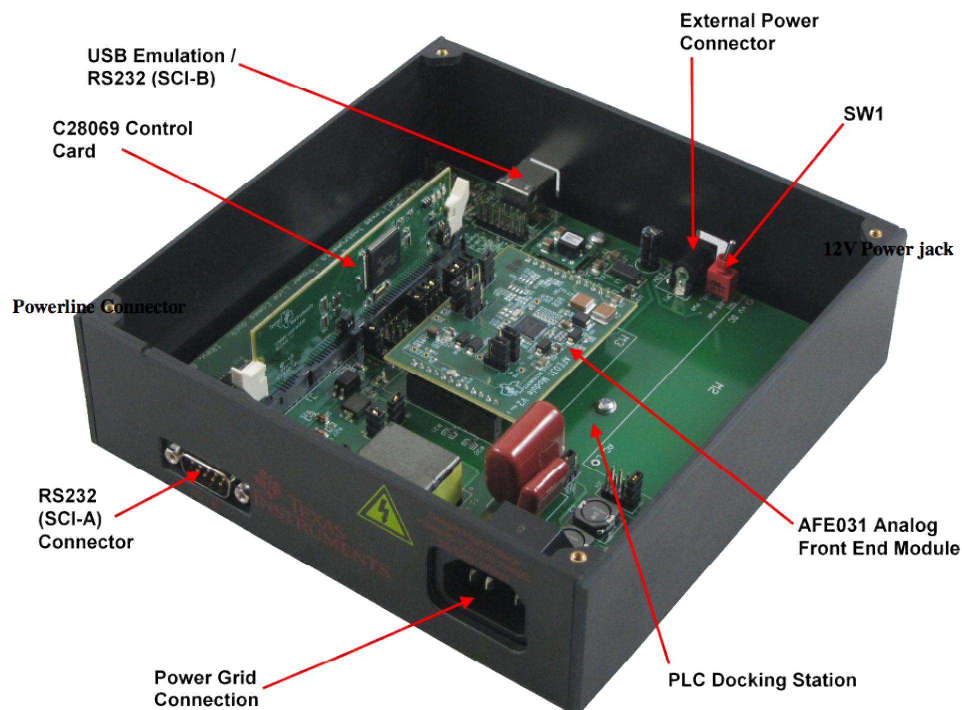


Fig. 2.2: Kit di sviluppo powerline TI MDSPLCKIT-V3.

Texas Instruments offre una varietà di piattaforme di sviluppo per PLC, utili alla sperimentazione e implementazione di software per Smart Grid. Il kit di sviluppo *TMDSPLCKIT-V3*, in Figura 2.2, contiene il necessario per la realizzazione di software per comunicazioni power line. Sono inclusi due modem, basati sul microcontrollore C2000 *TMS320F28069* ed il front-end analogico *AFE031*. Le librerie software fornite a corredo supportano varie tecniche di comunicazione, come PRIME, G3, FlexOFDM, e SFSK. Inoltre, con la

connessione JTAG integrata, si può programmare ed eseguire il debug attraverso la porta USB.

Lo stesso kit è stato utilizzato sia per realizzare il Base Node che i Service Node, che condividono anche gran parte del software.

DSP TMS320F28069

I microcontrollori della famiglia Piccolo™ C28x™, sono caratterizzati da un'architettura a 32 bit, con modulo per il calcolo a virgola fissa o mobile, adatta sia per compiti di elaborazione numerica che di controllo digitale. Il dispositivo include memorie Flash e RAM, per l'archiviazione ed esecuzione del software, oltre a numerose periferiche.

Il modello F28069, presenta le seguenti caratteristiche:

- Frequenza di lavoro fino a 90MHz.
- Operazioni a virgola mobile a singola precisione.
- 256KiB di memoria Flash.
- 100KiB di memoria RAM.
- Tre timer da 32 bit.
- Indirizzamento a parole di 16 bit.
- Tre porte seriali (UART).

La modalità di indirizzamento a 16 bit comporta alcuni problemi in fase di sviluppo, in quanto la più piccola unità di memoria utilizzabile è di due byte, ma in [15] è illustrato com'è possibile usare l'indirizzamento a byte.

NEMO

Il NEMO D4-L+, in Figura 2.3, è un monitor di rete multifunzione per bassa e media tensione. Può effettuare misure di tensione, corrente, potenza attiva, energia reattiva e altro [11]. Le grandezze possono essere visualizzate sul display

Fig. 2.3: IME NEMO D4-L+ usato come misuratore multifunzionale.

LCD, ma è disponibile un'interfaccia con protocollo JBUS/PROFIBUS su RS-485, per il collegamento ad altri dispositivi [10].

Questo strumento è stato impiegato, nel progetto, per eseguire una lettura a distanza della tensione di rete ai capi di un carico.

2.4 Software

Un nodo della microgrid è costituito da un dispositivo programmabile, il cui software implementa gli algoritmi decisionali che consentono alla rete di svolgere la propria funzione. La comunicazione può essere affidata a standard già esistenti, come PRIME, la cui implementazione è resa disponibile da TI. In questo progetto, però, è stato sviluppato uno *stack di protocolli* e la relativa implementazione software, che non corrisponde ad uno standard esistente. Si è sfruttato, però, il livello fisico di PRIME, perché adatto alla sperimentazione su power line e perché non rientra negli obiettivi del progetto la realizzazione di un diverso schema di modulazione.

Nei capitoli che seguono, è illustrata la struttura dello stack e vengono descritti in dettaglio i protocolli in esso implementati, mentre nel capitolo 11 sono presentati gli strumenti necessari allo sviluppo del software.

2.5 Setup

La rete è stata realizzata collegando tra loro i modem attraverso i relativi cavi elettrici e una ciabatta. Tuttavia, l'alimentazione dei nodi non viene fornita dallo stesso cavo impiegato per la comunicazione powerline, ma da un alimentatore secondario, per mantenere l'isolamento della rete.

Uno dei kit è stato scelto come Base Node, quindi ha un collegamento permanente, via USB, al computer di sviluppo, sia per la programmazione e debug, sia per l'esecuzione del software di controllo.

I restanti nodi, invece, fungono da Service Node e la loro programmazione può essere effettuata via USB (utile per il debug) o attraverso la funzione di aggiornamento remoto del firmware.

Infine, ad uno o più nodi, può essere connesso un dispositivo NEMO, grazie alla porta seriale del kit di sviluppo e ad un convertitore RS-232/RS-485. Poi, grazie ai protocolli sviluppati, il Base Node potrà comandare letture a distanza del NEMO e ricevere i risultati ottenuti.

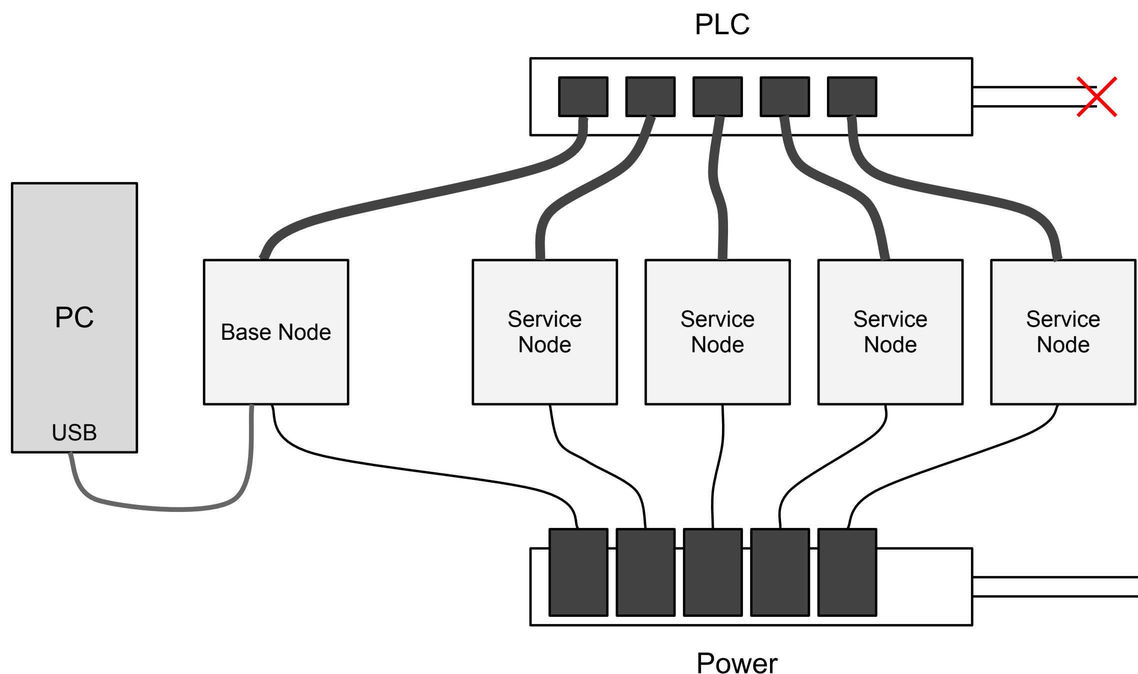


Fig. 2.4: Setup sperimentale utilizzato per lo sviluppo dello stack.

Capitolo 3

Stack

3.1 Introduzione

Le reti di telecomunicazioni, per il loro funzionamento, devono mettere in pratica diversi meccanismi e scambiare vari tipi di informazioni. Per organizzare e ridurre la complessità del sistema, le funzionalità sono suddivise su più livelli, andando a formare una *pila* o *stack* di protocolli. Ciascun livello dello stack sfrutta i servizi del livello inferiore, per offrire ulteriori funzionalità ai livelli superiori; andando dal basso verso l'alto, aumenta sempre più l'astrazione dei dati e l'indipendenza dalla tipologia di rete. Così facendo, il software può essere scritto in modo modulare e flessibile; ad esempio, è possibile cambiare il mezzo fisico andando a sostituire soltanto il primo livello, senza apportare modifiche alle restanti componenti dello stack.

Il *modello ISO/OSI* è uno standard che definisce una struttura di stack formato da sette livelli, riassunti in Figura 3.1, ma spesso, per semplicità, i tre livelli superiori vengono riuniti in un unico livello:

1. *Fisico*: si occupa di trasmettere un flusso di informazione attraverso un mezzo fisico, stabilendo le procedure elettroniche e/o fisiche necessarie.

2. *Collegamento*: permette di trasmettere dati in modo affidabile attraverso il mezzo fisico.

3. *Rete*: separa i livelli superiori dai meccanismi coinvolti nella trasmissione e permette la comunicazione tra due punti della rete anche non direttamente connessi.

4. *Trasporto*: rende disponibile una comunicazione affidabile e trasparente tra due nodi della rete.

5. *Applicazione*: costituisce l'interfaccia tra la rete e le applicazioni, per fornire servizi all'utente o ad altri elementi nella rete.

Fig. 3.1: Modello ISO/OSI per gli stack di protocolli.

I dati partono dal livello superiore e scendono attraverso la pila, fino ad arrivare alla trasmissione sul mezzo fisico; durante la discesa, ciascun livello aggiunge le informazioni necessarie al proprio funzionamento all'interno di una intestazione (o header), come stabilito dal relativo protocollo. Viceversa, un pacchetto che arriva dal livello fisico, risale attraverso lo stack, dove ogni livello estrae il proprio header prima di passare i dati al livello superiore, fino ad arrivare all'applicazione.

3.2 Struttura dello stack

Il modello fornito dallo standard ISO/OSI risulta eccessivamente complesso per il progetto presentato in questa tesi, perciò lo stack, in Figura 3.2, è suddiviso in tre livelli: Fisico, Collegamento e Applicazione.

Lo strato fisico si basa sul protocollo di primo livello dello standard PRIME, la cui implementazione è resa disponibile sottoforma di libreria da Texas Instruments. Al di sopra di questo, è stato sviluppato un protocollo *MAC* (Media Access Control), che si occupa della trasmissione affidabile di pacchetti dal Base Node ai Service Node e viceversa. Infine, sono stati implementati alcuni protocolli di tipo applicazione, che sfruttano il secondo livello per comunicare attraverso la rete:

- *Registrazione*: consente ad un Service Node di segnalare la propria presenza e le proprie funzionalità, e permette al Base Node di conoscere in tempo reale la composizione della rete.

- *Controllo*: è il protocollo con cui il nodo centrale gestisce il funzionamento ad alto livello della rete, come l'invio di comandi o la lettura dello stato di un sistema.

- *Aggiornamento firmware*: fornisce uno strumento utile all'aggiornamento del software che gestisce i nodi della smartgrid.

Vi è un ulteriore protocollo, necessario alla *sincronizzazione* del tempo tra i nodi della rete, che, concettualmente, potrebbe trovarsi nel livello applicazione, ma per le sue caratteristiche implementative, si trova all'interfaccia tra il livello MAC e il livello fisico.

Tutti i protocolli nominati in questo paragrafo, saranno descritti più a fondo nei prossimi capitoli.

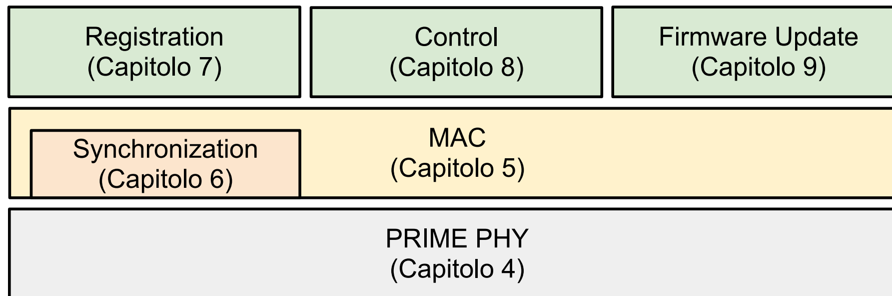


Fig. 3.2: Struttura a livelli del software.

3.3 Implementazione

L'implementazione di ogni protocollo consiste nella scrittura di varie funzioni, che ne consentono l'utilizzo da parte dei livelli superiori, o dal codice utente, e gestiscono i pacchetti in arrivo. Oltre a questo, però, lo stack deve eseguire continuamente altre operazioni, come la gestione dei timeout e delle ritrasmissioni, o la manutenzione di strutture interne; inoltre, l'esecuzione del codice dello stack deve convivere con il software vero e proprio, che determina il funzionamento del nodo. Per questi motivi, il programma deve essere impostato in modo tale da permettere l'esecuzione, virtualmente simultanea, di vari processi.

A tale scopo, si può sfruttare un vero e proprio sistema multitasking preemptive, come SYS/BIOS di Texas Instruments, ma per mantenere una maggiore semplicità, flessibilità e portabilità del software, si è scelto un meccanismo più semplice, ovvero un multitasking cooperativo con scheduler round-robin.

In un tale sistema, tutte le operazioni che vanno eseguite periodicamente costituiscono un task, e il programma principale (*main*) richiama in sequenza, all'infinito, tutti i task presenti, ad esempio quello dello stack e quello dell'utente. Ogni task deve avere una durata contenuta, per non occupare troppo la CPU, quindi è importante non vi siano funzioni che blocchino il flusso del programma. Spesso, i task sono costituiti da *macchine a stati* (*state machine*), ad esempio, per suddividere operazioni sequenziali da eseguire in più passi.

In ogni capitolo saranno presentati maggiori dettagli relativi all'implementazione delle diverse componenti del software.

Capitolo 4

Livello fisico

4.1 Librerie PRIME

A corredo della piattaforma hardware utilizzata in questo progetto, Texas Instruments mette a disposizione diverse librerie pre-compilate, tra cui l'implementazione dello standard PRIME, dal livello fisico a LLC. Tuttavia, per utilizzare lo stack TI nel suo intero, è necessaria la presenza di un Base Node PRIME, che non è stato qui impiegato, inoltre si è preferito implementare un livello MAC su misura. Il protocollo fisico, invece, può essere usato per realizzare una generica comunicazione via powerline, perciò le librerie del livello fisico PRIME sono state utilizzate come punto di partenza del progetto.

4.2 Livello fisico PRIME

L'obiettivo del livello fisico di PRIME è trasmettere dati in modo robusto attraverso la rete elettrica, nella banda CENELEC-A, raggiungendo una velocità di trasmissione fino a 128kbps. Viene impiegata una tecnica OFDM, con vari schemi di modulazione, e opzionalmente un codice FEC, per rendere più affidabile la trasmissione, a discapito del throughput [12].

Modulazioni

Lo standard PRIME è progettato per reagire in modo dinamico alle caratteristiche del mezzo fisico, scegliendo, per ogni pacchetto trasmesso, la modulazione migliore da utilizzare. Perciò, sono disponibili le configurazioni riportate in Tabella 4.1.

	DBPSK		DQPSK		D8PSK	
Convolutional Code (1/2)	On	Off	On	Off	On	Off
Information bits per subcarrier	0.5	1	1	2	1.5	3
Information bits per OFDM symbol	48	96	96	192	144	288
Raw data rate [kbps]	21.4	42.9	42.9	85.7	64.3	128.6

Tabella 4.1: Modulazioni disponibili per il livello fisico PRIME.

In questo progetto di tesi, però, non si sfrutta la gestione automatica della modulazione, implementata nel livello MAC di PRIME, ma si è preferito adottare lo schema più affidabile.

Formato del frame

La comunicazione a livello fisico avviene per *frame*, con caratteristiche adatte alla trasmissione sul mezzo fisico. Un frame è composto da tre parti:

- Preambolo (Preamble): indica l'inizio di un frame ed è fondamentale per la sincronizzazione della portante tra l'apparato di ricezione e quello di trasmissione.
- Intestazione (Header): contiene informazioni sulla struttura del frame, indispensabili per la corretta ricezione.
- Carico utile (Payload): è composto, fondamentalmente, dai dati utili trasportati dal frame.

In Figura 4.1 sono anche indicate le durate temporali delle tre sezioni.

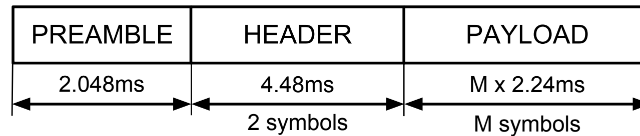


Fig. 4.1: Struttura del frame PHY di PRIME.

La struttura di Header e Payload è rappresentata in Figura 4.2 e i principali elementi, importanti per l'implementazione del livello MAC, sono i seguenti:

- *PROTOCOL*: indica lo schema di trasmissione utilizzato per modulare il payload, mentre l'intestazione è sempre modulata nel modo più robusto (DBPSK + FEC).
- *LEN*: rappresenta la lunghezza del payload in simboli, con un massimo di 63. A seconda della modulazione, cambia la massima lunghezza in bit che può essere inviata.
- *MAC_H*: contiene tutto, o in parte, l'header del livello MAC, in modo che abbia la stessa protezione dagli errori dell'intestazione del livello fisico. Ha una lunghezza di 54 bit, ovvero 7 byte meno 2 bit.
- *CRC*: consente di avere un controllo di integrità sulla sola intestazione.

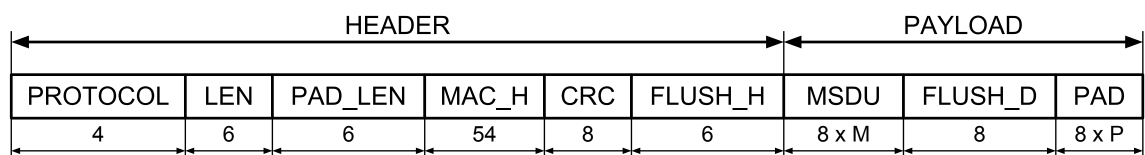


Fig. 4.2: Struttura dei campi Header e Payload di PRIME. Le dimensioni sono espresse in bit.

4.3 Implementazione

Le librerie sviluppate da Texas Instruments mettono a disposizione le funzioni necessarie alla trasmissione e ricezione di pacchetti attraverso la powerline, secondo le specifiche descritte in [16]. Tali funzioni, però, non vengono utilizzate direttamente dal livello MAC, ma è stata realizzata un'ulteriore interfaccia per la gestione dello strato fisico e per semplificarne l'accesso al livello superiore.

Inizializzazione

È stata realizzata un'unica funzione per l'inizializzazione di tutte le componenti del livello fisico, come il modulo AFE, gli interrupt, i buffer di trasmissione e le strutture dati. Inoltre, viene attivata la ricezione dei pacchetti.

Ricezione

La ricezione dei pacchetti è gestita in modo asincrono, cioè non avviene contestualmente alla chiamata di una funzione, ma è il livello fisico che segnala allo stack l'arrivo di un nuovo pacchetto, attraverso una *callback*.[1] Una volta ricevuto il pacchetto, viene effettuata l'operazione di cambio di *endianness*,[2] poi si inoltrano i dati al livello MAC, attraverso un'altra callback.

Trasmissione

Per inviare un pacchetto nella rete, sono messe a disposizione più funzioni, che vanno utilizzate nel giusto ordine; in breve, si controlla che il livello fisico non sia già occupato, poi si scrive il pacchetto nel buffer, infine si inizia la trasmissione. Queste funzioni si occupano di effettuare le operazioni necessarie, come la conversione di endianness, e di configurare le strutture dati del livello fisico per la trasmissione del payload, in modo da rendere indipendente il livello MAC dalle caratteristiche dello strato fisico.

[1] Una callback è una funzione chiamata da un'altra funzione, ad esempio al verificarsi di un evento.

[2] I due byte che formano una parola a 16 bit, nella CPU utilizzata, sono raparesentati in formato big-endian, mentre i dati nel pacchetto vengono trasmessi in formato little-endian

Task

Quanto descritto finora, non è sufficiente per il funzionamento delle librerie PRIME, infatti sono presenti due ulteriori funzioni, una legata alla ricezione e una alla trasmissione. Queste contengono delle macchine a stati che devono essere chiamate periodicamente per gestire le operazioni interne del livello. A questo scopo è stato realizzato un *task*, che viene periodicamente invocato dallo stack, per consentire la corretta esecuzione di queste due funzioni.

Capitolo 5

Livello MAC

5.1 Introduzione

La principale funzionalità di un livello MAC è consentire una comunicazione affidabile tra due entità della rete che sono fisicamente collegate tra loro. Nel caso particolare, l'obiettivo è di permettere al Base Node di trasmettere informazioni a uno o più Service Node e a questi ultimi di inviare dati al nodo centrale; non è previsto, invece, il dialogo diretto tra due Service Node.

Nel capitolo, si descrive il particolare protocollo di secondo livello sviluppato e implementato per questo stack. Le caratteristiche più importanti sono: il meccanismo di indirizzamento, la modalità di accesso al mezzo e la protezione dagli errori.

5.2 Struttura del protocollo

Le funzionalità di un protocollo MAC possono essere molteplici e dipendono fortemente dalle caratteristiche del mezzo fisico. Vi sono, inoltre, varie possibilità di implementazione, secondo specifiche esigenze, che influenzano, ad esempio, l'efficienza della rete e la complessità di realizzazione. Per questo progetto, è stato realizzato un protocollo semplice e leggero, che punta a limitare le collisioni con un opportuno accesso al mezzo.

Indirizzamento

Una caratteristica fondamentale del livello MAC è l'indirizzamento, che permette al nodo centrale di comunicare con uno specifico nodo e di conoscere la provenienza di un pacchetto ricevuto. Ad ogni Service Node della rete è associato un indirizzo univoco che lo identifica, specificato in fase di programmazione. Il Base Node, invece, essendo unico, non ha un indirizzo.

Esiste, poi, un indirizzo di *broadcast*, con il quale si indica un pacchetto destinato a tutti i nodi della rete.

Accesso al mezzo

Un metodo di accesso al mezzo fisico è, semplicemente, quello di trasmettere il pacchetto in modo immediato, senza effettuare nessun controllo; in questo modo, si ha sicuramente il minimo ritardo di consegna, ma è anche probabile che avvenga la collisione con un altra trasmissione, facendo perdere entrambi i pacchetti.

Il protocollo implementato tenta di ridurre la probabilità di collisione adottando una tecnica di *Collision Avoidance* (CA) con *Random Backoff*. In pratica, quando il livello superiore richiede l'invio di un pacchetto, l'operazione non viene subito eseguita, ma si attende un tempo casuale; poi, il dispositivo controlla che il canale trasmissivo sia libero e, in caso affermativo, il pacchetto viene trasmesso, altrimenti si ripete la procedura.

Esistono modalità alternative più efficaci e più complesse, ad esempio l'accesso al mezzo può essere interamente gestito dal Base Node, che coordina i Service Node con opportuni criteri, ma per reti di piccola entità si possono adottare schemi più snelli.

Controllo di integrità

Il livello MAC garantisce ai livelli superiori che i pacchetti arrivino a destinazione corretti o non arrivino affatto, ovvero impedisce che vengano processati dati che contengono errori. Per fare questo, viene aggiunto un controllo di integrità, implementato con un codice *Cyclic Redundancy Check* (CRC), in grado di rivelare un numero elevato di errori.

Pacchetto

La struttura del pacchetto si compone di un'intestazione, dei dati appartenenti al livello superiore e dal codice a ridondanza ciclica, come illustrato in Figura 5.1.

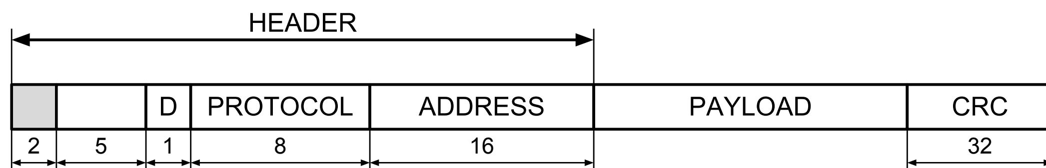


Fig. 5.1: Struttura del pacchetto MAC.

In particolare i campi dell'header sono:

- *Direction*: indica se il pacchetto proviene dal Base Node (D=1), o da un Service Node (D=0).
- *Protocol*: contiene l'identificativo del protocollo di livello superiore associato al payload.
- *Address*: è l'indirizzo del Service Node di destinazione (se D=1) o di provenienza (se D=0).

Vi sono alcuni bit liberi per sviluppi futuri, inoltre i due bit più significativi non possono essere utilizzati, perché non trasmessi sul mezzo fisico.

5.3 Implementazione

Analogamente a quanto visto per il livello fisico, l'implementazione del MAC si suddivide in: inizializzazione, ricezione, trasmissione e task.

Inizializzazione

La funzione di inizializzazione esegue alcune operazioni preliminari e registra la funzione di callback che gli permette di ricevere pacchetti dal livello fisico.

Ricezione

La procedura di ricezione di un pacchetto MAC è composta da alcuni passaggi. Come prima cosa, viene effettuato il controllo di integrità, in modo da scartare pacchetti che contengano errori. Successivamente, nel caso del Service Node, si verifica che il destinatario sia il nodo stesso o che il pacchetto sia di tipo broadcast. Una volta superati questi controlli, viene letto il tipo di protocollo contenuto nel payload e, quindi, si inoltrano i dati al livello superiore attraverso la corrispondente funzione.

Trasmissione

Prima di poter inviare dati attraverso il livello MAC, bisogna verificare che non sia già in corso una trasmissione, poi si può procedere alla costruzione del pacchetto.

Si inizia creando l'intestazione, per la quale vanno indicati l'indirizzo di destinazione (solo nel Base Node) e l'identificativo del protocollo di livello superiore. Poi, si scrive nel buffer di uscita il payload e si conclude chiamando la funzione con cui parte la procedura di trasmissione.

Negli istanti successivi, il task del livello MAC si occuperà di portare a termine l'operazione.

Task

La funzione principale svolta dal task del livello MAC è la gestione dell'accesso al mezzo fisico. In particolare, si occupa di controllare la scadenza del tempo di backoff e verificare che il canale sia libero. Quando si verificano entrambe queste condizioni, il pacchetto viene effettivamente trasmesso dal livello fisico sul canale.

Capitolo 6

Sincronizzazione

6.1 Introduzione

Nella smart microgrid, il Base Node dirige le operazione dei vari nodi, impartendo ordini ed elaborando misure, ed è importante che ciascun evento abbia un riferimento temporale; ad esempio, potrebbe essere necessario effettuare una misura nell'intera rete, simultaneamente in tutti i nodi, ad un preciso istante di tempo. Ciascun dispositivo, però, ha un proprio orologio interno (*clock*), indipendente da tutti gli altri, e quindi serve un meccanismo che permetta ai nodi di coordinarsi, in modo da avere un riferimento di tempo comune.

A tale scopo è stato sviluppato un algoritmo che consente ad ogni nodo di sincronizzarsi con il Base Node, di modo che tutti i nodi della rete abbiamo lo stesso tempo del nodo centrale.

In questa tesi si riprende il lavoro svolto in [9], passando da un contesto con un nodo Master e un nodo Slave, a una rete con molteplici dispositivi.

6.2 Principio di funzionamento

Vediamo, in questo paragrafo, i principali elementi che caratterizzano l'implementazione della sincronizzazione dei nodi.

Tempo locale e tempo remoto

Il tempo *remoto* si riferisce al clock del Base Node e, nel progetto, viene anche chiamato tempo *globale*, in quanto è quello imposto all'intera rete. Le azioni coordinate nella rete avvengono tutte rispetto al tempo globale.

Con tempo *locale*, invece, definiamo il tempo derivante direttamente dal clock del dispositivo. Questo viene utilizzato per gestire le temporizzazioni interne, come ad esempio i timeout di ricezione. Inoltre, tramite la sincronizzazione, dal tempo locale si potrà risalire al tempo globale.

Offset di tempo

Un generico nodo, per mantenere la sincronizzazione con il Base Node, utilizza come punto di riferimento l'istante di ricezione di un pacchetto. Infatti, conoscendo il tempo di trasmissione t_0 del pacchetto, il Service Node può ricavare il tempo t_1 al momento della ricezione, semplicemente sommando a t_0 il tempo di volo t_v, necessario al pacchetto per transitare nella rete; in breve $t_1 = t_0 + t_v$. Poi, il nodo può stimare il valore del tempo globale in qualunque istante, mettendo in relazione il tempo locale con quello del Base Node. Se chiamiamo x il clock di un Service Node e y il clock del Base Node, possiamo scrivere:

$$y = x + b$$

dove b è detto *offset* temporale. Ad esempio, se all'istante di ricezione $y = t_1$ il clock del nodo ha valore $x = T_1$, allora l'offset è $b = t_1 - T_1$, quindi, una volta ottenuto l'offset, è possibile ricavare il tempo del nodo centrale a partire dal tempo del Service Node.

Prendendo direttamente come riferimento x il tempo di trasmissione, l'espressione diventa:

$$y = x + t_v + b \, .$$

Offset di frequenza

Nelle precedenti formule, si suppone che i clock di tutti i dispositivi operino alla stessa frequenza, tuttavia, il tempo, in un microprocessore, viene scandito da un oscillatore al quarzo, che può deviare dalla propria frequenza nominale di alcune parti per milione. Per questo motivo, occorre compensare il clock anche con un *offset di frequenza*, denominato a, come segue:

$$y = ax + b + t_v \, .$$

Questo coefficiente varia per ogni dispositivo, ma può anche variare nel tempo, a causa di condizioni esterne, come la temperatura; perciò, l'algoritmo deve essere in grado di stimare il suo valore e aggiornarlo periodicamente.

RTT

Resta da vedere come il Service Node può calcolare il tempo t_v, perciò introduciamo il concetto di *Round Trip Time* (abbreviato RTT).

Per RTT si intende il tempo necessario a uno scambio completo di pacchetti tra due nodi, ovvero la trasmissione di un primo pacchetto dal nodo A al nodo B, e la ricezione di un pacchetto di risposta nel senso inverso.

L'operazione, denominata *Ping*, è caratterizzata da quattro istanti temporali, come rappresentato in Figura 6.1, che coincidono con i tempi di trasmissione e ricezione dei pacchetti ai due nodi:

- Al tempo t_0 in nodo A trasmette il pacchetto.
- Il nodo B riceve il pacchetto all'istante T_1.
- Dopo un periodo di elaborazione, il nodo B invia una risposta al tempo T_2.
- Il pacchetto giunge al nodo A al tempo t_3.

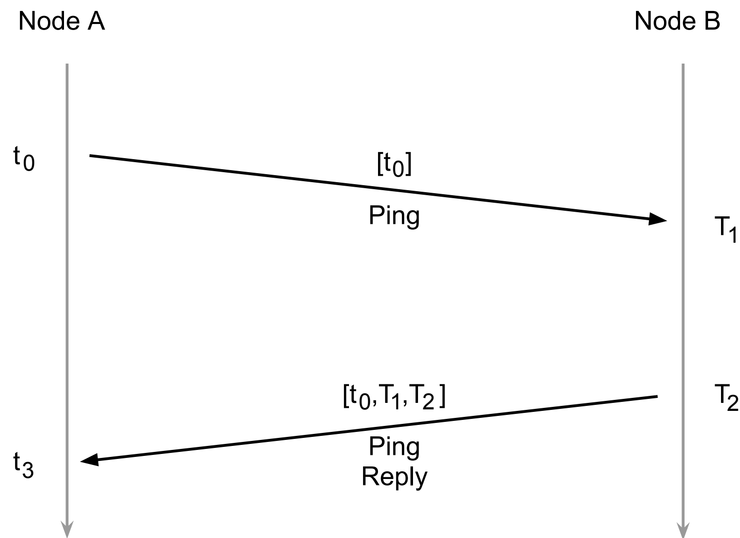


Fig. 6.1: Rappresentazione del protocollo Ping. Consiste nello scambio di due pacchetti, che contengono i tempi necessari per il calcolo del RTT.

Si può dunque definire

$$RTT = (t_3 - t_0) - (T_2 - T_1)$$

ovvero il tempo impiegato dal pacchetto per compiere andata e ritorno, meno il tempo di elaborazione del nodo B. Quindi, il tempo necessario per transitare da un nodo all'altro risulta

$$t_v = \frac{RTT}{2}$$

In conclusione, con un semplice scambio di pacchetti, è possibile avere una stima del tempo di volo, e si ottiene la relazione:

$$y = ax + b + RTT/2\,. \tag{6.1}$$

Tuttavia, il RTT, per uno specifico nodo, non ha un valore assoluto, ma è soggetto a piccole variazioni, e ciò comporta errori nella valutazione dei tempi. Per ovviare a questo problema, l'algoritmo dovrà anche compensare le variazioni del tempo di volo, per ogni pacchetto ricevuto.

6.3 Algoritmo e implementazione

La prima informazione di cui ha bisogno il Service Node, per attuare la sincronizzazione, è il tempo in cui il Base Node ha inviato un pacchetto. Per questo, nell'intestazione viene inserito il *timestamp*, ovvero l'indicazione del tempo a cui è stata effettuata l'operazione di trasmissione.

In fase di ricezione, si può quindi procedere al calcolo degli offset di tempo e di frequenza. Il valore del RTT, però, non si può ottenere da un singolo pacchetto, perché, come visto, è necessario uno scambio di informazioni, con l'operazione di ping. Per questo motivo, il nodo si occupa soltanto di stimare, separatamente, il valore medio del RTT, mentre le variazioni intorno a tale valore saranno automaticamente compensate dal calcolo dell'offset di tempo.

Stima del RTT

Il Service Node invia regolarmente un ping verso il Base Node; quando giunge una risposta, che contiene i quattro istanti temporali, viene aggiornato il valore medio del RTT come segue:

$$
\begin{aligned}
RTT_i &= (t_3 - t_0) - (T_2 - T_1) \\
\overline{RTT} &= \alpha \overline{RTT} + (1 - \alpha) RTT_i
\end{aligned}
$$

L'aggiornamento di questo valore avviene poco di frequente, per non occupare eccessiva banda, ma la media si stabilizza in pochi passi.

Offset di tempo e di frequenza

Per ogni pacchetto ricevuto, il Service Node ottiene una coppia di tempo remoto e corrispondete tempo locale. Una sequenza di questi valori, per le relazioni viste, delinea la retta

$$y = ax + b + \overline{RTT}/2$$

e tramite un algoritmo di regressione lineare, si possono dunque ricavare i valori dei coefficienti a e b. Le variazioni del RTT e della frequenza di clock, causano il discostamento dei punti dalla retta, ma ad ogni esecuzione dell'algoritmo, i cambiamenti vengono corretti.

In particolare, l'agoritmo utilizzato è il seguente:

$$\begin{aligned}
m_x &= \alpha m_x + (1-\alpha)x \\
m_y &= \alpha m_y + (1-\alpha)y \\
S_{xx} &= \alpha S_{xx} + (1-\alpha)(x-m_x)^2 \\
S_{xy} &= \alpha S_{xy} + (1-\alpha)(x-m_x)(y-m_y) \\
a &= S_{xy}/S_{xx} \\
b &= m_y - a m_x
\end{aligned}$$

Per realizzare l'implementazione, però, le precedenti equazioni vanno modificate, per evitare l'overflow e la divergenza delle variabili, che in un qualsiasi dispositivo digitale hanno dei limiti ben precisi. In particolare, anziché effettuare calcoli su valori assoluti, per il tempo locale e globale, si utilizzano valori relativi a quelli della precedente iterazione, memorizzati nelle variabili T_x e T_y. Nell'Algoritmo 1 sono elencati i passi che costituiscono l'implementazione realizzata.

Infine, ricordando l'equazione (6.1), si può ottenere il tempo globale in qualunque istante, a partire dal tempo locale, con la seguente relazione:

$$T_{\text{globale}} = T_y + a\left(T_{\text{locale}} - T_x\right) + b + RTT/2$$

Algoritmo 1 Sincronizzazione

1: $l = \text{localTime} - T_x$
2: $r = \text{remoteTime} - T_y$

3: $m_x = \alpha m_x + (1 - \alpha)l$
4: $m_y = \alpha m_y + (1 - \alpha)r$
5: $S_{xx} = \alpha S_{xx} + (1 - \alpha)(l - m_x)^2$
6: $S_{xy} = \alpha S_{xy} + (1 - \alpha)(l - m_x)(r - m_y)$

7: $m_x = m_x - l$
8: $m_y = m_y - r$

9: $a = S_{xy}/S_{xx}$
10: $b = m_y - am_x$

11: $T_x = \text{localTime}$
12: $T_y = \text{remoteTime}$

Coefficiente α

Il parametro α, visto in molte delle equazioni, è un coefficiente poco inferiore a 1, che permette di effettuare una media nel tempo delle varie grandezze. Il suo valore influenza la velocità di convergenza e la stabilità dell'algoritmo, ma, purtroppo, una migliore velocità corrisponde ad una stabilità inferiore. Un buon compromesso, quindi, consiste nell'utilizzare due diversi valori; in un primo momento, si punta ad una maggiore velocità di convergenza, utilizzando un valore di basso, ad esempio $\alpha = 0.8$, successivamente, quando i coefficienti hanno raggiunto valori prossimi a quelli corretti, si sceglie un valore che porta a maggiore stabilità, in questo caso si è utilizzato $\alpha = 0.96$.

Nell'implementazione realizzata, si eseguono i primi 100 passi con il primo coefficiente, poi si mantiene il secondo.

6.4 Risultati sperimentali

Una volta realizzata l'implementazione, si vuole verificare che l'algoritmo funzioni correttamente e con quale margine di errore. Perciò, sono state effettuate delle misure che mostrano quanto bene l'algoritmo riesca a stimare il tempo di volo t_v.

Descrizione della misura

L'esperimento consiste nell'invio di una serie di pacchetti ping dal Base Node ad un nodo target, ma il protocollo è stato modificato per includere, oltre ai quattro tempi locali (t_0, T_1, T_2, t_3), anche t_1, il tempo globale di ricezione al Service Node, ottenuto grazie al processo di sincronizzazione. In questo modo, dalla relazione $t_v = RTT/2 = t_1 - t_0$, si può ricavare il valore reale del tempo di volo, a partire dal RTT, e quello stimato dall'algoritmo, dai tempi t_0 e t_1. L'errore che si ottiene nel calcolo di t_v, ovvero $e = RTT/2 - (t_1 - t_0)$, è, sostanzialmente, l'errore di sincronizzazione a breve termine, cioè senza tener conto del contributo dell'offset di frequenza.

Risultati

In questo progetto, il RTT viene calcolato a partire da precisi istanti, però non rappresenta il semplice transito del pacchetto nel mezzo fisico, ma comprende anche parte del tempo di trasmissione e ricezione del pacchetto, perciò il suo valore è maggiore di quanto ci si potrebbe aspettare. In una configurazione back-to-back, il RTT è di circa 10 ms.

Come già osservato, il punto di riferimento di questo esperimento sono i tempi di volo reale e stimato: in Figura 6.2 si può osservare un esempio di andamento dei due tempi, e si nota come il valore stimato insegua quello reale.

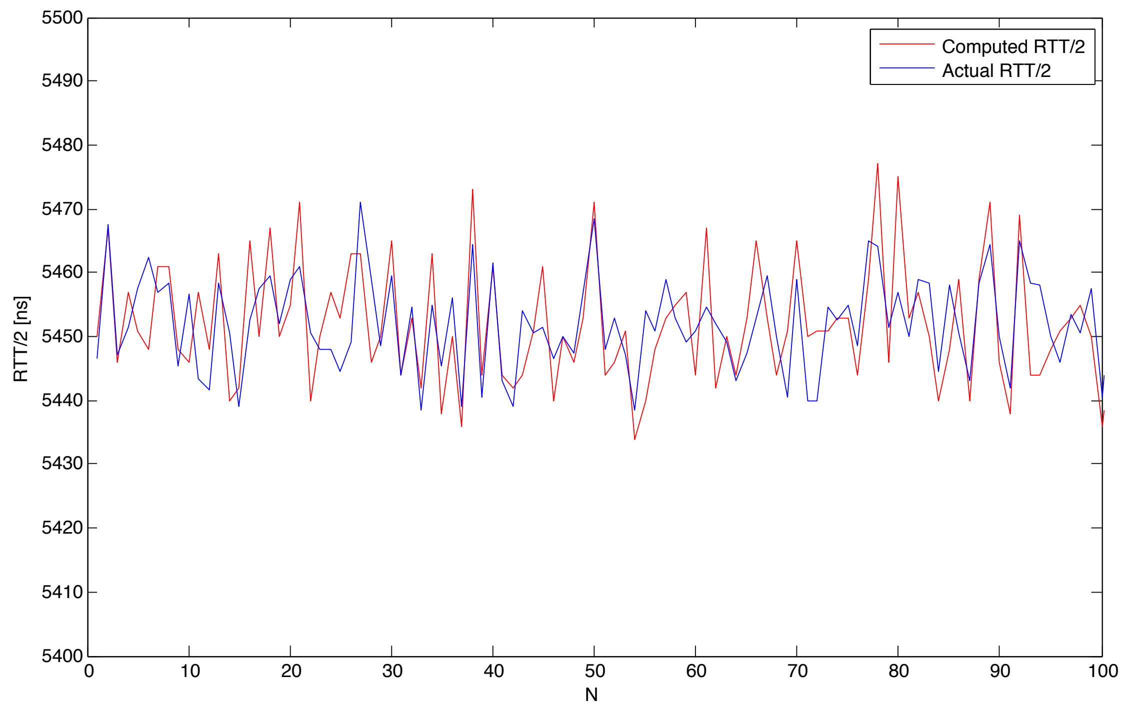


Fig. 6.2: Un esempio visivo di come l'algoritmo segua l'andamento del RTT. In blu è rappresentato $RTT/2$ reale, mentre in rosso quello stimato.

L'errore di stima è rappresentato in Figura 6.3 per una realizzazione di 3600 campioni, che corrisponde ad un'ora di funzionamento; le caratteristiche principali dell'errore sono riportate in Tabella 6.1.

Misura	Valore	Osservazioni
Media	$0\,ns$	L'errore medio è nullo
Deviaz.	$8.9\,ns$	La variazione dell'errore è nell'ordine dei 10ns
MSE	$79\,ns^2$	Coerente con i precendenti valori

Tabella 6.1: Risultati sperimentali sull'errore di sincronizzazione.

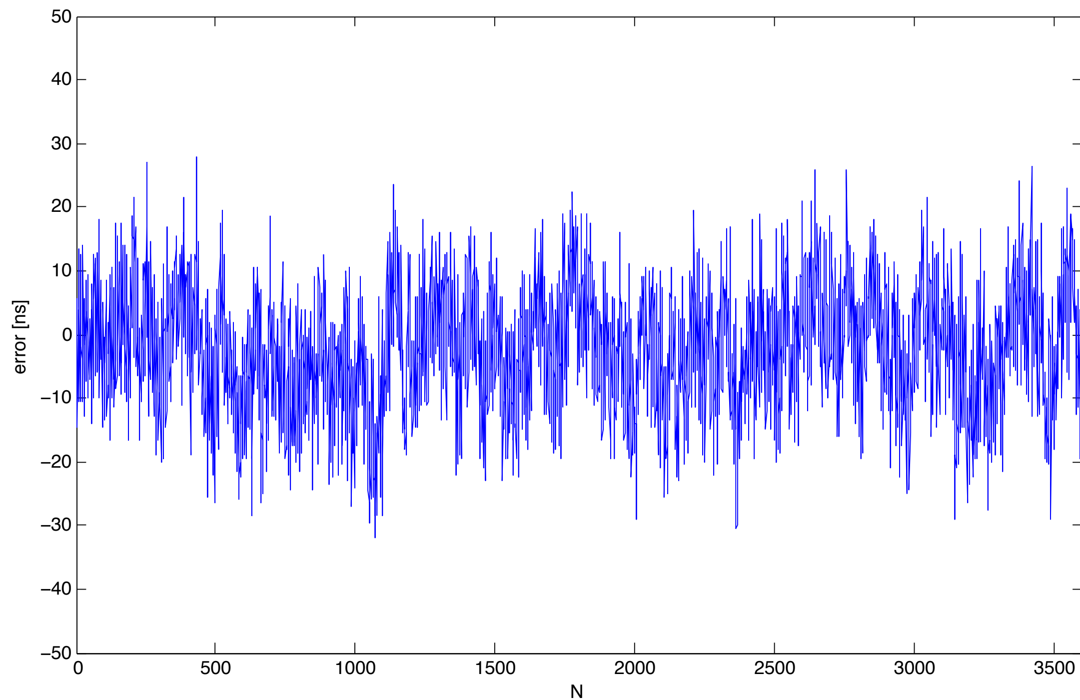


Fig. 6.3: Andamento dell'errore di stima di $RTT/2$. Si può notare la media nulla e una deviazione di circa 10 ns.

Osservazioni

È interessante osservare la relazione tra i tempo locale ed il tempo globale in un nodo. Un esempio è rappresentato in Figura 6.4, dove si può vedere la retta $y = ax + b$, con la particolarità della presenza dell'overflow di entrambi i clock. In ordine temporale, il primo tratto è quello sulla destra, che procede finché non avviene l'overflow del timer locale; segue il tratto sulla sinistra e, dopo l'overflow del tempo globale, si passa al tratto più in basso. Questo fenomeno, tuttavia, non influisce sul funzionamento dell'algoritmo.

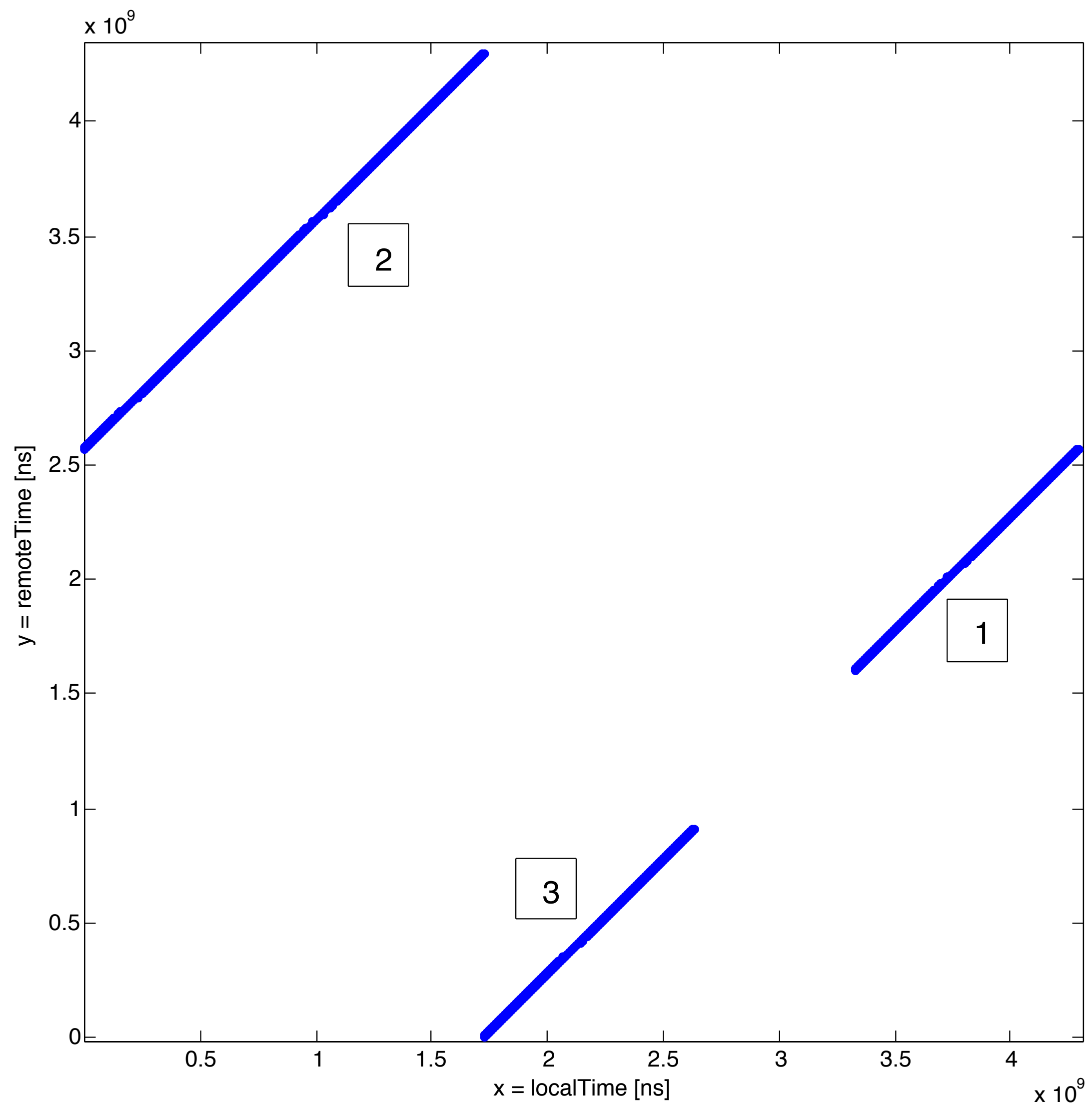


Fig. 6.4: Relazione tra il tempo locale e tempo remoto, descritta dalla retta $y = ax + b$. In questo esempio si può notare l'overflow di entrambi i timer.

Capitolo 7

Registrazione

7.1 Introduzione

La Smart Grid può avere una struttura dinamica, ovvero i nodi che la costituiscono possono essere aggiunti o rimossi in qualsiasi momento, e ognuno di essi può fornire servizi diversi alla rete. In questo caso, è necessario realizzare un sistema che permetta al nodo centrale di conoscere quali e quanti siano i nodi attivi nella rete, e le relative caratteristiche.

Con il protocollo di registrazione, ogni Service Node è in grado di segnalare la propria presenza e le sue funzionalità, mentre il Base Node può conoscere la struttura della rete in ogni momento.

7.2 Struttura del protocollo

Il protocollo si compone di due tipi di pacchetti: uno di registrazione, che va dal Service Node al Base Node, e uno di risposta, che transita nel senso inverso.

Ogni nodo, all'accensione, manifesta la sua presenza nella rete inviando un pacchetto di registrazione, e finché non riconosce una conferma da parte del Base Node, non effettuerà altre operazioni; la procedura è riassunta in modo grafico in Figura 7.1. Nel pacchetto, inoltre, vengono inserite le funzionalità, che verranno usate dal nodo centrale per impartire ordini adeguati alla tipologia di nodo. Ad esempio, il nodo può essere connesso ad una sorgente di energia elettrica, oppure

può monitorare un carico attraverso un misuratore di potenza, o fare entrambe le cose.

Infine, se un nodo rimane inattivo per un lungo tempo, il Base Node supporrà che esso non è più presente nella rete.

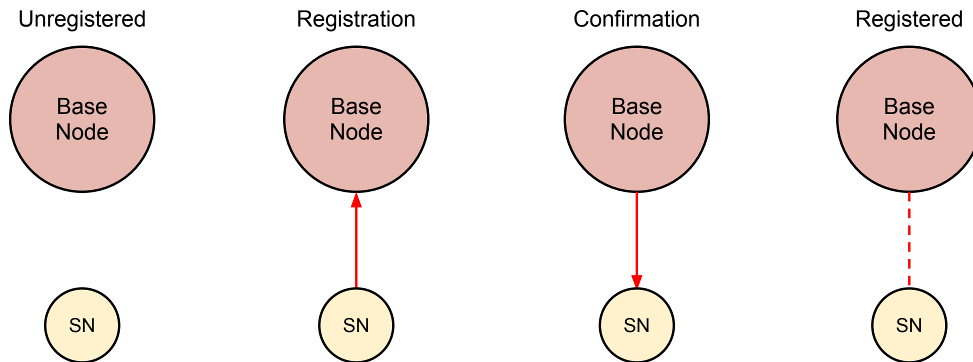


Fig. 7.1: Schema della procedura di registrazione

7.3 Implementazione

L'implementazione si distingue nettamente tra Base Node e Service Node, in quanto le funzioni svolte dal primo sono complementari a quelle di un generico nodo.

Base Node

Il nodo centrale deve mantenere in memoria un elenco di nodi attivi, con tutte le informazioni necessarie per la loro gestione. Per questo, è implementata una tabella, rappresentata in Figura 7.2, dove ciascuna riga contiene i seguenti campi:

- *Address*: è l'indirizzo del nodo associato alla riga della tabella. Se la cella è libera, allora Address vale *ADDRESS_NONE = 0*.

- *Features*: contiene le funzionalità offerte dal nodo.

- *Time*: rappresenta il tempo in cui si è verificata l'ultima attività del nodo.

- *Data*: sono presenti altri campi che dipendono dalla tipologia del nodo e utilizzati dall'algoritmo di controllo della rete.

Per mantenere sempre aggiornate queste informazioni, il Base Node svolge alcune operazioni. Innanzitutto, gestisce le richieste di registrazione, ovvero individua una riga libera della tabella e vi salva le informazioni del nodo, e successivamente invia un pacchetto di conferma. Inoltre, ad ogni pacchetto ricevuto da un nodo, di qualunque tipo, aggiorna il campo *time* corrispondente, per segnalare che il nodo è ancora attivo; infatti, periodicamente, il software controlla tutte le righe della tabella e libera quelle corrispondenti a nodi non più attivi da un certo periodo di tempo.

Fig. 7.2: Struttura della tabella dei nodi registrati nella rete.

Service Node

L'implementazione della procedura di registrazione, per quanto riguarda il Service Node, è piuttosto semplice: il nodo invia un pacchetto con le proprie features e attende una risposta; se quest'ultima non viene ricevuta dopo un tempo limite, il nodo ritenta la registrazione. Finché la procedura non è completa, non sono ammesse altre trasmissioni dal nodo verso l'esterno.

Capitolo 8

Controllo

8.1 Introduzione

La più importante funzione del Base Node è coordinare le operazioni eseguite da tutti i nodi della Microgrid. Ciò consiste, ad esempio, nel comandare la lettura dello stato dei carichi o nell'azionare un dispositivo oppure nell'attivare una diversa sorgente di energia.

Per fare questo, implementa una logica di controllo che definisce il comportamento della griglia e un protocollo di alto livello, per lo scambio di informazioni con in Service Node. Inoltre, grazie alla sincronizzazione, viene usato il *tempo globale* della rete per stabilire precisi istanti temporali per l'esecuzione dei comandi o per associare eventi al tempo in cui sono avvenuti.

In questo capitolo, si vedrà il più semplice esempio di implementazione, come dimostrazione delle varie funzionalità offerte dallo stack. In particolare, lo scopo è di rilevare la presenza dei monitor NEMO connessi ai nodi ed effettuare delle letture remote.

8.2 Struttura del protocollo

Il protocollo sviluppato è costituito da due tipi di pacchetto: uno trasmesso dal Base Node verso i Service Node e uno che transita nel senso inverso.

Il pacchetto del primo tipo, rappresentato in Figura 8.1, contiene il comando da eseguire e l'indicazione del tempo in cui l'operazione va compiuta; ulteriori parametri possono essere aggiunti, se necessario. Inoltre, a seconda del tipo di comando, il pacchetto può essere trasmesso in modalità *broadcast* piuttosto che indirizzarlo ad un singolo nodo.

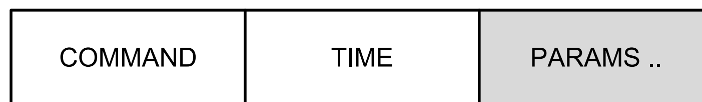


Fig. 8.1: Pacchetto di controllo da Base Node a Service Node.

I comandi usati in questo progetto sono soltanto due:

- *CMD_READ* (broadcast): indica ai nodi di effettuare una misura della rete elettrica in un determinato istante.

- *CMD_GET* (unicast): per ottenere da un Service Node gli ultimi dati rilevati.

Ai comandi di tipo *unicast* può essere associata una risposta inviata dal Service Node al Base Node. In questo caso, viene trasmesso un pacchetto che contiene i dati richiesti, come quello in Figura 8.2. Per il comando *READ*, l'informazione fornita è la tensione misurata nell'istante di tempo definito dal Base Node.

Fig. 8.2: Esempio di pacchetto di risposta da Service Node a Base Node.

8.3 Base Node

L'implementazione dello schema di controllo nel Base Node si compone principalmente di due parti:

- Un task con una macchina a stati che si occupa di trasmettere i comandi.
- Una funzione che processa i pacchetti in ingresso relativi al protocollo di controllo.

Il task è progettato in modo da inviare periodicamente un comando di tipo *READ*, in modalità broadcast; quindi, tutti i nodi che possiedono la capacità di eseguire il comando effettueranno una lettura del monitor NEMO al tempo specificato.

Successivamente, il task si occupa di reperire le misure, perciò invia a ciascun nodo un comando *GET* e attende la risposta (o scade il timeout). La funzione di ricezione elabora ogni pacchetto di risposta, poi salva i dati utili all'interno della tabella dei nodi, sfruttando un metodo messo a disposizione dal modulo di registrazione.

Una caratteristica della registrazione è quella che permette di conoscere le funzionalità di ciascun nodo, come la connessione ad un monitor di rete o ad altri dispositivi, perciò il Base Node può inviare comandi soltanto ai nodi che effettivamente li implementano, risparmiando tempo e risorse.

8.4 Service Node

Nel firmware non è stabilito a priori se al Service Node è collegato un monitor NEMO o meno, perciò, nella funzione di inizializzazione del modulo di controllo, si verifica la presenza del dispositivo attraverso la porta seriale del kit. In caso affermativo, il software aggiunge tra le sue funzionalità quella relativa alla lettura del NEMO e ciò viene fatto prima di iniziare la procedura di registrazione, dove si segnalano al Base Node le caratteristiche del nodo.

Analogamente a quanto visto prima, l'implementazione del protocollo di controllo prevede due parti:

- Una funzione di ricezione che elabora i comandi impartiti.
- Un task che esegue le operazioni programmate.

In particolare, alla ricezione di un pacchetto *READ*, la funzione segnala al task la presenza di una nuova operazione di lettura; nel momento in cui viene ricevuto un comando *GET*, invece, il nodo risponde con gli ultimi valori letti, indicando anche il tempo a cui si riferiscono.

Il task attende che vi sia un'operazione programmata e che giunga il tempo indicato per eseguirla. Nel caso specifico della lettura del NEMO, il valore di tensione viene letto e poi salvato, in attesa dell'arrivo di un comando *GET*.

Capitolo 9

Aggiornamento firmware

9.1 Introduzione

La rete presentata nei precedenti capitoli è costituita da una moltitudine di nodi, che possono essere dislocati fisicamente in un'area più o meno vasta, e talvolta collocati in luoghi difficilmente raggiungibili. Per questo, potrebbe essere complicato, se non impossibile, accedere al nodo in modo diretto, ad esempio tramite connessione USB. Quindi, asce l'esigenza di poter configurare e modificare il comportamento dei nodi anche a distanza, per cui è stato realizzato un protocollo per l'aggiornamento remoto del software.

L'aggiornamento consiste nella trasmissione, attraverso la rete powerline, di un file binario, costituito dal software compilato. Una volta che il nodo ha ricevuto correttamente il file, il vecchio software viene sostituito, quindi il nodo inizia ad operare con le nuove istruzioni.

9.2 Struttura del firmware

Con *firmware* si intende l'intero software che permette al dispositivo di svolgere i propri compiti; per questo progetto, si possono distinguere tre parti fondamentali:

- *Librerie*: si tratta delle librerie proprietarie di Texas Instruments che implementano il protocollo PRIME, nonché la libreria di supporto per le

funzionalità di base del microcontrollore.

- *Stack*: è la parte di software che realizza quanto descritto in questa tesi; si appoggia alle librerie di cui sopra per permettere ai nodi di comunicare ed eseguire alcune attività.

- *Codice utente*: composto da un eventuale software che sfrutta stack e librerie per svolgere ulteriori azioni.

Per effettuare l'aggiornamento, è possibile trasmettere il firmware nel suo intero, ma ciò richiede un notevole impiego di tempo ed è poco efficiente in termini di memoria utilizzata. Al contrario, risulta molto conveniente trasmettere soltanto quella parte di firmware che è stata effettivamente modificata.

Riguardo alla suddivisione presentata, si può notare che le librerie, essendo per natura non modificabili, non avranno mai bisogno di essere aggiornate; la parte relativa allo stack, invece, durante la fase di sviluppo potrebbe subire molti cambiamenti, mentre, una volta finalizzato, non necessiterà più di modifiche. Per tale ragione, in questo progetto, le librerie sono state separate dal resto del firmware e risiedono in una zona di memoria distinta, così da poter compilare e aggiornare la parte restante del firmware in modo indipendente.

Maggiori dettagli sulla separazione del firmware in due parti, si possono trovare nell'Appendice C.

9.3 Procedura di aggiornamento

Il file che contiene il firmware è memorizzato in un computer e, in qualche modo, deve essere trasmesso ai Service Node. Per raggiungere lo scopo, il PC deve affidarsi al Base Node, che è l'unico nodo della rete in grado di comunicare con tutti gli altri. Uno schema di principio è rappresentato in Figura 9.1.

Nel computer, quindi, è presente un programma per dialogare, via USB, con il Base Node e, analogamente, quest'ultimo avrà un codice utente che gli permette di comunicare con il PC.

Per questo progetto, è stato sviluppato un protocollo relativamente semplice, che permette di trasferire in modo affidabile il firmware, ma consente di aggiornare

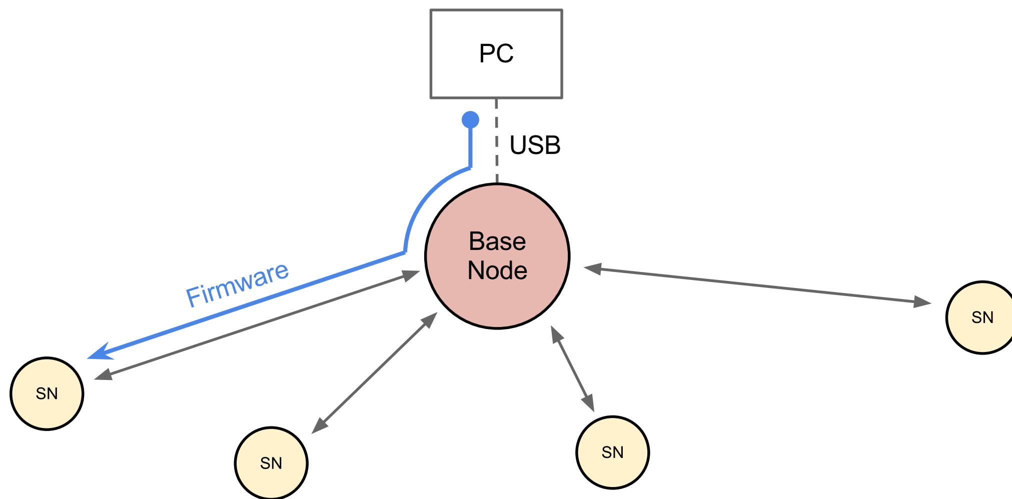


Fig. 9.1: Schema della procedura di aggiornamento firmware: il PC trasmette il nuovo binario al Service Node, attraverso la connessione USB con il Base Node.

un solo nodo alla volta. Infatti, nella procedura di aggiornamento, viene indicato l'indirizzo MAC di un Service Node, che chiameremo *target*.

9.4 Protocollo

L'aggiornamento del firmware si compone di alcune fasi:

- *Inizializzazione*: il Base Node comunica al nodo target che inizia la procedura di aggiornamento.
- *Invio dati*: il file viene trasferito, con uno o più pacchetti.
- *Finalizzazione*: indica la fine della trasmissione dati, eventualmente accompagnato da un codice per il controllo di integrità.
- *Riavvio*: comanda al target di effettuare un riavvio, che ha la funzione di attivare il firmware appena ricevuto.

A ciascuna di queste corrisponde un tipo di pacchetto, inoltre, per assicurare che i dati vengano correttamente processati dal destinatario, è implementato un meccanismo *ARQ Stop&Wait*, per cui, per ogni pacchetto trasmesso, il PC

attende una risposta da parte del Service Node. In caso la risposta non venga pervenuta entro un tempo limite, allora si ha la ritrasmissione del pacchetto. Più in particolare, anziché attuare una semplice ritrasmissione, viene utilizzato un ulteriore pacchetto, che permette di conoscere lo stato attuale del nodo target e, quindi, determinare se l'ultimo pacchetto è stato effettivamente perso: ad esempio, il nodo potrebbe aver impiegato più tempo del previsto per elaborare i dati.

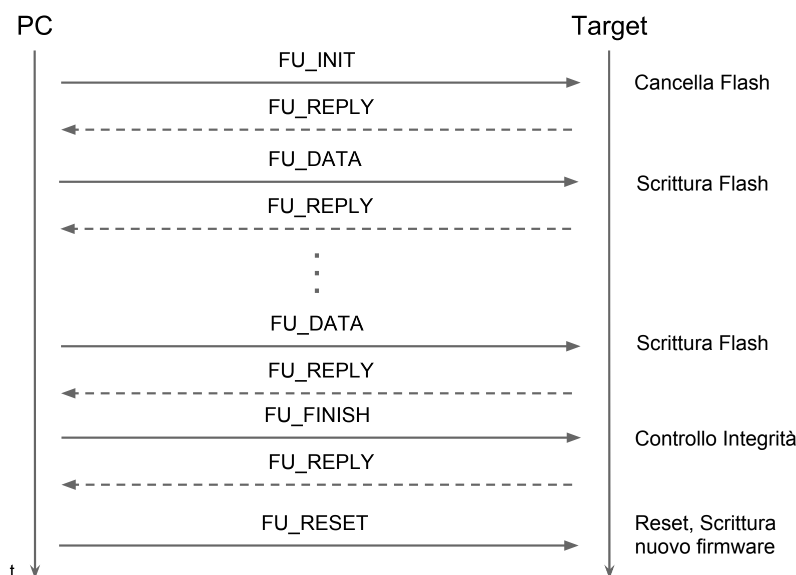


Fig. 9.2: Sequenza di pacchetti di un aggiornamento firmware.

In definitiva, i pacchetti che costituiscono il protocollo di aggiornamento firmware sono riassunti in Tabella 9.1, mentre in Figura 9.2 è rappresentata la sequenza di pacchetti che compongono l'intera procedura di aggiornamento.

Nome	**Descrizione**
INIT	Inizializzazione aggiornamento
DATA	Trasmissione dati firmware
FINISH	Fine invio dati
STATUS	Richiede lo stato del target
REPLY	Risposta di conferma del target

Tabella 9.1: Pacchetti che costituiscono il protocollo di aggiornamento firmware.

Rimane da vedere la struttura di questi pacchetti. Quelli diretti dal PC verso il nodo contengono un primo campo, denominato *CMD*, che identifica il tipo di pacchetto, come si può osservare in Figura 9.3 e Figura 9.4. Tra questi, solo il pacchetto di tipo *DATA* contiene ulteriori parametri: i dati, costituiti da porzioni di firmware, la lunghezza del precedente campo e l'indirizzo di memoria a cui vanno memorizzati i dati.

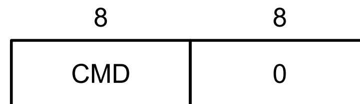


Fig. 9.3: Struttura del pacchetto per i comandi *INIT*, *STATUS*, *FINISH* e *RESET*.

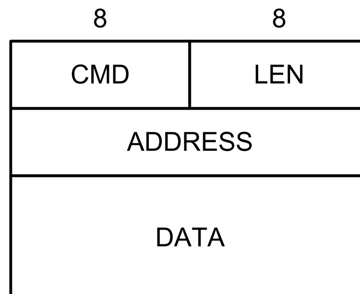


Fig. 9.4: Struttura del pacchetto *DATA*: contiene parte del firmware e l'indirizzo a cui questa corrisponde.

Il pacchetto *REPLY*, invece, ha una struttura diversa (Figura 9.5), ovvero contiene un campo *STATUS*, atto a segnalare lo stato attuale del nodo, più eventuali altri dati. In Tabella 9.2 sono elencati i possibili valori dello stato.

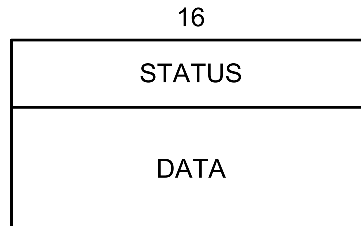


Fig. 9.5: Struttura del pacchetto *REPLY*: contiene lo stato attuale della procedura di aggiornamento.

Nome	Descrizione	Parametri
IDLE	Non è in corso alcuna operazione di aggiornamento	-
READY	Il nodo è pronto a ricevere i dati	-
DATA	Sono stati ricevuti dati	Ultimo indirizzo ricevuto
DONE	Procedura terminata	-

Tabella 9.2: Possibili valori del campo *STATUS*, in un pacchetto di tipo *REPLY*.

Riassumendo, la procedura di aggiornamento firmware è composta da una sequenza di pacchetti: si parte dall'inizializzazione, poi segue lo scambio dei dati, e si conclude con una finalizzazione; dopo il reset del dispositivo, il nuovo firmware inizia la sua esecuzione.

9.5 Implementazione

L'operazione di aggiornamento firmware, come visto, ha due principali attori: il PC e il nodo target. Il primo esegue un software che ha il compito di leggere il file binario, cioè il codice compilato, e inviarlo al nodo, con il protocollo qui presentato; il target, invece, man mano che riceve i dati, li deve immagazzinare in una zona temporanea, per poi sostituire il nuovo firmware al vecchio.

I dettagli sul programma per PC sono illustrati nel Capitolo 10.

Nodo target

Il nodo target ha bisogno di una memoria dove immagazzinare il nuovo firmware, finché non termina la procedura di aggiornamento. Le opzioni disponibili sono la memoria RAM e la memoria Flash del microcontrollore, ma la prima non è sufficientemente capiente per un firmware. Nella flash, d'altra parte, è già memorizzato e in esecuzione il firmware attuale, quindi la memoria deve essere partizionata. Vengono quindi create tre zone distinte: una per le librerie, una dove eseguire il firmware e una in cui salvare il nuovo firmware, che andrà poi a sovrascrivere il precedente. I dettagli su questa ripartizione vengono discussi nell'Appendice C.

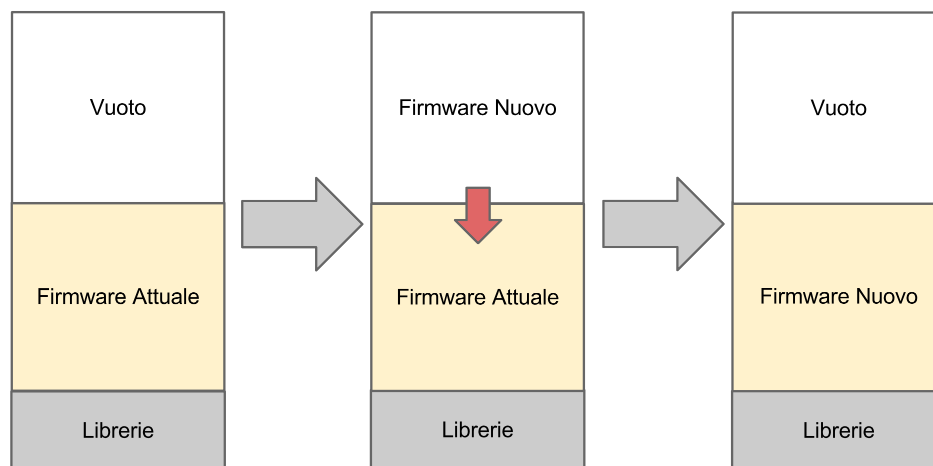


Fig. 9.6: Organizzazione della memoria flash. Il nuovo firmware viene memorizzato in una zona vuota, per poi essere copiato, sostituendo il firmware attuale.

Nel nodo target, ogni pacchetto di aggiornamento viene inoltrato dal livello MAC ad una funzione di livello superiore, che si occupa di elaborarlo e di inviare una risposta alla sorgente. A seconda del campo *CMD*, vengono eseguite diverse operazioni. In particolare, alla ricezione del comando *INIT*, la partizione di memoria flash che ospiterà il nuovo firmware viene cancellata, per essere pronta alla scrittura; con il comando *DATA*, le porzioni di file ricevute vengono scritte in memoria, ad un indirizzo associato all'indirizzo indicato nel pacchetto; all'arrivo del comando *FINISH*, l'aggiornamento termina, ma il nuovo firmware si trova

ancora in una zona di memoria distinta da quello attuale; l'ultimo importante passo è il comando *RESET* che fa riavviare il nodo.

Nel momento in cui il nodo si riavvia, prima di iniziare le normali operazioni, il software effettua un controllo in memoria, per verificare se è presente un nuovo firmware; se l'esito è positivo, allora il nuovo firmware viene copiato, sostituendo quello attuale, e il processore si riavvia nuovamente. Per eseguire questo controllo, il programma deve essere in grado di riconoscere la presenza di un firmware in memoria, per questo motivo i primi byte dello stesso sono costituiti da un'intestazione (*header*). In particolare è specificato il numero di versione, che viene confrontato con quello in esecuzione.

La sequenza di tutte queste operazioni è schematizzata in Figura 9.6.

Flash API

La memoria Flash non è di uso immediato, per quanto riguarda la scrittura dei dati, perciò ci si affida ad una libreria, denominata *Flash API*. La libreria è contenuta nel file *2806x_BootROM_API_TABLE_Symbols_fpu32.lib* e permette la gestione della memoria non volatile del microcontrollore. In pratica, offre funzioni per la cancellazione, scrittura e verifica della memoria.

Per l'utilizzo di tale libreria, vanno poste particolari attenzioni, sia nella configurazione preliminare, che nel codice da implementare, come descritto in dettaglio in [20]. Una questione importante da notare è che tutte le operazioni sulla memoria flash vanno eseguite in RAM, ciò implica che alcuni metodi andranno copiati nella memoria volatile prima di poterli eseguire.

9.6 Prestazioni

Il firmware sviluppato per questo progetto pesa complessivamente 50 KiB, ma grazie all'accorgimento di separare le librerie, si ottiene un file del peso di 7.28 KiB. Ciò comporta un netto miglioramento nella velocità di trasferimento del firmware al nodo target. Inoltre, la quantità di dati può essere ulteriormente ridotta nel caso in cui lo stack, o parte di esso, diventi parte integrante delle librerie statiche.

Ad esempio, utilizzando come modulazione una QPSK+FEC, l'aggiornamento viene trasmesso in 9.5 secondi, in assenza di ritrasmissioni.

9.7 Possibili sviluppi

Il principale svantaggio di questo protocollo è che l'aggiornamento coinvolge un solo nodo alla volta; se vi è la necessità di aggiornare più nodi con il medesimo firmware, l'operazione potrebbe richiedere molto più tempo del necessario. Per rendere più rapida la procedura, può essere realizzato un protocollo che trasmette il firmware in modalità broadcast. In questo modo, tutti i nodi interessati ricevono l'aggiornamento contemporaneamente, inoltre, con l'aggiunta di una qualche ridondanza, si possono compensare eventuali dati persi, senza ricorrere alla ritrasmissione [13].

Capitolo 10

Software PC

10.1 Introduzione

Nella rete transitano numerosi pacchetti e i nodi eseguono i propri algoritmi, ma non si ha un riscontro di quanto accade. Realizzando un'interfaccia tra utente e Smart Microgrid, invece, si può interagire con la rete, monitorando lo stato e comandando nuove operazioni.

Per questo progetto è stato realizzato un applicativo per PC, dotato di interfaccia grafica, che dialoga con il Base Node e mette a disposizione due funzioni:

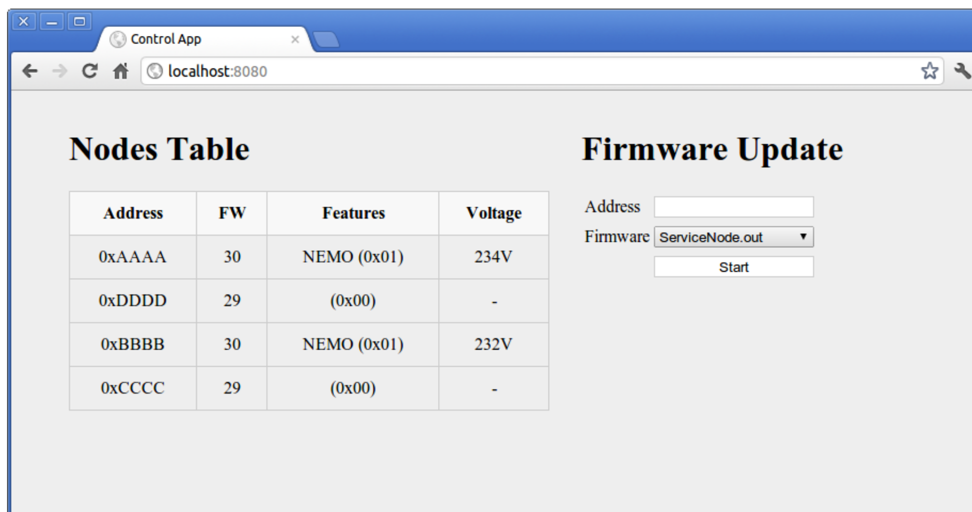
- *Tabella dei nodi*: mostra l'elenco dei nodi connessi alla rete e le relative funzionalità.
- *Aggiornamento firmware*: consente di aggiornare il firmware di un qualunque Service Node nella rete.

Questo programma, tuttavia, costituisce soltanto un punto di partenza per realizzare interfacce più complesse.

10.2 Interfaccia web

L'accesso al programma da parte dell'utente avviene attraverso un'interfaccia web, per mezzo di un qualsiasi *browser* internet. Ciò significa che l'interfaccia è disponibile non soltanto nel PC a cui è fisicamente collegato il Base Node, ma in qualsiasi computer connesso alla rete internet.

Quello che si presenta all'utente è mostrato in Figura 10.1 e si distinguono subito le due funzioni. Le istruzioni per l'utilizzo del software si possono trovare nell'Appendice D, mentre l'ambiente di sviluppo e le librerie utilizzate per la sua realizzazione saranno presentati nel Capitolo 11.



Address	FW	Features	Voltage
0xAAAA	30	NEMO (0x01)	234V
0xDDDD	29	(0x00)	-
0xBBBB	30	NEMO (0x01)	232V
0xCCCC	29	(0x00)	-

Fig. 10.1: Interfaccia web dell'applicazione di controllo.

10.3 Connessione USB

Come visto, il PC e il Base Node si scambiano dati per fornire informazioni all'utente e, per fare questo, si sfrutta la connessione USB del kit di sviluppo, che fornisce una porta seriale virtuale.

La comunicazione seriale consente la trasmissione di un flusso di caratteri, ma i dati scambiati dai due dispositivi sono costituiti da blocchi con un numero variabile di byte; per tale motivo è stato sviluppato un protocollo che organizza le informazioni in pacchetti, descritto con più dettaglio in Appendice A.

In ciascun pacchetto, come prima cosa, è indicato il tipo di comando, mentre i dati successivi vengono interpretati a seconda dell'azione da eseguire. Quindi, il PC può inviare diversi comandi al Base Node, il quale risponderà con un pacchetto entro un tempo limite.

10.4 Monitoraggio

La funzione di monitoraggio consiste nella visualizzazione della tabella dei nodi connessi alla rete. In particolare, vengono mostrate le letture di tensione eseguite dai Service Node a cui è collegato un dispositivo NEMO.

In pratica, questa funzionalità mostra in tempo reale il funzionamento dell'algoritmo di controllo, illustrato nel Capitolo 8, e del protocollo di registrazione, presentato nel Capitolo 7, dando un feedback visivo di ciò che avviene nella rete.

L'implementazione di questa funzionalità consiste soltanto nell'inviare un comando al Base Node, che subito trasmetterà al PC i dati presenti nella propria tabella dei nodi.

10.5 Aggiornamento firmware

Il secondo aspetto del programma è quello che permette di inviare un nuovo firmware ai nodi presenti nella microgrid e visualizzare l'avanzamento dell'operazione. Per questo, il software gestisce l'intero protocollo di aggiornamento, descritto nel Capitolo 9, mentre al Base Node è delegata soltanto la trasmissione dei pacchetti sul mezzo fisico. In breve, le operazioni effettuate sono:

1. Lettura ed elaborazione del file eseguibile.
2. Inoltro di un pacchetto di aggiornamento, specificando l'indirizzo di destinazione.
3. Attesa di un pacchetto di risposta o lo scadere del timeout.
4. Ripetizione dei punti 1, 2 e 3, finché tutto il firmware è stato trasmesso.

Un elemento fondamentale riguarda la lettura del file eseguibile generato dal compilatore. Tale file è prodotto in formato *COFF* (Common Object File Format) e contiene varie informazioni sul firmware, tra cui le zone di memoria con relativi indirizzi [17]. Il programma deve estrarre le porzioni di file che corrispondono al codice eseguibile e inviarle al target, indicando sempre il corrispondente indirizzo di memoria, in modo tale che il nodo possa riassemblarle all'interno della flash.

Capitolo 11

Ambiente di sviluppo

11.1 Introduzione

L'implementazione del progetto di tesi si divide principalmente in due parti, un relativa allo sviluppo del software dei nodi della rete, l'altra alla realizzazione del software di controllo per PC. Nel primo caso, sono stati utilizzati gli strumenti forniti da Texas Instruments per i propri dispositivi, tra cui *Code Composer Studio*; nell'altro caso, invece, ci si è affidati a programmi gratuiti, come l'ambiente integrato di sviluppo *Eclipse CDT*.

In questo capitolo, viene data una presentazione generale degli strumenti di sviluppo e si illustra come iniziare la scrittura di un programma per i kit di TI, mentre in Appendice D si trovano maggiori dettagli sui programmi utilizzati e sulla realizzazione del software.

11.2 Ambiente CCS

Code Composer Studio (CCS), in Figura 11.1, è un ambiente di sviluppo realizzato da Texas Instruments per le proprie famiglie di microcontrollori. CCS comprende tutta una serie di strumenti utili allo sviluppo di applicazioni embedded. Sono inclusi compilatori per i dispositivi TI, un editor di file sorgenti, un sistema di compilazione, il debugger e molto altro. La semplicità di configurazione permette di iniziare rapidamente lo sviluppo di software.

CCS è basato sul programma Eclipse e quindi rappresenta un ambiente familiare e completo per la programmazione in linguaggio C e Assembly.

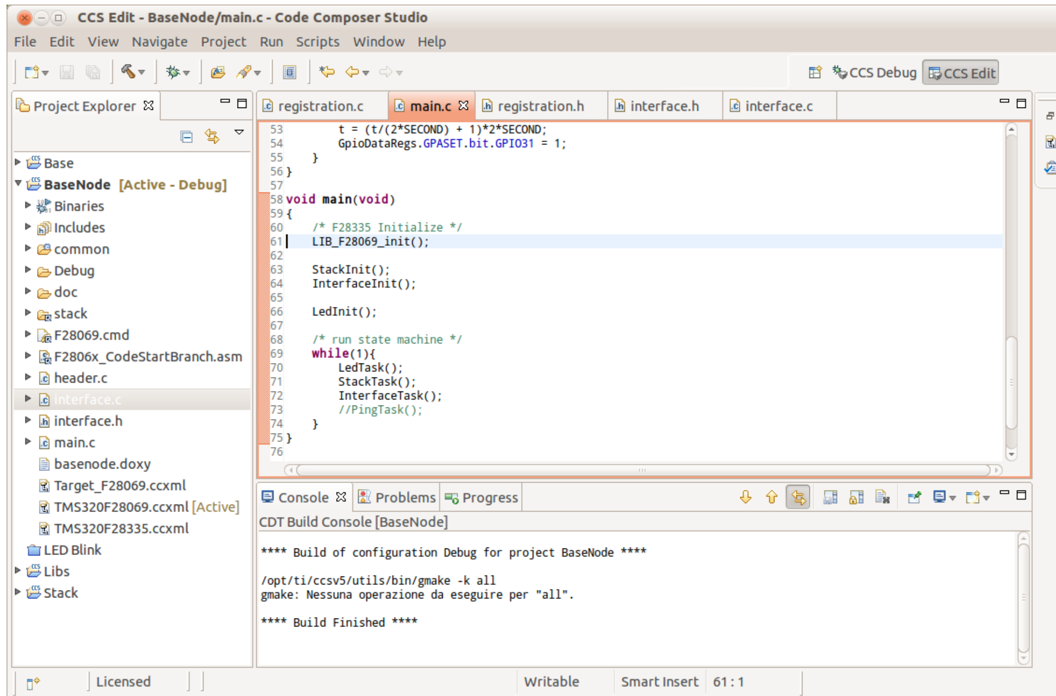


Fig. 11.1: Ambiente di sviluppo TI Code Composer Studio

Librerie

Alcune funzionalità importanti, specialmente quelle legate all'architettura del microcontrollore, vengono fornite da TI sotto forma di librerie. In questo modo si rende più rapido lo sviluppo e non si ha la necessità di conoscere a fondo il funzionamento della CPU.

In questo progetto sono state utilizzate le seguenti librerie:

- *CSL F28069*: per configurare correttamente la CPU e sfruttare le sue potenzialità, si utilizza la libreria di supporto del microcontrollore.
- *Flash*: le funzionalità per la cancellazione e scrittura della memoria flash, indispensabili per l'aggiornamento remoto del firmware, sono racchiuse in una libreria fornita da TI, chiamata *Flash API*, vista nel Capitolo 9.

- *PRIME*: come già visto nel Capitolo 4, viene usata una parte delle librerie del kit di sviluppo, allo scopo di usare il protocollo di livello fisico PRIME, per la comunicazione powerline.

Le prime due librerie si possono trovare all'interno del programma *controlSUITE*^{TM}, che permette di scaricare tutte le principali librerie per i processori della famiglia C2000$^{\mathrm{TM}}$; la terza, invece, viene data a corredo del kit di sviluppo.

Progetto CCS di base

Per questa tesi, tutti i file necessari sono stati riordinati e raccolti in una cartella da cui possono attingere i vari software, chiamata "PRIME".

Vediamo ora come si crea un progetto CCS di partenza per il microcontrollore F28069, con supporto allo strato fisico PRIME.

1. Si crea un nuovo progetto, indicando un nome e le seguenti impostazioni:
   - *Output type*: Executable
   - *Family*: C2000
   - *Variant*: TMS320F28069
   - *Connection*: JTAG XDS100
   - *Advanced -> Linker command file*: nessuno
2. Aprire le proprietà del progetto e abilitare il supporto FPU32 e VCU in *Build -> C2000 Compiler -> Processor Options*
3. Aggiungere in *Build->Variables* e in *Resource->Linked Resources*, il link chiamato *PRIME_ROOT* che punta alla cartella "PRIME"
4. In *Build -> C2000 Compiler -> Include Options* aggiungere alla *#include search path* le cartelle:

```
"${PRIME_ROOT}/DSP2806x/F2806x_common/include"
"${PRIME_ROOT}/prime/V6/include"
"${PRIME_ROOT}/DSP2806x/F2806x_headers/include"
```

```
"${PRIME_ROOT}/DSP2806x/F2806x_init/include"
"${CG_TOOL_ROOT}/include"
```

5. Aggiungere in *Build* -> *C2000 Compiler* -> *Advanced Options* -> *Predefined Symbols*:

   ```
   "_DEBUG"
   "LARGE_MODEL"
   "FLASH"
   "PHY_RX_ONLY=1"
   "TEST_RX_ONLY=1"
   "F2806X"
   ```

6. In *Build* -> *C2000 Linker* -> *Basic Options*:
   - *Stack size*: 0x380
   - *Heap size*: 0x400

7. Inserire i seguenti file al progetto, creando un link relativi a *PRIME_ROOT*:

   ```
   F2806x_Headers_nonBIOS.cmd        (parte del linker file)
   F2806x_GlobalVariableDefs.c       (registri della CPU)
   F28069_init.c                     (inizializzazione CPU)
   csl_f2806x.lib                    (supporto CPU)
   phy_lin_afe031_f2806x.lib         (livello fisico PRIME)
   hal_afe031_f2806x.lib             (AFE di PRIME)
   ```

8. Aggiungere al progetto file questi file, con l'opzione di copia:

   ```
   DSP2833x_CodeStartBranch.asm      (entry-point del sw)
   F28069.cmd                        (linker command file)
   ```

9. Creare un file "main.c" che contenga il metodo:

   ```
   void main(void) {
       // ...
   }
   ```

Dopo questi passaggi, il programma può essere compilato e caricato nella memoria di un kit di sviluppo, semplicemente connettendo il dispositivo al PC via USB, collegando l'alimentatore fornito e avviando l'operazione di debug.

Nel file "main.c", si possono includere i file *header* relativi alla libreria di PRIME, in modo da poter richiamare le funzioni per l'accesso al mezzo fisico, documentate in [16]. Così si può realizzare un programma che mette in comunicazione i dispositivi, grazie al protocollo powerline. Come riferimento, nel pacchetto PRIME di TI è incluso un esempio chiamato "test_tx_rx".

I progetti CCS realizzati per lo sviluppo dello stack posso presentare qualche differenza da questo progetto di base, ma eventuali modifiche sono descritte in Appendice D.

11.3 Ambiente Eclipse CDT

Eclipse è una piattaforma di sviluppo open-source, multi-linguaggio e multipiattaforma, ideata da un consorzio di grandi società, chiamato Eclipse Foundation.

Il *C/C++ Development Tooling* (in breve CDT) è l'ambiente per la programmazione in linguaggio C/C++ all'interno di Eclipse (Figura 11.2). Comprende molti strumenti utili e si interfaccia perfettamente a strumenti esterni, come compilatori e debugger, normalmente inclusi nelle distribuzioni Linux.

Librerie

Per realizzare l'interfaccia web dell'applicazione di controllo, bisogna implementare un server *HTTP* in grado di generare pagine *HTML* dinamiche, che contengono le informazioni sul Base Node e la rete. Questo può essere fatto con varie tecniche, ma in questo caso è stata adottata una semplice libreria open-source da intergrare nel programma, chiamata *mongoose* [7].

Mongoose offre varie funzionalità e permette sia di generare pagine in modo dinamico, sia di fornire file statici.

Per rendere l'interfaccia grafica più funzionale, sono state incluse nelle pagine alcune funzioni in codice *Javascript* che sfruttando la libreria *JQuery* [6].

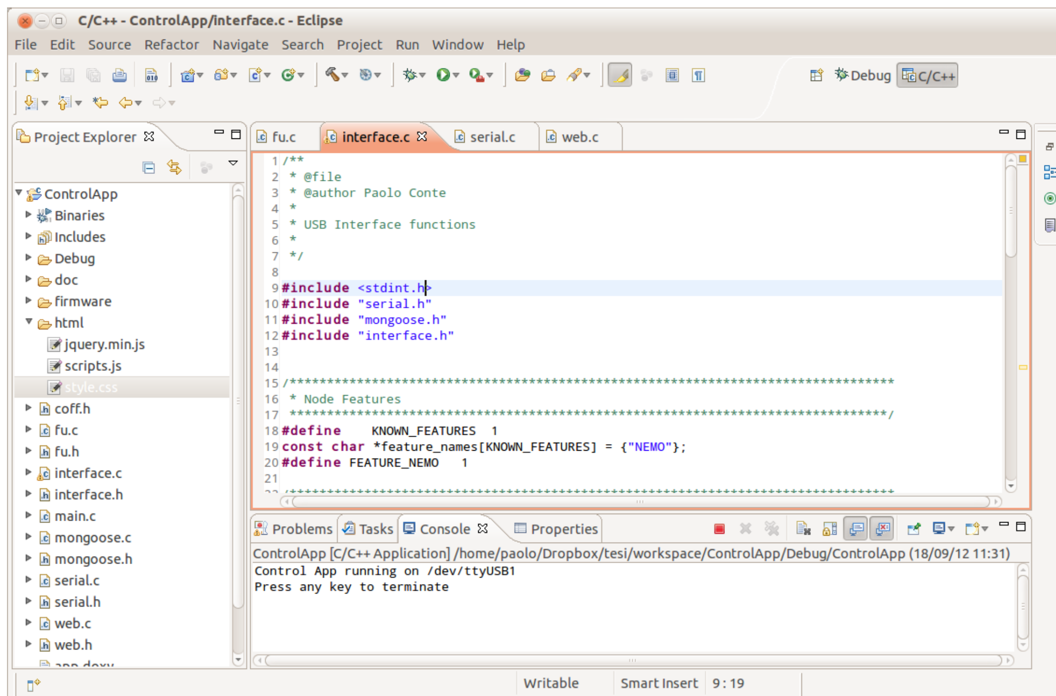


Fig. 11.2: Ambiente di sviluppo Eclipse CDT.

Ad esempio, la tabella dei nodi viene periodicamente aggiornata, in modo da fornire indicazioni in tempo reale, mentre la sezione di aggiornamento firmware mostra il progresso dell'operazione.

Impostazione progetto

Per configurare il progetto Eclipse per l'utilizzo della libreria mongoose, bastano poche, ma essenziali, impostazioni:

- Inserire della libreria *pthread* nelle impostazioni del linker, per il supporto al multi-threading.
- Aggiungere la libreria *dl*, oppure creare il simbolo pre-definito *NO_SSL*, che disabilita il supporto alle connessioni sicure.
- Includere i file “mongoose.c” e “mongoose.h” al progetto.

Inoltre, nella cartella “html” sono stati inseriti alcuni file statici, tra cui la libreria JQuery.

Capitolo 12

Conclusioni

In questa tesi, sono stati presentati diversi protocolli che nell'insieme costituiscono una base per lo sviluppo di reti intelligenti.

Si è visto un sistema di controllo che sfrutta altre funzioni dello stack, come la registrazione e la sincronizzazione del tempo, per impartire comandi e raccogliere dati. Inoltre, attraverso l'interfaccia web, un utente può osservare lo stato dei nodi e intervenire nel funzionamento della rete.

I possibili sviluppi sono numerosi e la strada seguita nella realizzazione di questo progetto non è di certo l'unica. Qui si è utilizzata una struttura centralizzata, dove un nodo dirige tutte le operazioni, ma, al contrario, si può costruire una struttura decentralizzata o distribuita. Inoltre, vi sono diverse possibilità per lo schema da utilizzare per il livello fisico.

Il sistema di controllo può essere ampliato completando il supporto ad altri dispositivi da connettere ai nodi e realizzando un algoritmo di gestione più complesso, in grado di comandare i nodi per raggiungere uno o più scopi prefissati.

In definitiva, è stato fornito un esempio concreto di Micro Smartgrid, che implementa le funzionalità più importanti e può essere utilizzato per effettuare ulteriori studi.

Bibliografia

[1] C2000^{TM} Power Line Modem Developer's Kit. http://www.ti.com/tool/tmdsplckit-v3.

[2] Code Composer Studio^{TM}. http://www.ti.com/tool/ccstudio.

[3] controlSUITE^{TM}. http://www.ti.com/controlsuite.

[4] Doxygen. http://www.stack.nl/~dimitri/doxygen/.

[5] Eclipse CDT. http://www.eclipse.org/cdt/.

[6] JQuery. http://jquery.com.

[7] Mongoose. https://github.com/valenok/mongoose.

[8] WineHQ. http://www.winehq.org/download/.

[9] Massimo Gallina. Sincronizzazione tra dispositivi su powerline communications per smartgrid. Master's thesis, Università degli studi di Padova, 2012.

[10] IME S.p.A. *MF6FT Communication Protocol*. http://www.imeitaly.com/protocolli/PR109.pdf.

[11] IME S.p.A. *NT695 Nemo D4-L+*. http://www.imeitaly.com/docs/NT695.pdf.

[12] PRIME Alliance. *Draft Specification for PoweRline Intelligent Metering Evolution*.

[13] Michele Rossi, Giovanni Zanca, Luca Stabellini, Riccardo Crepaldi, Albert F. Harris III, and Michele Zorzi. Synapse: A network reprogramming protocol for wireless sensor networks using fountain codes. *Proc. of IEEE SECON 2008, San Francisco, CA, USA*, 2008.

[14] Michele Tasca. Power line communications per smart grid: studio e test su rete artificiale. Master's thesis, Università degli studi di Padova, 2012.

[15] Texas Instruments Inc. *Byte Accesses with the C28x CPU.* `http://processors.wiki.ti.com/index.php/Byte_Accesses_with_the_C28x_CPU`.

[16] Texas Instruments Inc. *PRIME Physical Layer API Specifications.*

[17] Texas Instruments Inc. *SPRAAO8 Common Object File Format.*

[18] Texas Instruments Inc. *SPRU513D TMS320C28x Assembly Language Tools.*

[19] Texas Instruments Inc. *SPRUGU2B controlSUITE^{TM} Getting Started Guide.*

[20] Texas Instruments Inc. *TMS320F2806x Piccolo^{TM}A Flash API.*

Appendice A

Protocollo seriale

La comunicazione tra Base Node e computer avviene tramite cavo USB, che fornisce una porta seriale virtuale, perciò la comunicazione è costituita da un flusso di byte. Se, però, si devono trasmettere una serie di informazioni, costituite da più byte, per di più di lunghezza variabile, è necessario organizzare questi dati in pacchetti. Per questo motivo, è stato implementato un semplice protocollo, descritto in questa appendice.

La struttura del pacchetto è composta da un byte di inizio (*START*), una sequenza di byte,e un byte di fine pacchetto (*END*).

Per evitare che i valori associati ai simboli *START* e *STOP* compaiano all'interno del pacchetto, confondendo il ricevitore, si utilizza un ulteriore simbolo (*ESCAPE*). Quando nel flusso dati compare un simbolo uguale a *START*, *STOP* o *ESCAPE*, si trasmette un byte *ESCAPE* e poi si invia il dato con il bit 5 invertito; in ricezione, il simbolo *ESCAPE* si scarta, ma al byte successivo si inverte il bit 5.

Esempio

Di seguito, un esempio pratico di pacchetto costruito con il meccanismo presentato; in Tabella A.1 sono riportati i valori utilizzati nel progetto per i simboli speciali.

Dati:		0x12	0x34	0x7E		0x56	
Pacchetto:	0x7F	0x12	0x34	0x7D	0x5E	0x56	0x7E

Nome	**Valore**
START	0x7F
ESCAPE	0x7D
END	0x7E

Tabella A.1: Valori speciali del protocollo.

Appendice B

Overflow mode

Durante lo sviluppo del progetto, è stato riscontrato un problema nell'eseguire le operazioni di sottrazione, che porta al malfunzionamento di diverse operazioni. In particolare, quando si effettua la differenza tra due variabili senza segno, anche se l'operazione ha subito un overflow, ci si aspetta un risultato corretto; in alcuni casi, invece, si sono ottenuti valori errati.

La CPU dei microcontrollori C2000 di Texas Instruments, contiene un flag, denominato *OVM*, che definisce il comportamento dell'unità aritmetica in caso di overflow. Il valore predefinito è zero, e in questo caso si ottiene il normale comportamento delle operazioni di addizione e sottrazione, ma se il flag è attivo, allora in caso di overflow, il risultato satura al massimo (in caso di addizione) o minimo (in caso di sottrazione) valore rappresentabile.

Il compilatore suppone sempre che il flag non sia attivo, ma alcune funzioni delle librerie PRIME attivano il flag senza resettarlo successivamente, compromettendo il funzionamento dell'intero firmware.

Non essendo disponibili i sorgenti delle librerie, è stata inserita un'istruzione, per il reset del flag, all'uscita di alcune funzioni e all'inizio delle callback.

Appendice C

Librerie, linker e map file

Per comprendere meglio i temi trattati in questa appendice, riassumiamo brevemente qual è il processo di compilazione di un progetto.

Quando si costruisce il file eseguibile (operazione *Build*), si susseguono le seguenti operazioni:

1. Ogni file sorgente viene *pre-processato*
2. Ciascun sorgente viene tradotto in codice *assembly*
3. I file assembly vengono compilati in *file oggetto*
4. Il *linker* unisce tutti i file oggetto e le librerie in un unico eseguibile
5. Viene restituita una mappa degli indirizzi di memoria allocati (*map file*)

Le librerie non sono altro che un insieme di file oggetto, ovvero file compilati in cui gli indirizzi di memoria, relativi a variabili, codice macchina e costanti, non sono ancora definiti. Il compito del linker è quello di assegnare lo spazio di memoria a tutti questi elementi. Per fare ciò, ha bisogno di conoscere quali sono le zone di memoria disponibili e come associare gli indirizzi alle varie parti che compongono l'eseguibile e queste informazioni sono contenute nel *Linker Command File*. Le specifiche di quest'ultimo si possono trovare all'interno di [18].

Separazione delle librerie

Nel Capitolo 9, si illustra la struttura del firmware e si sottolinea la distinzione tra librerie statiche e il codice che compone lo stack, che vengono separati per ottimizzare la procedura di aggiornamento del firmware. L'idea è quella di assegnare due zone di memoria flash diverse per le librerie e per il resto del codice sorgente, in modo da poter trasmettere, attraverso la rete, soltanto il software che viene effettivamente modificato. Il programma potrà comunque accedere sempre alle funzioni delle librerie perché, non essendo state modificate, avranno sempre gli stessi indirizzi di memoria. Questo risultato si potrebbe ottenere istruendo il linker in modo che collochi i file oggetto delle librerie in una specifica area della Flash, però niente ci assicura che gli indirizzi delle funzioni non cambino tra una compilazione all'altra, specialmente se vengono modificate alcune opzioni, perciò questa tecnica non può essere usata.

Per ovviare il problema, è stato impiegato un diverso approccio, che consiste nel compilare le librerie in un progetto CCS separato da quello del firmware vero e proprio.

Organizzazione della memoria

Prima di esaminare la configurazione dei due progetti, vediamo com'è stata organizzata la memoria flash e quali sono le varie componenti. Prima di tutto, va osservato che la memoria del microcontrollore utilizzato è divisa in otto settori e la cancellazione, necessaria per la riprogrammazione, avviene di settore in settore, perciò è necessario collocare librerie e firmware in settori diversi. In Tabella C.1 è riassunta la suddivisione della flash, mentre la Figura C.1 mostra gli indirizzi di memoria assegnati e la sequenza di avvio, che sarà vista più in dettaglio in seguito.

Componente	**Settore**
Entry Point, Indirizzo MAC e Librerie	FLASHA + FLASHB
Firmware + Header	FLASHC + FLASHD + FLASHE
Archiviazione aggiornamento FW	FLASHF + FLASHG + FLASHH

Tabella C.1: Organizzazione della memoria

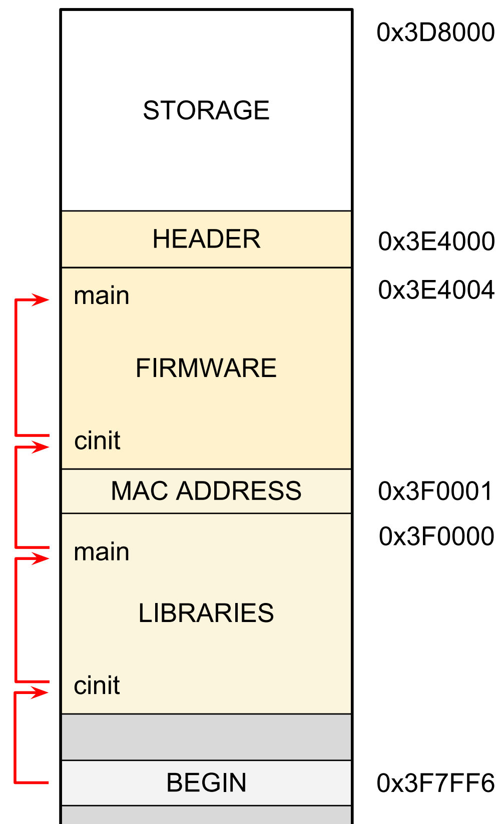


Fig. C.1: Mappa della memoria e sequenza di avvio.

Inoltre, anche la memoria RAM è stata suddivisa, perché un progetto non è consapevole delle sezioni di memoria utilizzate dall'altro; in particolare, alle librerie sono stati assegnati i settori di RAM L7 e L8, e al firmware tutti gli altri.

Compilazione liberie

Il progetto CCS contiene vari file, tra cui le librerie dello strato fisico PRIME, ma quelli importanti, per la separazione delle librerie dal firmware, sono il linker command file e il *main*. Con il primo, si specificano quali zone di memoria flash e RAM utilizzare per la compilazione del file eseguibile, mentre il secondo serve per passare all'esecuzione del firmware dopo l'inizializzazione delle librerie.

In Tabella C.2 sono elencati i file del progetto, con una breve descrizione.

Per implementare quanto detto, il command file è stato modificato, prima di tutto, per delimitare le zone di memoria assegnate alle librerie, con il codice riporato di seguito:

```
MEMORY
{
    PAGE 0:
        ...
        MAC_ADDRESS : origin = 0x3F0000, length = 0x000001
        LIBS        : origin = 0x3F0001, length = 0x007F7F
        ...
    PAGE 1:
        RAML78      : origin = 0x010000, length = 0x004000
    ...
}
```

Poi, si è assegnato alle varie sezioni la zona di memoria flash o RAM corrispondente; di seguito alcuni esempi:

```
SECTIONS
{
    ...
    .cinit              : > LIBS,    PAGE = 0
    .pinit              : > LIBS,    PAGE = 0
    .text               : > LIBS,    PAGE = 0
    codestart           : > BEGIN,   PAGE = 0
    ...
    PHY_RESERVED_BUF    : > RAML78,  PAGE = 1
    PHY_DATA_BUF        : > RAML78,  PAGE = 1
    ...
}
```

L'altro importante elemento di questo progetto, è il file "main.c", che effettua due operazioni:

- Inizializza alcune funzioni delle librerie PRIME, che necessitano di essere eseguite in memoria RAM.
- Esegue un salto all'entry-point del firmware, che si è scelto di posizionare all'indirizzo 0x003E4004.

Infatti, all'avvio del dispositivo, viene prima eseguita l'inizializzazione della parte relativa alle librerie, che poi si occupa di passare all'inizializzazione del firmware, per finire nella funzione *main* del firmware stesso. Questa sequenza di operazioni è anche rappresentata in Figura C.1.

File	**Descrizione**
F28069_init.c	Contiene la funzione di inizializzazione della CPU
F28069.cmd	Il linker commando file di questo progetto
F2806x_CodeStartBranch.asm	Punto di partenza dell'eseguibile
F2806x_GlobalVariableDefs.c	Contiene dichiarazione di variabili della CPU
F2806x_Headers_nonBIOS.cmd	Parte del command file relativo alla CPU
main.c	Contiene la funzione *main*
csl_f2806x.lib	Libreria di base della CPU
hal_afe031_f2806x.lib	Libreria dell'AFE powerline
phy_lin_afe031_f2806x.lib	Libreria dello strato fisico PRIME

Tabella C.2: File utilizzati nel progetto Libreria.

Compilazione firmware

Il firmware viene ora compilato senza includere i file delle librerie PRIME, ma, così facendo, il compilatore non è più in grado di associare le funzioni contenute nelle librerie al loro indirizzo di memoria. Per tale motivo, è stato fornito manualmente l'indirizzo per ogni funzione utilizzata.

Innanzitutto, si è esaminato il *map file* dove sono presenti gli indirizzi relativi a tutte le funzioni, e si sono presi quelli utili al firmware. Poi, è stato creato un file header, dove sono inseriti i puntatori alle funzioni di libreria, utilizzando gli indirizzi appena trovati, secondo questo meccanismo:

```
PHY_status_t PHY_rxPpduStart(PHY_cbFunc_t cb_p)
                        diventa
#define LIB_PHY_rxPpduStart \
             ((PHY_status_t(*)(PHY_cbFunc_t cb_p))0x003f1d6f)
```

Quindi, nel firmware, tutte le chiamate a queste funzioni dovranno avere il prefisso "LIB_", in modo da compilare correttamente il progetto.

Primo avvio

Il progetto Libreria va compilato una sola volta, altrimenti gli indirizzi ricavati precedentemente potrebbero variare, dopodiché può essere caricato su tutti i dispositivi via USB.

Per quanto riguarda il firmware, invece, è fondamentale impostare il progetto CCS in modo che non vengano cancellati i settori A e B della flash, altrimenti si andrebbero a cancellare le librerie. Poi, va caricato almeno una volta attraverso la porta USB e, successivamente, si può sfruttare l'aggiornamento remoto.

Durante la programmazione attraverso l'ambiente CCS, viene anche scritto nella flash (senza cancellarla) l'indirizzo MAC del dispositivo, che quindi va scelto prima di questa operazione. Nell'aggiornamento remoto, invece, l'indirizzo non viene modificato, perché risiede nella stessa memoria delle librerie.

Appendice D

Quick start

In questa sezione sono elencati dettagli relativi all'utilizzo degli strumenti di sviluppo, all'organizzazione dei file e alla verifica del funzionamento dello stack.

Software e librerie

Come prima cosa, è necessario scaricare ed installare gli ambienti di sviluppo CCS ed Eclipse. In [2] e [5] vi sono i link di riferimento dei due programmi, dove sono disponibili versioni sia per sistemi Windows che Linux.

Con l'applicazione controlSUITE™ (Figura D.1), si possono ottenere le librerie di supporto della CPU e le Flash API, seguendo le istruzioni riportate in [19], ma il programma è compatibile soltanto con Windows; si può comunque eseguire anche in Linux grazie al programma WineHQ [8]. La pagina di riferimento si trova all'url [3].

Infine, si reperisce il pacchetto software del kit di sviluppo powerline alla pagina [1]. Quest'ultimo contiene documentazione, librerie, schemi elettrici ed esempi software.

Organizzazione dei file

I file sono organizzati all'interno di una cartella dove sono collocati i *workspace* (cartelle di lavoro) di CCS ed Eclipse, le librerie e la documentazione.

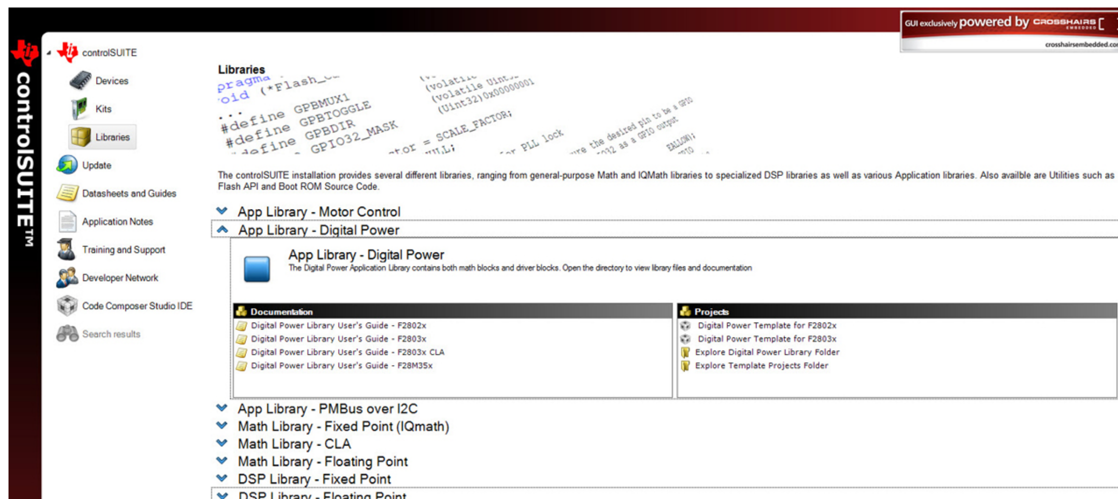


Fig. D.1: Applicazione TI controlSUITE$^{\text{TM}}$.

Nella cartella “PRIME”, si trovano i file di supporto della CPU F28069 e le librerie dello standard PRIME. Non tutti i file presenti sono stati utilizzati, ma principalmente quelli citati nel Capitolo 11 per la creazione del progetto di base. Inoltre, è disponibile una cartella che contiene tutti i file relativi alle Flash API.

I workspace creati sono tre: uno per Eclipse, che contiene i programmi da eseguire nel PC; un altro, chiamato “workspace_a”, racchiude i progetti CCS per il Base Node e per le librerie PRIME, e un progetto *Stack* che contiene i sorgenti della pila di protocolli, che possono essere inclusi in altri progetti; infine, “workspace_b” per il progetto CCS del Service Node. La ragione per cui i progetti per le due tipologie di nodi si trovano in workspace separati verrà spiegata più avanti.

Impostazione dei progetti

La configurazione dei progetti in CCS rispecchia, in generale, quanto detto nel Capitolo 11. In particolare, va posta attenzione alla corretta impostazione delle variabili *PRIME_ROOT*, in modo che puntino al percorso della cartella “PRIME”.

Il progetto per il Base Node e per le librerie hanno una configurazione del tutto simile a quella presentata, ma il primo ha un simbolo predefinito denominato

STACK_BASENODE, che permette una compilazione condizionale di alcuni elementi.

Il progetto per il Service Node non contiene i file relativi a PRIME, in quanto viene applicata la separazione delle librerie, come illustrato in Appendice C.

Compilazione

Per procedere alla compilazione si attiva il comando *Build*, quindi si può vedere il progresso dell'operazione nella sezione *Console*; eventuali errori vengono presentati lì e nella tabella *Problems*.

Al termine vengono prodotti diversi file, tra cui il file eseguibile e il *map file*, utile per studiare l'occupazione di memoria del software e gli indirizzi delle funzioni.

Debug

Prima di poter effettuare il debug di un progetto CCS, è necessario disporre di un *target file*, che istruisce il programma su come collegarsi al kit di sviluppo. Una versione predefinita del file viene automaticamente fornita quando viene creato il progetto ed è sufficiente al debug di una singola scheda.

Se si ha la necessità di controllare più kit con un singolo computer, però, il target file va configurato in modo da specificare il numero seriale del kit a cui si deve connettere. Il seriale si può ricavare usando l'utility *xds100serial*, che fa parte degli strumenti forniti da CCS. L'ambiente di sviluppo, però, non può eseguire il debug di due o più software contemporaneamente, perciò sono necessarie più istanze di CCS, ognuna con il proprio workspace.

Il debug del microcontrollore è preceduto dalla cancellazione e programmazione della memoria Flash. Il progetto può essere configurato per cancellare solo determinate aree della memoria, fondamentale per l'uso del firmware con librerie separate; infatti, bisogna disabilitare la cancellazione di *FLASHA* e *FLASHB*.

Nota importante

Normalmente, il debug inizia dal punto di ingresso (*entry-point*) del programma, che conduce poi alla funzione *main*. Nell'uso delle librerie separate, però, si deve prima eseguire l'inizializzazione delle librerie, che poi cedono il controllo al firmware; per fare questo, bisogna forzare il *Reset* della CPU con l'apposito comando di CCS e poi passare ala funzione *Run*.

Setup

Per mettere in funzione il software realizzato, tutti i nodi, sia Base Node che Service Node, devono essere preparati caricandovi il progetto con le librerie PRIME. Per fare ciò, è sufficiente eseguire il debug del corrispondente progetto, con un target file privo di numero seriale.

Subito dopo, in un kit prescelto può essere caricato il programma del Base Node. Gli altri nodi, invece, possono essere programmati con il firmware del Service Node, con l'accortezza di indicare, per ciascuno di essi, un indirizzo MAC, scritto nell'apposita variabile del file *main.c*. È bene tenere a mente che la programmazione via USB sovrascrive l'indirizzo MAC del dispositivo, mentre l'aggiornamento remoto del firmware lo lascia inalterato.

A questo punto, connettendo i cavi powerline a una ciabatta e alimentando tutti i nodi, prima il Base Node, si avvia il funzionamento della rete. Il primo riscontro che si ottiene è il lampeggio simultaneo dei LED delle schede, grazie all'algoritmo di sincronizzazione. Poi, con l'interfaccia web del programma di controllo, si possono ottenere ulteriori informazioni sullo stato dei dispositivi.

Software PC

Le funzionalità del software per PC sono due: l'aggiornamento remoto del firmware e il monitoraggio della rete attraverso il Base Node. Il relativo progetto Eclipse è situato nella cartella "workspace".

Il programma di gestione, chiamato *Control App*, si avvia da terminale e rimane in esecuzione finché non fermato dall'utente; l'unico parametro richiesto è il device della porta USB a cui è collegato il Base Node. Durante la sua esecuzione,

l'applicazione mette a disposizione un'interfaccia web, raggiungibile da browser digitando l'indirizzo:

```
http://localhost:8080
```

Qui si possono visualizzare i nodi connessi alla rete e le rispettive funzionalità offerte, nonché le letture eseguite dai monitor NEMO, come illustrato nel Capitolo 10.

Per eseguire l'aggiornamento firmware, invece, l'interfaccia elenca i file eseguibili presenti nella cartella "firmware" del progetto, quindi è sufficiente inserire l'indirizzo MAC del nodo da aggiornare, selezionare il file desiderato e avviare la procedura.

Si può accedere all'interfaccia web anche da remoto, sostituendo a *localhost* l'indirizzo IP della macchina, se ci si trova all'interno della medesima subnet, oppure l'indirizzo IP pubblico del router, se il PC è dietro NAT. Nel caso della rete del DEI, si può accedere autenticandosi al server *login* e realizzando un redirect, con il comando

```
ssh -L 80:<nome-pc>:8080 -l <nome-utente> login
```

e collegandosi all'indirizzo

```
http://localhost
```

Versione stand-alone

Per la funzione di aggiornamento firmware vi è anche una versione del programma separata dall'interfaccia web, chiamato *Firmware Updater* (in breve *fu*), che si esegue da terminale con il seguente comando:

```
./fu porta indirizzo inizio fine file
```

I parametri del programma sono:

- *Porta*: indica la porta USB a cui è connesso il Base Node.
- *Indirizzo*: è l'indirizzo MAC del nodo a cui è destinato l'aggiornamento.
- *Inizio*: indirizzo di memoria flash da cui inizia il firmware.

- *Fine*: indirizzo di memoria flash in cui termina il firmware.
- *File*: percorso del file eseguibile del firmware.

Un esempio pratico:

```
./fu /dev/ttyUSB0 0x12AB 0x3E4000 0x3F0000 ServiceNode.out
```

Questo programma non deve essere eseguito contemporaneamente all'applicazione di controllo.

Nota di accesso alle porte USB

In alcuni sistemi Linux, come Ubuntu, l'accesso alle porte USB può essere effettuato solo con privilegi *root*, quindi le applicazioni che ne fanno uso andrebbero eseguite con *sudo*. Questo problema, tuttavia, può essere eliminato creando il file "/etc/udev/rules.d/50-ttyusb.rules" con il contenuto:

```
KERNEL=="ttyUSB[0-9]*", NAME="tts/USB%n", SYMLINK+="%k", ↘
 GROUP="uucp", MODE="0666"
```

Appendice E

Documentazione

Ogni file sorgente che costituisce il software realizzato offre delle funzioni pubbliche, utilizzate dal programma stesso o richiamabili per lo sviluppo di nuove funzionalità. Queste funzioni sono documentate nel codice stesso, ma è anche disponibile una documentazione in formato pdf e html (Figura E.1), che ne raccoglie il nome, lo scopo e le modalità d'uso.

In particolare, all'interno dei progetti *Base Node* e *Stack*, vi è una cartella "doc" che racchiude i documenti.

Doxygen

La documentazione è stata creata in modo automatico, direttamente dai file sorgente, con il programma *doxygen* [4].

Per generare i documenti sono necessari due accorgimenti:

- I file sorgenti devono essere commentati usando un'apposita sintassi, così doxygen è in grado di interpretare automaticamente le informazioni, come i parametri delle funzioni.

- Si deve creare e modificare un file di configurazione, che fornisce a doxygen impostazioni fondamentali, come la locazione dei file e le funzionalità richieste.

In conclusione, la generazione della documentazione avviene con il comando:

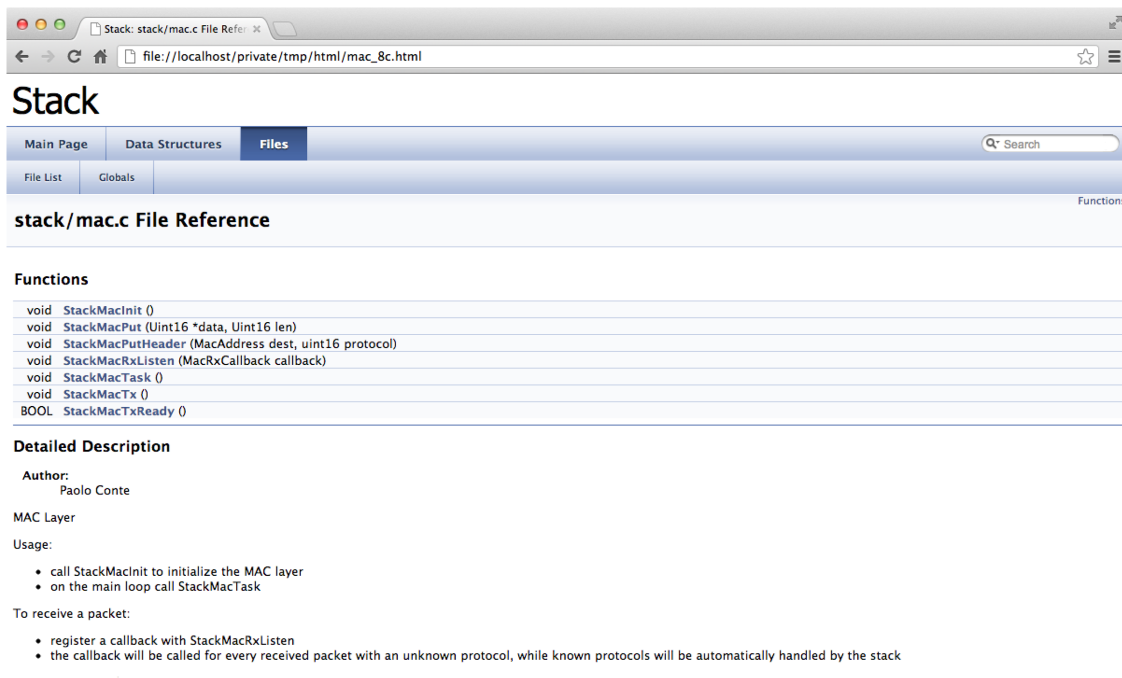


Fig. E.1: Documentazione generata con Doxygen.

```
doxygen stack.doxy
```

dove l'unico parametro è il nome del file di configurazione.