Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Elettronica

Registratore di dati per dispositivi di accumulo energetico

Laureando

Dario Marchetto

Relatore

Prof. Simone Buso

Correlatore

Ing. Marco Stellini

Anno Accademico 2012/2013

Indice

Capitolo 1

Obiettivo del progetto

Questa tesi riguarda la realizzazione di un dispositivo per l'acquisizione dei dati ed il monitoraggio di un sistema di accumulo energetico presente nel Dipartimento di Ingegneria dell'Informazione.

Il suddetto sistema di accumulo si basa su un gruppo di batterie che possono venire caricate e scaricate, simulando il comportamento di un generico utilizzatore.

Il sistema di acquisizione ha il compito di acquisire periodicamente e registrare su un supporto di memoria FLASH (scheda microSD) i dati salienti che caratterizzano lo stato di accumulo energetico, e di fornire una stima dello stato di carica delle batterie.

Inoltre il sistema assume anche la funzione di Web *Server*, fornendo all'utente finale (*Client*) la possibilità di accedere ai dati acquisiti tramite una connessione Internet, usando un browser Web.

Da qui in poi si farà riferimento ad un 'record' per indicare l'insieme dei dati acquisiti dal sistema in un certo istante: ogni record è costituito dalla tensione, dalla corrente, dalla temperatura e dallo stato di carica delle batterie, oltre che ad una 'etichetta temporale' che associa univocamente il record al preciso istante in cui viene salvato.

1.1 Il sistema di accumulo energetico

L'energia elettrica viene accumulata in 21 batterie al piombo da 12 Volt connesse in serie, modello 12HX135 prodotto dalla EnerSys. Ogni singola batteria è costituita da 6 celle ed ha una capacità nominale pari a 28 Ah.

Queste batterie sono state prodotte per essere efficenti nell'immagazzinamento di energia elettrica: possono trovare impiego nei sistemi per la produzione di energia elettrica rinnovabile oppure per avere a disposizio-

ne una riserva di energia da fornire ad apparecchi elettrici di emergenza, nel caso l'ente fornitore dell'energia elettrica ne sospendesse temporaneamente l'erogazione (questo è proprio quello che fanno i gruppi di continuità, o UPS).

Le batterie al piombo (singole o connesse in gruppo) sono tuttora il mezzo più diffuso che viene utilizzato per questi scopi, nonostante la relativamente bassa potenza per unità di volume che riescono ad erogare; un loro fondamentale pregio è il basso costo e la potenza che possono erogare.

In particolare, nel settore delle energie rinnovabili (come il solare o l'eolico) l'accumulo energetico riveste un fattore di primaria importanza, e la sua efficenza dipende in gran parte dal tipo e dalla tecnologia delle batterie utilizzate per conservare l'energia prodotta fino al momento in cui viene richiesta. Inoltre, in questo settore, solitamente non sono noti a priori i periodi in cui il sistema di accumulo dovrà erogare energia, oppure quelli in cui ne dovrà acquisire: questo porta ad un utilizzo delle batterie spesso piuttosto aggressivo, con continui cicli di carica e scarica. Gli stress fisici e/o chimici che ne derivano ne causano la graduale ed inevitabile diminuzione della durata di vita.

Capitolo 2

Sezione Hardware

Il sistema di acquisizione progettato deve acquisire la tensione, la corrente e la temperatura istantanee del banco di batterie in esame, e fornire i dati di misura agli utenti che ne fanno richiesta: lo schema di principio su cui si basa è mostrato in figura 2.1.

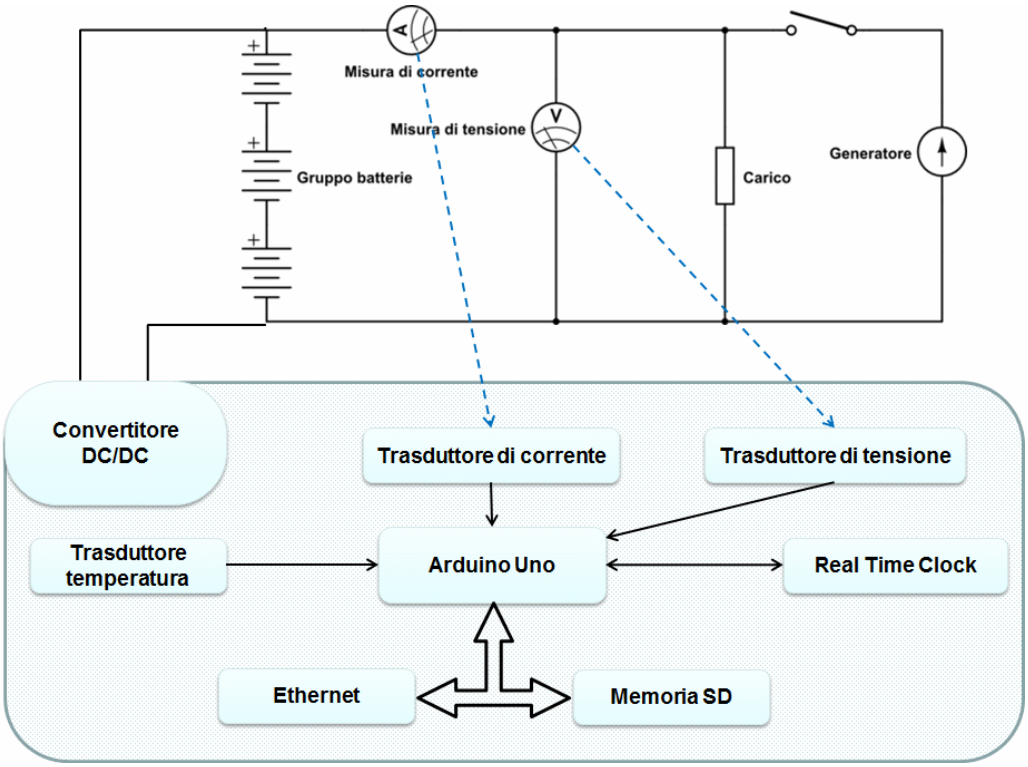


Figura 2.1: Schema a blocchi

Vengono ora descritte separatamente le varie parti hardware che compongono il sistema.

2.1 Alimentazione

Si è deciso di alimentare il sistema connettendolo alle batterie stesse, le quali sono connesse in serie senza la possibilità di accedere ai loro morsetti, ad eccezione che per i due alle estremità della serie: con le batterie a piena carica ai loro capi si trova una tensione intorno ai 280 Volt.

Adottando un convertitore di tensione switching isolato (per motivi di sicurezza), che supporti questo valore di tensione di ingresso, la tensione totale viene ridotta a quella richiesta dal sistema di misura.

La scelta del tipo switching è stata obbligata dal vincolo di ottenere un basso assorbimento di potenza per alimentare il sistema; l'alta efficenza dei convertitori a commutazione consente infatti di raggiungere basse perdite.

La scelta è ricaduta sul modello TMP10212, prodotto dalla TracoPower: consente di avere all'ingresso una tensione continua (nel range $120 \div 370\ V$) oppute alternata (nel range $85 \div 264\ V$), risultando adatto sia come convertitore AC/DC che DC/DC.

Fornisce in uscita i $+12\ V$ ed i $-12\ V$ riferiti alla stessa massa; inoltre presenta una basso valore di ondulazione di tensione (ripple) dichiarato, pari all' 1% della tensione di uscita. Alle sue uscite non vengono perciò aggiunti dei condensatori antidisturbo per stabilizzare meglio la tensione di alimentazione al sistema di acquisizione.

2.2 La scheda Arduino Uno

La scheda Arduino Uno fornisce agli sviluppatori di semplici sistemi embedded un veloce strumento di prototipazione, grazie alla discreta potenza di calcolo ed alla possibilità di essere interfacciata con altre schede o sensori (presenta infatti ingressi e uscite analogici e digitali), consentendo la rapida creazione di sistemi relativamente complessi.

Può venire alimentata:

- attraverso la connessione USB;
- fornendo una tensione compresa nel range $+7 \div +12\ V$ al connettore jack apposito;
- attraverso il pin V_{IN} presente sul connettore a bordo scheda.

Negli ultimi due casi la tensione di alimentazione ($+5\ V$) richiesta dal chip ATmega328 viene ricavata attraverso il regolatore di tensione lineare $NCP1117$ già presente sulla scheda.

In questo progetto viene adottata l'ultima delle tre opzioni; è stato necessario interporre tra il convertitore DC/DC e l'ingresso V_{IN} dell'Arduino un regolatore di tensione lineare $L7808$ che abbassi la tensione $+12\ V$ a $+8\ V$, per evitare i problemi di surriscaldamento del regolatore $NCP1117$.

Si ricorda che i regolatori lineari si comportano come dei resistori variabili posti in serie tra una tensione di ingresso ed una di uscita: variando il loro valore di resistenza equivalente, varia anche la caduta di tensione ai loro capi.

Ne risulta che devono dissipare, in calore, una potenza pari a:

$$P = (V_{IN} - V_{OUT}) \cdot I_{OUT}\ ;$$

quindi, diminuendo la tensione d'ingresso V_{IN} da $+12V$ a $+8V$, il regolatore $NCP1117$ a bordo dell'Arduino deve dissipare circa metà della potenza iniziale, a parità di corrente di uscita.

L'altra metà di potenza viene invece dissipata dal regolatore $L7808$, ai cui capi si trova una caduta di tensione di $12\ V - 8\ V = 4\ V$.

Il cuore dell'Arduino Uno è l'ATmega328, un microcontrollore che lavora alla frequenza di 16 MHz; è un microcontrollore di tipo RISC (Reduced Instruction Set Computer) a 8 bit e si basa sull'architettura Harvard, la quale prevede che la memoria in cui risiede il programma sia fisicamente separata dalla memoria di elaborazione (o memoria dati).

Questo implica la coesistenza di due bus, ovvero uno per l'accesso alla memoria dati ed uno per quella programma; il core del microcontrollore può accedere in contemporanea ad entrambe, raggiungendo così prestazioni più spinte rispetto all'architettura Von Neumann (dove esiste un unico bus per l'accesso al programma ed alla memoria dati). Su quest'ultima architettura si basano invece vari microprocessori.

Il microcontrollore esegue quindi il programma salvato sulla propria memoria interna (di tipo FLASH e ampia 32 *kbyte*), mentre utilizza per le elaborazioni una memoria SRAM (che ammonta a 2 *kbyte*); dispone anche di una memoria EEPROM non volatile di 1 *kbyte*.

Come la quasi totalità dei microcontrollori in commercio, dispone internamente di varie periferiche, che consentono di realizzare sistemi completi senza un uso eccessivo di componenti esterni; tra le più importanti ci sono:

- due timer da 8 bit ed uno da 16 bit, ognuno con relativo prescaler;
- un convertitore Analogico-Digitale (ADC) single-ended a 10 bit, che può acquisire un valore analogico di tensione da sei diversi ingressi. Il

riferimento di tensione per la conversione può essere impostato sui $+5\,V$ (prodotti dal regolatore di tensione con un tolleranza dichiarata dell' 1%), sul riferimento di tensione interno di $+1.1\,V$, o su una tensione di riferimento fornita esternamente sul pin $AREF$;

- un modulo per la comunicazione su bus Serial Peripheal Interface (SPI) ed uno per il bus Inter Integrated Circuit (I^2C);
- fino a sei uscite configurabili in modalità PWM.

L'utilizzo di ogni periferica è reso molto semplice grazie alle svariate librerie di funzioni disponibili.

La programmazione del chip ATmega328 presente sull'Arduino Uno avviene tramite una connessione seriale; a questo scopo sulla scheda è presente un piccolo microcontrollore (ATmega8U2), programmato in modo da convertire la comunicazione USB in una seriale compatibile con i livelli fisici dello standard TTL (supportata dall'ATmega328).

Durante il caricamento di un nuovo programma, il chip ATmega8U2 fa lavorare l'ATmega328 in modalità bootloader, ovvero viene eseguito una parte di firmware speciale (che risiede in una parte di memoria programma dell'ATmega328 disabilitata alla scrittura) che provvede a ricevere il nuovo programma (tramite comunicazione seriale) e a salvarlo in memoria sovrascrivendo quello presente.

Tutto il procedimento di caricamento di un nuovo programma richiede pochi secondi, e viene reso molto semplice dall'interfaccia utente, eseguibile su sistemi operativi Windows, Linux o Mac. La stessa interfaccia utente provvede anche a compilare il codice scritto in linguaggio C.

2.3 La scheda Arduino Ethernet Shield

La scheda Arduino Ethernet Shield è una scheda di espansione che si innesta sui connettori dell'Arduino Uno; consente al microcontrollore dell'Arduino di interfacciarsi con la rete Internet, collegandole un comune cavo *Ethernet*, e di comunicare con una scheda di memoria *microSD*, la quale viene inserita nell'apposito slot.

Nella memoria SD risiedono le pagine HTML che costituiscono l'interfaccia utente del sistema; la scelta di dislocarle sulla memoria di SD, piuttosto che codificarle all'interno del firmware dell'ATmega328, è stata obbligata dalla limitata quantità di memoria dello stesso chip.

Con il collegamento alla rete si rende possibile il monitoraggio dello stato delle batterie anche da remoto, attraverso un semplice computer su cui sia installato un browser Web.

Per quanto riguarda la parte relativa alla comunicazione con la rete, la scheda Ethernet Shield monta il controller Ethernet Wiznet W5100, il quale comunica con il microcontrollore dell'Arduino Uno attraverso il bus seriale *SPI*. Anche la comunicazione con la scheda di memoria microSD viene gestita con il bus *SPI*, e quindi le due comunicazioni non possono avere luogo contemporaneamente.

La tensione di alimentazione per il chip W5100 viene ricavata dai $+5\ V$ dell'Arduino Uno, dopo essere ridotta a $+3.3\ V$ da un regolatore di lineare.

Il numero di connessioni contemporanee socket massimo supportato è limitato a 4; un socket rappresenta un canale di comunicazione tra un dispositivo Client ed un Server.

2.4 I trasduttori di tensione e corrente

Per acquisire i valori di tensione e corrente sono stati utilizzati due trasduttori isolati prodotti dalla LEM. L'isolamento consente di eseguire misure senza entrare in contatto fisico con potenziali pericolosi; infatti in questi trasduttori il circuito primario di ingresso (collegato al circuito di potenza, sul quale si effettua la misura) ed il circuito secondario di uscita (che fornisce un certo segnale in uscita proporzionale a quello in ingresso a primario) sono elettricamente isolati.

Entrambi i trasduttori usano la tecnologia con sensore ad effetto Hall, che consente di effettuare misure di correnti a partire da quelle di campo magnetico.

Con questo metodo di misura è possibile misurare in modo diretto valori di corrente (e in modo indiretto anche valori di tensione) senza influenzare minimamente il circuito sotto misura: infatti se è noto il campo magnetico H prodotto dalla corrente I, che scorre in un filo posto ad una distanza r dal punto di misura, è possibile ricavare il valore stesso di corrente secondo la legge di Biot-Savart:

$$B = \mu \cdot H = \frac{\mu}{2\pi} \frac{I}{r} \tag{2.1}$$

dove $\mu = \mu_0 \cdot \mu_r$ è la permeabilità magnetica del mezzo in cui viene misurato il campo magnetico e B è l'induzione elettromagnetica.

Ovviamente il fatto di eseguire una misura di corrente senza interferire con il segnale da misurare consente di annullare le dissipazioni di potenza che si otterrebbero invece con la classica misura di corrente che utilizza lo shunt resistivo; quest'ultimo metodo è stato scartato in partenza visto che

le potenze in gioco in questa applicazione avrebbero portato a perdite e problemi di dissipazione non trascurabili.

Nella pratica della misura, i sensori ad effetto Hall che si possono trovare in commercio, lavorano in modalità ad 'anello aperto' o ad 'anello chiuso'.

Per effettuare una misura di corrente incognita I_x, che scorre nel circuito primario, con la prima modalità si pone una sottile lastra di materiale semiconduttore di spessore d (elemento di Hall) nel traferro di un circuito magnetico; questa lastra viene percorsa longitudinalmente da una corrente di controllo I_c, fornita da un apposito circuito.

La corrente I_x, percorrendo un conduttore avvolto con un certo numero di spire attorno al circuito magnetico, genera un flusso che viene concentrato nel circuito magnetico stesso (grazie alla sua elevata permeabilità) e rilevato dall'elemento di Hall; ai suoi capi si genera una differenza di potenziale V_H dovuta alla diversa distribuzione dei portatori di carica per colpa del campo magnetico H (la forza di Lorentz risulta diretta perpendicolarmente alla direzione dei portatori di carica che compongono la corrente I_c).

La V_H, detta 'tensione di Hall' in onore del fisico statunitense Edwin Herbert Hall che nel 1879 scopri l'effetto Hall, risulta essere proporzionale alla corrente di controllo I_c e all'induzione elettromagnetica B, ed inversamente proporzionale allo spessore d, secondo una costante K_H che tiene conto delle proprietà del materiale utilizzato per la lastra:

$$V_H = K_H \frac{I_c \cdot B}{d} \ . \tag{2.2}$$

Da qui è possibile ricavare il valore di induzione elettromagnetica B, e ricavare successivamente dall'equazione 2.1 il valore della corrente incognita.

La modalità ad anello chiuso, invece, si basa sulla compensazione del flusso magnetico prodotto dal circuito primario.

Il circuito di partenza di un trasduttore di corrente che lavora ad anello chiuso è lo stesso di quello ad anello aperto, al quale però viene aggiunto un controllo in retroazione.

L' elemento di Hall, inserito nel traferro, risente del flusso magnetico generato dalla corrente I_P nel circuito primario e ai suoi capi si genera una tensione ad essa proporzionale; questa tensione viene convertita in una corrente I_S, da un circuito di controllo, e viene fornita ad una bobina avvolta sul circuito magnetico. Si fa in modo che il flusso generato dalla I_S annulli quello generato dalla I_P.

In questo modo l'andamento della corrente I_S insegue quello della I_P, a meno di una costante moltiplicativa che dipende dal rapporto spire esistente

tra circuito primario e secondario, e dal condizionamento del segnale svolto dal circuito di controllo.

Solitamente i trasduttori di corrente ad anello chiuso offrono prestazioni migliori a livello di accuratezza: risultano anche meno influenzati dalle variazioni di temperatura (sono compensati).

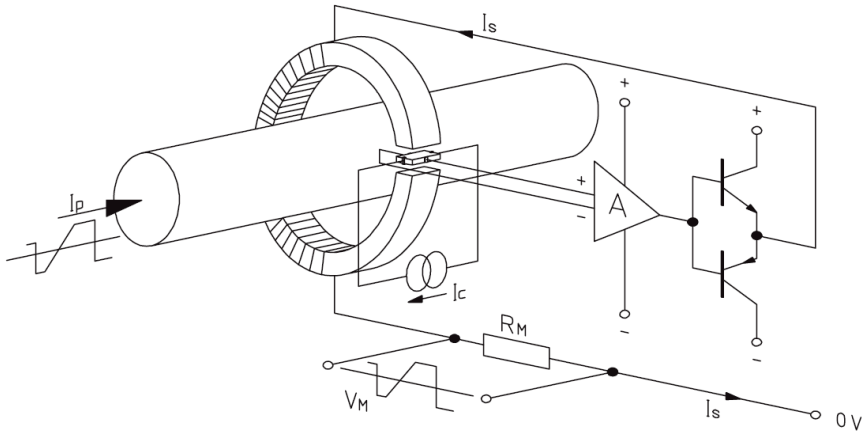


Figura 2.2: Trasduttore di corrente ad effetto Hall a catena chiusa

Il trasduttore di corrente utilizzato è il circuito integrato LEM FHS 40-P, montato sulla scheda di valutazione KIT 9 (figura 2.4).

Viene alimentato a $+5V$ e fornisce in uscita una tensione nel range $+0.5 \div +4.5V$, a seconda della corrente in ingresso: la sua uscita assume il valore di $2.5V$ (pari alla tensione di riferimento V_{ref} interna) quando la corrente è nulla.

In alternativa, la tensione di riferimento interno può essere impostata dall'esterno, applicando al pin V_{ref} una tensione compresa nell'intervallo $2 \div 2.8\ V$.

È possibile porre in modalità standby il LEM FHS 40-P, portando a $+5\ V$ il suo pin *Standby*, per ridurre il consumo di corrente (che in condizioni operative non dovrebbe superare i 19 mA).

È del tipo a 'catena aperta' e consente la misura di correnti fino a circa $\pm 78A$ con una sensibilità di 25.8 mV/A (valori dettati dalla geometria costruttiva del KIT 9).

La corrente da misurare scorre sulla pista presente sul un lato del KIT 9; in sua corrispondenza, ma dall'altro lato del circuito stampato, si trova l'integrato FHS 40-P che rileva e misura il campo magnetico prodotto.

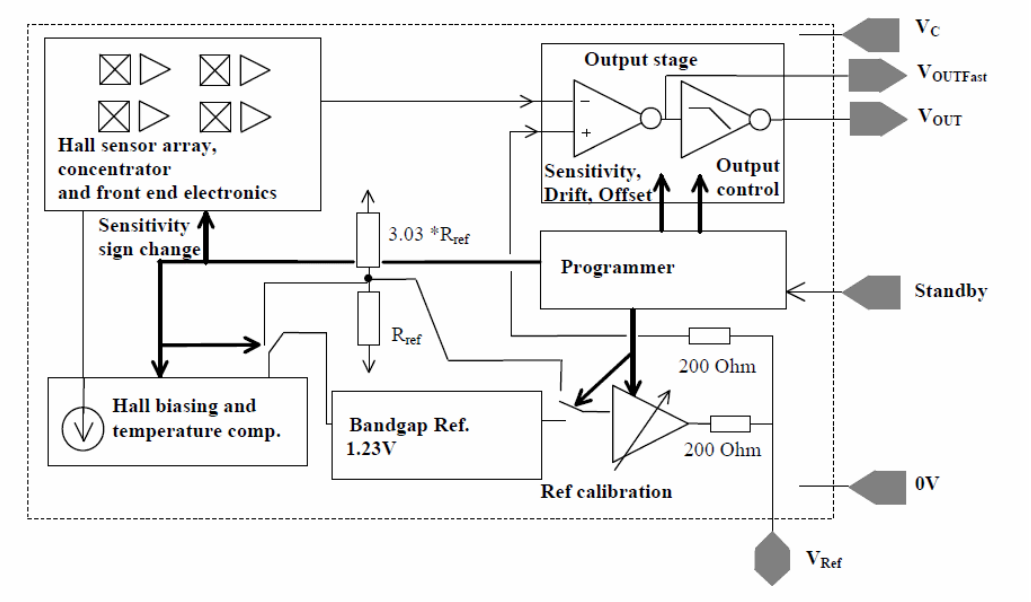


Figura 2.3: Struttura interna LEM FHS 40-P

Il trasduttore di tensione utilizzato è il LEM LV 25-P (figura 2.5); è del tipo a 'catena chiusa'. Viene alimentato con una tensione duale di $\pm 12\ V$ e consente la misura di tensioni da ± 10 fino a $\pm 500\ V$.

Misura indirettamente la tensione grazie ad una resistenza R_1 montata esternamente in serie all'avvolgimento primario, la quale crea la corrente I_P proporzionale alla tensione da misurare V_{IN}; trascurando quindi la resistenza dell'avvolgimento primario si ottiene:

$$I_P = \frac{V_{IN}}{R_1}$$

Nell'avvolgimento secondario di uscita viene generata una corrente I_S ad essa proporzionale, che dipende dal rapporto di conversione $K_N = \frac{I_S}{I_P} = \frac{2500}{1000}$; una resistenza esterna di misura R_M fornisce la tensione di uscita V_{OUT} che viene poi misurata:

$$V_{OUT} = R_M \cdot I_S = R_M \cdot K_N \cdot I_P = R_M \cdot K_N \cdot \frac{V_{IN}}{R_1} \tag{2.3}$$

Invertendo la 2.3 si ottiene la 2.4, che permette di conoscere il valore di tensione applicata in ingresso a partire dalla misura di quella di uscita:

$$V_{IN} = \frac{R_1}{R_M \cdot K_N} V_{OUT} \tag{2.4}$$

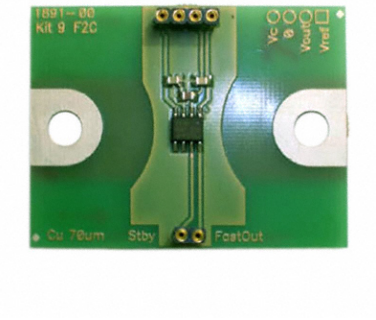


Figura 2.4: LEM FHS 40-P su scheda KIT 9

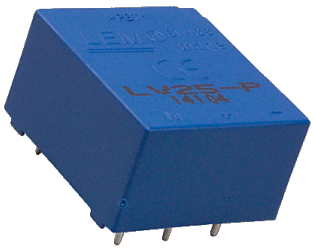


Figura 2.5: LEM LV 25-P

Come specifica il datasheet, il valore di R_1 deve essere calcolato in modo da produrre una corrente nominale a primario $I_{P_N} = 10mA$, alla tensione nominale da misurare. Viene scelto perciò:

$$R_1 = \frac{300}{0.01} = 30\ k\Omega.$$

Per la scelta del valore di R_M il datasheet specifica che, con una corrente a primario massima di $\pm 10\ mA$ e alimentazione a $\pm 12\ V$, non si può andare oltre i 190 Ω: viene scelto il resistore di valore commerciale più prossimo con una tolleranza del 1% , cioè 187 Ω.

2.5 Il sensore di temperatura

È stato necessario collegare all'Arduino anche un sensore di temperatura.

Essa produce un effetto diretto di variazione della capacità delle batterie, secondo la tabella riportata dal costruttore (figura 2.6).

A questo scopo viene utilizzato l'integrato LM35, il quale necessita solamente di essere alimentato (da 4 a 20 V) per fornire in uscita un segnale analogico di tensione, che viene poi acquisito tramite una conversione A/D dall'Arduino. La relazione tra tensione di uscita prodotta dal sensore e la temperatura è lineare: con l'aumento di un grado centigrado la tensione generata aumenta di 10 mV.

Temperature	Correction Factor
5°C	0.84
10°C	0.88
15°C	0.93
20°C	0.97
25°C	1.00
30°C	1.03
35°C	1.05
40°C	1.07

Figura 2.6: Fattore di correzione della capacità in base alla temperatura

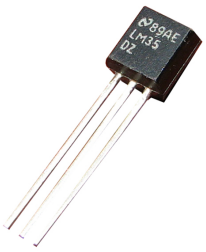


Figura 2.7: Sensore di temperatura LM35

2.6 Il bus I^2C

Il bus I^2C è un bus seriale che consente la comunicazione tra almeno un dispositivo che viene chiamato MASTER (che gestisce il bus) ed uno SLAVE.

Richiede la presenza di due sole linee collegate in comune tra i dispositivi che comunicano (chiamate SDA ed SCL), oltre al riferimento comune di massa.

Alla linea di alimentazione, che può essere condivisa da tutti i dispositivi, si collegano le due resistenze di pull-up, che in stato di riposo (nessun dispositivo che trasmette) mantengono a livello alto le linee SDA ed SCL.

Ogni dispositivo deve avere un indirizzo univoco che lo identifica nel bus.

Solo un dispositivo alla volta può comunicare, ed è sempre uno di tipo MASTER che inizia la trasmissione.

Esso fornisce il segnale di clock agli altri dispositivi sulla linea SCL: si tratta quindi di un bus sincrono.

Ogni bit inviato prevede che la linea SDA possa commutare mentre il segnale su SCL assume valore basso; fanno eccezione i bit di START e di STOP, che delimitano una trasmissione, i quali seguono le regole sottostanti:

- lo start è costituito da una transizione da alto a basso della linea SDA mentre SCL è alto;

- lo stop è rappresentato da una transizione da basso ad alto di SDA mentre SCL è alto.

La comunicazione su questo bus può avvenire a diverse velocità, dipendentemente dalle velocità di trasmissione supportate dai dispositivi: esistono le modalità *standard mode* (velocità di trasmissione di 100 $kbit/s$), $fast\ mode$ (fino a 400 $kbit/s$) e *high speed* (per collegamenti veloci fino a $3,4\ Mbit/s$).

Il numero massimo di dispositivi che possono essere presenti sullo stesso bus è limitato dai 7 bit riservati all'indirizzo: sono perciò supportati fino a $2^7 = 128$ dispositivi connessi contemporaneamente, ognuno con un indirizzo identificativo diverso.

Nella pratica, però, è raro trovare così tanti dispositivi connessi ad un bus I^2C perchè un ulteriore limite è costituito dalla capacità massima sopportata dal bus: 400 pF.

Bisogna quindi dimensionare correttamente le resistenze di pull-up per garantire la corretta propagazione del segnale da un dispositivo all'altro.

2.6.1 Il modulo RTC

Il modulo Real Time Clock (figura 2.8) consente all'Arduino di avere a disposizione un riferimento temporale da assegnare ai vari campioni analogici acquisiti. Consiste in una piccola schedina che monta l'integrato DS1307, un orologio/calendario a basso consumo che dialoga con la scheda Arduino sul bus seriale I^2C, al quale accede in modalità SLAVE.

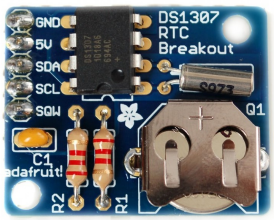


Figura 2.8: Modulo RTC con DS1307

Il conteggio del tempo si basa su un oscillatore che lavora con un quarzo da 32768 kHz, il quale garantisce una buona precisione.

Per poter comunicare con il circuito esterno, l'integrato DS1307 deve essere alimentato da una sorgente di tensione primaria V_{cc} di $+5$ Volt, ma consente di mantenere il conteggio del tempo anche con una sola batteria di backup da 3 Volt (V_{bat}).

L'integrato DS1307 sceglie automaticamente la sua sorgente di alimentazione: seleziona quella primaria se $V_{cc} > V_{bat} + 0.2\ V$; in questo caso permette la comunicazione I^2C solo se $V_{cc} > 1.25 \cdot V_{batt}$. Invece, se $V_{cc} < V_{bat} + 0.2\ V$ seleziona la batteria di backup, la quale gli consente di lavorare per svariati anni senza però poter comunicare con il circuito esterno (modalità basso consumo).

2.6.2 Il convertitore A/D MCP3221

Si tratta di un convertitore A/D a 12 bit a basso consumo che comunica in digitale sul bus I^2C.

Supporta le modalità di comunicazione *standard* e $fast$; nel modo $fast$, lavorando in acquisizione continua, riesce ad arrivare ad una frequenza di campionamento di 22.3 $ksps$.

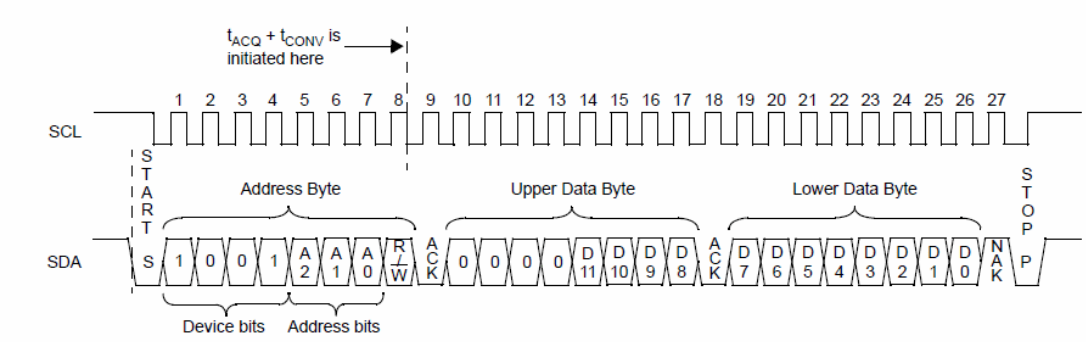


Figura 2.9: Modalità acquisizione singola MCP3221

Come per la comunicazione con il modulo RTC, per iniziare la comunicazione con il chip MCP3221 bisogna che il MASTER invii il bit di START seguito dall'indirizzo identificativo *0x4d*.

L'acquisizione e la conversione del segnale in ingresso iniziano con il fronte di discesa del bit $R/\overline{W}$; a questo punto il clock interno inizia la gestione del campionamento e della conversione, indipendentemente dal clock di trasmissione usato sul bus.

Vengono poi inviati al MASTER i due byte contenenti il risultato della conversione A/D, partendo dal byte più significativo, come mostra il diagramma di figura 2.9.

Capitolo 3

Sezione Software

Il software del sistema di acquisizione consiste in una parte di firmware scritto in linguaggio C, eseguito dall'Arduino, e da una parte composta da pagine Web scritte in linguaggio HTML, che risiedono nella scheda di memoria inserita nello slot dell'Ethernet Shield.

Le due parti dialogano tra loro scambiandosi dati e/o comandi.

3.1 Il firmware

Un tipico programma Arduino (o sketch) prevede di essere diviso in due funzioni principali: *setup*() e *loop*().

La prima contiene tutte le istruzioni che hanno lo scopo di inizializzare il microcontrollore, e per questo viene eseguita una sola volta subito dopo l'accesione (o dopo un evento di RESET hardware o software).

La seconda contiene le istruzioni del vero e proprio programma che l'Arduino esegue ciclicamente dopo aver eseguito *setup*().

Il flusso di esecuzione semplificato del firmware dell'Arduino viene riassunto nella figura 3.1.

Vengono ora analizzate separatamente le varie funzionalità del firmware, facendo riferimento alle librerie utilizzate.

3.1.1 La memoria microSD

A causa della limitata quantità di RAM del chip ATmega328, la presenza della scheda SD risulta di fondamentale importanza per questo progetto; in mancanza della stessa infatti non sarebbe possibile salvare i risultati delle acquisizioni, ed il firmware si limiterebbe a fornire al Client solamente l'ultimo record presente in memoria.

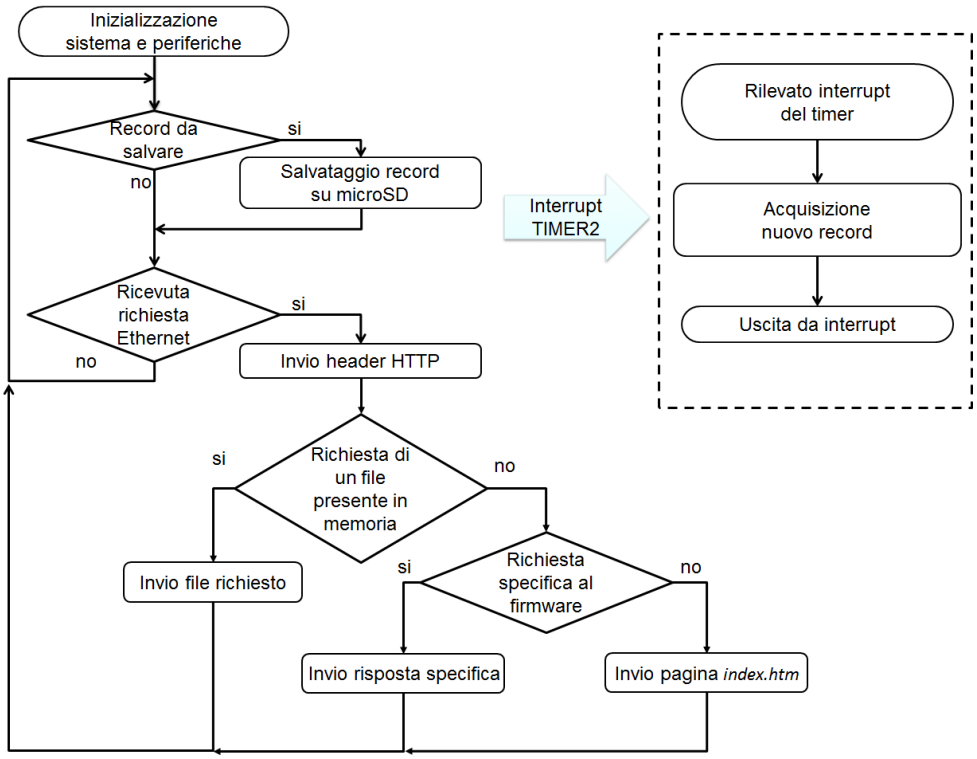


Figura 3.1: Diagramma di flusso del firmware

Come già detto, sia la comunicazione con la scheda di memoria SD che quella con il controller Ethernet avvengono sul bus SPI. Viene perciò usata la relativa libreria $SPI.h$ che ne implementa il protocollo.

La libreria $SD.h$ invece consente di effettuare operazioni di lettura/scrittura su schede di memoria con filesystem FAT, con la limitazione di usare nomi dei file compatibili con la convenzione 8.3: si possono cioè gestire solamente file il cui nome sia composto al massimo da 8 caratteri di nome ed ulteriori 3 per l'estensione.

Di questa libreria vengono usate le funzioni:

- $SD.begin()$: provvede ad inizializzare la comunicazione con la scheda e restituisce un valore di tipo *boolean* pari a *true* se la scheda è stata rilevata ed inizializzata correttamente, $false$ in caso contrario. Riceve come argomento il numero del pin dell'ATmega328 (pin 4) connesso alla linea *Chip Select* della scheda;

- $SD.exixts(filename)$: restituisce *true* se sulla scheda è presente il file richiesto, $false$ altrimenti;

- $SD.open(filepath, mode)$: apre un file presente su scheda in modalità lettura o scrittura e posiziona il puntatore all'inizio del file. Restituisce un oggetto di tipo $File$ su cui possono essere eseguite le funzioni di lettura e scrittura;

- $SD.close(filepath)$, chiude un file precedentemente aperto;

- $file.read()$, restituisce il prossimo carattere letto dal file ed avanza il puntatore di lettura;

- $file.write(char)$, scrive il carattere passato come argomento nel file ed avanza il puntatore di scrittura.

A causa di una limitazione della libreria $SD.h$, nel caso la scheda venisse inserita solamente in un momento successivo all'accensione del sistema, si dovrebbe procedere al RESET del sistema stesso, al fine di rilevare correttamente la presenza della scheda ed abilitare il salvataggio dei dati.

Il metodo di salvataggio dei dati acquisiti prevede la creazione giornaliera di un file in cui vengono salvati tutti i record relativi al giorno stesso: il nome del file corrisponde alla data a cui si riferisce.

Ogni file viene scritto in formato testo rispettando la struttura del formato Comma Separated Values (CSV), per renderne agevole l'apertura con programmi di calcolo, ed è composto di tante righe quanti sono i record salvati in esso, oltre che da una riga iniziale che ne costituisce l'intestazione.

Come già detto, ogni record è a sua volta composto, nell'ordine, dal riferimento temporale della singola acquisizione, dalla tensione, dalla corrente, dalla temperatura e dallo stato di carica stimato; all'interno del singolo file ogni campo viene separato con un delimitatore (;).

È possibile calcolare lo spazio occupato dai file di log se è nota la frequenza di salvataggio del singolo record, $f_{nuovo\ record}$, e la sua dimensione in byte, $N_{byte\ record}$.

Considerando che viene salvato un nuovo record ogni 5 secondi ($f_{nuovo\ record} = \frac{1}{5} = 0.2Hz$) dopo un certo tempo t dall'avvio del sistema sulla memoria verranno salvati:

$$N_{byte}(t) = (f_{acquisizione} \cdot N_{byte\ record}) \cdot t \tag{3.1}$$

nuovi byte.

Quindi sapendo che $N_{byte\ record}$ può valere al massimo 38 byte (8 byte per l'ora + 6 byte per la tensione + 6 byte per la corrente + 6 byte per la temperatura + 6 byte per lo SoC + 2 byte per il fineriga + 4 byte per

i separatori), si ottiene che viene prodotto un file di circa 0.63 Mbyte ogni giorno.

In questo modo, considerando una registrazione senza interruzioni, verrebbero memorizzati circa 230 Mbyte in un anno; dato che la scheda di memoria utilizzata ha una capacità di 2 Gbyte il sistema di registrazione ha un'autonomia maggiore di 8 anni.

3.1.2 La comunicazione Ethernet

Grazie alla scheda Arduino Ethernet Shield, l'Arduino Uno riesce a connettersi alla rete Internet come Client o come Server; in questa applicazione viene configurato come Server, che invia dati ai Client che ne fanno richiesta.

Per abilitare l'Arduino alla comunicazione con il protocollo TCP, bisogna definire nel firmware il valore dell'indirizzo MAC (Media Access Control) e l'indirizzo IP (Internet Protocol) che l'Arduino utilizzerà durante la comunicazione.

La libreria che si occupa di gestire la comunicazione Ethernet è la *Ethernet.h*; in questo progetto l'Arduino lavora come Server su un IP statico, ma la libreria supporta anche la funzionalità DHCP, che consente di assegnare automaticamente al Server Arduino un IP tra quelli disponibili.

Ovviamente è possibile variare i parametri funzionali per la connessione, quali MAC, indirizzo IP e porta di comunicazione (listato 3.1).

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // indirizzo
    MAC
IPAddress ip(147, 162, 11, 104); // indirizzo IP
EthernetServer server(80); // porta di comunicazione
```

Listing 3.1: Parametri per la connessione Ethernet

Dopo aver abilitato la comunicazione con l'Ethernet Shield, portando a livello alto la corrispondente linea *Chip Select* del bus *SPI* (connessa al pin 10 dell'ATmega328), viene eseguita la funzione *Ethernet.begin(mac, ip)*, che abilita la comunicazione con i valori di MAC e IP specificati, seguita da *server.begin()*, che attiva l'ascolto delle connessioni in entrata.

Successivamente, il Server controlla ciclicamente se qualche Client ha inviato qualche richiesta (mediante l'istruzione *server.available()*); in caso affermativo il Server attende di aver ricevuto una richiesta HTTP completa.

Il contenuto della richiesta HTTP viene salvato in una stringa nella memoria RAM dell'Arduino, per poterne estrarre la funzione specifica richiesta al Server, con gli eventuali parametri.

Viene sempre assunto che la richiesta che il Client rivolge al Server sia di tipo GET (per semplicità il firmware non considera l'eventualità di ricevere una richieste di altro tipo, come il POST).

Per rispondere al Client, il Server usa la funzione *client.print(char *)* (o la *client.println(char *)*) che serve a inviare una stringa di risposta.

Normalmente le stringhe passate come argomento alla funzione *client.print(char *)* vengono memorizzate nella memoria RAM, ma con la macro *F(char *)* è possibile indicare al compilatore di salvarle direttamente nella memoria programma FLASH, liberando così preziosa memoria RAM richiesta per le elaborazioni.

Una tipica richiesta di tipo GET, sul protocollo HTTP, può essere la seguente:

```
GET /pagina.htm?parametro1=a&parametro2=b HTTP/1.0
```

Listing 3.2: Esempio di richiesta GET

In essa vengono specificati:

- il tipo di richiesta (*GET*);
- il nome completo del file che il Client richiede (*/pagina.htm*) seguito da parametri opzionali (che nell'esempio sono *parametro1=a* e *parametro2=b*);
- la versione del protocollo HTTP utilizzato dal Client (*HTTP/1.0*).

Ogni campo viene separato da uno spazio; possono seguire anche altre informazioni facoltative, ma che non vengono prese in considerazione dal firmware.

Il firmware dell'Arduino analizza quindi la richiesta HTTP ricevuta con il metodo *richiestaGET(char *)*, visibile nel listato 3.3, individuando il nome del file richiesto e gli eventuali parametri.

Poi controlla che il nome del file richiesto corrisponda ad un file presente nella scheda di memoria: in caso affermativo lo invia al Client, ma non prima di aver inviato la tipica intestazione delle risposte HTTP, che segnala al Client lo stato della richiesta. Il testo contenuto nell'intestazione viene mandato direttamente con le istruzioni viste sopra (consiste in più stringhe salvate nella memoria programma dell'Arduino, come mostra il listato 3.4); ad esso segue il contenuto vero e proprio della pagina HTML.

Ogni file memorizzato sulla scheda SD (comprese le pagine HTML) viene inviato al Client usando la funzione *invioFile()* dopo essere stato caricato in memoria RAM in blocchi di byte (listato 3.5).

```
// Estrae il nome del file richiesto ed i parametri dalla
    stringa passata come argomento
boolean richiestaGET(char * buffer) {
  boolean parametriPresenti=false;
  unsigned char cursore=5;
  nomeFileRichiesto = &buffer[5];
  while ( (buffer[cursore] != ' ') && (cursore <
     BYTE_BUFFER_HTTP) ) {
    if( ( (buffer[cursore] == '&') || (buffer[cursore] == '?')
       ) && (cursore < BYTE_BUFFER_HTTP-1) ) {
      buffer[cursore]=NULL; // terminatore
      if(!parametriPresenti) {
        parametriRichiestaGET = &buffer[cursore+1];
        parametriPresenti=true;
      }
    }
    cursore++;
  }
  buffer[cursore]=NULL; // terminatore
  if(!parametriPresenti)
    parametriRichiestaGET = &buffer[cursore];

  if(nomeFileRichiesto[0] == NULL)
    return false;
  return true;
}
```

Listing 3.3: Funzione per l'analisi della richiesta GET

Se invece il nome del file che il Client richiede con il metodo GET non rappresenta un file presente in memoria, il firmware verifica se corrisponde a un comando per cui è stata prevista una risposta specifica, confrontando la stringa ricevuta con l'insieme dei comandi definiti mediante il metodo *comandoGET*(*char* *) (listato 3.6).

In questo caso, i comandi che prevedono una risposta da parte del firmware sono i seguenti:

- *nuovoDato*: il Server invia al Client solamente l'ultimo record acquisito usando il metodo *invioUltimoRecord(client)* (listato 3.7);
- *cercaFile*: il Server verifica la presenza del file passato come parametro sulla scheda di memoria e risponde al Client con un '1' se la verifica ha ha esito positivo, oppure con uno '0' altrimenti;
- *eliminaFile*: il Client fa richiesta al Server di eliminare un file presente su scheda di memoria (anche in questo caso il nome del file corrisponde

```
client.println(F("HTTP/1.1 200 OK"));
client.println(F("Content-Type: text/html"));
client.println(F("Connnection: keep-alive"));
client.println();
```

Listing 3.4: Istruzioni per l'invio dell'intestazione HTTP

al parametro della richiesta): il Server risponde con un '1' se riesce ad eliminare il file, con uno '0' altrimenti;

- *data*: il Server manda al client la data del file su cui sta attualmente salvando i record: corrisponde ad una stringa nel formato *dd-MM-yy*;
- *RTC*: il Client fa richiesta di aggiornare la data/ora del Real Time Clock, passandone per parametri i nuovi valori, la cui validità è stata già verificata dal Client mediante le funzione Javascript *dataValida()* e *oraValida()* discusse più avanti;
- *reset*: il Client richiede il reset della scheda Arduino tramite WDT (vedi sezione 3.1.5).

Per fare un esempio, nel caso il Client inviasse una richiesta di tipo *cercaFile* al fine di verificare la presenza del file *log_file.txt*, allora la parte di richiesta GET analizzata dall'Arduino sarà costituita dalla stringa:

```
GET /cercaFile?log_file.txt HTTP/1.0
```

Listing 3.8: Esempio di comando attraverso richiesta GET

3.1.3 La comunicazione I^2C

La libreria *Wire.h* viene inclusa per poter comunicare, sul bus I^2C, con l'integrato DS1307 e col convertitore A/D MCP3221 addetto alla misura della corrente.

All'interno della libreria, le funzioni *Wire.beginTransmission()* e *Wire.endTransmission()* si occupano di inizializzare e terminare una trasmissione, gestendo le tempistiche e le modalità di trasmissione del bus, mentre le funzioni *Wire.read()* e *Wire.write(char)* servono a ricevere e inviare un byte ad un dispositivo connesso al bus.

L'integrato DS1307 memorizza l'ora e la data attuali in formato BCD, salvandoli in più registri interni da 8 bit ciascuno: dispone infatti di una piccola quantità di memoria, di cui i primi 8 byte (indirizzi *00h-07h*) sono

```
// Invia al client il file fileRichiesto
boolean invioFile(char * nomeFile, EthernetClient client) {
  if (!schedaPresente)
    return false;
  File file=SD.open(nomeFile, FILE_READ);
  if(file) {
    boolean continua=true;
    while(continua) {
      char blocco[BYTE_BLOCCO];
      for(unsigned char i=0; i<BYTE_BLOCCO; i++) {
        if(file.available())
          blocco[i]=(char)file.read();
        else {
          continua=false;
          blocco[i]=NULL;
        }
      }
      blocco[BYTE_BLOCCO]=NULL;
      client.print(blocco);
    }
    file.close();
    return true;
  }
  else
    return false;
}
```

Listing 3.5: Funzione per l'invio dei file presenti su scheda di memoria. Questa funzione ha permesso di aumentare di 4 volte la velocità di invio dei dati rispetto all'invio di un singolo carattere: si è passati da circa 4 kbyte/s fino a circa 16 kbyte/s.

riservati al Real Time Clock, mentre i seguenti 56 (indirizzi *08h-FFh*) non sono utilizzati e possono essere sfruttati come RAM non volatile. La lettura/scrittura dei registri viene gestita internamente usando un puntatore al registro in uso (REGISTER POINTER).

Alla prima accensione dell'integrato DS1307 è necessario che il dispositivo che accede in modalità MASTER al bus (l'Arduino) gli imposti la data e l'ora, seguendo la procedura di scrittura su bus I^2C indicata nel datasheet:

- il dispositivo MASTER invia una condizione di START seguita dall'indirizzo dello SLAVE (*0x68* per il DS1307) e dal bit di direzione $R/\overline{W}$ (impostato a 0 per un'operazione di scrittura)

- lo SLAVE risponde al MASTER con un bit di ACKNOWLEDGE

```
// Confronta la stringa passata per parametro e restituisce "true
    " se corrisponde al contenuto di nomeFileRichiesto
boolean comandoGET(char * stringa1) {
  unsigned char lunghezza1=0;
  unsigned char lunghezza2=0;
  while( (stringa1[lunghezza1] != NULL) && (lunghezza1 <
      BYTE_BUFFER_HTTP) )
    lunghezza1++;
  while( (nomeFileRichiesto[lunghezza2] != NULL) && (lunghezza2 <
      BYTE_BUFFER_HTTP) )
    lunghezza2++;
  if(lunghezza1 != lunghezza2)
    return false; // stringhe di lunghezza diversa sono diverse
  unsigned char i=0;
  while( (stringa1[i] == nomeFileRichiesto[i]) && (i < lunghezza1
      ) )
    i++;
  if(i == lunghezza1)
    return true;
  return false;
}
```

Listing 3.6: Funzione che identifica il comando ricevuto

- il MASTER invia un primo byte di dati, contenente l'indirizzo del registro interno dal quale iniziare la scrittura
- lo SLAVE salva l'indirizzo ricevuto nel REGISTER POINTER e risponde al MASTER con un bit di ACKNOWLEDGE
- il MASTER invia in sequenza i byte di dati da scrivere nei registri dello SLAVE
- lo SLAVE risponde al MASTER con un bit di ACKNOWLEDGE ad ogni byte ricevuto ed incrementa il valore del REGISTER POINTER; termina la comunicazione quando riceve una condizione di STOP dal MASTER

Ovviamente questa procedura viene seguita anche nel caso in cui venisse a mancare l'alimentazione e si dovesse reimpostare la data/ora del RTC.

Per implementare la procedura di scrittura è stata scritta la funzione

void scriviRTC(byte ore, byte minuti, byte giorno, byte mese, byte anno)

```
// Invia al client l'ultimo record presente in memoria
void invioUltimoRecord(EthernetClient client) {
  client.print(ora);
  client.print(';');
  client.print(tensioneMedia);
  client.print(';');
  client.print(correnteMedia);
  client.print(';');
  client.print(temperaturaMedia);
  client.print(';');
  client.print(SoC);
}
```

Listing 3.7: Funzione per l'invio dell'ultimo record memorizzato

ADDRESS	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	FUNCTION	RANGE
00H	CH	10 Seconds			Seconds				Seconds	00–59
01H	0	10 Minutes			Minutes				Minutes	00–59
02H	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/AM							
03H	0	0	0	0	0	DAY			Day	01–07
04H	0	0	10 Date		Date				Date	01–31
05H	0	0	0	10 Month	Month				Month	01–12
06H	10 Year				Year				Year	00–99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08H-3FH									RAM 56 x 8	00H–FFH

Figura 3.2: Registri interni DS1307

che serve ad impostare la data/ora del RTC, passando come parametri i nuovi valori, in formato esadecimale.

Per leggere, invece, il contenuto dei registri interni al fine di ottenere la data/ora attuali bisogna preliminarmente effettuare un'operazione di scrittura nella quale il MASTER si limita a mandare allo SLAVE l'indirizzo del registro interno di partenza per l'inizio della lettura, e seguire poi la procedura di lettura da bus I^2C:

- il dispositivo MASTER invia una condizione di START seguita dall'indirizzo dello SLAVE (col quale intende comunicare) e dal bit di di direzione $R/\overline{W}$ (impostato a 1 per un'operazione di lettura);
- lo SLAVE risponde al MASTER con un bit di ACKNOWLEDGE;
- lo SLAVE inizia a trasmettere il contenuto dei registri interni a partire da quello indicato del REGISTER POINTER. Ad ogni byte inviato incrementa il REGISTER POINTER;

- il MASTER risponde allo SLAVE con un bit di ACKNOWLEDGE, se intende ricevere altri dati; altrimenti manda un bit di NOT ACKNOWLEDGE se intende terminare il trasferimento.

Per la procedura di lettura, l'Arduino esegue la funzione

*void leggiRTC(char * data, char* ora)*

che acquisisce i valori attuali di data ed ora dal DS1307, e li restituisce come stringhe salvandoli nelle variabili passate per argomento.

3.1.4 L'interrupt temporizzato e l'acquisizione dei dati

La libreria *MsTimer2.h* permette di eseguire una funzione periodicamente, appoggiandosi all'evento di interrupt generato dal TIMER2 dell' ATmega328.

Viene usata per temporizzare le acquisizioni A/D con le due istruzioni inserite nella parte di inizializzazione dell'Arduino:

```
MsTimer2::set(PERIODO_ACQUISIZIONE, nuovaAcquisizione); //
    eseguo la funzione nuovaAcquisizione() ogni 50 ms
MsTimer2::start(); // abilito il timer
```

Listing 3.9: Assegnazione della funzione *nuovaAcquisizione* al TIMER2

Grazie all'interrupt l'esecuzione del programma principale si interrompe e viene eseguito il metodo *nuovaAcquisizione()* (listato 3.10), che acquisisce i nuovi valori di tensione e di temperatura (con una conversione eseguita dal convertitore A/D interno all'Arduino).

```
void nuovaAcquisizione() { // acquisizione di tensione e
    temperatura tramite ADC ed aggiornamento dello SoC
  double tensione = ((Vref*analogRead(0))/1024) / Gtensione; //
     acquisisco la tensione
  richiestaADC=true; // attivo il flag per acquisizione di
     corrente
  analogRead(1); // istruzioni necessaria per sensore ad alta
     impedenza
  double temperatura = ( ( Vref * analogRead(1) ) / 1024 ) /
     Gtemperatura; // acquisisco la temperatura
  accumulatoreTensione += tensione;
  accumulatoreCorrente += corrente;
  accumulatoreTemperatura += temperatura;
  punti++;
  if (punti == PUNTI_MEDIA) {
    recordPronto=true; // segnalo la presenza di un nuovo
       campione da salvare
```

```
    punti=0;
    tensioneMediaPrecedente=tensioneMedia;
    tensioneMedia = accumulatoreTensione/PUNTI_MEDIA;
    correnteMedia = accumulatoreCorrente/PUNTI_MEDIA;
    temperaturaMedia = accumulatoreTemperatura/PUNTI_MEDIA;
    accumulatoreTensione=0;
    accumulatoreCorrente=0;
    accumulatoreTemperatura=0;
  }

  capacitaMassima=CAPACITA_NOMINALE*coefficenteTemperatura(
      temperaturaMedia); // aggiorno il valore di capacita
      massima in base alla temperatura

  // aggiorno lo SoC
  if((statoSistema==INIZIALE) || (statoSistema==EQUILIBRIO))
    SoC=SoC_from_OCV(tensione);
  else {
    double contatoreCarica = (corrente*PERIODO_ACQUISIZIONE)
        /3600000.0; // il periodo di acquisizione viene
        convertito in ore
    SoC += 100*(contatoreCarica/capacitaMassima);
  }
  if(SoC <= 0)
    SoC=0;
  if(SoC >= 100)
    SoC=100;

  // aggiorno lo stato del sistema
  if (corrente > CORRENTE_LIMITE)
    statoSistema = CARICA;
  else if (corrente < (-CORRENTE_LIMITE) )
    statoSistema = SCARICA;
  else if( ( (tensioneMedia-tensioneMediaPrecedente)<
      DERIVATA_LIMITE) && ( (tensioneMedia-
      tensioneMediaPrecedente)>-DERIVATA_LIMITE) )
       statoSistema=EQUILIBRIO;
  else
       statoSistema=TRANSIZIONE;
}
```

Listing 3.10: Funzione *nuovaAcquisizione*()

Il valore di corrente viene invece richiesto al convertitore A/D MCP3221, che opera a fianco del trasduttore di corrente, in seguito al controllo del flag *richiestaADC* (listato 3.11): questa operazione risulta necessaria perchè non è possibile eseguire una lettura dal bus I^2C all'interno della routine di interrupt del TIMER2, visto che la libreria *Wire.h* usa lei stessa gli interrupt.

```
double leggiADC() {
  unsigned char msbyte; // byte piu significativo
  unsigned char lsbyte; // byte meno significativo
  Wire.requestFrom(INDIRIZZO_ADC, 2); // richiedo i 2 byte della
      conversione ADC
  msbyte=Wire.read();
  lsbyte=Wire.read();
  unsigned int correnteADC=lsbyte+(256*msbyte);
  richiestaADC=false;
  return ( ( ( Vref * correnteADC ) / 4092 ) - Vref_LEM_I ) /
      Gcorrente;
}
```

Listing 3.11: Funzione *leggiADC*()

I valori istantanei vengono utilizzati per calcolare i rispettivi valori medi su un certo periodo; questi ultimi sono i valori che vengono scritti nei file di log.

Il valore istantaneo di corrente è utile per ottenere una miglior precisione nel conteggio della carica entrante/uscente delle batterie (di cui si parla nel capitolo 4).

Per quanto riguarda la temperatura, la sua misura consente di determinare il coefficente moltiplicativo, che indica la variazione della capacità delle batterie, interpolando i dati forniti dal costruttore (tabella 2.6) con la funzione *coefficenteTemperatura()* (listato 3.12).

3.1.5 Il Watch Dog Timer

Il WDT è una funzionalità del chip ATmega328: si tratta di un particolare timer che lavora con una sorgente di clock indipendente dal clock di sistema. Genera un interrupt o un reset di sistema quando si verifica un overflow.

Risulta utile in quei programmi in cui il flusso di esecuzione può incontrare qualche condizione imprevista e far bloccare l'intero sistema: in questi casi lo si imposta per generare un reset di sistema dopo un certo periodo, se il timer non viene azzerato nel frattempo.

In questo progetto viene usato come sorgente di reset di tipo software e viene abilitato solo nel momento in cui il Client effettua una richiesta di *reset* al Server, per effettuare il reset del sistema di acquisizione da remoto, senza accedere fisicamente alla scheda.

Includendo la libreria *wdt.h* si può usare la funzione *wdt_enable*(), specificando un valore di tempo, allo scadere del quale viene generato il reset.

```
const unsigned char arrayTemperatura[PUNTI_TEMPERATURA]={ 0, 5,
    10, 15, 20, 25, 30, 35, 40 };
const unsigned char arrayCoefficenteTemperatura[PUNTI_TEMPERATURA
    ]={ 0, 84, 88, 93, 97, 100, 103, 105, 107 };

// Restituisce il coefficente moltiplicativo per cui va
    moltiplicata la capacita', in base alla temperatura
double coefficenteTemperatura(double temperaturaMisurata) {
  unsigned char indice;
  for(indice=0; indice < PUNTI_TEMPERATURA-1; indice++)
    if ( (temperaturaMisurata < arrayTemperatura[indice+1]) && (
        temperaturaMisurata > arrayTemperatura[indice]) )
      break;
  double coefficenteAngolare = ( ( arrayCoefficenteTemperatura[
      indice+1] - arrayCoefficenteTemperatura[indice] ) / 100.0 )
       / ( arrayTemperatura[indice+1] - arrayTemperatura[indice]
      ) ;
  return ((arrayCoefficenteTemperatura[indice]) + (
      coefficenteAngolare*temperaturaMisurata))/100.0;
}
```

Listing 3.12: Funzione $coefficenteTemperatura()$

3.2 Le pagine Web

Le pagine Web permettono di visualizzare rapidamente su una finestra di un browser Web i dati analogici che il firmware acquisisce, e di effettuare dei controlli sul sistema di acquisizione; al loro interno contengono anche delle parti in linguaggio JavaScript: queste linee di codice vengono interpretate ed eseguite dal browser che le riceve.

Per questo si dice che Javascript è un linguaggio interpretato (non deve essere compilato) e di tipo Client-side (cioè viene proprio eseguito dal Client e non dal Server).

È stato predisposto un piccolo menù di navigazione per rendere semplice l'accesso alle funzioni del firmware; è composto da poche pagine HTML (i nomi delle pagine rispettano la convenzione 8.3 sul nome dei file, come già detto).

Per aggiornare parti del contenuto di una pagina (senza doverla ricaricare interamente) o per permettere all'utente di interagire con la stessa, tramite semplici controlli, viene sfruttata la tecnica di trasferimento dati AJAX che permette al Server di mandare al Client solo una piccola quantità di dati.

Con questa tecnica la richiesta dati viene effettuata dal Client in maniera asincrona, in modo invisibile all'utente che accede al Server.

3.2.1 La pagina *index.htm*

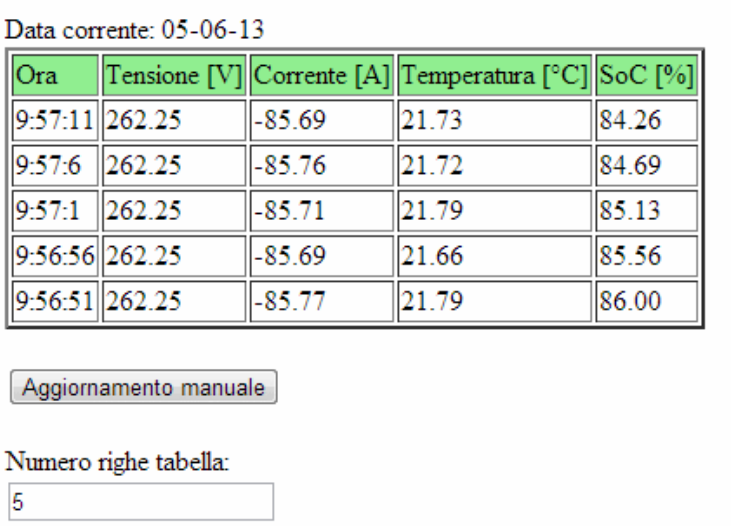
Da questa si accede alle altre: contiene solamente una lista di collegamenti ipertestuali.

Viene visualizzata in modo predefinito come pagina principale del Web Server.

3.2.2 La pagina *tabella.htm*

Mostra una tabella (figura 3.3) con le ultime acquisizioni; il suo contenuto viene aggiornato automaticamente ad intervalli regolari, tramite una richiesta GET di tipo *nuovoDato* (sezione 3.1.2) all'Arduino.

È possibile variare il numero di righe che vengono visualizzate ed è presente un bottone per l'aggiornamento manuale nel caso il browser usato non supportasse la tecnologia AJAX (come per i dispositivi mobili).


Data corrente: 05-06-13

Ora	Tensione [V]	Corrente [A]	Temperatura [°C]	SoC [%]
9:57:11	262.25	-85.69	21.73	84.26
9:57:6	262.25	-85.76	21.72	84.69
9:57:1	262.25	-85.71	21.79	85.13
9:56:56	262.25	-85.69	21.66	85.56
9:56:51	262.25	-85.77	21.79	86.00

Figura 3.3: Pagina *tabella.htm*

3.2.3 La pagina *grafico.htm*

Mostra un grafico con i dati delle ultime acquisizioni.

Il grafico viene tracciato grazie all'uso delle librerie JavaScript open-source jQuery e Flot.

In particolare la libreria jQuery facilita la creazione di pagine Web dinamiche, anche con effetti grafici di animazione e transizione che rendono

più gradevoli i contenuti delle pagine; la libreria Flot si appoggia invece alla libreria jQuery consentendo di effettuare operazioni di disegno, come la creazione di svariati tipi di grafici.

Per poter usare le suddette librerie, vengono inclusi nel file *grafico.htm* i riferimenti ai file di script necessari, memorizzati sulla scheda di memoria: verranno richiesti dal browser del Client in sequenza non appena termina l'invio del file *grafico.htm*:

```
<script language="javascript" type="text/javascript" src="
    jquery.js"></script>
<script language="javascript" type="text/javascript" src="flot.
    js"></script>
<script language="javascript" type="text/javascript" src="time.
    js"></script>
```

Listing 3.13: Inserimento dei plugin nella pagina HTML con i TAG *script*

In questo caso viene usato un grafico a linee che visualizza gli andamenti nel tempo delle grandezze fisiche misurate.

Praticamente lo script presente nella pagina si occupa di aggiornare periodicamente ed in modo automatico il contenuto del grafico con i dati richiesti all'Arduino tramite una richiesta di tipo *nuovoDato* (come per il caso della tabella): i dati ricevuti vengono memorizzati in più array, di lunghezza pari al massimo numero di punti presenti nel grafico.

Dopo aver acquisito un nuovo record, lo script controlla che abbia un'etichetta temporale diversa dal precedente: in caso affermativo lo riconosce come un nuovo record da inserire nel grafico e lo divide in più parti (corrispondenti ai campi del record), ognuna delle quali viene accodata al rispettivo array.

È stata inserita la possibilità di impostare, tramite due caselle di testo, il numero massimo di punti che vengono interpolati e l'intervallo di aggiornamento; questi valori sono limitati a rimanere entro i limiti indicati.

Cambiando il numero di punti del grafico, gli array contenenti i campi dei record vengono ridimensionati.

Infine il grafico è impostato per adattarsi automaticamente alle dimensioni della finestra del browser.

Purtroppo è stato rilevato che raramente qualche browser richiede nell'ordine sbagliato i file di script all'Arduino, non riuscendo a visualizzare il grafico: in questi casi basta aggiornare manualmente la pagina affinchè il browser richieda nuovamente i file richiesti dal grafico e li ricarichi.

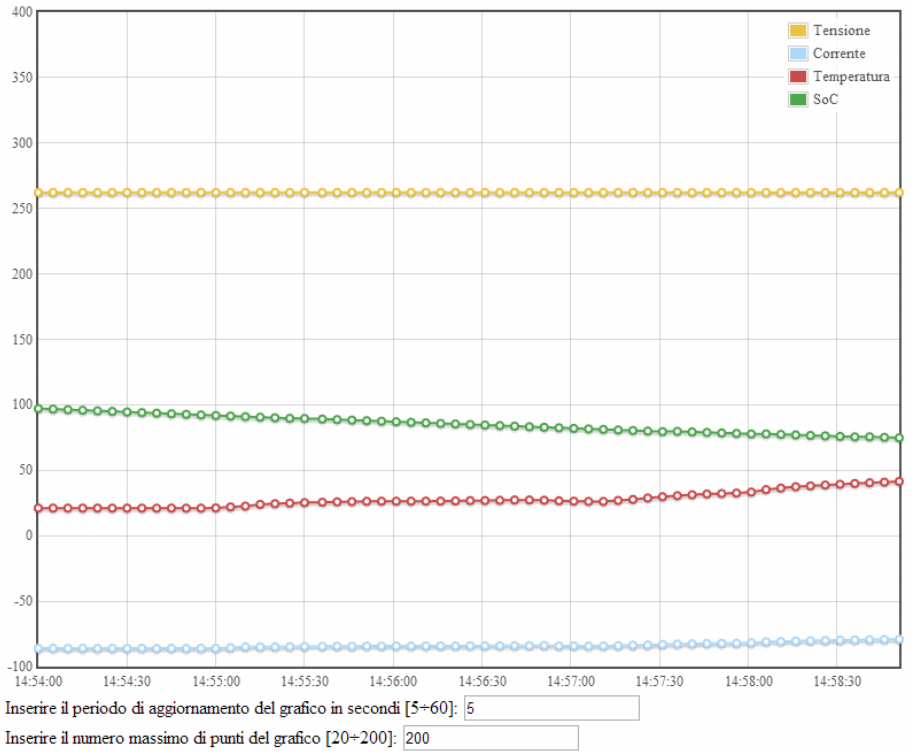


Figura 3.4: Pagina *grafico.htm*

3.2.4 La pagina *RTC.htm*

Permette di reimpostare il Real Time Clock inserendo data e ora. I nuovi dati devono essere inseriti in apposite caselle di testo (figura 3.5).

Viene effettuato un controllo sulla correttezza dei dati inseriti: ogni parametro deve essere composto di due caratteri altrimenti non viene inviata all'Arduino la richiesta di aggiornamento del Real Time Clock.

Tale controllo viene implementato con una funzione JavaScript (riportata nel listato 3.14) eseguita quando l'utente clicca il bottone per impostare i nuovi valori di data/ora; il controllo viene quindi eseguito dal browser del Client, liberando risorse di calcolo dell'Arduino che si deve limitare ad inviare i nuovi dati al chip *DS1307*, sapendo già che quelli inseriti sono validi.

Si nota l'uso della funzione JavaScript *Date(anno, mese, giorno)*, la quale crea un oggetto di tipo *Date* che include un riferimento temporale UTC corrispondente al numero di millisecondi passati dal 1 gennaio 1970 fino alla data specificata come argomento.

Nel caso venisse passata una data che non esiste nel calendario (ad esempio il 31 febbraio 2013) allora il riferimento temporale creato farà riferimento

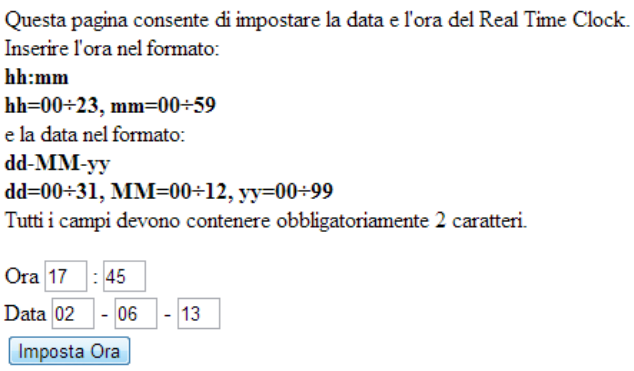


Figura 3.5: Pagina *RTC.htm*

al successivo giorno veramente esistente (nell'esempio sarà 1 marzo 2013); quindi confrontando i parametri inseriti nella caselle di testo, da parte dell'utente, con quelli del riferimento temporale UTC, creato con la funzione *Date()*, si determina se la data inserita risulta valida.

3.2.5 La pagina *file.htm*

Permette di accedere ad un singolo file di log inserendone la relativa data, di eliminarlo dalla scheda, oppure di eliminare tutti i file relativi ad uno specifico anno (figura 3.6).

Anche qui viene eseguito un controllo sulla data inserita con la funzione *dataValida()* presente nella pagina *RTC.htm*.

```
function dataValida(stringaGiorno, stringaMese, stringaAnno) {
  if ( (stringaGiorno.length == 2) && (stringaMese.length == 2)
     && (stringaAnno.length == 2) ) {
    var giorno=parseInt(stringaGiorno);
    var mese=parseInt(stringaMese);
    var anno=parseInt(stringaAnno);
    var dataReale=new Date(anno+2000, mese-1, giorno);
    if ( (giorno == dataReale.getDate()) && (mese == (dataReale.
       getMonth() + 1)) && (anno == (dataReale.getFullYear() -
       2000)) )
      return true; // data valida
    else
      return false; // data NON valida
  }
  else
    return false; // data NON valida
}

function oraValida(stringaOre, stringaMinuti) {
  if ( (stringaOre.length == 2) && (stringaMinuti.length == 2) &&
      (parseInt(stringaOre) <= 23) && (parseInt(stringaMinuti)
     <= 59) )
    return true; // ora valida
  else
    return false; // ora NON valida
}
```

Listing 3.14: Script per il controllo sulla correttezza dei dati; il metodo *getMonth*() dell'oggetto *dataReale* restituisce un numero da 0 a 11 che indica il mese

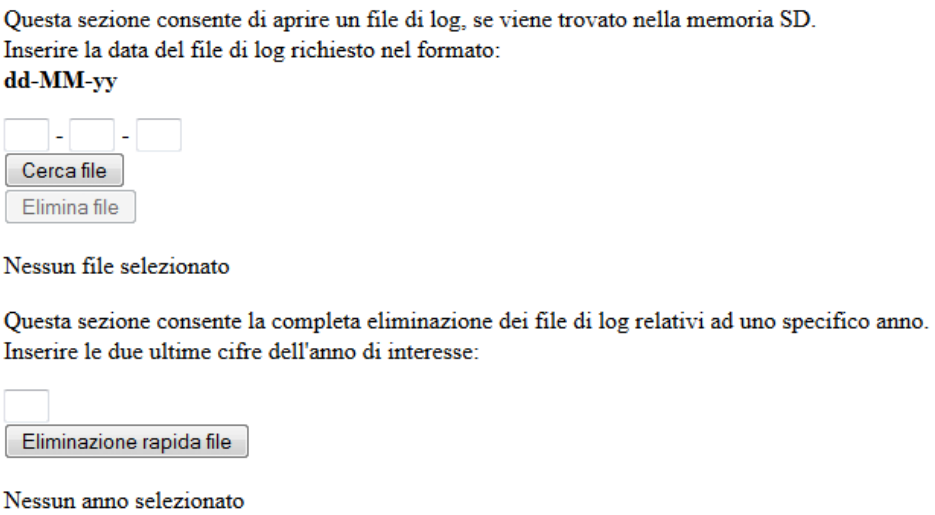


Figura 3.6: Pagina *file.htm*

Capitolo 4

La stima dello Stato di Carica

4.1 Il comportamento delle batterie

Una batteria ricaricabile si può modellizzare con il modello equivalente mostrato in figura 4.1: in esso è presente un generatore di forza elettromotrice EMF con una resistenza in serie R_S (che non è altro che la resistenza interna propria della batteria), più varie celle R-C che servono a tener conto del comportamento delle reazioni chimiche che avvengono all'interno della batteria.

Questo, noto come modello Randle, approssima il comportamento reale delle batterie in modo sempre migliore all'aumentare del numero di celle R-C considerato.

La tensione che si può prelevare ai morsetti della batteria, senza connetterla ad alcun carico, viene indicata con OCV (Open Circuit Voltage).

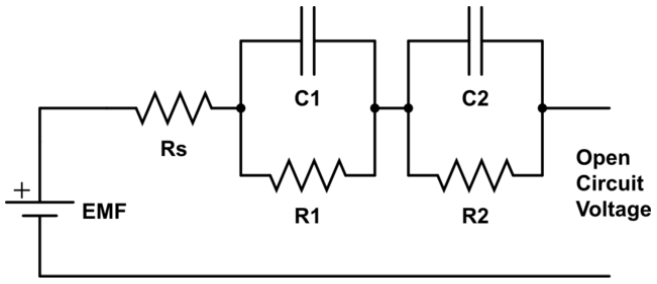


Figura 4.1: Schema equivalente delle batterie

Applicando una corrente costante uscente o entrante ai morsetti delle batterie, si può notare un comportamento simile a quello mostrato nelle figure 4.2 e 4.3: durante la fase di carica/scarica si nota la presenza di un divario

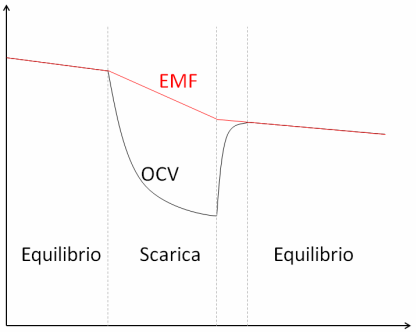


Figura 4.2: Curve di tensione durante la scarica

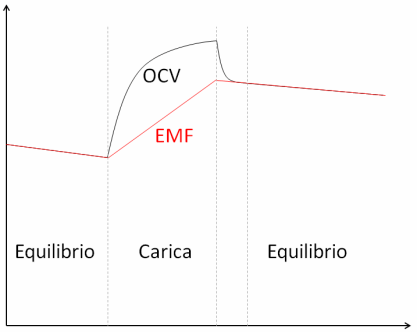


Figura 4.3: Curve di tensione durante la carica

tra i valori di EMF e OCV, docuto alle cadute di tensione sulla resistenza R_s e sulle celle R-C.

Al termine di questa fase i valori di EMF ed OCV tornano a coincidere, ma non prima che sia passato un certo periodo (periodo di rilassamento).

La leggera pendenza delle curve di EMF e OCV visibile nello stato di equilibrio è dovuta all'effetto dell'auto-scarica, dovuta alle perdite interne della batteria.

4.2 Lo stato di carica

Lo stato di carica, o SoC (acronimo di State of Charge), al tempo t di una batteria è definito come la carica residua $Q(t)$ (espressa in Coulomb) che la batteria può ancora erogare al carico in rapporto alla sua capacità nominale (intesa come quantità massima di carica), e solitamente viene indicata in percentuale:

$$SoC(t)\ [\%] = \frac{Q(t)}{Q_{nominale}} \times 100 \tag{4.1}$$

La capacità nominale non è un parametro facile da definire, perchè dipende da molti fattori (per esempio la temperatura o l'invecchiamento delle singole celle). Per questo, si è deciso che per definire il valore di capacità nominale di un accumulatore si faccia riferimento ad un tempo di scarica e ad una temperatura ambiente (alla quale avviene la scarica) prefissati.

Le normative europee stabiliscono così che la capacità di una batteria al piombo, indicata con C_{10}, si debba riferire ad un periodo di scarica di 10 ore ad una temperatura costante di 25 ℃, dopo le quali la tensione della batteria scende dal valore massimo ad un minimo, chiamato 'tensione di soglia'. Oltre

la tensione di soglia la batteria riduce rapidamente la corrente che riesce a fornire al carico.

Inoltre solitamente i costruttori di batterie esprimono la capacità nominale in Ampere-ora (*1 Ah=3600 Coulomb*), quindi se si considerano le specifiche delle batterie in esame:

$$C_{10} = 28\ Ah,\ V_{soglia,singola\ cella} = 1.80\ V,\ T = 25°C$$

si può prevedere approssimativamente che una singola batteria risulterà scarica se da essa viene prelevata una corrente costante di $2.8A$ per un tempo di 10 ore; dopo questo periodo la singola batteria presenterà una tensione di:

$$1.80\ V \times 6 = 10.8\ V.$$

Gli standard stabiliti per la definizione della capacità nominale di una batteria variano a seconda del tipo di batteria (ad esempio per le batterie Ni-Cd viene stabilito un periodo di scarica di 5 ore, quindi nei loro datasheet i costruttori specificano il valore di C_5) o a seconda delle normative del paese di produzione (per esempio la normativa USA definisce un periodo di scarica di 20 ore per le batterie al piombo, e indica quindi la capacità nominale delle stesse con C_{20}).

Esistono vari metodi per la stima dello SoC di un accumulatore, tra i quali si ricordano:

- la misura della densità dell'elettrolita: è un metodo abbastanza preciso ma la gran parte delle volte risulta scomodo da mettere in pratica, in quanto necessita di avere accesso alle batterie. A batteria carica la densità dell'elettrolita (che è l'acido solforico per le batterie al piombo) risulta maggiore;

- la misura della EMF: come già detto, sapendo che in condizioni di equilibrio i valori di EMF e OCV sono uguali, attraverso le tabelle che mettono in relazione OCV e SoC (rilasciate solitamente dai costruttori) è possibile risalire allo SoC con una semplice misura di tensione. Questo metodo risulta molto rapido, ma presenta dei punti deboli: bisogna scollegare qualsiasi carico dalle batterie in esame (cosa non sempre pratica e/o fattibile) e attendere che le batterie siano state a riposo (anche per varie ore) prima di considerare la misura di tensione affidabile per la stima dello SoC.

  Questo metodo, nelle condizioni operative sopra indicate, risulta efficace anche se le batterie rimangono per lungo tempo a riposo e l'effetto dell'auto-scarica ne produce il lento calo dello SoC;

- il 'conteggio di carica' (o Coulomb counting): questo metodo prevede di conoscere lo stato iniziale di carica dell'accumulatore in esame ($SoC_{iniziale}$, o equivalentemente la sua carica iniziale $Q_{iniziale}$) e che si misuri la quantità di carica entrante/uscente dallo stesso. Per fare ciò è possibile misurare costantemente la corrente $i(t)$ entrante/uscente dalla batteria, e ricavare la carica residua $Q(t)$, al tempo t, scambiata con il circuito esterno calcolando l'integrale della corrente nel tempo:

$$Q(t) = Q_{iniziale} + \int_0^t i(t)dt \tag{4.2}$$

Dividendo ogni membro dell'equazione 4.2 per $Q_{nominale}$ si ottiene l'equazione 4.3, con la quale si può stimare lo SoC.

$$SoC(t) = SoC_{iniziale} + \frac{\int_0^t i(t)dt}{Q_{nominale}} \tag{4.3}$$

Risulta essere un valido metodo per stimare lo SoC nel caso la corrente richiesta vari rapidamente e risulti impossibile l'utilizzo del metodo precedente; presenta però dei problemi non trascurabili che ne limitano l'accuratezza: gli inevitabili errori che si accumulano nell'operazione di integrazione, la difficoltà di avere a disposizione un valore abbastanza affidabile dello stato di carica per inizializzare il sistema di integrazione e il rendimento di carica delle batterie non unitario.

Il rendimento di carica, definito come il rapporto tra l'energia fornita in un ciclo completo di carica e l'energia disponibile per il successivo ciclo completo di scarica, risulta sempre minore del 100% a causa delle perdite interne (che si manifestano nel surriscaldamento della batteria stessa) perciò il conteggio di carica entrante nella batteria durante la fase di carica produce inevitabilmente un risultato maggiore della carica effettivamente erogabile in seguito.

Per questi motivi il conteggio di carica deve essere ricalibrato ad intervalli regolari, ed in condizioni ben definite.

In conclusione con il conteggio di carica non è possibile tener conto dell'auto-scarica, che è un fenomeno interno delle batterie.

4.3 Il metodo utilizzato

Per riuscire a realizzare un algoritmo efficace per la stima dello SoC, da implementare su un sistema a microcontrollore, vengono uniti i pregi del metodo di misura di tensione a circuito aperto e del conteggio di carica.

L'algoritmo descritto nel seguito è riportato in vari libri e pubblicazioni riguardanti i sistemi di gestione delle batterie.

In particolare, all'accensione del sistema (STATO INIZIALE) si assume che il gruppo di batterie sia stato a riposo fino a quell'istante, e che la misura diretta di tensione si possa considerare attendibile per la stima dello SoC iniziale, il quale viene determinato interpolando i dati forniti dal costruttore (tabella 4.4).

State of charge	Voltage
100%	2.12 to 2.14 V/Cell
80%	2.09 to 2.11 V/Cell
60%	2.05 to 2.08 V/Cell
40%	2.01 to 2.04 V/Cell
20%	1.97 to 2.00 V/Cell

Figura 4.4: Relazione SoC-OCV delle batterie 12HX135; i dati si riferiscono alla singola cella, rimasta a riposo per 24 ore a 25℃

Poi l'algoritmo transita verso uno di tre stati, come mostra il diagramma di figura 4.5, in base al valore misurato della corrente; la decisione per effettuare le transizione di stato viene presa in riferimento a un certo valore limite I_{limite} di corrente, abbastanza piccolo.

Si può quindi transitare verso:

- lo STATO DI EQUILIBRIO se la corrente è, in modulo, minore del valore limite definito e la tensione rimane pressochè costante; qui lo SoC viene determinato tramite la relazione $SoC - OCV$, come è avvenuto per lo STATO INIZIALE;
- lo STATO DI CARICA se la corrente risulta maggiore del valore limite; qui lo SoC viene calcolato con il metodo 'conteggio di carica', partendo dal valore precedente la transizione di stato;
- lo STATO DI SCARICA se la corrente risulta minore dell'opposto del valore limite; come per lo STATO DI CARICA viene usato il 'conteggio di carica'.

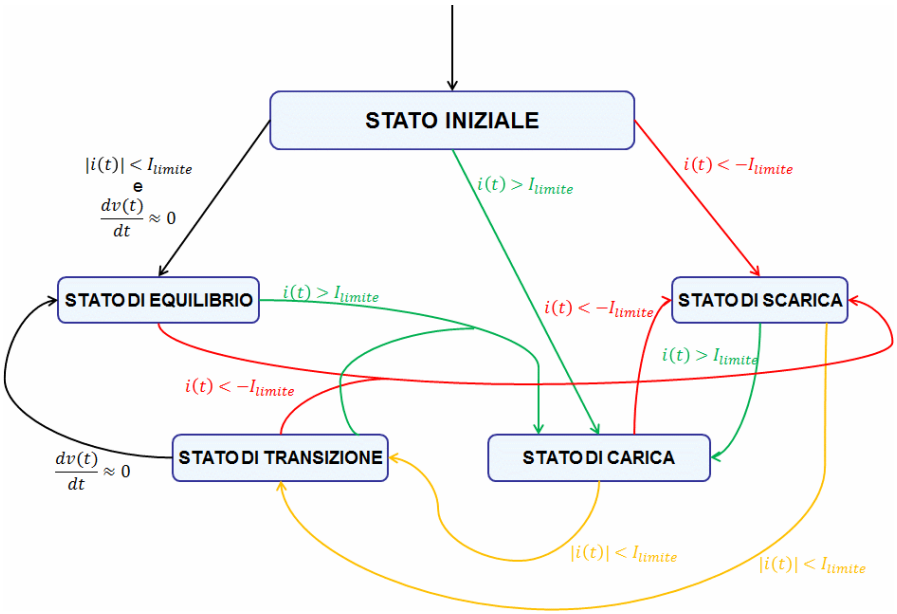


Figura 4.5: Diagramma a stati dell'algoritmo

Vi è un ultimo stato, al quale si può accedere dallo STATO DI CARICA o di SCARICA: lo STATO DI TRANSIZIONE. In esso si giunge se la corrente torna a calare, fino a rientrare in una fascia di valori tali da poterla considerare trascurabile (cioè se $|i(t)| < I_{limite}$).

In questo stato il valore di SoC stimato viene aggiornato tramite il conteggio di carica, e si deve attendere finchè la tensione misurata risulti stabile (cioè che le batterie si possano considerare 'a riposo') oppure che la corrente torni a crescere, superando (in modulo) il valore I_{limite}.

Nel primo caso il sistema transita nello stato di EQUILIBRIO, mentre nel secondo in quello di SCARICA o CARICA.

Per quanto riguarda l'implementazione di questo metodo sulla scheda Arduino, non ci sono problemi per la misura di tensione a circuito aperto, mentre per il conteggio di carica bisogna adottare un sistema per il calcolo dell'integrale in ambito digitale.

Sapendo che il microcontrollore acquisisce i valori di corrente (e tensione) con una frequenza nota $f_{acquisizione}$, corrispondente ad un periodo $T_{acquisizione} = \frac{1}{f_{acquisizione}} = 50\ ms$ fissato nel firmware, è possibile approssimare l'equazione a tempo continuo 4.3 con l'analoga a tempo discreto 4.4:

$$SoC(n+1) = SoC(n) + \frac{I(n) \cdot T_{acquisizione}}{Q_{nominale}} \tag{4.4}$$

dove con n viene indicato l'istante n-esimo, a cui corrisponde la corrente $I(n)$. Questa equazione viene usata dal firmware per aggiornare la stima dello SoC dopo ogni nuova acquisizione.

Praticamente la condizione di 'tensione stabile', che si deve verificare per transitare nello STATO DI EQUILIBRIO, si ottiene quando la derivata della tensione è nulla, o ha lente variazioni dovute all'auto-scarica.

Per questo viene scelto un valore limite $derivata_{limite}$, abbastanza piccolo, usato come soglia di decisione.

Siccome i dati su cui opera l'algorimtmo sono discretizzati nel tempo, allora la derivata della tensione si può esprimere come differenza tra il valore assunto in un istante, $V(n)$, e quello precedente, $V(n-1)$.

Per cui se $V(n) - V(n-1) \leq derivata_{limite}$ allora la tensione di batteria viene considerata stabile.

Capitolo 5

Conclusioni

Il sistema descritto e sviluppato in questa attività di tesi verrà installato al fine di monitorare l'andamento energetico delle batterie.

Il sistema è già registrato presso le infrastrutture informatiche del dipartimento con i seguenti parametri di connessione:

- indirizzo IP statico: 147.162.11.104
- Net Mask: 255.255.255.255.0
- Gateway: 147.162.11.254
- DNS: 147.162.2.100

e per accedergli basta digitare:

logbatterie

nella barra indirizzi di un browser collegato alla rete dipartimentale.

La realizzazione finale è visibile in figura 5.1.

Il sistema realizzato si può adattare benissimo anche ad altre configurazioni di batterie riprogrammando l'Arduino Uno con i parametri caratteristici delle batterie sotto misura, i quali sono definiti nel codice sorgente attraverso le macro e gli array riportati nel listato 5.1; in particolare gli array che vengono definiti costituiscono le tabelle delle specifiche tecniche delle batterie (riportate nelle figure 4.4 e 2.6).

Anche i valori di soglia I_{limite} e $derivata_{limite}$, che servono a regolare l'algoritmo di stima dello SoC, possono essere impostati tramite riprogrammazione dell'Arduino, variando il valore assegnato loro dalle apposite macro (listato 5.2).

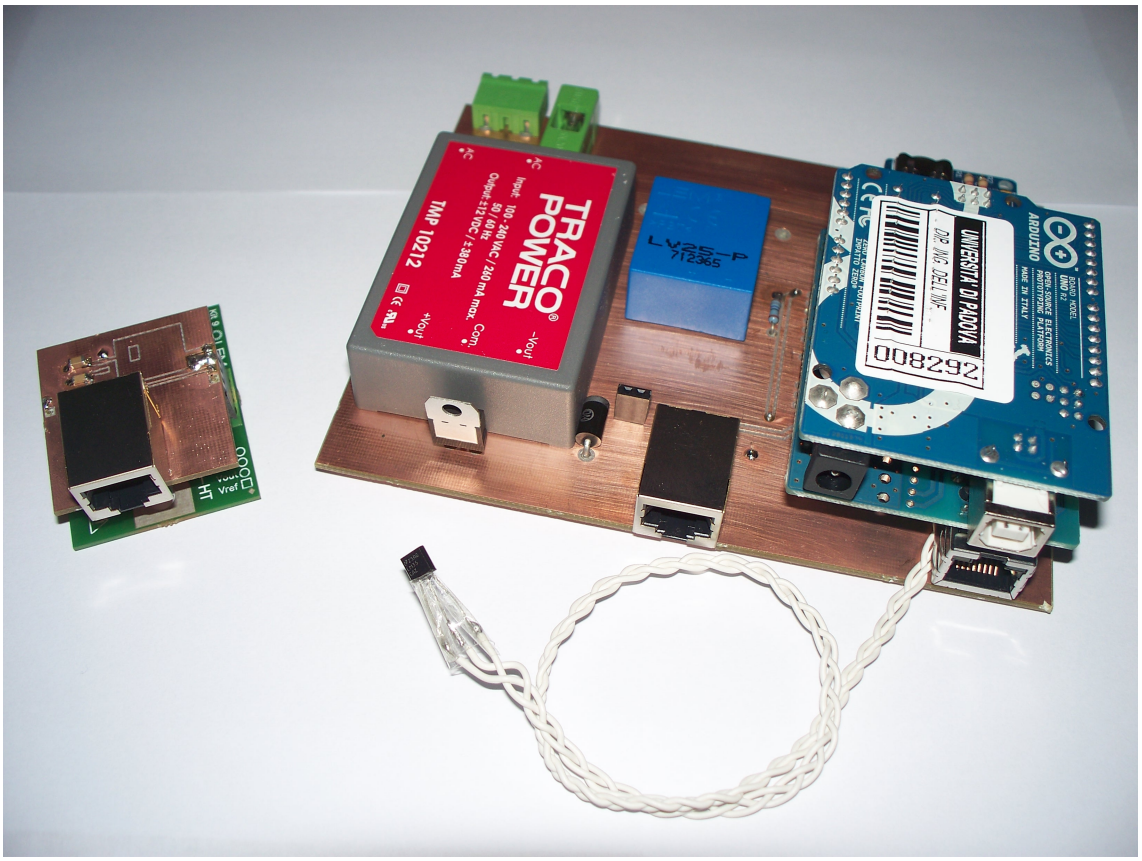


Figura 5.1: Realizzazione finale del sistema: a sinistra si vede la piccola scheda che comprende il LEM FHS 40-P ed il convertitore A/D MCP3221, mentre a destra è visibile la scheda che monta il convertitore DC/DC, il LEM LV 25-P ed il trasduttore di temperatura. Su quest'ultima viene innestato l'Arduino Uno, grazie agli appositi connettori.

5.1 Possibili sviluppi futuri

L'impiego del controller Ethernet W5100 comporta un consumo non trascurabile, pari a circa 180 mA, anche in standby; ciò comporta una lenta, ma costante, scarica delle batterie.

Un possibile miglioramento a questo aspetto del progetto consiste nel sostituire il modulo Arduino Ethernet Shield in uso con la nuova versione, che monta il controller W5200.

Il controller W5200 permette, a differenza del W5100, di essere messo in modalità standby, permettendo un notevole risparmio di energia da parte del sistema di monitoraggio, nei lunghi periodi inattività.

Un ulteriore aspetto riguarda la possibilità di monitorare lo stato di salute (State of Health) delle batterie. Infatti, a seguito di numerosi cicli di carica e scarica le batterie tendono a diminuire la loro capacità e conseguentemente falsare la stima dello stato di carica.

Per risolvere, seppur parzialmente, questo inconveniente è possibile implementare appositi algoritmi di valutazione. Tali algoritmi consentono il calcolo della diminuzione della capacità delle batterie mediante la valutazione del bilancio energetico.

```
// PARAMETRI DELLE BATTERIE
#define CELLE_PER_BATTERIA 6 // numero di celle per batteria
#define NUMERO_BATTERIE_SERIE 21 // numero di batterie collegate
    in serie
#define CAPACITA_NOMINALE 28 // Capacita nominale della serie,
    espressa in Ah

// TABELLA PER IL CALCOLO DELLO SoC IN FUNZIONE DELLA TENSIONE A
    CIRCUITO APERTO (A RIPOSO)
#define PUNTI_OCV_SoC 7
const unsigned char arraySoC[PUNTI_OCV_SoC]={ 0, 0, 20, 40, 60,
    80, 100 };
const double arrayOCV[PUNTI_OCV_SoC]={0,
                                1.800 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                1.985 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                2.025 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                2.065 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                2.100 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                2.130 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE };

// TABELLA PER IL CALCOLO DEL COEFFICENTE DI CORREZIONE PER LA
    CAPACITA IN FUNZIONE DELLA TEMPERATURA
#define PUNTI_TEMPERATURA 9
const unsigned char arrayTemperatura[PUNTI_TEMPERATURA]={ 0, 5,
    10, 15, 20, 25, 30, 35, 40 };
const unsigned char arrayCoefficenteTemperatura[PUNTI_TEMPERATURA
    ]={ 0, 84, 88, 93, 97, 100, 103, 105, 107 };
```

Listing 5.1: Macro e array di configurazione delle batterie

```
// PARAMETRI PER LA STIMA DELLO STATO DI CARICA
#define CORRENTE_LIMITE 1.0 // corrente limite in Ampere per
    decidere la soglia dello STATO TRANSIZIONE
#define DERIVATA_LIMITE 1.0 // derivata limite in Volt per
    decidere la soglia dello STATO EQUILIBRIO
```

Listing 5.2: Macro per controllare i parametri dell'algoritmo di stima dello SoC

Capitolo 6

Sorgente completo in C

```
/*
  Web Server + SD Card Logger + Real Time Clock

 Circuito:
 * Ethernet shield connesso ai pin 10, 11, 12, 13
 * RTC DS1307 e ADC MCP3221: SDA connesso ad AN4, SCL connesso ad
     AN5
 * Sensore di tensione connesso ad AN0
 * Sensore di temperatura connesso ad AN1

 creato il 19 marzo 2013
 ultima modifica 10 luglio 2013
 da Dario Marchetto

*/

#include <SPI.h> // libreria per la comunicazione su bus SPI (
    richiesta dalla scheda di memoria SD e dal modulo Ethernet)
#include <Ethernet.h> // libreria per la gestione del modulo
    Ethernet
#include <SD.h> // libreria per la gestione della scheda di
    memoria Secure Digital
#include <MsTimer2.h> // libreria per gestire la temporizzazione
    dell'acquisizione attraverso il Timer2
#include <Wire.h> // libreria per la comunicazione I2C
#include <avr/wdt.h> // libreria per l'uso del Watch Dog Timer

// PARAMETRI DELLE BATTERIE
#define CELLE_PER_BATTERIA 6 // numero di celle per batteria
#define NUMERO_BATTERIE_SERIE 21 // numero di batterie collegate
    in serie
#define CAPACITA_NOMINALE 28 // Capacita nominale della serie,
    espressa in Ah
```

```
// PARAMETRI PER LA STIMA DELLO STATO DI CARICA
#define CORRENTE_LIMITE 1.0 // corrente limite in Ampere per
    decidere la soglia dello STATO TRANSIZIONE
#define DERIVATA_LIMITE 1.0 // derivata limite in Volt per
    decidere la soglia dello STATO EQUILIBRIO

// PARAMETRI TRASDUTTORI E ADC
#define Vref 5.0 // tensione riferimento ADC
#define Vref_LEM_I 2.5 // tensione di riferimento interna del LEM
     FHS 40-P
#define Gtensione 0.015583333 // Guadagno del LEM di tensione
#define Gcorrente 0.0258 // Guadagno del LEM di corrente
#define Gtemperatura 0.010 // Guadagno del sensore di temperatura
    : 10 mV/Â°C

// COSTANTI DI PROGRAMMA
#define numeroByteTimestamp 7 // numero di byte di cui Ã¨
    composto il timestamp
#define PERIODO_ACQUISIZIONE 50 // tempo che trascorre tra una
    acquisizione e la successiva, in millisecondi
#define PERIODO_SALVATAGGIO 5000 // tempo che trascorre tra il
    salvataggio di un record ed il suo successivo, in
    millisecondi
#define BYTE_BUFFER_HTTP 40 // byte contenuti nel bufferHTTP (40)
#define BYTE_BLOCCO 16 // numero di byte in un pacchetto (
    divisione del file) (32)
#define CHIP_SELECT_ETHERNET 10 // numero pin dell'Arduino
    collegato al Chip Select del modulo Ethernet (interfaccia SPI
    )
#define CHIP_SELECT_SD 4 // numero pin dell'Arduino collegato al
    Chip Select della scheda SD (interfaccia SPI)
#define INDIRIZZO_RTC 0x68 // indirizzo del modulo RTC per
    comunicare attraverso il bus I2C
#define INDIRIZZO_ADC 0x4d // indirizzo del convertitore A/D per
    comunicare attraverso il bus I2C

// TABELLA PER IL CALCOLO DELLO SoC IN FUNZIONE DELLA TENSIONE A
    CIRCUITO APERTO (A RIPOSO)
#define PUNTI_OCV_SoC 7
const unsigned char arraySoC[PUNTI_OCV_SoC]={ 0, 0, 20, 40, 60,
    80, 100 };
const double arrayOCV[PUNTI_OCV_SoC]={0,
                                1.800 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                1.985 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                2.025 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
```

```
                                2.065 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                2.100 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE,
                                2.130 * CELLE_PER_BATTERIA *
                                    NUMERO_BATTERIE_SERIE };

// TABELLA PER IL CALCOLO DEL COEFFICENTE DI CORREZIONE PER LA
    CAPACITA IN FUNZIONE DELLA TEMPERATURA
#define PUNTI_TEMPERATURA 9
const unsigned char arrayTemperatura[PUNTI_TEMPERATURA]={ 0, 5,
    10, 15, 20, 25, 30, 35, 40 };
const unsigned char arrayCoefficenteTemperatura[PUNTI_TEMPERATURA
    ]={ 0, 84, 88, 93, 97, 100, 103, 105, 107 };

// PARAMETRI PER LA COMUNICAZIONE ETHERNET
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // indirizzo
     MAC
IPAddress ip(147, 162, 11, 104); // indirizzo IP
EthernetServer server(80); // porta di comunicazione

char blocco[BYTE_BLOCCO+1]; // buffer per l'invio dei file al
    Client
char * parametriRichiestaGET=NULL;
char * nomeFileRichiesto=NULL;
double capacitaMassima=CAPACITA_NOMINALE*1.0; // in seguito viene
     corretta con la temperatura
enum stato {INIZIALE, EQUILIBRIO, TRANSIZIONE, CARICA, SCARICA}
    statoSistema; // stati del sistema di monitoraggio
double SoC; // stato di carica espresso in percentuale (0%-100%)

#define PUNTI_MEDIA (PERIODO_SALVATAGGIO/PERIODO_ACQUISIZIONE)
    *1.0 // acquisizioni su cui fare la media
unsigned int punti=0;
double tensioneMedia=0;
double tensioneMediaPrecedente=0;
double correnteMedia=0;
double temperaturaMedia=0;
double accumulatoreTensione=0;
double accumulatoreCorrente=0;
double accumulatoreTemperatura=0;

char nomeFileLog[]="00-00-00.CSV"; // nome iniziale (poi viene
    aggiornato con la data)
char data[]="00-00-00";
char ora[]="00:00:00";
boolean recordPronto=false; // flag che indica la presenza di un
    nuovo record che deve essere salvato
```

```
boolean schedaPresente=false; // flag che indica la presenza
    della scheda SD dopo il tentativo di inizializzazione

// Acquisizione della corrente dal ADC MCP3221
double corrente;
boolean richiestaADC=false; // flag per acquisizione di corrente
double leggiADC() {
  unsigned char msbyte; // byte piu significativo
  unsigned char lsbyte; // byte meno significativo
  Wire.requestFrom(INDIRIZZO_ADC, 2); // richiedo i 2 byte della
      conversione ADC
  msbyte=Wire.read();
  lsbyte=Wire.read();
  unsigned int correnteADC=lsbyte+(256*msbyte);
  richiestaADC=false;
  return ( ( ( Vref * correnteADC ) / 4092 ) - Vref_LEM_I ) /
      Gcorrente;
}

// Acquisisce i dati attuali dei trasduttori
void nuovaAcquisizione() { // acquisizione di tensione e
    temperatura tramite ADC ed aggiornamento dello SoC
  double tensione = ((Vref*analogRead(0))/1024) / Gtensione; //
      acquisisco la tensione
  richiestaADC=true; // attivo il flag per acquisizione di
      corrente
  analogRead(1); // istruzioni necessaria per sensore ad alta
      impedenza
  double temperatura = ( ( Vref * analogRead(1) ) / 1024 ) /
      Gtemperatura; // acquisisco la temperatura
  accumulatoreTensione += tensione;
  accumulatoreCorrente += corrente;
  accumulatoreTemperatura += temperatura;
  punti++;
  if (punti == PUNTI_MEDIA) {
    recordPronto=true; // segnalo la presenza di un nuovo
        campione da salvare
    punti=0;
    tensioneMediaPrecedente=tensioneMedia;
    tensioneMedia = accumulatoreTensione/PUNTI_MEDIA;
    correnteMedia = accumulatoreCorrente/PUNTI_MEDIA;
    temperaturaMedia = accumulatoreTemperatura/PUNTI_MEDIA;
    accumulatoreTensione=0;
    accumulatoreCorrente=0;
    accumulatoreTemperatura=0;
  }

  capacitaMassima=CAPACITA_NOMINALE*coefficenteTemperatura(
      temperaturaMedia); // aggiorno il valore di capacita
```

```
      massima in base alla temperatura

  // aggiorno lo SoC
  if((statoSistema==INIZIALE) || (statoSistema==EQUILIBRIO))
    SoC=SoC_from_OCV(tensione);
  else {
    double contatoreCarica = (corrente*PERIODO_ACQUISIZIONE)
        /3600000.0; // il periodo di acquisizione viene
        convertito in ore
    SoC += 100*(contatoreCarica/capacitaMassima);
  }
  if(SoC <= 0)
    SoC=0;
  if(SoC >= 100)
    SoC=100;

  // aggiorno lo stato del sistema
  if (corrente > CORRENTE_LIMITE)
    statoSistema = CARICA;
  else if (corrente < (-CORRENTE_LIMITE) )
    statoSistema = SCARICA;
  else if( ( (tensioneMedia-tensioneMediaPrecedente)<
     DERIVATA_LIMITE) && ( (tensioneMedia-
     tensioneMediaPrecedente)>-DERIVATA_LIMITE) )
      statoSistema=EQUILIBRIO;
  else
      statoSistema=TRANSIZIONE;
}

// Acquisisce dal Real Time Clock la data e l'ora attuali
void leggiRTC(char * data, char * ora) {  // acquisizione RTC dal
    DS1307
  Wire.beginTransmission(INDIRIZZO_RTC); // inizio a trasmettere
     al RTC
  Wire.write((byte)0x00); // indirizzo di partenza del register
     pointer del RTC
  Wire.endTransmission(); // fine trasmissione
  Wire.requestFrom(INDIRIZZO_RTC, numeroByteTimestamp); //
     richiedo "numeroByteTimestamp" byte dal RTC
  unsigned char timestamp[numeroByteTimestamp];
  for(unsigned int i=0; i<numeroByteTimestamp; i++)
    timestamp[i]=Wire.read();
  data[0]=(timestamp[4]/16) + '0'; // giorno
  data[1]=(timestamp[4]%16) + '0';
  data[3]=(timestamp[5]/16) + '0'; // mese
  data[4]=(timestamp[5]%16) + '0';
  data[6]=(timestamp[6]/16) + '0'; // anno
  data[7]=(timestamp[6]%16) + '0';
  ora[0]=(timestamp[2]/16) + '0'; // ore
```

```
  ora[1]=(timestamp[2]%16) + '0';
  ora[3]=(timestamp[1]/16) + '0'; // minuti
  ora[4]=(timestamp[1]%16) + '0';
  ora[6]=(timestamp[0]/16) + '0'; // secondi
  ora[7]=(timestamp[0]%16) + '0';
}

// Reimposta il Real Time Clock tramite i parametri passati
    attraverso una richiesta GET
void scriviRTC(byte ore, byte minuti, byte giorno, byte mese,
    byte anno) {
  Wire.beginTransmission(INDIRIZZO_RTC);
  Wire.write((byte)0x00); // il primo byte stabilisce il registro
      iniziale da scivere: inizio salvando i secondi
  Wire.write((byte)0x00); // azzero i SECONDI
  Wire.write(minuti); // MINUTI da 0x00 a 0x59
  Wire.write(ore); // ORE da 0x00 a 0x24
  Wire.endTransmission();
  Wire.beginTransmission(INDIRIZZO_RTC);
  Wire.write((byte)0x04); // inizio salvando il giorno del mese
  Wire.write(giorno); // GIORNO del mese da 0x00 a 0x31
  Wire.write(mese); // MESE da 0x00 a 0x12
  Wire.write(anno); // ANNO 0x00 a 0x99
  Wire.endTransmission();
}

// Mappa la tensione a riposo (Open Circuit Voltage) allo SoC (
    usa l'interpolazione lineare)
double SoC_from_OCV(double tensione) {
  unsigned char indice;
  for(indice=0; indice < PUNTI_OCV_SoC-1; indice++)
    if ( (tensione < arrayOCV[indice+1]) && (tensione > arrayOCV[
        indice]) )
      break; // esco dal ciclo
  double coefficenteAngolare = ( (arraySoC[indice+1] - arraySoC[
      indice])/100.0 ) / (arrayOCV[indice+1] - arrayOCV[indice])
      ;
  return (arraySoC[indice] + coefficenteAngolare*tensione); //
      calcolo lo SoC attuale con interpolazione lineare
}

// Restituisce il coefficente moltiplicativo per cui va
    moltiplicata la capacita, in base alla temperatura
double coefficenteTemperatura(double temperaturaMisurata) {
  unsigned char indice;
  for(indice=0; indice < PUNTI_TEMPERATURA-1; indice++)
    if ( (temperaturaMisurata < arrayTemperatura[indice+1]) && (
        temperaturaMisurata > arrayTemperatura[indice]) )
      break;
```

```
  double coefficenteAngolare = ( ( arrayCoefficenteTemperatura[
      indice+1] - arrayCoefficenteTemperatura[indice] ) / 100.0 )
       / ( arrayTemperatura[indice+1] - arrayTemperatura[indice]
      ) ;
  return ((arrayCoefficenteTemperatura[indice]) + (
      coefficenteAngolare*temperaturaMisurata))/100.0;
}

// Estrae il nome del file richiesto ed i parametri dalla stringa
    passata come argomento
boolean richiestaGET(char * buffer) {
  boolean parametriPresenti=false;
  unsigned char cursore=5;
  nomeFileRichiesto = &buffer[5];
  while ( (buffer[cursore] != ' ') && (cursore < BYTE_BUFFER_HTTP
      ) ) {
    if( ( (buffer[cursore] == '&') || (buffer[cursore] == '?') )
       && (cursore < BYTE_BUFFER_HTTP-1) ) {
      buffer[cursore]=NULL; // terminatore
      if(!parametriPresenti) {
        parametriRichiestaGET = &buffer[cursore+1];
        parametriPresenti=true;
      }
    }
    cursore++;
  }
  buffer[cursore]=NULL; // terminatore
  if(!parametriPresenti)
    parametriRichiestaGET = &buffer[cursore]; // nessun parametro
  if(nomeFileRichiesto[0] != NULL)
    return true; // richiesto un file
  return false; // nessun file richiesto
}

// Invia al client il file fileRichiesto
boolean invioFile(char * nomeFile, EthernetClient client) {
  if (!schedaPresente)
    return false;
  File file=SD.open(nomeFile, FILE_READ);
  if(file) {
    boolean continua=true;
    while(continua) {
      for(unsigned char i=0; i<BYTE_BLOCCO; i++) {
        if(file.available())
          blocco[i]=file.read();
        else {
          continua=false;
          blocco[i]=NULL;
        }
```

```
      }
      blocco[BYTE_BLOCCO]=NULL;
      client.print(blocco);
    }
    file.close();
    return true;
  }
  else
    return false;
}

// Confronta la stringa passata per parametro e restituisce "true
   " se corrisponde al contenuto di nomeFileRichiesto
boolean comandoGET(char * stringa1) {
  unsigned char lunghezza1=0;
  unsigned char lunghezza2=0;
  while( (stringa1[lunghezza1] != NULL) && (lunghezza1 <
     BYTE_BUFFER_HTTP) )
    lunghezza1++;
  while( (nomeFileRichiesto[lunghezza2] != NULL) && (lunghezza2 <
      BYTE_BUFFER_HTTP) )
    lunghezza2++;
  if(lunghezza1 != lunghezza2)
    return false; // stringhe di lunghezza diversa sono diverse
  unsigned char i=0;
  while( (stringa1[i] == nomeFileRichiesto[i]) && (i < lunghezza1
     ) )
    i++;
  if(i == lunghezza1)
    return true;
  return false;
}

// Invia al client l'ultimo record presente in memoria
void invioUltimoRecord(EthernetClient client) {
  client.print(ora);
  client.print(';');
  client.print(tensioneMedia);
  client.print(';');
  client.print(correnteMedia);
  client.print(';');
  client.print(temperaturaMedia);
  client.print(';');
  client.print(SoC);
}

void setup() {
  wdt_disable(); // disabilito il Watch Dog Timer
  // INIZIALIZZO LA SCHEDA SD
```

```
  pinMode(CHIP_SELECT_ETHERNET, OUTPUT);
  digitalWrite(CHIP_SELECT_ETHERNET, HIGH);
  if(SD.begin(CHIP_SELECT_SD)) // inizializzo scheda SD
    schedaPresente=true;
  // INIZIALIZZO IL MODULO ETHERNET
  Ethernet.begin(mac, ip); // inizializza la libreria Ethernet e
      le impostazioni di rete
  server.begin(); // il server inizia ad "ascoltare" le
      connessioni in arrivo
  // INIZIALIZZO IL MODULO PER LA COMUNICAZIONE I2C
  Wire.begin(); // accesso al bus I2C in modalitÃ  MASTER
  // INIZIALIZZO IL SISTEMA DI STIMA DELLO STATO DI CARICA (
      assumo che il sistema di accumulo sia a riposo)
  statoSistema=INIZIALE;
  while(recordPronto);
    //nuovaAcquisizione();
  SoC=SoC_from_OCV(tensioneMedia);
  // INIZIALIZZO IL TIMER PER L'ACQUISIZIONE
  MsTimer2::set(PERIODO_ACQUISIZIONE, nuovaAcquisizione); //
      eseguo la funzione nuovaAcquisizione() ogni 50 ms
  MsTimer2::start(); // abilito il timer
  leggiRTC(data, ora); // ora di partenza
}

void loop() {
  if(richiestaADC)
    corrente=leggiADC();
  if(recordPronto) { // se c'Ã¨ un nuovo dato acquisito da
      memorizzare lo salvo nella memoria SD
    recordPronto=false; // azzero il flag che segnala la presenza
        di un nuovo dato da salvare
    leggiRTC(data, ora); // acquisisco data e ora correnti dal
       RTC
    for(unsigned char i=0; i<8; i++)
      nomeFileLog[i]=data[i];
    // SALVO IL NUOVO DATO NELLA MEMORIA SD
    if (schedaPresente) { // se la scheda SD Ã¨ presente...
      boolean filePresente=false;
      if(SD.exists(nomeFileLog))
        filePresente=true;
      File logFile=SD.open(nomeFileLog, FILE_WRITE);
      if (logFile) { // se riesco ad aprire il file di log salvo
          i dati
        if(filePresente) // se il file esiste inizio una nuova
            riga
          logFile.println();
        else // altrimenti scrivo l'intestazione del file
          logFile.println(F("ora;tensione;corrente;temperatura;
              SoC"));
```

```
        logFile.print(ora);
        logFile.print(';');
        logFile.print(tensioneMedia);
        logFile.print(';');
        logFile.print(correnteMedia);
        logFile.print(';');
        logFile.print(temperaturaMedia);
        logFile.print(';');
        logFile.print(SoC);
        logFile.close(); // chiudo il file
      }
    } // fine salvataggio dati su memoria SD
  } // fine if(recordPronto)

  EthernetClient client = server.available(); // cerca i client
     che hanno qualche richiesta
  if (client) { // se c'Ã¨ qualche client che ha fatto richiesta
    // una richiesta HTTP termina con una riga vuota
    char bufferHTTP[BYTE_BUFFER_HTTP];
    unsigned char indice=0; // indice di scrittura per array
       bufferHTTP
    boolean currentLineIsBlank = true;
    while (client.connected() && client.available()) { // se
       client connesso e manda richieste al server
      char c = client.read();
      if (indice < BYTE_BUFFER_HTTP)
        bufferHTTP[indice++]=c;
      if (c == '\n' && currentLineIsBlank) { // fine richiesta
         HTTP: ora rispondo al client
        client.println(F("HTTP/1.1 200 OK"));
        client.println(F("Content-Type: text/html"));
        client.println(F("Connnection: keep-alive"));
        client.println();
        if(richiestaGET(bufferHTTP)) {
          if(!invioFile(nomeFileRichiesto, client)) { //; //
             invio il file richiesto se viene trovato nella
             memoria SD
            if(comandoGET("nuovoDato"))
              invioUltimoRecord(client);
            if(comandoGET("cercaFile")) { // richiesto il file di
                log attuale
              if (SD.exists(parametriRichiestaGET))
                client.print('1'); // il file richiesto esiste
              else
                client.print('0'); // il file richiesto non
                   esiste
            }
            if(comandoGET("eliminaFile")) {
              if (SD.remove(parametriRichiestaGET))
```

```
            client.print('1'); // file eliminato con successo
          else
            client.print('0');
        }
        if(comandoGET("data"))
          client.print(data);
        if(comandoGET("RTC")) { // richiesta la modifica del
            RTC
          byte ore = (parametriRichiestaGET[3]-'0') + 16*((
              parametriRichiestaGET[2]-'0'));
          byte minuti = (parametriRichiestaGET[8]-'0') +
              16*((parametriRichiestaGET[7]-'0'));
          byte giorno = (parametriRichiestaGET[13]-'0') +
              16*((parametriRichiestaGET[12]-'0'));
          byte mese = (parametriRichiestaGET[18]-'0') + 16*((
              parametriRichiestaGET[17]-'0'));
          byte anno = (parametriRichiestaGET[23]-'0') + 16*((
              parametriRichiestaGET[22]-'0'));
          scriviRTC(ore, minuti, giorno, mese, anno);
        }
        if(comandoGET("reset")) // richiesto il reset
          wdt_enable(WDTO_1S); // Reset Arduino tramite Watch
              Dog Timer
      }
    }
    else {
      if(schedaPresente) // azione predefinita in assenza di
          comandi
        invioFile("index.htm", client);
      else{
        client.print(F("<html><meta␣http-equiv=\"refresh\"␣
            content=\"10\">SoC="));
        client.print(SoC);
        client.print(F("<br>Tensione="));
        client.print(tensioneMedia);
        client.print(F("<br>Corrente="));
        client.print(correnteMedia);
        client.print(F("<br>Temperatura="));
        client.print(temperaturaMedia);
        client.print(F("<br>Scheda␣di␣memoria␣non␣presente<br
            ><a␣href=\"reset\">reset</a><html>"));
      }
    }
  }
  if (c == '\n') { // sta iniziando una nuova linea
    currentLineIsBlank = true;
  }
  else if (c != '\r') { // Ã¨ stato ricevuto un carattere
      sulla linea corrente
```

```
            currentLineIsBlank = false;
          }
        }
        // lascio al browser il tempo di ricevere i dati
        delay(1);
        // chiudo la connessione
        client.stop(); // disconnetto il client dal server
    }
}
```

Bibliografia

[1] H. J. Bergveld. «Battery management systems : design by modelling». Tesi di dott. Enschede, 2001. URL: http://doc.utwente.nl/41435/.

[2] Arduino. *Arduino IDE 1.0.5*. 2013. URL: http://arduino.cc/en/Main/Software.

[3] LEM. *Current transducer, voltage transducer, sensor, power measurement*. 2013. URL: http://www.lem.com.