**ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA**

**SEDE DI CESENA**

**SECONDA FACOLTA' DI INGEGNERIA CON SEDE A CESENA**

**CORSO DI LAUREA IN INGEGNERIA ELETTRONICA E TELECOMUNICAZIONI**

TITOLO DELL'ELABORATO

# Progetto di un sistema di gestione e comunicazione dati per un generatore di forme d'onda basato su microcontrollore

Elaborato in

**Elettronica dei sistemi digitali**

| Relatore | Correlatore | Presentato da |
|---|---|---|
| Prof. Ing. Aldo Romani | Dott. Michele Dini | Fiumana Mattia |

Sessione II

Anno Accademico 2011/2012

# Indice generale

# Introduzione

Il progetto, la cui parte di gestione è descritta in questo testo, ha come scopo lo sviluppo di un sistema per la riproduzione, attraverso un sistema vibrante elettrodinamico (shaker), di vibrazioni acquisite dall'ambiente circostante in situazioni di vita quotidiana (esempio le vibrazioni prodotte da un treno in movimento o da una camminata) o ideali (ottenute tramite strumenti di laboratorio), al fine di caratterizzare trasduttori piezoelettrici per studiarne il funzionamento, le caratteristiche e il loro comportamento sotto stress.

Nell'elaborato viene descritta la realizzazione di un sistema di gestione e controllo di forme d'onda, finalizzata al pilotaggio dello shaker.

La gestione avviene per mezzo di un programma installato su un PC che consente all'utente di trasmettere e ricevere dal sistema di controllo dello shaker. Per far ciò è stata realizzata un interfaccia grafica che consente all'utente un'interazione semplice ed efficiente col sistema.

Allo stesso modo è stato creato un programma, eseguito dal microcontrollore presente sulla scheda di controllo, che consente di comunicare col PC e con le altre periferiche presenti nel sistema, come il salvataggio o la lettura dei dati sulla memoria del sistema.

Nei capitoli successivi viene descritto come questo progetto opera nello studio di nuove fonti energetiche per elettronica mobile e portatile.

Nel primo capitolo viene descritta la finalità per cui è stato realizzato il sistema. Vengono inoltre riportate le specifiche tecniche che occorre rispettare per realizzare un sistema in grado di operare correttamente sia con l'utente che con le periferiche.

Nel capitolo riguardante il software viene ampiamente descritta la struttura del sistema, come esso interagisce con ogni suo componente (librerie dedicate per ogni periferica) e come l'utente può operare nella gestione dei dati.

Il terzo contiene la descrizione della struttura hardware della scheda nei suoi particolari.

Il capitolo conclusivo racchiude le considerazioni finali e i risultati ottenuti al termine del progetto.

# Capitolo 1 - Il progetto

## 1.1 - Energy harvesting

Negli ultimi anni il problema energetico è stato messo in primo piano a causa di una ricerca sempre crescente volta a ridurre gli sprechi, a fronte di una costante diminuzione delle risorse non rinnovabili presenti sul nostro pianeta [1].

Anche l'elettronica sta subendo questo mutamento: se infatti in passato si è sempre cercato di aumentare le prestazioni dei dispositivi senza tenere in considerazione i consumi, al giorno d'oggi i due aspetti viaggiano di pari passo. Complice di questa evoluzione è la miniaturizzazione dei dispositivi elettronici e la necessità di avere sistemi portabili (ad esempio smartphone, computer portatili, ecc...).

E' proprio qui che nasce il concetto di e*nergy harvesting*, ossia l'acquisizione di energia da sorgenti alternative disponibili nell'ambiente (fig. 1.1). Ad esempio una semplice camminata oppure lo star seduti in una carrozza di un treno possono essere sfruttati per immagazzinare piccole quantità di energia che possono poi essere utilizzate per diversi scopi, come il monitoraggio remoto di pazienti, il controllo efficiente dell'energia negli uffici oppure l'alimentazione di sensori che tengono monitorato il traffico su un ponte. Alcuni esempi di sorgenti utilizzate in ambito di Energy Harvesting sono:

- Celle solari;
- Trasduttori piezoelettrici;
- Convertitori di onde elettromagnetiche;
- Generatori termoelettrici [2].

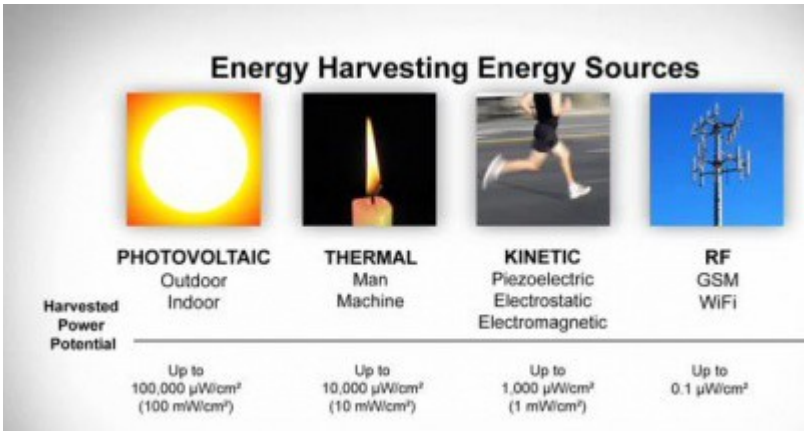


*fig. 1.1: Esempi di sorgenti tipicamente sfruttate per l'energy harvesting [3]*

## 1.2 - Descrizione del progetto

Il lavoro svolto durante la realizzazione del progetto è destinato allo studio e all'implementazione di parte di un sistema vibrante (shaker elettrodinamico) per la caratterizzazione di trasduttori piezoelettrici e la replica di vibrazioni acquisite in ambienti reali.

Le tipologie di sorgenti trattate in questo progetto sono trasduttori piezoelettrici, in grado di recuperare energia da vibrazioni a bassa frequenza. Benché eroghino bassissime potenze (dell'ordine delle centinaia di µW) sono comunque in grado di alimentare dispositivi portatili a microcontrollore, come sensori wireless e sistemi di monitoraggio ambientale.

Per lo studio in laboratorio dei trasduttori piezoelettrici ci si avvale di uno shaker elettrodinamico. Il suo movimento trasmetterà vibrazioni ai sensori piezoelettrici i quali genereranno una potenza proporzionale all'accelerazione a loro imposta. Quest'accelerazione è conosciuta e resa regolabile all'utente per permettergli la caratterizzazione del trasduttore in determinate condizioni e per la simulazione del comportamento in ambiente reale [4].

## 1.3 - Struttura del sistema

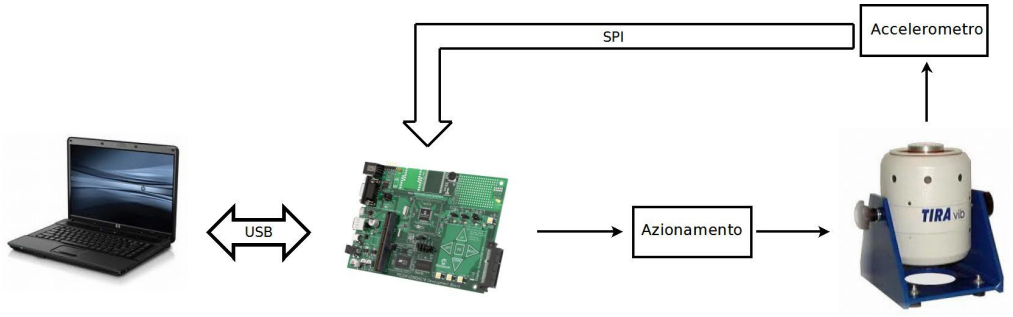


*fig. 1.2: Schema del sistema*

Il controllo dello shaker avviene per mezzo di un microcontrollore che, tramite la generazione di alcuni segnali, impone e tiene costantemente monitorata l'accelerazione imposta alla struttura da testare e acquisisce i dati necessari per l'elaborazione successiva.

In questo elaborato non verranno trattate ulteriormente le modalità con cui il microcontrollore interagisce con lo shaker, ma come verrà programmato il microcontrollore del sistema di controllo per poter interagire con l'utente in maniera semplice e dinamica. Questo perché non occorre solo che il dispositivo elabori le informazioni, ma che esse possano anche essere lette e interpretate da un operatore che effettua la misura.

Si mostrerà in dettaglio come il microcontrollore e il PC interagiscono (fig. 1.3) e il modo in cui un operatore possa gestire le informazioni che sono presenti nel sistema di controllo dello shaker.

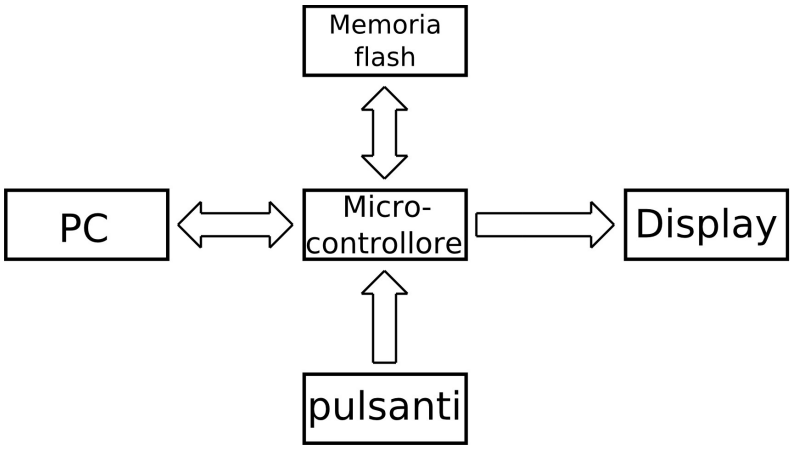


*fig. 1.3: Schema a blocchi del sistema relativo al progetto*

Tramite PC si ha una comunicazione bidirezionale con la scheda utilizzando l'interfaccia USB, permettendo così sia la scrittura che la lettura dati. In fase di scrittura verranno caricate le ampiezze della vibrazione che dovrà riprodurre lo shaker, mentre in lettura verranno salvati su file i dati essenziali all'elaborazione, quali tensione e accelerazione.

Il display è usato per la visualizzazione dell'avanzamento delle fasi di scrittura e lettura. Esso permette anche di interagire con l'utente tramite l'utilizzo di menù a scorrimento. L'utilizzo di pulsanti si rende quindi necessario per poter scorrere tra le funzionalità messe a disposizione ed eseguirle.

Una memoria non volatile è essenziale per permettere di salvare in maniera duratura i dati per le misure oltre ai risultati delle stesse.

## 1.4 - Specifiche e scelte progettuali

E' stato necessario in fase di progettazione stabilire alcuni vincoli per poter poi scegliere in maniera corretta i diversi componenti, in particolare il microcontrollore, il display, la memoria e per riuscire ad impostare correttamente il software in modo da rispettare precise regole.

### 1.4.1 - Specifiche hardware

La scelta dell'hardware è indispensabile per la realizzazione del progetto perché la scelta di componenti che non rispecchiano le esigenze progettuali ne potrebbe comportare il fallimento.

L'elemento principale è il microcontrollore, a causa della sua posizione centrale su ogni operazione, come si può vedere in fig. 1.2 e in fig. 1.3. Per il progetto ci si concentrati sulle serie ad 8 e 16 bit della Microchip sostanzialmente per due motivi:

- Elevata quantità di materiale messo a disposizione dalla casa produttrice, come librerie, documentazione tecnica e di supporto.
- Vasta gamma di prodotti.

Per prima cosa è necessario avere alta velocità computazionale e una disponibilità di memoria sufficiente a contenere tutti i dati necessari alle misure. Questo ha spinto verso la gamma di microcontrollori a 16 bit ed escludendo a priori quella minore ad 8 bit.

Oltre alla velocità e alla capacità di memorizzazione è necessario che il microcontrollore disponga di almeno due periferiche: USB e SPI. La prima serve per

comunicare con il PC mentre la seconda è necessaria per ricevere i dati da parte dell'accelerometro presente sullo shaker.

Successivamente ci si è concentrati sulle modalità di salvataggio dei dati caricati dal PC e i risultati delle varie misure. La scelta fatta è quella di utilizzare una memoria flash esterna con interfaccia SPI, così da condividere la stessa periferica usata per l'accelerometro. L'accesso in memoria si ha infatti esclusivamente a misura ultimata escludendo così conflitti in fase di comunicazione e accesso alle risorse .

Fortunatamente Microchip rende disponibile una scheda (fig. 1.4) contenente già tutti gli elementi necessari per la corretta esecuzione del progetto, in particolare:

- Microcontrollore PIC24FJ256DA210;
- Connettore USB;
- Memoria flash SST25VF016B;
- Pulsanti.

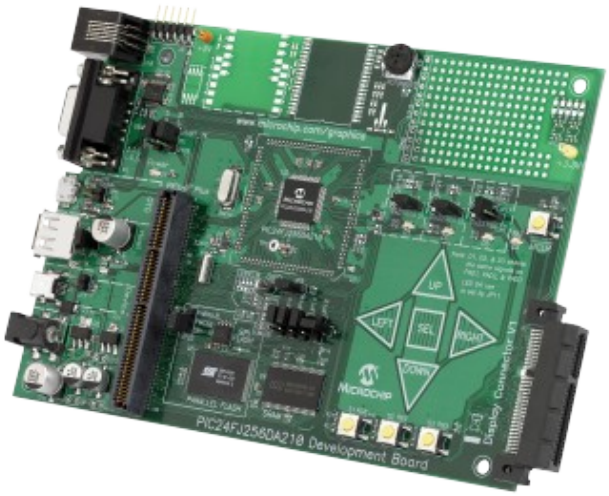


*fig. 1.4: Scheda di sviluppo Microchip: PIC24FJ256DA210 Development Board*

Sempre a livello hardware è necessario disporre di un display LCD per poter comunicare con l'utilizzatore della scheda. Il display scelto è il modello MC42005A6W-BNMLW, in grado di mostrare venti caratteri per riga su quattro righe. Qualora fosse necessario è possibile utilizzare anche la retroilluminazione, effettuando gli opportuni collegamenti.

### 1.4.2 - Specifiche software

Analogamente alla parte hardware, è necessario stabilire alcuni punti di partenza per poter realizzare il programma che andrà successivamente implementato nel PIC.

Inanzi tutto è stato stabilito che la dimensione massima dei campioni di una forma d'onda da riprodurre sarà di 10000 elementi (due byte a campione), pertanto è stato realizzato un buffer in RAM in grado di contenerne fino a 12500 per permettere flessibilità sul numero di campioni utilizzabili. In ogni caso il numero massimo di campioni caricati non sarà mai superiore a 10000 vista la scelta di realizzare onde arbitrarie di durata dieci secondi con una frequenza di campionamento di 1000 Hz.

Infine per uniformare la gestione dei buffer in memoria è stata creata una struttura fissa nello spazio di memoria flash composta da 10 buffer di dimensione fissa (fig. 1.6) divisi in 3 sezioni. La prima contiene la dimensione dei dati salvati ed è composta da 2 Byte, la seconda contiene il nome del buffer ed è di dimensione fissa di 8 Byte, nella terza sono contenuti tutti i campioni relativi alla forma d'onda in questione.

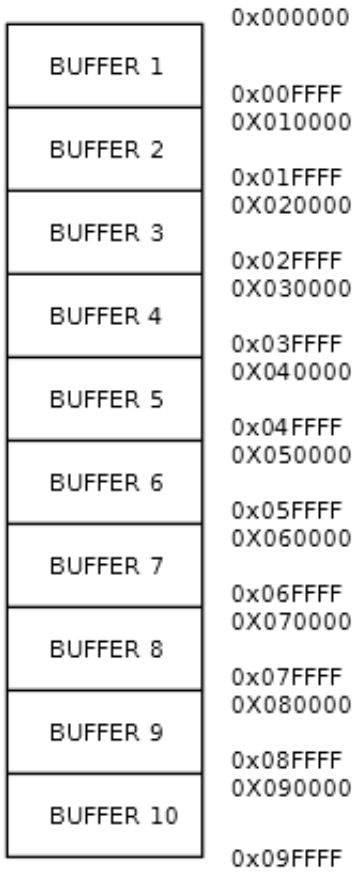


*fig. 1.5: Indirizzamento dei buffer in memoria*

I buffer inizieranno rispettivamente all'indirizzo 0x000000, 0x010000 fino a 0x090000 (fig. 1.5).

L'area dati ha dimensione fissa nonostante il numero di campioni sia variabile. Questa scelta è stata fatta per evitare di dover cancellare la memoria ad ogni cancellazione di un buffer.

Il campo *lunghezza* è un indice fondamentale per sapere se il buffer contiene dati o è vuoto, infatti se posta a 0 il sistema è in grado di capire che lo spazio in memoria è utilizzabile e non impegnato. Questo procedimento si è reso necessario perché la cancellazione di tutto il banco comporterebbe l'usura prematura e non giustificata della memoria. In questo modo si ha, tramite una semplice scrittura, la cancellazione "virtuale" del banco selezionato.

Una struttura C che descrive il buffer realizzato è la seguente:

```
struct wave_buffer {
        unsigned int lunghezza;
        char nome[8];
        char dati[25000];
}
```

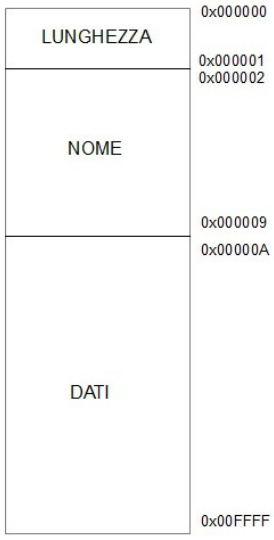


*fig. 1.6: Struttura del buffer 1*

### 1.4.3 - Specifiche di progetto

Sempre in fase di progettazione sono state decise alcune specifiche utili alla realizzazione del programma.

Il valore numerico dei singoli campioni da caricare in flash deve essere compreso da 0 a 4095, a causa della risoluzione del DAC (LTC1450) scelto per la generazione del segnale analogico con cui pilotare l'azionamento dello shaker. Difatti esso effettua conversioni D/A a 12 bit e non è possibile fornirgli un valore in ingresso superiore 4095 falserebbe la misura non potendo essere riprodotto correttamente.

L'intervallo di campionamento delle forme d'onda è di 1 ms, pertanto le temporizzazioni da riportare sui file sia da caricare che da salvare avranno questo intervallo fisso tra un campione e il successivo.

## 1.5 - Schematico

Lo schema (fig. 1.7) mostra come sono state realizzate le connessioni tra i dispositivi utilizzati nel progetto. Quello mostrato è relativo alla parte di progetto, le altre connessioni sono presenti sulla guida[1] della scheda di sviluppo.

La memoria flash presenta i pin *Hold* e *WP* sempre non attivi, permettendo una migliore velocità di comunicazione. Infatti in protezione alla scrittura è già presente il registro di stato e il comando di *write_enable*.

I pulsanti presentano una configurazione di tipo *pull-down*, portando la tensione a zero solo nell'istante in cui viene premuto.

Il display comunica col microcontrollore con quattro pin (D4-D7) e i pin di controllo sono il *Register Select Signal* (RS) e l'*enable* (EN). Il display può essere configurato anche in lettura, per monitorare il suo stato. Per il progetto, non essendo richiesta tale funzione, è utilizzato in sola scrittura. Il potenziometro permette la regolazione del contrasto e il valore di 1 KOhm è stato scelto per poter settare correttamente questo parametro.

1 Disponibile sul sito di Microchip: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en547654

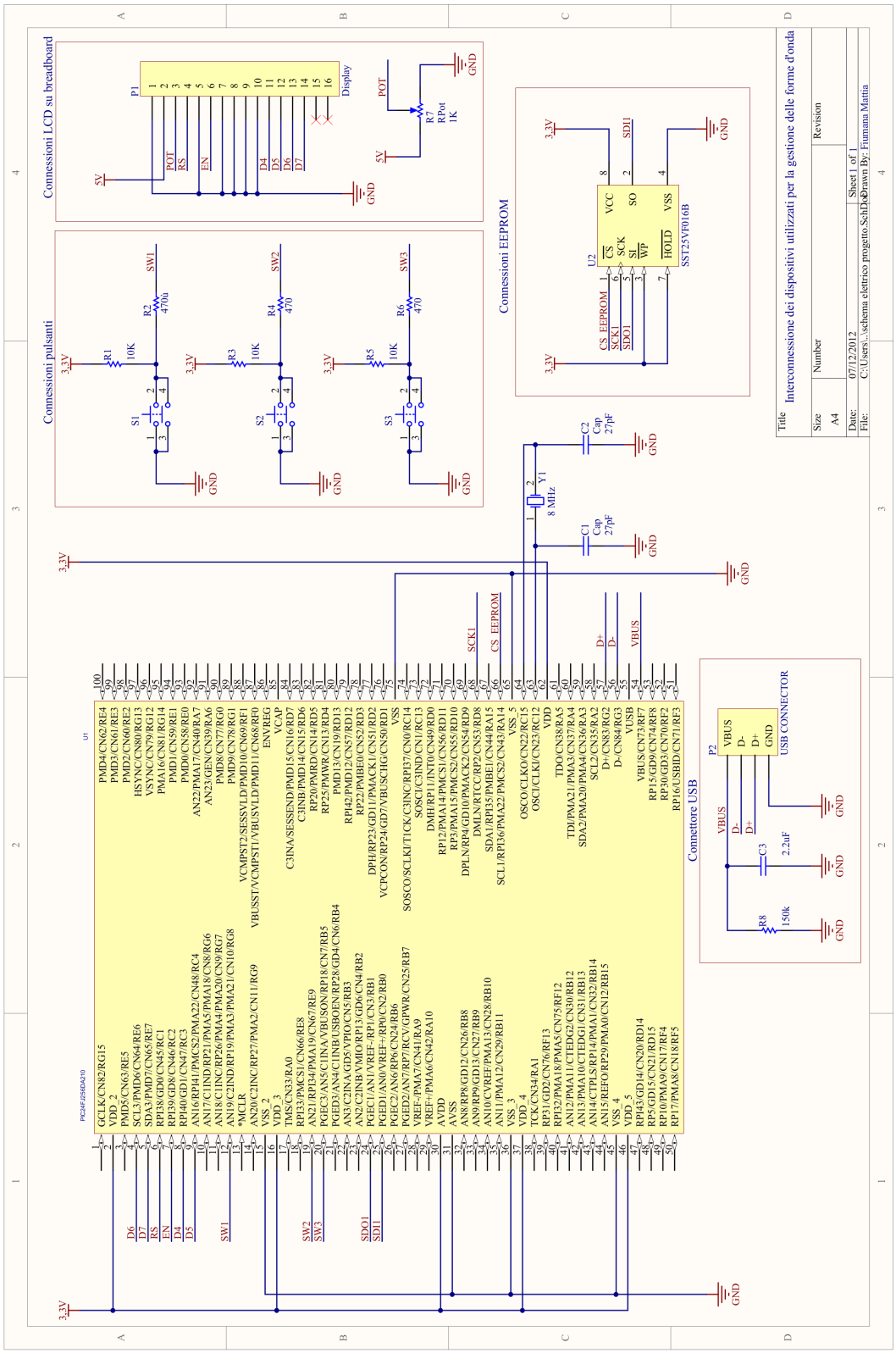


*fig. 1.7: Schema elettrico relativo al progetto. I pin che in questo schema risultano non connessi rispettano le configurazioni presenti sulla documentazione della development board.*

# Capitolo 2 - Software

## 2.1 - Introduzione

Il programma per la comunicazione PC-scheda di sviluppo è stato scritto in Perl-Tk, in maniera tale da dotarlo di interfaccia grafica. Il codice del programma del microcontrollore presente nella scheda di sviluppo è invece stato scritto in C (compilato tramite compilatore C30 di Microchip) [5].

Il Perl fu creato come ausilio ai sistemisti, inizialmente nato come linguaggio di manipolazione di testo e file si è poi evoluto grazie a moduli aggiuntivi per l'elaborazione grafica, manipolazione dei banche dati, gestione dei processi di comunicazione via rete. Esso è utilizzabile in tutti quegli ambiti in cui non siano strettamente necessarie le performance di un linguaggio compilato, offrendo al contempo tempi di sviluppo molto rapidi.

Questo linguaggio è stato pensato per essere pratico e duttile, trascurando la compattezza. Il Perl infatti è noto per il detto "c'è più di un modo per farlo", proprio perché per raggiungere un obiettivo il programmatore può ottenere gli stessi risultati usando metodi diversi. Questo fa sì che il linguaggio sia utilizzabile da una vasta gamma di utenti, da chi si è appena cimentato nel suo utilizzo e da chi ci lavora da anni. Grazie a ciò si è in grado di risolvere problemi che con altri linguaggi di programmazione risulterebbero molto più complessi.

Con l'utilizzo di Perl-Tk, cioè l'estensione grafica di Perl, è stata creata l'interfaccia grafica con la quale l'utente interagisce con il software.

Il linguaggio C è utilizzato per la programmazione del microcontrollore. Microchip mette a disposizione un ambiente di sviluppo gratuito (MPLAB) per l'utilizzo dei suoi prodotti, unitamente ad un compilatore per il linguaggio C (diverso per ogni famiglia di microcontrollori). Il compilatore utilizzato in questo lavoro è il C30 per la famiglia di microcontrollori a 16 bit.

## 2.2 - Emulazione della porta seriale tramite USB

La comunicazione tra microcontrollore e PC avviene tramite interfaccia USB [6]. Per motivi di semplicità, si è scelto di emulare una porta seriale standard (RS-232) sulla connessione USB: dopo l'installazione degli opportuni driver forniti da Microchip il sistema operativo riconosce l'avvenuta connessione con la scheda di sviluppo mostrando un nuovo dispositivo di comunicazione seriale (ad esempio una nuova porta COM sui sistemi operativi Microsoft) nell'elenco dei dispositivi di sistema.

La possibilità di utilizzare questo tipo di connessione ci permette di trasmettere e ricevere dati con facilità senza il bisogno di implementare librerie aggiuntive sul compilatore Perl.

## 2.3 - Programma installato su PC

### 2.3.1 - Pre-requisiti

Il programma scritto può essere eseguito su qualsiasi calcolatore con sistema Linux. Se si dispone di una macchina Windows occorre installare il software Cygwin, un emulatore in grado di eseguire su macchine Windows diversi comandi Linux.

In entrambi i casi, sia che si lavori su Linux che su Cygwin, si deve procedere con l'installazione di Perl [7] e della sua estensione Tk.

La comunicazione tra scheda di sviluppo e PC avviene tramite l'emulazione di una porta seriale. Per permettere che il computer riconosca questa connessione bisogna installare il driver fornito da Microchip (mchpcdc.inf)[2].

Il programma Perl è impostato per comunicare con la porta seriale /dev/ttyS3 (in Windows COM4). E' necessario verificare che il PC riconosca la seriale in maniera corretta, altrimenti si deve procedere a rinominare la porta.

### 2.3.2 - TK

L'estensione Tk [8] ha come scopo realizzare un interfaccia grafica con cui l'utente potrà interagire col programma. Con comandi dedicati si è in grado di creare pulsanti, finestre di dialogo e di testo e molto altro. Un semplice esempio di come questo motore funziona può essere la creazione di una finestra è il seguente:

2 Il driver è reperibile a questo indirizzo: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en547784

```
#!/usr/local/bin/perl
use Tk;
my $mw = new MainWindow;
my $label = $mw -> Label (-text => "Clicca per uscire") -> pack();
my $button = $mw -> Button(-text => "Esci", -command => sub {exit} ) -> pack ();
MainLoop;
```

Il risultato del codice scritto è il seguente:

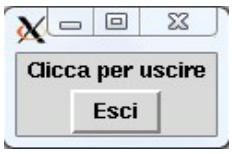


*fig. 2.1: Esempio di finestra con l'utilizzo di Tk*

Partendo da questo piccolo esempio si può arrivare alla realizzazione di programmi complessi, come è avvenuto per questo progetto.

### 2.3.3 - Funzionamento

Lo scopo del progetto è quello di creare un sistema di facile utilizzo da parte dell'utente, in grado di comunicare col microcontrollore e svolgere diverse funzioni utili alla gestione dei dati.

L'utente potrà caricare un qualsiasi file di testo (vedi paragrafo 2.3.5) presente sul proprio terminale ed inviarlo tramite comunicazione su porta seriale. Quest'operazione può essere eseguita per tutti i buffer presenti in memoria.

Allo stesso modo l'operatore può decidere di caricare il contenuto presente su un determinato buffer e salvarlo sul proprio terminale. Il file sarà sempre di tipo testo e con caratteristiche spiegate nel paragrafo 2.3.5.

Oltre a questo, è possibile cancellare il contenuto di un buffer intero qualora il suo contenuto non sia più di utilità, in modo da permettere il salvataggio di altre forme d'onda o eventuali dati acquisiti dalla misura. Allo stesso modo è possibile cancellare il contenuto dell'intera memoria.

E' infine possibile ricevere lo stato di tutti i buffer in modo da poter tener monitorato l'utilizzo della memoria. Se un buffer risulta pieno verrà riportato il suo nome e la

dimensione, viceversa verrà mostrato che il buffer è libero e quindi utilizzabile (fig. 2.3).

### 2.3.4 - Interfaccia grafica

Un elemento fondamentale per rendere il programma accessibile a chiunque debba eseguire la misura è avere a disposizione un interfaccia grafica semplice che permetta di capire cosa sta succedendo e cosa si sta facendo. L'interfaccia grafica è visibile in fig. 2.2.

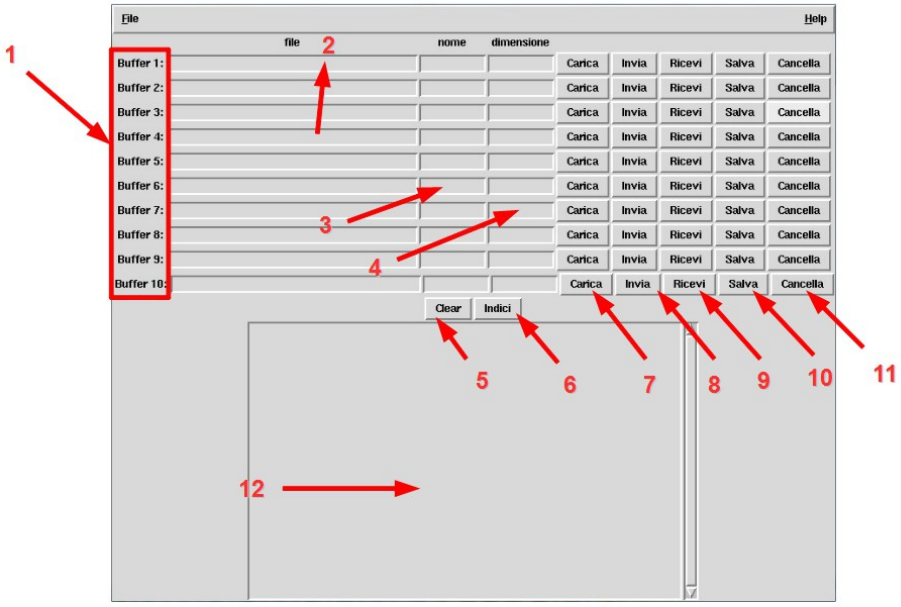


*fig. 2.2: Interfaccia grafica del programma*

Di seguito viene descritta la finestra:

1. Indica il buffer di riferimento, ad ogni buffer è associato uno spazio in memoria di 25KB.
2. Percorso del file: una volta selezionato il file di testo (paragrafo 2.3.5) desiderato tramite il pulsante “Carica” (7) verrà visualizzato il suo percorso. E' possibile scegliere un file diverso per ogni buffer.
3. Nome: si può inserire il nome da assegnare al buffer. Esso verrà salvato in memoria (fig. 1.6) ed il numero massimo di caratteri è 8, pertanto ogni carattere

in più verrà scartato.

4. Dimensione: una volta caricato il file verrà riportata la sua dimensione (in numero di campioni). Una dimensione superiore a 12500 porterà ad un errore e al mancato salvataggio dell'onda sulla memoria flash.
5. Clear: cancella la memoria flash presente sulla scheda di sviluppo.
6. Indici: riceve dalla scheda lo stato dei buffer e li riporta nella finestra di dialogo (12).
7. Carica: carica un file presente sul disco.
8. Invia: I dati vengono trasferiti dal PC alla scheda di sviluppo per il salvataggio in memoria
9. Ricevi: legge il contenuto di un buffer solo qualora esso contenga dati.
10. Salva: Una volta ricevuto il file tramite (9) viene salvato su un file di testo a scelta.
11. Cancella: Effettua la cancellazione del buffer.
12. Finestra di dialogo: Vengono riportati tutti i messaggi della trasmissione. Nel caso venga premuto il tasto Indici verranno riportati i dati relativi ai singoli buffer (fig. 2.3)

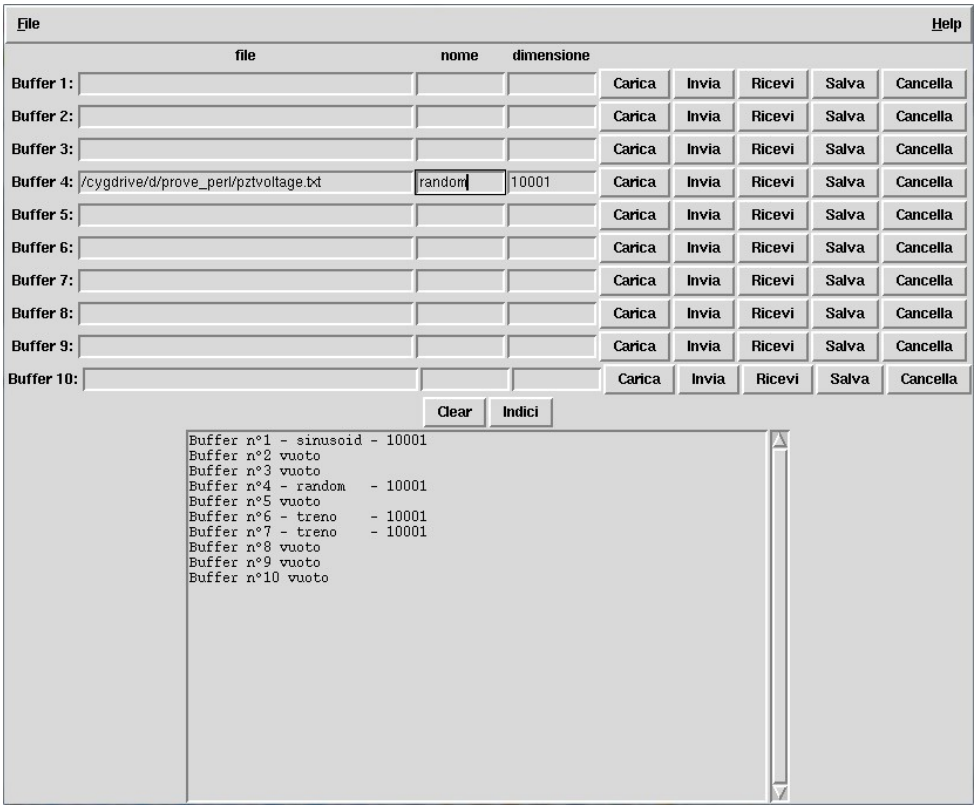


*fig. 2.3: Esempio di funzionamento del programma*

### 2.3.5 - Struttura del file di testo

Per poter caricare e di conseguenza inviare correttamente la forma d'onda desiderata sono indispensabili queste specifiche.

Di default il programma cerca solo file con estensione .txt, anche se è consentito utilizzare ogni tipo di file di testo. Essi devono essere composti da tante righe quanti sono i campioni da inviare e ogni riga deve contenere due dati, il tempo e il valore del campione, entrambi in formato scientifico (esempio 4.004E-03).

La riga deve iniziare con tre spazi vuoti seguiti dal valore temporale, successivamente dovrà essere inserito un carattere di tabulazione e due spazi vuoti. Il terzo spazio è facoltativo e utile qualora si voglia implementare l'invio di caratteri negativi (serve solo al fine di una lettura più confortevole), però per questo progetto non sarà necessario a seguito delle specifiche progettuali. Infine dovrà essere presente il valore del campione seguito da un carattere di tabulazione e da un invio (il carattere ASCII 0x0A) (fig. 2.4).

```
···1.9000000e-02 » ···5.9450714e-01 » ¶
···2.0000000e-02 » ···1.0937500e+00 » ¶
···2.1000000e-02 » ···1.5133929e+00 » ¶
···2.2000000e-02 » ···1.9330357e+00 » ¶
···2.3000000e-02 » ···2.2991071e+00 » ¶
···2.4000000e-02 » ···2.5580357e+00 » ¶
···2.5000000e-02 » ···2.6562500e+00 » ¶
···2.6000000e-02 » ···2.6150000e+00 » ¶
···2.7000000e-02 » ···2.5012500e+00 » ¶
···2.8000000e-02 » ···2.3300000e+00 » ¶
···2.9000000e-02 » ···2.1162500e+00 » ¶
```

*fig. 2.4: Esempio di struttura del file di testo*

Un altro elemento fondamentale per la corretta creazione del documento di testo è il valore di ampiezza del campione che dovrà essere compreso da 0 a 4,095e+00. Questo è necessario perché il programma effettuerà automaticamente una moltiplicazione x1000 una volta caricato il file.

Qualora si decida di non seguire questa specifica occorrerà in fase di misura implementare una funzione di condizionamento dell'onda salvata per portarne il campo di variazione da 0 a 4095.

Analogamente quando un file viene letto dalla memoria e salvato, viene ricreato un file di testo impaginato come il precedente. I campioni saranno sempre a distanza di 1 ms come da specifiche e in maniera opposta a ciò che avviene per il caricamento, il dato ricevuto dal PIC verrà diviso per 1000.

### 2.3.6 - Funzioni

Nella tabella sottostante sono racchiuse le funzioni implementate nel programma:

| Nome funzione | Codice | Descrizione | Pulsante |
|---|---|---|---|
| push_bufferN | None | Carica un file da disco e lo prepara all'invio. Il file caricato è associato al buffer N. | Carica |
| push_send_bufferN | 0x6562 | Invia i campioni della forma d'onda caricata e li salva nel buffer N. | Invia |
| read_bufferN | 0x6563 | Il PIC legge in memoria flash il buffer N e lo invia al PC. | Ricevi |
| push_salva_bufferN | None | Salva su un file .txt scelto i dati ricevuti. | Salva |
| push_cancella_bufferN | 0x6566 | Inizializza il buffer N | Cancella |
| push_Index | 0x6561 | Il PIC legge gli indici in memoria e li invia al PC (fig. 9) | Indici |
| push_Clear | 0x6564 | Inizializza tutta la memoria | Clear |

*Tabella 1: Funzioni relative al programma in Perl-Tk*

La colonna “codice” rappresenta il valore esadecimale indicativo della chiamata, quando il microcontrollore riceve questo codice eseguirà la funzione ad esso associata, come sarà mostrato nel dettaglio successivamente.

#### 2.3.6.1 - push_bufferN

Questa funzione permette di caricare un file di testo dal PC e prepararlo all'invio. Selezionando il tasto “carica” viene aperta la finestra (fig. 2.5) che permette di scegliere il file desiderato all'interno del PC.

Il formato che viene impostato di default è .txt anche se ai fini del progetto ogni file di tipo testo viene elaborato correttamente.

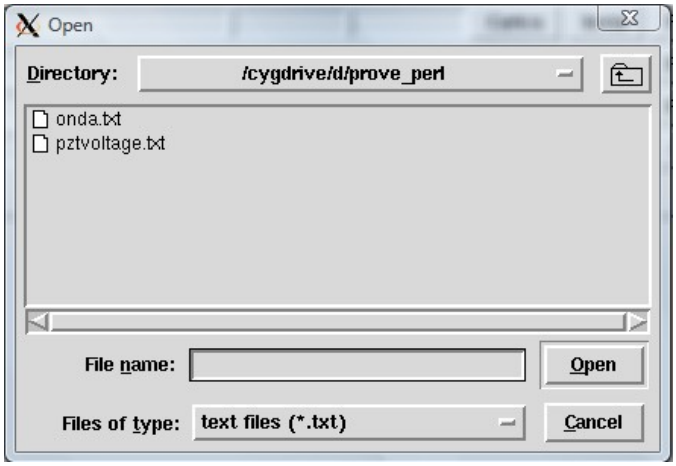

*fig. 2.5: Finestra di caricamento*

Il percorso del file viene riportato nella casella relativa al buffer utilizzato e alla voce "file", mentre in lunghezza viene scritto il numero dei campioni presenti (vedi fig. 2.3).

Questa procedura è eseguibile per ogni buffer permettendo il caricamento di un massimo di dieci file contemporaneamente, uno per ogni buffer presente in memoria.

Un esempio di codice per far capire la semplicità di utilizzo del codice Perl-Tk è il seguente:

```
my $types = [['text files', '.txt'], ['All Files', '*'],];
$open1 = $mw → getOpenFile(-filetypes => $types, -defaultextension => '.txt');
```

Queste due righe di codice permettono la visualizzazione della finestra in fig. 2.5 e il salvataggio del percorso del file selezionato nella variabile *open1*.

Caricato il file occorre dare un nome al buffer su cui si andrà a scrivere. Deve contenere al massimo otto caratteri vista la struttura della memoria, eventuali lettere aggiuntive verranno cancellate (esempio: Sinusoide diventerà Sinusoid).

### 2.3.6.2 - push_send_bufferN

La funzione permette l'invio dei dati relativi al buffer utilizzato al microcontrollore tramite porta seriale.

E' necessario aver già caricato un file tramite *push_bufferN* prima di procedere con l'invio per evitare che il sistema dia errore.

La trama richiede l'invio dei dati seguendo una struttura fissa (fig. 2.6).

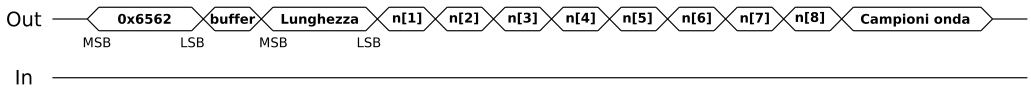


*fig. 2.6: Trama della funzione push_send_bufferN*

Per rispettare la trama in fig. 2.6 la funzione effettua separatamente l'invio di tutti i parametri seguendo il diagramma in fig. 2.7.

Per prima cosa recupera il contenuto del file caricato precedentemente e ne effettua la codifica, eliminando la colonna dei tempi e trasformando il formato scientifico dei campioni in intero a 16 bit.


*fig. 2.7: diagramma a blocchi della funzione push_send_bufferN*

Per esempio la conversione di tutti i campioni da formato scientifico ad intero avviene in questo modo:

```
for ($i=0; $i < $size_entity; $i++){
    $lines[$i] = sprintf("%.10g", $lines[$i]);
    $lines[$i] = $lines[$i] * 1000;
    $lines[$i] = sprintf ("%.0f", $lines[$i]);
    $lines[$i] = "$lines[$i]" ;
    $lines[$i] = pack ("n*", $lines[$i]);
}
```

Prima si elimina la dicitura scientifica lasciando comunque intatti i valori decimali. Viene poi convertito in intero in modo da avere valori compresi tra 0 e 4095 per poi generare tramite *pack* i caratteri codificati per la trasmissione.

Elaborati tutti i campioni da inviare inizia la trasmissione. La porta seriale può essere interpretata dal programma come un semplice file e potervi scrivere utilizzando le funzioni di base del linguaggio.

```
open TRANS, ">/dev/ttyS3" or die $!;
```

In questo modo la porta */dev/ttyS3* viene aperta in sola scrittura, in modo che il programma possa "scriverci" sopra. Ora basta usare la funzione *print TRANS* per inviare un qualsiasi dato tramite seriale. Al termine di ogni trasmissione è opportuno chiudere la comunicazione tramite *close TRANS*.

Dalla fig. 2.7 si vede come vengono eseguite le trasmissioni in maniera separata, prima viene inviato il codice della funzione seguito dal numero del buffer su cui scrivere, lunghezza, nome e valore di tutti i campioni.

Degno di nota è l'utilizzo delle regular expressions per modificare il contenuto delle variabili come, ad esempio, una riga del file di testo caricato:

```
for ($i = 0; $i < $size_entity; $i++){
   $lines[$i] =~ s/^.+[\011]  //g;
   $lines[$i] =~ s/^ //g;
   $lines[$i] =~ s/[\011].+$//g;
}
```

La prima operazione del ciclo elimina tutti i caratteri precedenti al primo segno di tabulazione (*[\011]*). Nella seconda riga si procede con la rimozione di tutti gli spazi concludendo con la cancellazione di tutti gli elementi successivi all'ultimo segno di

tabulazione. Il risultato di questa operazione è un vettore contenente il valore in formato scientifico dei soli campioni della forma d'onda.

Un processo analogo avviene per il nome, consentendo di eliminare eventuali caratteri in più oppure aggiungere degli spazi vuoti fino ad ottenere un numero complessivo di otto caratteri.

### 2.3.6.3 - read_bufferN

Lo scopo della funzione è ricevere dal microcontrollore il valore dei campioni presenti in memoria. La trama (fig. 2.8) prevede l'invio del codice associato alla funzione seguito dal numero del buffer. La risposta del sistema è costituita dagli elementi contenuti nel buffer seguito dalla stringa di terminazione.

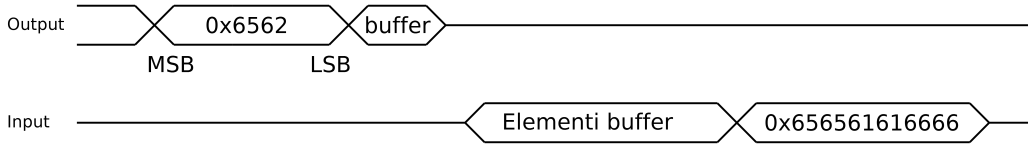


*fig. 2.8: Trama per la funzione read_bufferN*

La funzione ha una struttura simile alla precedente (fig. 2.9). Prima esegue l'invio del codice della funzione e il numero del buffer selezionato tramite la porta seriale, esattamente come avviene per la funzione push_send_bufferN.

Per ricevere i dati dalla porta seriale occorre utilizzare il comando *open*, analogamente a come avviene per la trasmissione:

*open TRANS, "</dev/ttyS3" or die $!;*
Per ricevere i dati il programma si mette in attesa utilizzando questa funzione:

*if ($n = read FILE, $data, 256 != 0)*
La condizione si verifica ogni volta che almeno un byte viene ricevuto dalla porta seriale (da ricordarsi che viene considerata come un file) salvandone il valore nella variabile *data*.

La ripetizione di questo controllo, fino alla ricezione della stringa di terminazione, permette al sistema di trasferire tutti i dati dal microcontrollore al PC.

Il salvataggio dei dati ricevuti viene effettuato su un array che ha come finalità l'intermediazione tra questa funzione e quella di salvataggio ( 2.3.6.4 -push_salva_bufferN).

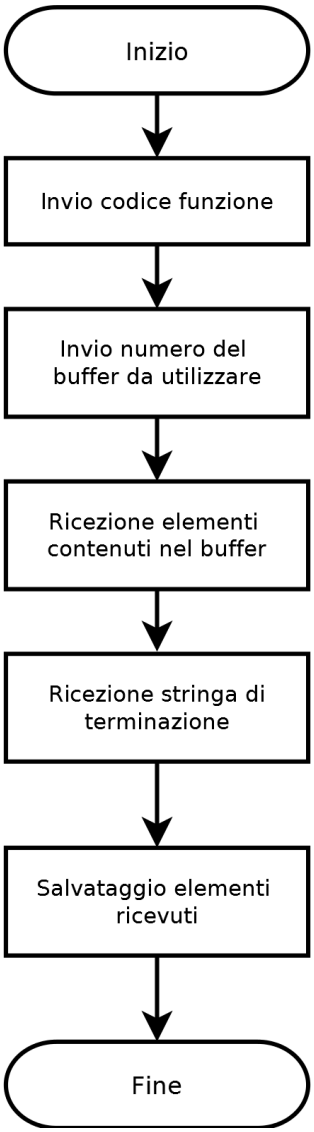

*fig. 2.9: Struttura della funzione*

### 2.3.6.4 - push_salva_bufferN

I dati ricevuti con la funzione precedente non sono utilizzabili direttamente dall'utente. Occorre creare un file di testo contenente il valore dei campioni memorizzati utilizzando la stessa struttura descritta precedentemente (paragrafo 2.3.5).

Selezionando il pulsante “Salva” il sistema chiede all'utente di selezionare un file esistente o di crearne uno nuovo (fig. 2.10). Di default viene usata l'estensione .txt, ma selezionando, alla voce “Files of type”, l'elemento “All File” permette l'utilizzo di estensioni diverse.

Per utilizzare questa funzione occorre eseguire la funzione read_bufferN.

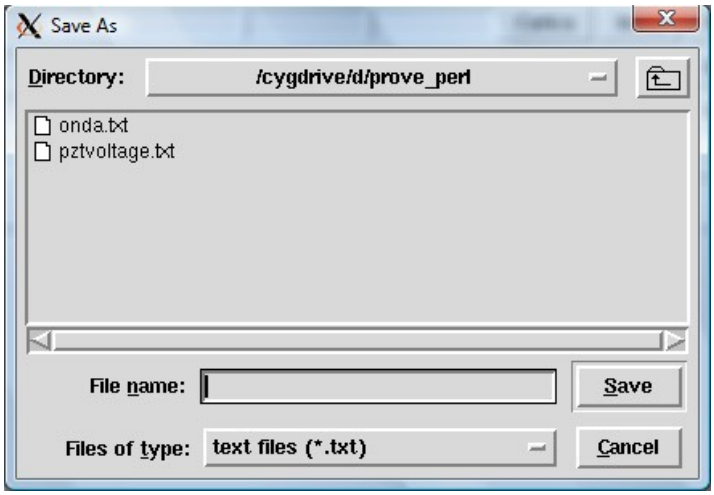


*fig. 2.10: finestra di salvataggio*

Grazie alle regular expressions descritte prima viene ricreata la struttura del file composta dalla colonna tempo e valore del campione in formato scientifico, come riportato in fig. 2.11.

```
··1.9000e-02 » ···5.9500e-01»¶
··2.0000e-02 » ···1.0940e+00»¶
··2.1000e-02 » ···1.5130e+00»¶
··2.2000e-02 » ···1.9330e+00»¶
··2.3000e-02 » ···2.2990e+00»¶
··2.4000e-02 » ···2.5580e+00»¶
··2.5000e-02 » ···2.6560e+00»¶
··2.6000e-02 » ···2.6150e+00»¶
··2.7000e-02 » ···2.5010e+00»¶
··2.8000e-02 » ···2.3300e+00»¶
··2.9000e-02 » ···2.1160e+00»¶
··3.0000e-02 » ···1.8750e+00»¶
```

*fig. 2.11: esempio di documento di testo salvato*

### 2.3.6.5 - push_cancella_bufferN

Lo scopo della funzione è liberare uno spazio in memoria da destinare al salvataggio di altre forme d'onda o dei dati acquisiti dalla misura.

Tramite la porta seriale emulata viene comunicato al sistema di inizializzare un determinato buffer. La comunicazione prevede l'invio del codice identificativo e il numero del buffer (fig. 2.12).

Essendo il processo irreversibile, è opportuno verificare che il buffer selezionato non contenga dati essenziali.

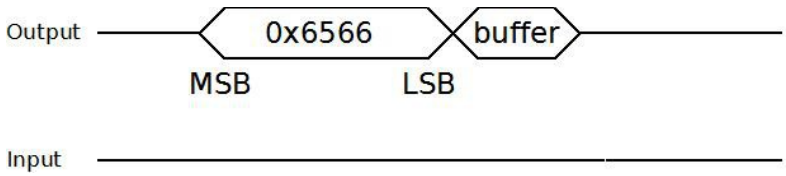


*fig. 2.12: frame del comando push_cancella_bufferN*

### 2.3.6.6 - push_Index

Selezionando il pulsante “Indice” si ha la possibilità di visualizzare lo stato di ogni buffer presente in memoria.

Il PC comunica al microcontrollore il codice della funzione ricevendo lo stato di tutti i buffer presenti in memoria. Il risultato della comunicazione viene scritto all'interno della finestra di dialogo (fig. 2.3) e riporta il nome e la lunghezza del buffer se questo contiene dei dati, altrimenti riporta la dicitura “buffer vuoto”.

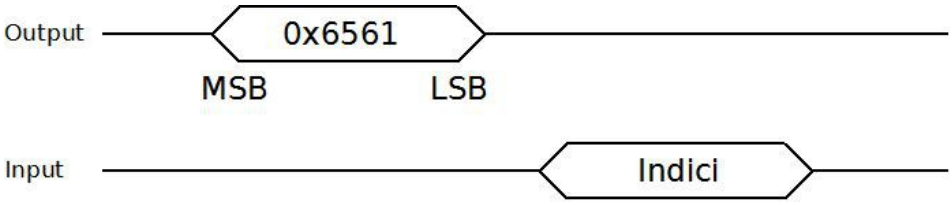


*fig. 2.13: Frame relativo al comando push_Index.*

### 2.3.6.7 - push_clear

La funzione è simile a push_cancella_bufferN con l'unica differenza che al posto di inizializzare il singolo buffer inizializza tutta la memoria. Anche questo procedimenti è irreversibile, è opportuno verificare di non aver in memoria informazioni essenziali.

La trama di questa funzione consiste nella sola trasmissione del codice identificativo (fig. 2.14).

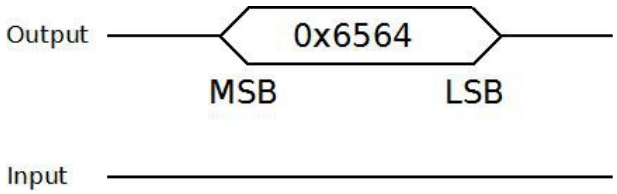

*fig. 2.14: Frame relativo alla funzione push_Clear*

## 2.4 - Codice microcontrollore

Per la gestione delle varie funzionalità che deve svolgere il microcontrollore sono state create tre librerie in grado di gestire le periferiche essenziali per questo progetto.

Lo scopo di quanto descritto è la creazione di un sistema in grado di trasferire informazioni da un PC al microcontrollore e viceversa, memorizzandole in una memoria flash per permettere l'utilizzo dei dati caricati anche dopo lo spegnimento.

### 2.4.1 - Funzionamento

All'accensione e tutte le volte che il sistema viene riavviato il PIC procede con l'eseguire le istruzioni presenti nella funzione *main* (fig. 2.15) che solitamente si distinguono in inizializzazione ed esecuzione del programma.

L'inizializzazione consiste nella configurazione dei registri di sistema e delle periferiche. Al suo interno viene configurata la porta SPI1 [9], l'USB, i pin di comunicazione col display e il chip select della memoria flash.

La funzione *ProcessIO()* contiene il corpo del programma. Non comprendendo un ritorno da questa funzione il *main* non verrà mai completato.

In *ProcessIO()* è presente il codice che permette all'utente di scegliere tra due modalità, ***run*** o ***usb,*** (fig. 2.16) sfruttando la funzione *init_menu()*. La prima modalità è destinata alle procedure di misura e non verrà trattata in questo testo, la seconda è destinata a gestire la comunicazione col computer.

Come si vede dallo schema la funzione *USB_comunication()* è un ciclo che non prevede uscita. Infatti, una volta eseguita, il sistema resterà in attesa di istruzioni ed eseguirà i comandi ricevuti fino alla terminazione della comunicazione seriale. Il diagramma in fig. 2.17 rappresenta le operazioni svolte dalla funzione.

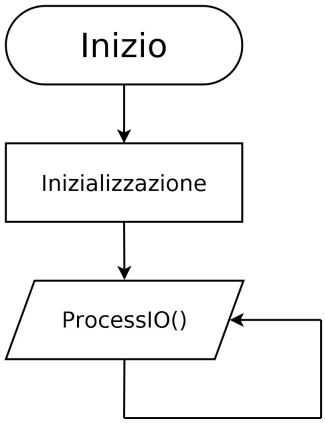


*fig. 2.15: Schema della funzione main()*

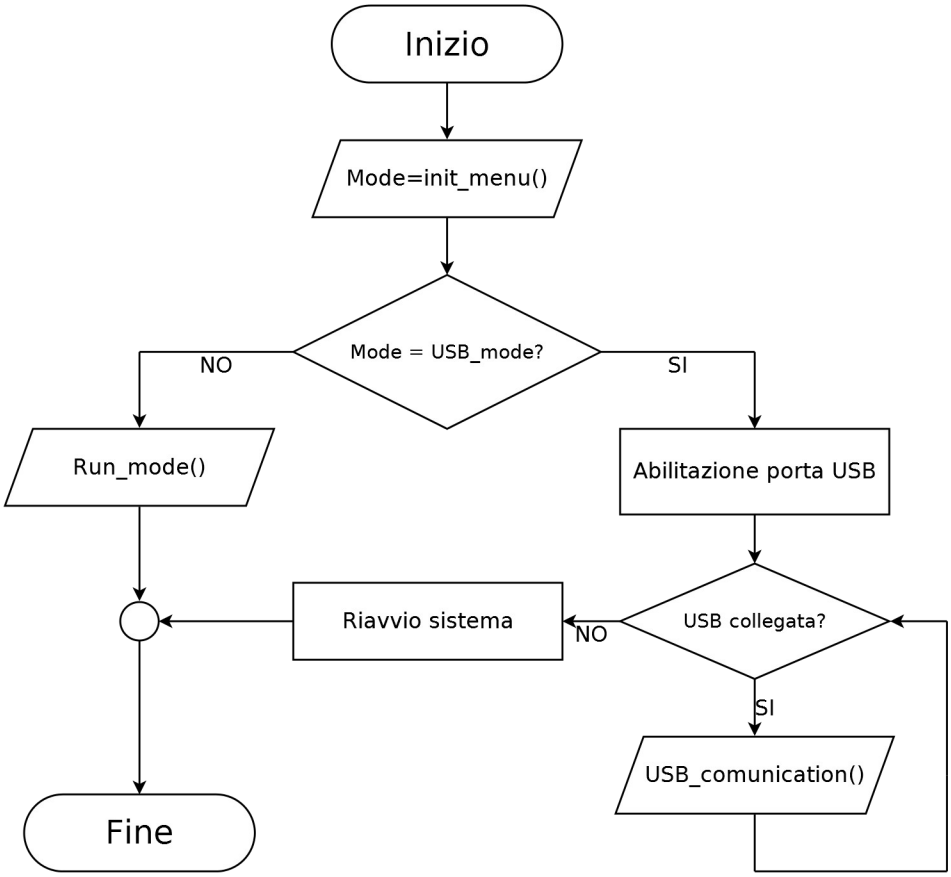


*fig. 2.16: schema a blocchi della funzione ProcessIO*

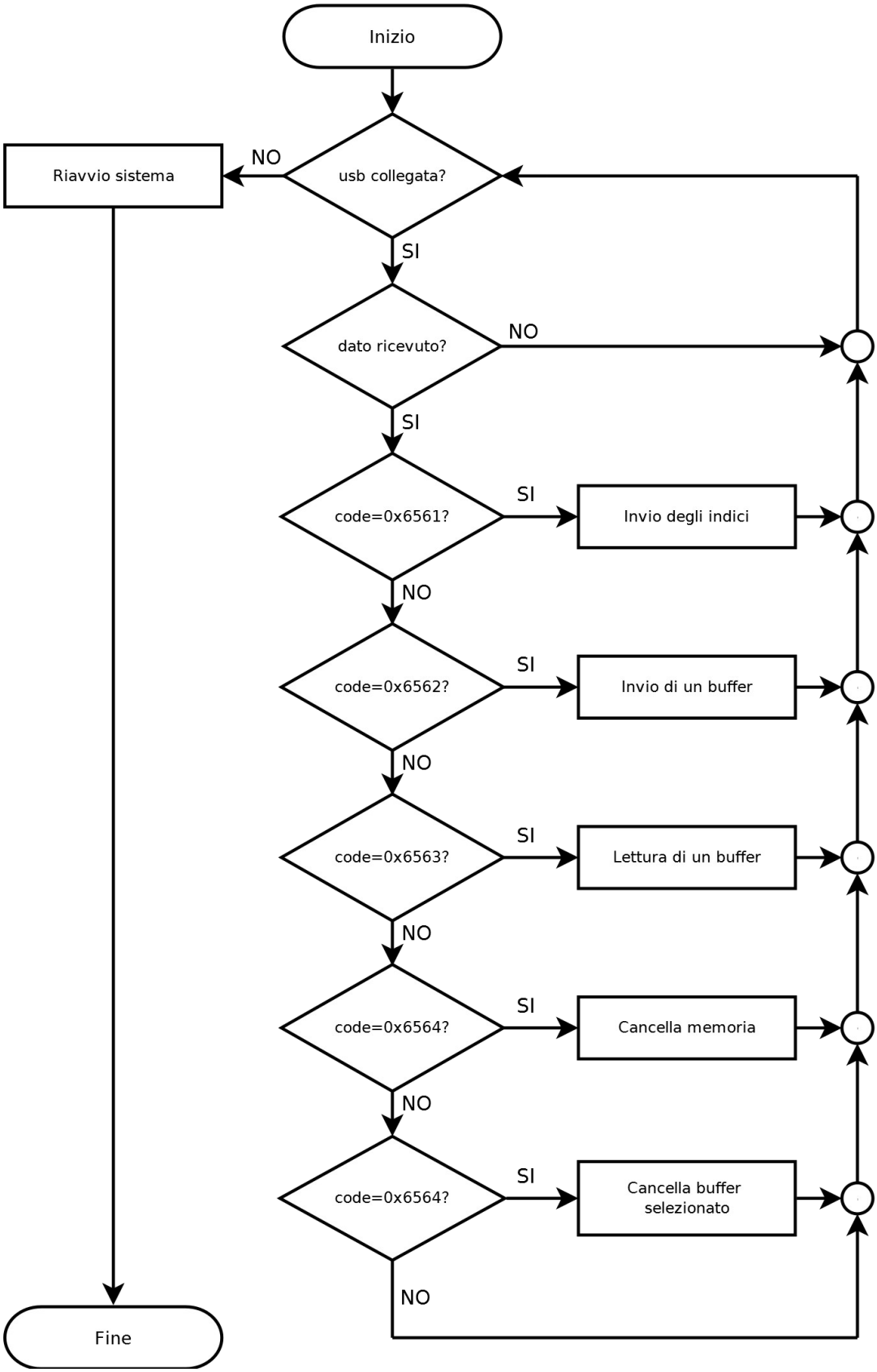


*fig. 2.17: Shema a blocchi della funzione USB_comunication()*

Il primo passo è verificare che sia attiva la comunicazione, essendo inutile l'utilizzo

della funzione senza avere una connessione fisica tra PC e scheda. Verificata la connessione si attende che il sistema comunichi l'istruzione da eseguire sempre monitorando l'eventuale terminazione del collegamento.

```
while ( numBytesRead == 0){
        if(!USB_BUS_SENSE)
        {
       USBDeviceDetach();
       Reset();
        }
        if(USBUSARTIsTxTrfReady())
        {
                numBytesRead = getsUSBUSART(USB_Out_Buffer,255);
        }
        CDCTxService();
}
```

L'esempio mostra il ciclo col quale il sistema attende la trasmissione di un dato. Prima verifica l'effettivo collegamento fisico del cavo, andando a controllare il valore logico sul pin collegato al VUSB (alimentazione trasportata dall'usb), per poi verificare l'eventuale ricezione di un dato. Il programma resterà in attesa finché almeno un byte non viene trasmesso al microcontrollore.

Alla ricezione del codice il sistema verifica a quale funzione corrisponde e ne esegue il codice corrispondete. Per esempio se riceve il codice 0x6561 eseguirà la funzione relativa all'invio degli indici relativi ai buffer in memoria.

Terminata l'esecuzione della funzione richiesta il microcontrollore ritorna in attesa di ulteriori istruzioni.

### 2.4.2 - Libreria per la gestione della comunicazione seriale – USB_COMUNICATION.h

La comunicazione tra PC e microcontrollore avviene tramite la porta USB che emula una seriale. La Microchip fornisce già una libreria[3] contenente le funzioni e le configurazioni necessarie per l'emulazione.

Il trasferimento dati è invece gestito dalla libreria *USB_COMUNICATION.h* sfruttando le funzioni fornite dalla casa produttrice. La tabella seguente mostra le funzioni presenti

3 E' inclusa nel pacchetto Microchip Solutions v2012-04-03, reperibile dal sito della Microchip. La libreria utilizzata è Device - CDC - Basic Demo.

in questa libreria:

| Nome | Descrizione |
|---|---|
| get_index | Viene invocata quando il PIC riceve l'istruzione 0x6561. Permette l'invio degli indici di tutti i buffer. |
| write_buffer | Viene invocata quando il PIC riceve l'istruzione 0x6562. Permette la ricezione di un intero buffer e il successivo salvataggio in memoria. |
| read_buffer | Viene invocata quando il PIC riceve l'istruzione 0x6563. Leggerà in memoria un intero buffer e lo invierà tramite USB al PC. |
| erase_buffer_usb | Viene invocata quando il PIC riceve l'istruzione 0x6566. Effettuerà l'inizializzazione del buffer selezionato. |

*Tabella 2: Funzioni contenute nella libreria USB_COMUNICATION.h*

#### 2.4.2.1 - get_index

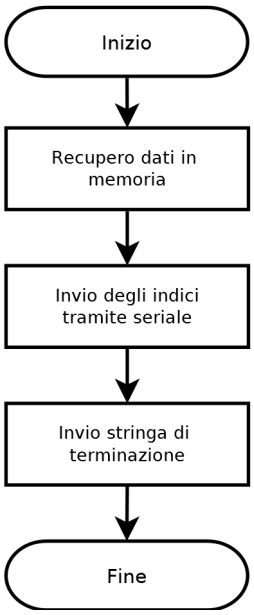


*fig. 2.18: Diagramma della funzione get_index*

La funzione get_index richiede in ingresso un array di tipo *char* nel quale verranno salvati tutti i dati da inviare. Tutte le volte che verrà chiamata questa funzione l'array utilizzato come ingresso viene modificato, pertanto è opportuno crearne uno utilizzabile solo per queste funzioni senza che contenga dati sensibili o necessari per altre operazioni.

Il suo scopo è recuperare lo stato dei buffer presenti in memoria e comunicarlo al PC,

pertanto prima di tutto legge le lunghezze di ogni banco e se diverse da zero ne recupera il nome.

Le trasmissioni tramite seriale richiedono dati in formato char, occorre quindi convertire eventuali numeri in stringhe di caratteri:

*itoa(dim, a, 10);*
*itoa(buf, b + 1, 10);*
dove *dim* e *buf* sono gli array che conterranno il valore convertito, *a* e *b* i valori interi e *10* è la base con cui viene convertita.

Recuperate le informazioni richiese, il sistema trasmette tramite seriale le informazioni da riportare a video e il risultato è quello riportato in fig. 2.3.

Non conoscendo numero esatto di caratteri trasmessi la comunicazione viene conclusa con l'invio della stringa di terminazione visibile in fig. 2.8.

#### 2.4.2.2 - write_buffer

Lo scopo della funzione è ricevere i dati relativi ad un buffer (lunghezza, nome e valore dei campioni) e salvarli in memoria.

L'esecuzione richiede l'utilizzo di due array, da immettere all'ingresso nel momento della chiamata:

*void write_buffer (char * USB_Out_Buffer, char * Out_Buffer)*
Il primo è necessario per la comunicazione seriale, perché vi vengono salvati di dati ricevuti. La sua dimensione deve essere di almeno 255 elementi, numero massimo consentito dalla libreria fornita da Microchip. Il secondo array viene usato per il salvataggio dei campioni, pertanto deve avere dimensione non inferiore a 25000 elementi. Il loro contenuto verrà modificato all'interno di questa funzione perciò è opportuno che non contenga dati necessari per altre operazioni.

In fig. 2.19 è mostrato il funzionamento del processo. Dovendo ricevere i dati del buffer, il sistema resta in attesa che venga effettuata una trasmissione.

Quando un elemento viene ricevuto si procede con l'identificazione, cioè il microcontrollore verifica di quale elemento del buffer si tratta, salvandolo nella rispettiva variabile.

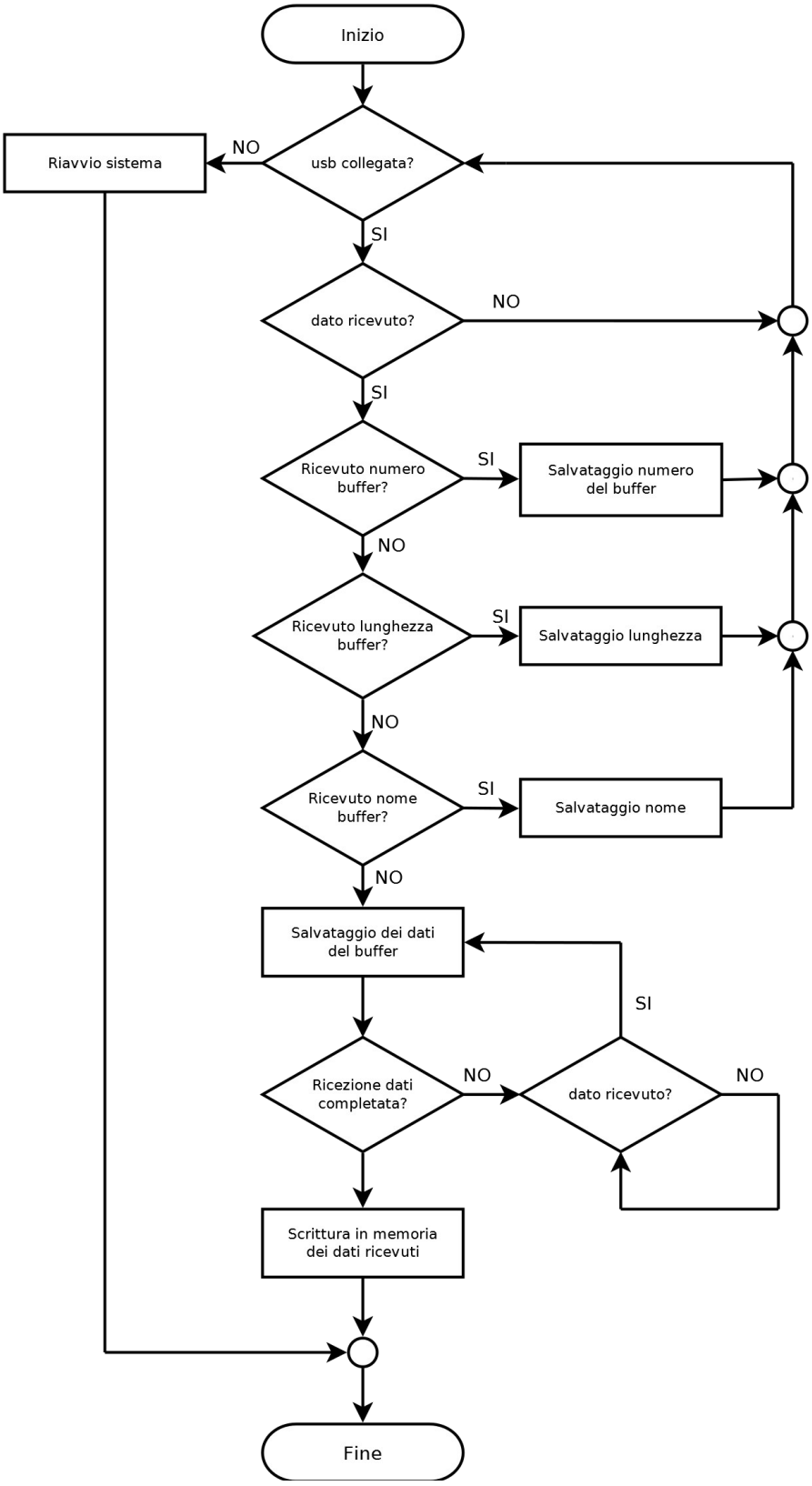


*fig. 2.19: Diagramma a blocchi della funzione write_buffer*

Completata la ricezione del numero del buffer selezionato, il numero dei campioni da salvare e il nome da attribuirgli si procede con la ricezione dei campioni. Conoscendone

il numero il sistema effettuerà un controllo per verificare il corretto trasferimento dei dati, provocando il riavvio del sistema nel caso si verifichino errori.

La funzione termina col salvataggio dei dati in memoria, eseguito solo nel caso in cui il trasferimento sia avvenuto correttamente.

#### 2.4.2.3 - read_buffer

Lo scopo della funzione è trasmettere tramite porta seriale il contenuto di un buffer precedentemente scelto.

Al momento della chiamata è necessario fornire due array:

*void read_buffer (char * USB_Out_Buffer, char * Out_Buffer)*

Il primo è necessario per la comunicazione seriale, perché vi vengono salvati di dati ricevuti o inserite le informazioni da trasmettere. La sua dimensione deve essere di almeno 255 elementi, numero massimo consentito dalla libreria fornita da Microchip. Il secondo array viene usato per il salvataggio dei campioni, pertanto deve avere dimensione non inferiore a 25000 elementi. Il loro contenuto verrà modificato all'interno di questa funzione pertanto è opportuno che non contenga dati necessari per altre operazioni.

Il suo funzionamento, descritto in fig. 2.20, mostra la sequenza delle operazione svolte. Dovendo ricevere dalla porta seriale il numero del buffer da caricare, il sistema attende che la comunicazione venga portata a termine. Un eventuale errore nella trasmissione provocherà il riavvio del sistema.

Completato questo passaggio viene recuperato il contenuto del buffer e inviato al PC tramite la comunicazione seriale. Non prevedendo l'invio del numero dei campioni da trasmettere al termine della comunicazione viene inviata la stringa di terminazione (fig. 2.8).

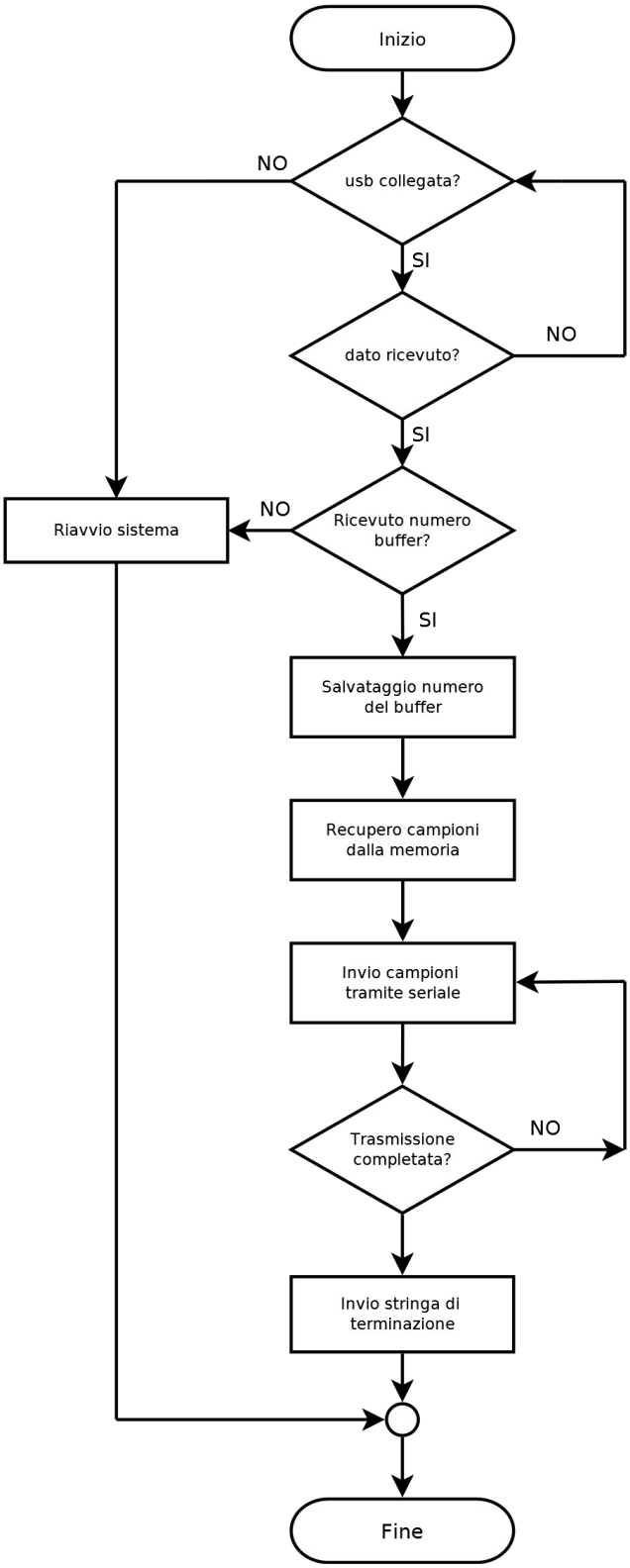


*fig. 2.20: Schema a blocchi della funzione read_buffer*

### 2.4.2.4 - erase_buffer_usb

Questa funzione ha come finalità l'inizializzazione di un buffer presente in memoria. Il PC comunica il numero del banco da eliminare ed il microcontrollore provvede a porre a zero la sua lunghezza (fig. 2.21).

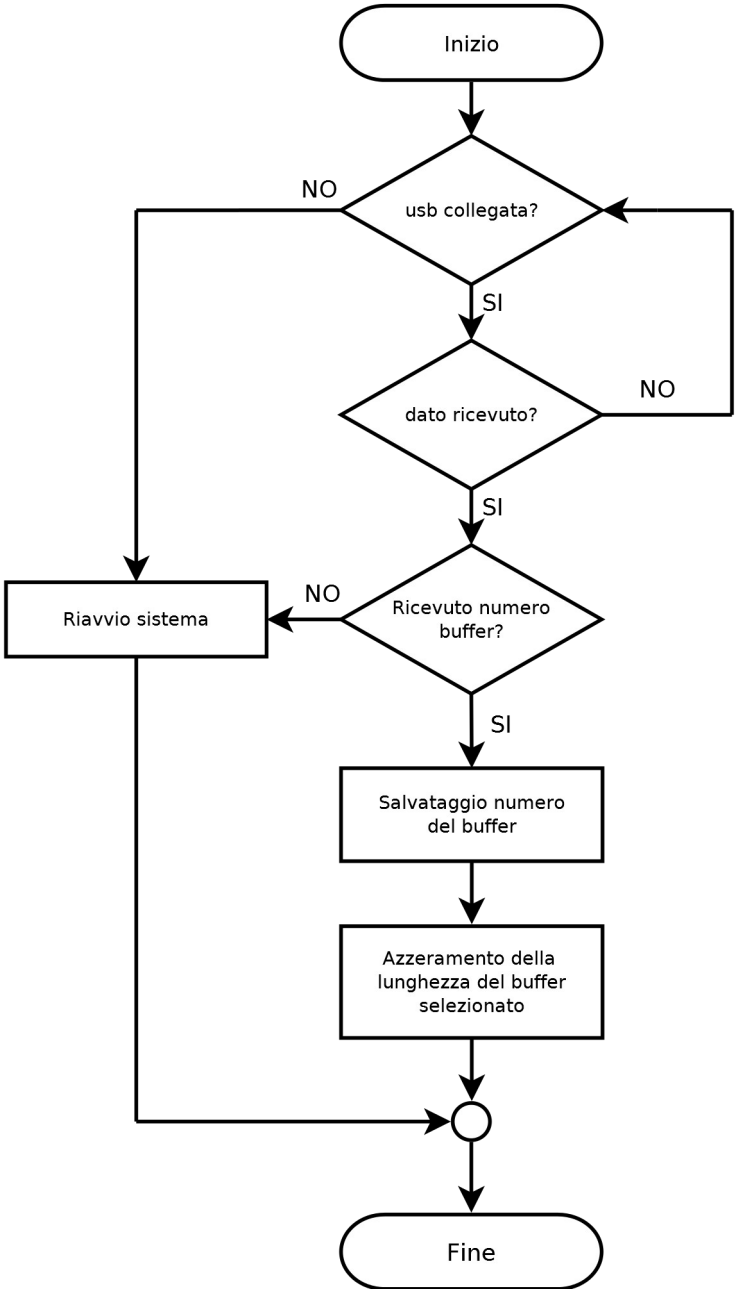


*fig. 2.21: Schema a blocchi della funzione erase_buffer_usb*

### 2.4.3 - Gestione della memoria flash – comunicazione_spi.h

L'interazione tra microcontrollore e memoria flash è indispensabile per le operazioni di salvataggio e lettura dei dati. Se la memoria non venisse utilizzata non sarebbe possibile conservare nel tempo i dati ed un eventuale riavvio del sistema o la perdita dell'alimentazione comporterebbe la perdita di tutti i dati presenti nel sistema.

Le funzioni presenti nella libreria sono le seguenti:

| Nome | Descrizione |
|---|---|
| abilita_porta | Attiva la porta SPI1, all'interno del file .h è possibile modificare i diversi valori di configurazione. |
| erase_eeprom | Cancella tutti i contenuti del buffer e pone tutte le lunghezze (fig. 6) a 0. |
| initialize_eeprom | Pone a 0 le lunghezze di tutti i buffer senza cancellare il contenuto degli stessi. Questo permetterà una minore usura della memoria. |
| erase_buffer | Cancella il contenuto del buffer selezionato e pone a 0 la lunghezza. |
| initialize_buffer | Pone a 0 la lunghezza del buffer selezionato senza cancellare il contenuto del buffer stesso. Questo permetterà una minore usura della memoria. |
| write_enable | Configura i registri interni alla memoria in modo da permettere la scrittura di dati al suo interno. La mancata invocazione di tale funzione impedirà la scrittura. |
| write_disable | Configura i registri interni alla memoria in modo da non permettere la scrittura di dati al suo interno. |
| write_byte | Consente la scrittura in memoria di un byte. |
| read_byte | Legge in memoria un byte e lo salva su una variabile. |
| read_word | Legge in memoria due byte consecutivi e ne ritorna il valore a 16 bit salvandolo su una variabile. |
| is_free | Verifica se il buffer selezionato è pieno o vuoto. |
| first_free | Ci ritorna il primo buffer libero, qualora non ce ne siano ritorna 0xFF. |
| get_size | Ritorna la lunghezza del buffer selezionato |
| get_name | Ritorna il nome del buffer selezionato |
| get_buffer | Ritorna il contenuto del buffer selezionato |
| write_buffer_ee | Permette la scrittura del buffer selezionato. |

*Tabella 3: Funzioni contenute nella libreria comunicazione_spi.h*

#### 2.4.3.1 - abilita_porta

Per poter comunicare con la flash è necessario abilitare la comunicazione seriale con protocollo SPI. Per far ciò questa funzione imposta i vari registri tramite la seguente funzione:

*OpenSPI1 (conf1, conf2, conf3);*
I valori posti in ingresso alla funzione hanno lo scopo di configurare i tre registri utilizzati per la porta SPI, in particolare:

- *conf1* configura il registro **SPI1CON1**.
- *conf2* configura il registro **SPI1CON2**.
- *conf3* configura il registro **SPI1STAT**.

#### 2.4.3.2 - erase_eeprom

Ha come scopo la cancellazione di tutti i contenuti presenti nella memoria e di inizializzare i buffer in essa presenti.

Il processo è irreversibile perché pone al valore di 0xFF tutte le celle, tranne le prime due di ogni buffer (rappresentano la lunghezza) che vengono poste a 0x00.

Questa operazione oltre che a usurare la memoria a causa del numero limitato di cicli di scrittura raccomandato dal costruttore, richiede del tempo per essere eseguita ( 60 ms ). E' sconsigliato utilizzarla durante tutti quei processi che richiedono un monitoraggio.

#### 2.4.3.3 - initialize_eeprom

La funzione effettua l'azzeramento delle lunghezze contenute in tutti i buffer presenti in memoria. Le specifiche richiedono di utilizzare la lunghezza come indice per il controllo della presenza di dati nel buffer, ponendola a zero il sistema capisce che all'interno di quel banco non è presente alcun dato. Questo ci permette di abbassare notevolmente il tempo di cancellazione, che si riduce a qualche μs. Inoltre l'utilizzo di questa funzione al posto di *erase_eeprom* permette una minor usura della memoria.

#### 2.4.3.4 - erase_buffer

L'architettura della memoria impedisce scritture consecutive sulla stessa cella perché la logica interna non è in grado di effettuare la transizione da 0 a 1 logico. Per questo, prima di ogni processo di scrittura, è indispensabile resettare il banco di memoria interessato.

La funzione *erase_buffer(n° buffer)* esegue la cancellazione dello spazio di memoria associato al buffer (compresa la lunghezza) ponendo tutte le celle a 0xFF permettendo così la scrittura.

#### 2.4.3.5 - initialize_buffer

Funzione simile ad *initialize_eeprom*, con la differenza di porre a zero la lunghezza del solo buffer selezionato invece della lunghezza di tutti i buffer:

*unsigned int erase_buffer (BYTE buffer)*
Da non confondere con *erase_buffer* perché questa non cancella tutti gli elementi contenuti nello spazio di memoria scelto ma pone a zero solo la lunghezza.

#### 2.4.3.6 - write_enable

La memoria dispone di un registro di stato che protegge i dati in essa salvati. Di default il registro è configurato in modo da impedire la scrittura o la cancellazione del contenuto della memoria.

La funzione *write_enable* imposta i bit del registro in modo da permettere la scrittura o la cancellazione della memoria.

#### 2.4.3.7 - write_disable

La funzione svolge l'operazione opposta a quella precedente, imposta i bit del registro di stato in modo da impedire la scrittura o la cancellazione dei dati nella memoria.

Una volta completato un ciclo di scrittura è sempre consigliato chiamare questa funzione per evitare che operazioni svolte tramite la porta SPI possano modificare il contenuto della memoria stessa.

#### 2.4.3.8 - write_byte

La funzione permette la scrittura di un byte in memoria. In ingresso richiede tre parametri:

*unsigned int write_byte (BYTE data, unsigned int add, BYTE upper_add)*
Il primo, *data*, è il valore da salvare e deve essere di dimensione pari ad un byte. La variabile *add* permette l'indirizzamento all'interno del buffer e può andare da 0 a 65535 mentre *upper_add* indica il numero del buffer scelto e deve essere compreso tra 0 e 9.

Prima di chiamare questa funzione è necessario eseguire *write_enable,* altrimenti la scrittura non andrà a buon fine.

### 2.4.3.9 - read_byte

La funzione effettua la lettura di un byte presente in memoria e ne riporta il contenuto. In ingresso richiede l'indirizzo su cui effettuare la lettura:

*unsigned int read_byte (unsigned int add, BYTE upper_add)*
La variabile *add* permette l'indirizzamento all'interno del buffer e può andare da 0 a 65535 mentre *upper_add* indica il numero del buffer scelto e deve essere compreso tra 0 e 9.

La lettura non necessita la chiamata di nessun'altra funzione perché non effettua alcuna modifica in memoria.

### 2.4.3.10 - read_word

La funzione effettua l'interpolazione tra due byte presenti in memoria, generando così un valore intero a 16 bit. Come per read_byte in ingresso occorre fornire l'indirizzo e in numero del buffer.

La lettura e l'interpolazione avviene in questo modo:

```
up_byte = read_byte (add, upper_add);
lo_byte = read_byte (add + 1 , upper_add);
data_out = ((up_byte << 8) & 0xff00) | (lo_byte & 0x00ff);
```

Il byte presente nella prima cella è quello più significativo mentre il successivo rappresenta quello meno significativo.

### 2.4.3.11 - is_free

Semplice funzione che, andando a leggere la lunghezza del buffer selezionato, ritorna “TRUE” se risulta vuoto oppure “FALSE” nel caso contrario.

### 2.4.3.12 - first_free

Funzione che  permette di ritornare il numero del primo buffer vuoto:

```
BYTE i;
for (i = 0; i < N_BUFFER; i++){
        if ( is_free (i) ){
                return i;
        }
}
return 0xFF;
```

Si nota come la funzione finisca non appena viene trovato un buffer vuoto grazie a

is_free riportandone il numero. Se la memoria risulta piena il valore ritornato è 0xFF.

#### 2.4.3.13 - get_size

La funzione effettua la lettura in memoria della lunghezza del buffer selezionato e la ritorna. L'operazione viene svolta con l'ausilio della funzione read_word.

#### 2.4.3.14 - get_name

La funzione svolge la lettura delle celle contenenti i caratteri del nome e li ricompone all'interno di un array. Per essere chiamata occorre che vengano usati in ingresso i seguenti dati:

*void get_name (BYTE * name, unsigned int size, BYTE buffer)*
Il primo elemento corrisponde al vettore nel quale viene salvato il nome mentre *size* corrisponde alla sua lunghezza. Per quanto riguarda il progetto, *size* deve corrispondere ad 8 a causa della lunghezza fissa del nome. La variabile *buffer* è il numero del buffer da cui si desidera recuperare il nome.

#### 2.4.3.15 - get_buffer

La funzione *get_buffer* effettua la lettura dei campioni presenti in memoria e li salva all'interno di un array. Per essere chiamata occorre utilizzare in ingresso i seguenti dati:

*void get_buffer (BYTE * out_buffer, BYTE buffer)*
Il vettore *out_buffer* è usato per il salvataggio dei campioni e deve essere costituito da almeno 25000 elementi, mentre *buffer* indica il numero del buffer da leggere.

#### 2.4.3.16 - write_buffer_ee

La funzione in questione effettua la scrittura di un intero buffer in memoria. Occorre porre in ingresso tutti i dati relativi al buffer:

*void write_buffer_ee (char * in_buffer, unsigned int size, char *nome, BYTE buffer)*
L'elemento *in_buffer* corrisponde all'array contenente tutti i campioni da salvare, *size* indica il numero di elementi di cui è composto *in_buffer* e *nome* contiene i caratteri che compongono il nome del buffer. La variabile *buffer* indica il numero del banco su cui scrivere.

Prima di chiamare questa funzione è necessario utilizzare write_enable così da abilitare la scrittura all'interno della memoria.

### 2.4.4 - Gestione dei menù di interazione con l'utente – menu.h

Per permettere all'utente di interagire col sistema questa libreria mette a disposizione tre menù. Tramite il display LCD il sistema interroga l'utente, il quale tramite i pulsanti opera le sue scelte.

| Nome | Descrizione |
|---|---|
| end_menu | Permette al PIC di chiedere se l'utente è intenzionato a riavviare il sistema o se preferisce continuare |
| init_menu | Utilizzato all'inizio del programma per chiedere all'utente se vuole entrare in modalità USB o in modalità RUN |
| select_buffer | Interroga l'utente su quale buffer è intenzionato a selezionare per eseguire una determinata operazione. |

*Tabella 4: Funzioni contenute nella libreria menu.h*

#### 2.4.4.1 - end_menu

L'esecuzione di questa funzione permette all'utente di scegliere se riavviare il sistema o se proseguire con l'esecuzione del programma.

Sul display viene visualizzata la richiesta e tramite i pulsati si scorrono le due opzioni.

Riavviando il sistema tutti i dati presenti in RAM vengono cancellati. E' opportuno salvare tutti i dati utili prima di chiamare questa funzione.

#### 2.4.4.2 - init_menu

All'avvio il sistema interroga l'utente per decidere in quale modalità entrare. Questa funzione permette di visualizzare sul display LCD la richiesta di scelta della modalità con cui iniziare ad operare

L'utente, con l'ausilio dei pulsati, sceglie con quale modalità operare (**run** o **usb**).

#### 2.4.4.3 - select_buffer

Il menù creato con questa funzione permette all'utente di selezionare un buffer presente in memoria. Mentre scorre il menù, l'utente può vedere se lo stato del buffer visualizzato è vuoto (“buffer vuoto”) oppure contiene dei dati (“nome_buffer – lunghezza”).

Premuto il tasto di conferma la funzione ritorna il numero del buffer scelto.

# Capitolo 3 - Conclusioni

Realizzando il circuito presente in fig. 1.7, il programma di controllo del microcontrollore descritto nel paragrafo 2.4 e l'interfaccia grafica vista nel paragrafo 2.3 si è riusciti creare un sistema in grado di scambiare informazioni tra un PC e un microcontrollore. Questi dati vengono salvati all'interno di una memoria flash con interfaccia SPI in modo da permetterne l'utilizzo nel tempo.

La comunicazione svolta tramite l'emulazione di una porta seriale (RS-232) consente una comunicazione bidirezionale consentendo all'utente di immettere e recuperare dati dal sistema in maniera comprensibile. In particolare è possibile ricreare un file di testo con un impaginazione che permette a chi interagisce col sistema di avere a portata di mano tutti i dati in un formato leggibile e comprensibile.

L'utilizzo di dieci buffer è essenziale per la realizzazione di più misure, consentendo il salvataggio di forme d'onda diverse e allo stesso tempo dare spazio al salvataggio dei dati acquisiti dalla misura.

La realizzazione di librerie specifiche per ogni periferica permette, a chiunque decida di portare modifiche al sistema, di operare in maniera chiara e veloce. Basta infatti la chiamata ad una funzione per recuperare il contenuto di un buffer, leggerne il nome o semplicemente cancellarne il contenuto.

In conclusione il progetto è riuscito a rispettare tutte le specifiche senza dover scendere a compromessi.

# Riferimenti bibliografici

[1] Tratto da wikipedia: http://it.wikipedia.org/wiki/Energy_harvesting

[2] M. Belleville et al. *Energy Autonomous Systems: Future Trends in Devices, Technology, and System,* CATRENE, 2009, ISBN 978-88-904-399-0-2.

[3] Immagine tratta dal sito http://it.emcelettronica.com/energy-harvesting-progetto-di-riferimento-da-silicon-labs

[4] A. Romani, R. P. Paganelli, E. Sangiorgi, M. Tartagni, *A Rapid Modeling and Prototyping Technique for Piezoelectric Energy Harvesting Systems*, in: SENSORDEVICES 2011 : The Second International Conference on Sensor Device Technologies and Applications, s.l, IARIA, 2011, pp. 86 – 90.

[5] Lucio Di Jasio, *Programming 16-Bit PIC Microcontrollers in C, Learning to Fly the PIC 24*, Elsevier Inc, 2007, ISBN: 978-0-7506-8292-3.

[6] Application Note Microchip AN1247: *Communication Device Class (CDC) Host*.

[7] Larry Wall, Tom Christiansen, Jon Orwant, *Programming Perl*, O'Reilly Media, 2000, ISBN: 978-0-596-00027-1.

[8] Jon Orwant, *Web, Graphics & Perl/Tk*, O'Reilly & Associates, Inc., 2003, ISBN: 0-596-00311-0.

[9] Reference Manual Microchip, *PIC24F Family Reference Manual, Sect. 23 Serial Peripheral Interface (SPI).*

# Indice delle tabelle

# Indice delle immagini