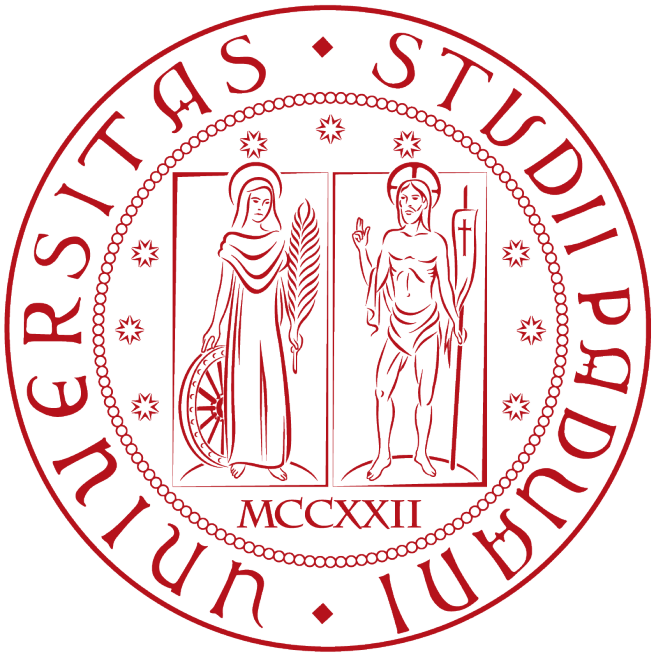
UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'Informazione



SCUOLA DI INGEGNERIA

Corso di laurea in Ingegneria dell'Informazione

TESI DI LAUREA

Datalogger realtime basato su Pinguino32 con supporto USB, comunicazione ZigBee e memoria MMC

Relatore: *Prof. Paolo Tenti*

Correlatore: *Dott.Ing. Marco Stellini, PhD*

Laureando: *Paride Miola*

ANNO ACCADEMICO 2012-2013

Sommario

Nel contesto dell'acquisizione dati per applicazioni di energy meter uno dei requisiti è la capacità di memorizzare dati con un elevato bit rate e di trasmetterli ad un sistema di controllo principale.

Oggetto di questo lavoro di tesi è proprio la realizzazione di un datalogger basato su hardware compatibile con l'ambiente di sviluppo Pinguino32.

Nel dettaglio vengono sviluppate le librerie e le funzioni necessarie per poter accedere, sia in lettura sia in scrittura, ad una memory card tramite microcontrollore. Viene affrontata anche la parte di gestione del timestamp e di comunicazione, wired e wireless, dei dati registrati. Sarà infatti possibile utilizzare il supporto USB del microcontrollore, per stabilire una comunicazione bidirezionale con il computer tramite un semplice terminale. Inoltre verrà implementata la comunicazione wireless tramite moduli radio basati sul protocollo ZigBee. In questo caso l' esempio applicativo è basato su un device master che trasmette i dati ed uno slave che si occupa della ricezione e dell'inoltro dello stream al terminale del computer.

Per completezza viene presentato anche un esempio di comunicazione wired verso dispositivi mobili dotati di sistema operativo Android.

Infine, vista la scarsa documentazione relativa all'ambiente di sviluppo Pinguino32, viene presentata una breve guida all'installazione e configurazione dell'IDE, in modo da poter facilmente utilizzare il codice e l'hardware presentato in questo documento.

Indice

1. Introduzione

Per la realizzazione del datalogger si è scelto di utilizzare una scheda di sviluppo basata sul microcontrollore Microchip® PIC32MX440F256H, su cui è stato precaricato il bootloader Pinguino32.

Il lavoro è stato suddiviso in due parti, la prima relativa alla lettura e scrittura dei dati registrati su memory card e la seconda dedicata alla comunicazione wired e wireless verso altri dispositivi.

Per la gestione della memory card, nello specifico una MicroSD, si è deciso di utilizzare il File System FAT32, in modo da poter facilmente leggere la scheda di memoria anche con altri lettori e sistemi operativi.

Per rendere facile la manipolazione dei dati registrati, questi vengono salvati sulla scheda di memoria direttamente in formato CSV[1], in modo da poter essere facilmente letti tramite lo stesso datalogger o importati nei più diffusi fogli di calcolo o software di analisi dei dati.

La parte relativa alla comunicazione fa uso delle librerie USB CDC[2] per creare, tramite terminale, una semplice interfaccia utente e di debug. La comunicazione wireless invece è affidata a due moduli radio MRF24J40MB-I/RM, di cui saranno presentati brevemente peculiarità e stack del protocollo. Verrà inoltre presentato il codice sviluppato per creare un ricevitore, basato sul protocollo ZigBee, in modo da poterne mostrare un esempio applicativo.

Fig. 1 Scheda di sviluppo su cui e' stato prototipato il datalogger

[1] Comma Separated Values.
[2] USB Communications Device Class.

1.1 Progetto Arduino

La piattaforma di sviluppo Arduino rappresenta uno degli esempi di maggior successo di framework *Open Source* ed *Open Hardware,* nato con l'intento di creare una scheda di prototipazione che fosse flessibile e di facile utilizzo.

La presenza di una vasta documentazione, schemi elettrici e sorgenti dell'intero framework, nonchè una folta community di appassionati e sviluppatori, ha permesso al progetto di espandersi sia dal punto di vista software, con la creazione di librerie dedicate ai più disparati utilizzi, sia dal punto di vista hardware, con la nascita dei così detti *shields,* cioè delle schede elettroniche di espansione che si adattano al pinout delle schede Arduino aumentandone le funzionalità.

1.1.1 Architettura Arduino

Nonostante esistano numerosissime librerie software e shields di espansione che permettono di interfacciarsi ad una grande varietà di dispositivi esterni (LCD, porte ethernet, motor driver, ecc) quello che frena le schede Arduino è la limitata potenza di calcolo. Infatti la maggior parte delle schede di sviluppo monta dei microcontrollori della serie ATmega, cioè MCU ad 8 bit prodotte dalla Atmel.

E' evidente che con una frequenza di clock di soli 16Mhz, 1024byte di RAM e 16Kbyte[3] di memoria flash disponibile per l'utente, i processori alla base delle schede Arduino ne limitano notevolmente il campo di applicazioni realizzabili, soprattutto quando gli shields, utilizzati per farsi carico del grosso del lavoro, non riescono a sostituire la mancanza di un'architettura di base a 16 o 32 bit.

Per fare un esempio concreto, la porta USB di cui sono equipaggiate le schede Arduino può essere utilizzata solo come porta seriale tramite un chip FTDI che si occupa della conversione USB/Seriale. Infatti con soli 16Mhz di clock il processore non è in grado di mantenere una connessione USB, nemmeno in modalità *half speed*. Questo rappresenta un chiaro limite funzionale anche per l'applicazione oggetto di questa tesi.

[3] In realtà dei 16Kbyte disponibili per l'applicazione utente, 2Kbyte vengono riservati al bootloader Arduino.

1.2 Progetto Pinguino32

Visto il successo del progetto Arduino, sono nati numerosi progetti paralleli che imitano il form factor delle schede Arduino e la filosofia di creare un ambiente di sviluppo semplice ed user friendly. Tra le più famose vi sono le schede della serie ChipKit, prodotte dalla Microchip® utilizzando MCU a 32bit, che rendono possibile l'esecuzione di programmi ben più complessi, senza l'utilizzo di schede di espansione esterne. A differenza del progetto Arduino però, le schede ChipKit non hanno adottato la stessa ottica Open Source e Open Hardware, restando vincolate ad un IDE della Microchip® e sistemi di sviluppo e debug proprietari.

Da un gruppo di appassionati nasce così il progetto Pinguino32, ovvero un framework compatibile con gli shields Arduino, ma che utilizza l'architettura più performante offerta dai microcontrollori a 32 bit.

Le potenzialità e la versatilità offerta sono significativamente superiori rispetto al più noto Arduino sebbene il progetto si trovi ancora in uno stadio pre-alpha e molte librerie presentano bug dovuti ad un porting[4] in evoluzione.

1.2.1 Architettura Pinguino32

A differenza delle schede Arduino, il progetto Pinguino32 non ha deciso di vincolarsi ad un particolare tipo di hardware, ma ha scelto una famiglia più ampia di processori. In particolare sono nate due versioni, una basata sull'architettura ad 8bit dei processori Microchip®, pensata prevalentemente per applicazioni semplici e che non richiedono un'elevata potenza di calcolo, ed una seconda versione incentrata sull'architettura a 32bit offerta dai processori Microchip® della serie MIPS PIC32MX e dai processori della STMicroelettronics® della serie Cortex M3 STM32.

Essendo il progetto svincolato da un singolo produttore Hardware è possibile trovare sul mercato diverse soluzioni, in modo che si possa scegliere la scheda di sviluppo che meglio si adatta alle proprie esigenze.

In questo progetto è stato scelto di utilizzare una *proto-board* realizzata dalla Olimex ed equipaggiata con un processore a 32bit della Microchip®.

[4] Molte delle librerie utilizzate dal framework Pinguino32 sono basate su quelle presenti nel framework Arduino e riscritte o adattate per funzionare sulla nuova architettura a 32bit.

2. IDE Pinguino32 X.2

L'IDE del progetto Pinguino (uno screen-shot è riportato in figura 2) offre un ambiente estremamente user friendly e minimalistico, tanto da essere assenti anche funzioni di debug e terminale.

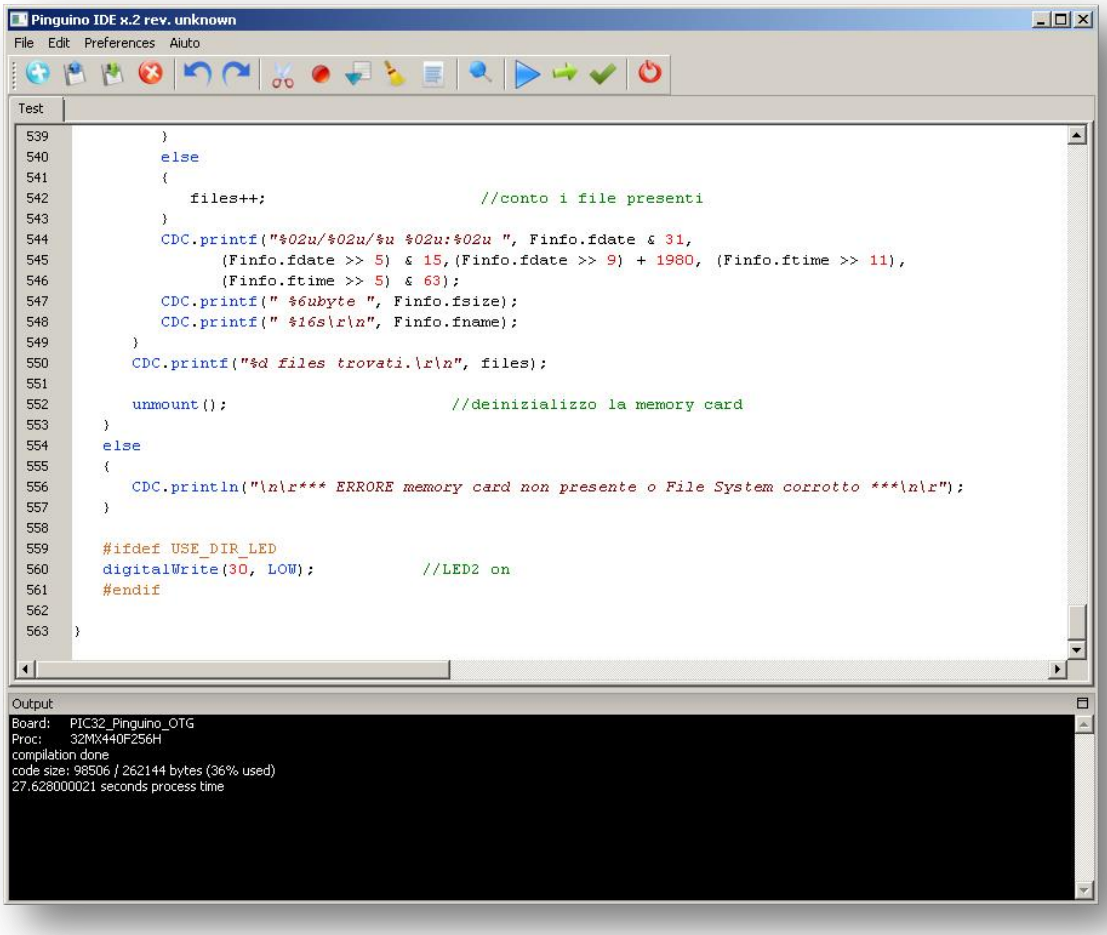


Fig. 2 Ambiente di sviluppo Pinguino32 X.2

Si trovano i comuni comandi di editing e ricerca nel testo[5], un menù da cui è possibile scegliere l'hardware da programmare, ed i comandi di compilazione ed invio del sorgente alla scheda di sviluppo.

Il compilatore utilizzato dall'IDE, varia in funzione dell'hardware di destinazione selezionato, in questo progetto è stato utilizzato il compilato gcc-MIPS.

[5] Attualmente nella versione X.2 dell'IDE la funzione presenta qualche bug e non è compatibile con l'ambiente windows.

2.1 Installazione dell'IDE

Al momento della stesura di questa relazione, l'ultima versione disponibile del framework Pinguino32 è la X.2 rev305 e non è disponibile una guida all'installazione, né tantomeno una documentazione relativa ai prerequisiti. Per semplificare l'utilizzo dei sorgenti qui descritti, si è deciso di presentare al lettore una breve guida per la corretta installazione e configurazione dell'IDE in ambiente Windows.

2.1.1 Prerequisiti

Prima di installare l'IDE è necessario verificare che sul proprio computer siano disponibili tutti i pacchetti necessari alla corretta esecuzione dell'ambiente di sviluppo, in particolare è necessario scaricare ed installare l'interprete Python 2.5.2[6], le librerie wxpython 2.8[7], quelle pyserial[8] ed in fine le librerie pyusb[9].

Per poter utilizzare correttamente la scheda di sviluppo è necessario installare il driver USB fornito dalla Microchip® e modificato dal gruppo HackingLab[10].

2.1.2 Setup dell'IDE

Per prima cosa è necessario scaricare l'installer dell'ambiente di sviluppo[11]. E' importante che l'IDE venga installato in un percorso senza caratteri di spazio nel path, infatti il compilatore è ancora in versione Alpha e va in crash nel caso in cui venga installato in una directory o in un percorso contenente spazi vuoti.

[6] http://www.python.org/ftp/python/2.5.2/python-2.5.2.msi
[7] http://downloads.sourceforge.net/wxpython/wxPython2.8-win32-unicode-2.8.9.1-py25.exe
[8] http://pypi.python.org/pypi/pyserial
[9] http://bleyer.org/pyusb/PyUSB-1.5.win32-py2.5.exe
[10] http://hackinglab.org/pinguino/download/driver%20pinguino%20windows/
[11] http://code.google.com/p/pinguino32/downloads/list

2.2 Aggiornamento parziale alla versione X.3

La versione X.3 è stata rilasciata da poco, tramite SVN, ai soli programmatori che si sono uniti al gruppo di sviluppo del progetto ed è ancora lontana dal poter essere rilasciata in forma ufficiale.

La versione X.2 è la più stabile attualmente disponibile, ed offre un buon numero di librerie e funzioni direttamente utilizzabili. Tuttavia per l'applicazione in esame, la versione X.2 non è in grado di sfruttare tutte le potenzialità offerte dal processore PIC32MX440F256H. Inoltre alcune funzioni risultano in conflitto con altre; in particolare viene a mancare tutta la parte relativa alla gestione del punto di mount delle memory card, la gestione dell'EOF[12] ed altre funzioni necessarie alla gestione del file system FAT32.

La versione X.3, d'altra parte, è altamente instabile e presenta numerosi bug: non funziona la comunicazione tramite porta USB e la manipolazione delle stringhe e la gestione dell'output tramite le funzioni printf() e sprintf(). Per ovviare a questi problemi e raggiungere lo scopo della tesi si è scelto di utilizzare come base la versione X.2 del framework, modificando le librerie esistenti e fondendo i file utili presenti della versione X.3, in modo da creare una nuova versione più completa ed adatta a compilare il sorgente qui presentato.

2.2.1 File X.3

Per la corretta compilazione del sorgente è necessario appoggiarsi ad alcune librerie non presenti nella versione X.2. Dopo aver installato la versione X.2 dell'IDE, è necessario collegarsi tramite SVN al gruppo di sviluppo del progetto[13], navigare fino alla posizione svn/branches/x.3/p32/include/pinguino/libraries/sd e scaricare i file diskio.c, diskio.h, ff.c, ff.h, fileio.c, fileio.h[14]. Questi file vanno poi sostituiti nella stessa directory della propria installazione dell'IDE.

[12] End Of File.

[13] http://code.google.com/p/pinguino32/source/browse/

[14] Rispettivamente revision r250, r246, r250, r246, r318, r318.

2.2.2 Modifiche alle librerie

Per poter utilizzare correttamente le nuove librerie prese dalla versione X.3 è necessario modificare anche il file sd.pdl32 presente nella directory /Pinguino32X.2/p32/lib/sd.pdl32, della propria installazione. Di seguito sono state messe in evidenza le modifiche apportate per rendere compatibile il porting dalla versione X.2 a quella X.3 dell'IDE.

Nella figura seguente è riportato nel lato sinistro il contenuto originale del file, mentre nel lato destro il contenuto del file modificato. In giallo sono quindi evidenziate le porzioni aggiornate, in arancione quelle aggiunte, in verde le parti non modificate ed in rosso le parti non piu' necessarie e quindi eliminate.

Fig. 3 Modifiche al contenuto del file sd.pdl32

2.3 Microchip® USB Driver

Dopo aver installato l'IDE, è necessario installare le librerie libusb presenti nella directory /Pinguino32X.2/extra, con il nome setup-libusb-win7-8bits. Nella stessa directory è presente l'archivio mchpcdc.zip contente i driver CDC rilasciati dalla Microchip® per poter utilizzare la comunicazione bidirezionale tramite porta USB.

Nel dettaglio questi driver si occupano di creare una porta COM virtuale, in modo che sia possibile dialogare con la scheda Pinguino come fosse un dispositivo seriale, utilizzando un semplice terminale[15] settato su velocità fino a 115Kbps, 8 bit per pacchetto, 1 bit di stop e nessun bit di parità. La porta COM è identificabile tramite il pannello di gestione dispositivi come riportato in figura 4.

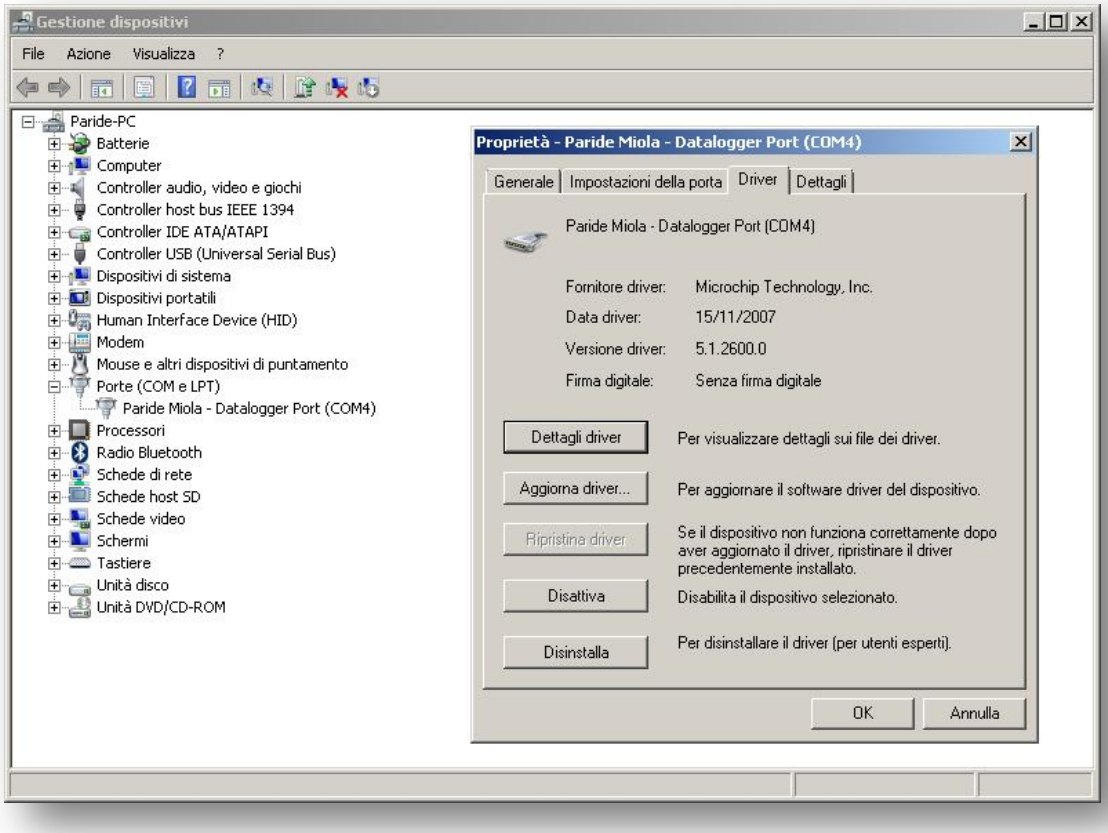


Fig. 4 Dettaglio riconoscimento dispositivo in ambiente Windows

[15] Nei test è stato utilizzato il terminale Microsoft Hyper Terminal presente nel CD allegato.

2.3.1 Modifiche ai driver USB

Nel corso dei primi test di comunicazione, i driver rilasciati dalla Microchip® hanno mostrato dei problemi di compatibilità con Windows 7. Questi sono stati risolti apportando una modifica ai driver in modo che il dispositivo fosse correttamente riconosciuto dal sistema operativo. Dopo aver estratto l'archivio è necessario sostituire il file mchpcdc.cat [16], successivamente si apra il file mchpcdc.inf con un semplice editor di testo e si modifichino le righe 89 e 92 come mostrato nella seguente figura:

Contenuto originale mchpcdc.inf

```
[DeviceList]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_FEAB
[DeviceList.NTamd64]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_FEAB
```

Contenuto modificato mchpcdc.inf

```
[DeviceList]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_000A
[DeviceList.NTamd64]
%DESCRIPTION%=DriverInstall, USB\VID_04D8&PID_000A
```

Fig. 5 Modifiche al contenuto del file mchpcdc.inf

A questo punto l'installazione e la configurazione dell'ambiente di sviluppo è completata. Come verifica si può provare quindi a lanciare l'IDE e controllare che nel file pinguino.log, presente nella directory principale dell'installazione, non compaiano messaggi di errore.

[16] http://dl.dropbox.com/u/8407374/Paride_Miola_Pubblica/mchpcdc.cat

Contenuto pinguino.log

```
Pinguino started at Sat Apr 21 21:02:18 2012

Python version is good (>=2.5)
System host is Windows
8-bit compiler OK.
32-bit compiler OK.
wx.python successfully loaded
wx.aui successfully loaded
Regex successfully loaded
Shutil successfully loaded
Subprocess successfully loaded
GetText successfully loaded
Locale successfully loaded
WebBrowser successfully loaded
Editeur successfully loaded
ArgParse successfully loaded
Pinguino Boards List successfully loaded

Everything is OK.
```

Fig. 6 Log installazione corretta

Per semplificare l'installazione e la configurazione dell'IDE senza apportare manulamente le modifiche ai vari file, si è realizzato un archivio ZIP contenente tutto il materiale necessario al corretto setup dell'ambiente di sviluppo. Tale archivio (circa 55Mb) è scaricabile all'indirizzo:

http://dl.dropbox.com/u/8407374/Paride_Miola_Pubblica/Ready_To_Go.zip

Fig. 7 QR-Code per il download rapido del pacchetto di configurazione realizzato

3. Hardware datalogger

Software e librerie illustrate in questo documento sono state sviluppate, utilizzando come piattaforma per i test, la scheda elettronica PIC32-PINGUINO-OTG prodotta dalla Olimex e compatibile con l'ambiente di sviluppo Pinguino32.

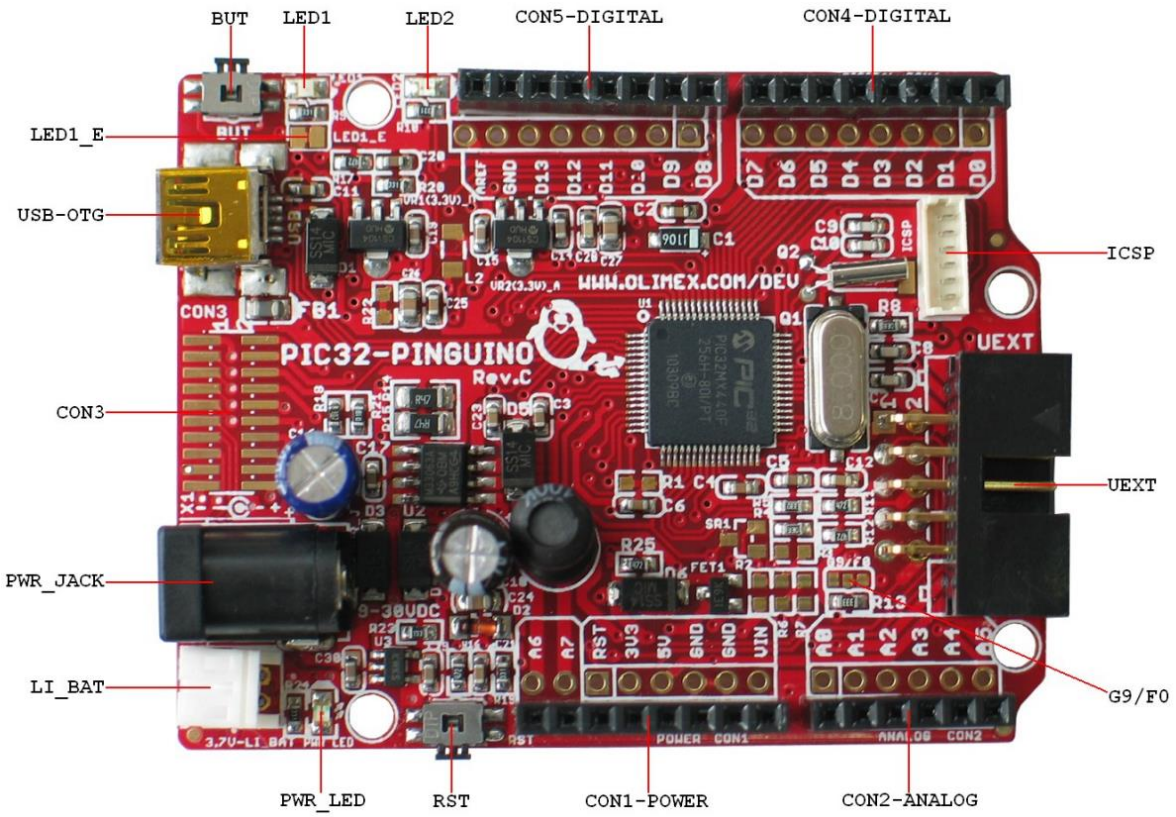


Fig. 8 Vista frontale della scheda Olimex PIC32-PINGUINO-OTG

Come si osserva in figura 8 è montato il processore PIC32MX440F256H della Microchip®, che offre 32Kb di RAM e 256Kb di memoria flash disponibile per l'utente. La possibilità di avere un clock base di 80Mhz permette inoltre di gestire una connessione USB full speed da 12Mb/s, tramite il connettore miniUSB presente nella scheda (usb-otg).

In basso a sinistra si trova la parte che si occupa di gestire l'alimentazione dell'elettronica, che può avvenire direttamente tramite USB, oppure alimentando il tutto con una tensione compresa tra 9V e 30V (pwr-jack). La scheda dispone anche di un carica batterie per una singola cella Li-Ion, collegabile al connettore LI_BAT, ed un circuito che si occupa dello "switch" tra l'alimentazione principale e quella tramite batteria.

Sui restanti lati si trovano vari header di cui, quelli superiori ed inferiori, compatibili con lo standard Arduino, mentre le nuove porte di I/O offerte dal PIC vengono raccolte nel molex sul lato destro e denominato UEXT.

Osservando più attentamente si può notare la presenza di un connettore per l'ICSP [17] che permette di "flashare" il microprocessore senza passare per il bootloader USB. Vicino al connettore è presente un quarzo più piccolo di quello utilizzato per il clock del PIC e che serve un integrato che si occupa di gestire l'RTCC[18] e di cui ne verrà fatto uso in questo progetto.

Come si può notare ispezionando la parte che si occupa della comunicazione USB non sono presenti integrati che eseguono una conversione USB-Seriale, si tratta infatti di una porta USB completa e non una emulazione seriale tramite FTDI. Questo permette di utilizzare la porta USB sia come host sia come client e di poter realizzare una serie di applicazioni HID[19] e CDC.

Infine, alimentando la scheda, è possibile notare la presenza di 3 LED, uno rosso che segnala che la scheda è alimentata e gli altri due, uno verde (LED1) ed uno giallo (LED2), che segnalano quando è attiva la funzione *bootloader*[20], ma che comunque possono essere utilizzati dall'applicazione utente per segnalazioni visive.

Sul retro (fig. 9) invece si trova un connettore per memory card nel formato MicroSD ed il piccolo integrato che si occupa del real time clock calendar.

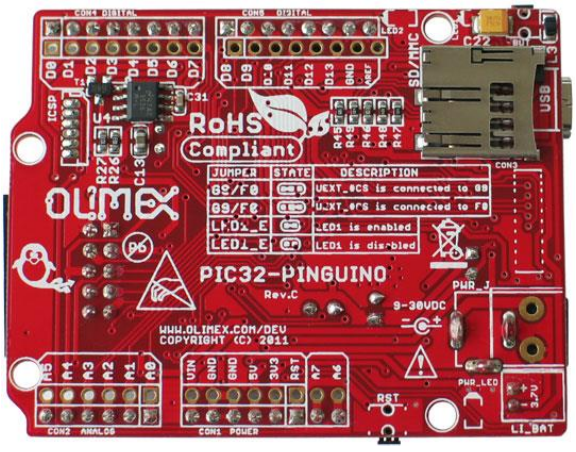


Fig. 9 Vista posteriore della scheda Olimex PIC32-PINGUINO-OTG

[17] In Circuit Serial Programming
[18] Real Time Clock Calendar.
[19] Human Input Device
[20] Per avviare la modalità bootloader è necessario premere il pulsante BUT durante la fase di boot della scheda, fino a quanto entambi i led, verde e giallo, iniziano a lampeggiare.

E' evidente che l'hardware utilizzato in questo progetto è di gran lunga superiore anche rispetto a quello offerto dalla più potente delle schede Arduino, la Arduino Mega 2560. Nella tabella vengono infatti riportate per confronto alcune caratteristiche.

Specifiche	Pinguino32 (PIC32MX440F256H)	Arduino Mega 2560
Architettura	32 bit	8 bit
Clock	80 Mhz	16 Mhz
Program Memory Size	512 Kb	256 Kb
RAM	32 Kb	8 Kb
Canali DMA	4	No
USB	1.1 & 2.0 Full Speed	No
UART	2	4
SPI	2	1
I2C	2	No
Interrupt	96	57
Canali ADC 10-bit	16	16
ADC Sample Rate	1000	1000
32-bit Timers	2	No
16-bit Timers	5	4
8-bit Timers	No	2
Porta parallela	PMP16	No
Comparatori	2	1
RTCC	Yes	No
I/O Pins	85	54
Pin Count	100	100
Low power	40 uA (battery)	500 uA

Tab. 1 Confronto Pinguino / Arduino

3.1 Pinout lettore MMC

Prima di poter utilizzare il lettore di memory card è necessario conoscere quali saranno i pin utilizzati dal microcontrollore per evitare pericolosi conflitti. Il pinout è stato individuato utilizzando lo schema elettrico mostrato in figura A1 allegata in appendice ed è stato riassunto nella seguente tabella.

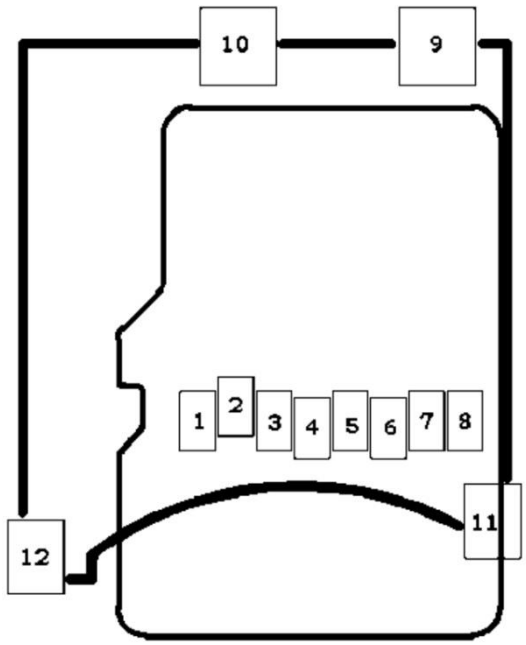


Fig. 10 Pinout lettore memory card

Pin #	Segnale
1	MCIDAT2
2	D8_MMC_#SS
3	D11 (MOSI)
4	+3.3V
5	D13 (SCK / LED1)
6	GND
7	D12 (MISO)
8	MCIDAT1
9	NC
10	NC
11	NC
12	NC

Tab. 2 Corrispondenza segnali/pin

4. Codice sorgente datalogger

Il codice sorgente è stato strutturato in modo da offrire una semplice interfaccia, visualizzabile tramite terminale, in modo da poter testare singolarmente le varie funzioni che compongono il programma.

Nella figura 9 vengono mostrate le impostazioni nel caso in cui si scelga di utilizzare come terminale il software Microsoft HyperTerminal. Per come sono strutturati i driver CDC, il *bitrate* della porta virtuale non è significativo, infatti è possibile impostare anche velocità minori di 115Kbps senza compromettere il funzionamento del datalogger.

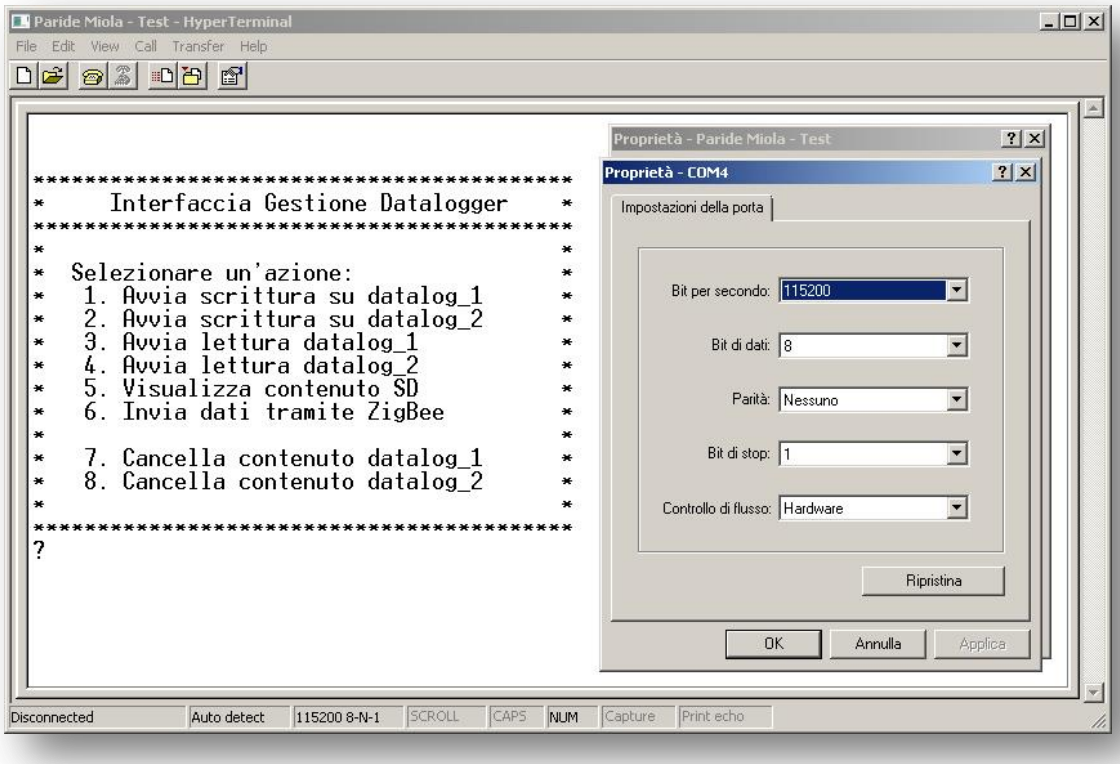


Fig. 11 Vista principale interfaccia di gestione

4.1 Header

Nell'header, riportato di seguito, sono state riassunte alcune informazioni salienti del progetto: i dettagli relativi ai pin riservati ed utilizzati dal datalogger ed alcune note riguardo le funzioni del sorgente.

```
/*
********************************************************************
INFO:
  Datalogger con supporto per memory card SD/MMC su File System FAT32 e
  trasmissione dati tramite USB e protocollo ZigBee.

Developer:     Paride Miola 578216 Univeristà di Padova
Tested on:     Olimex PIC32-PINGUINO-OTG
IDE:           x.2 trunk r305 + sd.pdl32 (x.3 r246)
               + diskio.c, ff.c, fileio.c (x.3 r250)
               + ieee154.h + mrf24j40.c (x.2 r270)

Based on Regis Blanchot, Alfred Broda, Mark Harper and Pierre Mandon
documentation.

ZigBee Pinout to UEXT connector
SD Pinout to SPI:  MOSI - pin 11
                   MISO - pin 12
                    CLK - pin 13
                     CS - pin 8

********************************************************************
CHANGELOG:
  1. code size: 91332 / 262144 bytes (34% used)
  2.

********************************************************************
NOTE:
  1. La dimensione del buffer di lettura modifica il tempo necessario a
     leggere il file. Il valore ottimale è legato alla dimensione
     dell'unità di allocazione del File System della SD.
  2. Dichiarare il buffer di lettura localmente anzicchè globalmente
     comporta un aumento della dimensione del codice di 60byte.

********************************************************************
METODI:
  1. write_to_sd(filename, *data)     //data come puntatore ad array char
  2. create_dummy_data(filename)
  3. delete_from_sd(filename)         //e. "Cartella/data.csv"
  4. read_sd(filename)
  5. get_rtc()
  6. dir_sd()

********************************************************************
*/
```

4.2 Impostazioni, variabili e costanti

Di seguito è riportata la sezione iniziale del codice in cui sono stati raccolti tutti i parametri di interesse, che possono essere assegnati per configurare i componenti interni della scheda. In particolare si dividono in quattro gruppi: una parte relativa alla dimensione in byte del buffer di lettura del MCR[21], una parte relativa ai parametri di configurazione del modulo radio MRF24J40MB-I/RM, una serie di impostazioni di debug ed infine una parte riguardante il setup iniziale del modulo RTCC.

```
#include <stdbool.h>              //definizioni boolean datatype


//--- IMPOSTAZIONI MCR ---
#define READ_BUFFER_SIZE 32       //dimensione buffer di lettura
#define WRITE_BUFFER_SIZE 32      //dimensione buffer di scrittura


//--- IMPOSTAZIONI ZigBee ---
//#define USE_ZIG_BEE             //commentare per disattivare ZigBee
#define ZIG_CHANNEL 20            //il canale va scelto tra 11 e 26
#define ZIG_NETWORK_ID 0x003C     //personal area network 0~65535 (PAN)
#define BASE_ADDRESS 0x0001       //indirizzo modulo trasmittente 0~65535
#define DEST_ADDRESS 0x0002      //indirizzo modulo ricevente 0~65535


//--- IMPOSTAZIONI DEBUG ---
#define USE_WRITE_LED             //attiva LED2 durante la scrittura
#define USE_READ_LED              //attiva LED2 durante la lettura
#define USE_SEND_LED              //attiva LED2 durante la trasmissione
#define USE_DIR_LED               //attiva LED2 durante enum directory


//--- IMPOSTAZIONI RTCC ---
#define ORARIO 0x10300000         //orario dell'RTC nel formato 0xHHMMSS00
#define DATA 0x12011703           //data dell'RTC nel formato 0xAAMMGG0S
                                  //S=giorno della settimana Sab=0 Ven=6
```

[21] Memory Card Reader

```
//--- VARIABILI GLOBALI ---
char        read_buffer[READ_BUFFER_SIZE];      //buffer di lettura Nota2
bool        menu_ok;                            //flag menu
u8          input_menu;                         //scelta menu
FIL         file;                               //struttura file object
FRESULT     mount_error, dummy_error;           //risposte dal fs
DWORD       file_size;                          //dimensione file (EOF)
rtccTime    temp_orario, orario;                //orario dall'rtcc
rtccDate    temp_data, data;                    //data dall'rtcc
#ifdef USE_ZIG_BEE
u8   n;                                         //ZigBee dummy data
#endif


//--- COSTANTI ---
const u16 drift = 180;                          //correzione RTC
const u32 setup_orario = ORARIO;
const u32 setup_data = DATA;


//--- PROTOTIPI ---
void write_to_sd(char filename[], int *buffer_pointer);
void create_dummy_data(char filename[]);
void delete_from_sd(char filename[]);
void read_sd(char filename[]);
void get_rtcc();
void dir_sd();
```

4.3 Entry Point

Dopo aver alimentato la scheda, la prima istruzione, eseguita dal microcontrollore nella posizione 0x0000, è un salto alla porzione finale della memoria flash del PIC. Qui infatti risiede il bootloader Pinguino32, che esegue un controllo sullo stato dell'input collegato al pulsante BUT. Se il pulsante è premuto viene avviato il modulo di programmazione tramite USB, altrimenti un altro salto porta il PC[22] esattamente nella posizione 0x0004 dove risiede l'inizio dell'applicazione utente.

4.3.1 Setup()

Il setup, sotto riportato, è la prima routine richiamata all'avvio dell'applicazione utente e contiene il codice necessario alla configurazione della scheda e dell'hardware ad essa collegato. In questo caso si occupa di inizializzare il Real Time Clock ed il modulo radio MRF24J40MB-I/RM, richiamando rispettivamente le funzioni RTCC.open e ZIG.init. I parametri con cui inizializzare l'RTCC e il modulo radio possono essere facilmente modificati operando sulle definizioni e le costanti presenti tra le impostazioni iniziali.

Nota: Il modulo ZigBee non viene inizializzato nel setup(), ma viene reinizializzato ogni volta che viene richiamata la funzione di trasmissione, questo perché i pin utilizzati dal modulo radio sono condivisi con il bus SPI utilizzato dalla memory card e questo provoca dei problemi di sincronia nel caso si provasse a trasmettere dopo aver avuto accesso alla scheda di memoria.

```
//*********************************************************************/
//_____Setup_____________________________________________________________/
//*********************************************************************/
void setup()
{
  int i;

  //imposto i pin collegati ai led verde e giallo come output
  pinMode(13, OUTPUT);
  pinMode(30, OUTPUT);

  digitalWrite(13, HIGH);      //LED1 on
  digitalWrite(30, HIGH);      //LED2 on
```

[22] Program Counter

```
  //Inizializzo ed avvio il Real Time Clock Calendar
  RTCC.open(setup_orario, setup_data, drift);

  //Pausa iniziale, contente a Windows di montare il dispositivo senza
  //mandare in overload il buffer in ingresso della porta USB
  delay(5000);

  //routine led blink
  for (i = 0; i < 25; i++)
  {
       digitalWrite(30, HIGH);
       digitalWrite(13, HIGH);
       delay(100);
       digitalWrite(30, LOW);
       digitalWrite(13, LOW);
       delay(100);
  }

  CDC.println("\n\r*** Premere invio per avviare l'applicazione.");
  while(CDC.getKey() != '\r');     //attendo finchè non ricevo un return

}
```

4.4 Funzioni accessorie

Le routine presentate in questa sezione non fanno propriamente parte del cuore del software di datalogging, ma sono state scritte per mostrare un esempio applicativo di interazione con le funzioni di accesso alla memory card e realtime clock.

4.4.1 get_rtcc()

Più che una vera e propria funzione, get_rtcc è un blocco di codice separato. Si è scelto di utilizzare questo stile di programmazione modulare per sopperire alla mancanza di un debugger nell'IDE e semplificare così la lettura e correzione del sorgente. Il codice si occupa semplicemente di interrogare l'RTCC e salvare i valori di orario e data in formato decimale.

L'RTCC è un piccolo integrato che si occupa di gestire e tenere memoria del timestamp concatenato alla stringa da memorizzare dopo ogni misurazione. La sua inizializzazione è molto semplice ed avviene tramite le definizioni iniziali del sorgente; una volta impostato è necessario che l'integrato resti alimentato, anche con una semplice batteria tampone, per poter mantenere aggiornato l'orario di sistema.

```
//*****************************************************************/
//_____RTCC_____________________________________________________/
//*****************************************************************/
void get_rtcc()
{
  RTCC.getTimeDate(&temp_orario, &temp_data);
  //conversione da bcd in decimale di orario e data
  orario = RTCC.convertTime(&temp_orario);
  data = RTCC.convertDate(&temp_data);
}
```

4.4.2 dir_sd()

Per offrire una migliore interfaccia utente (fig. 13) e mostrare le potenzialità delle librerie per la gestione della memory card, è stata scritta una funzione che permette di visualizzare il contenuto della scheda di memoria, mostrando sul terminale i file e le directory presenti al suo interno.

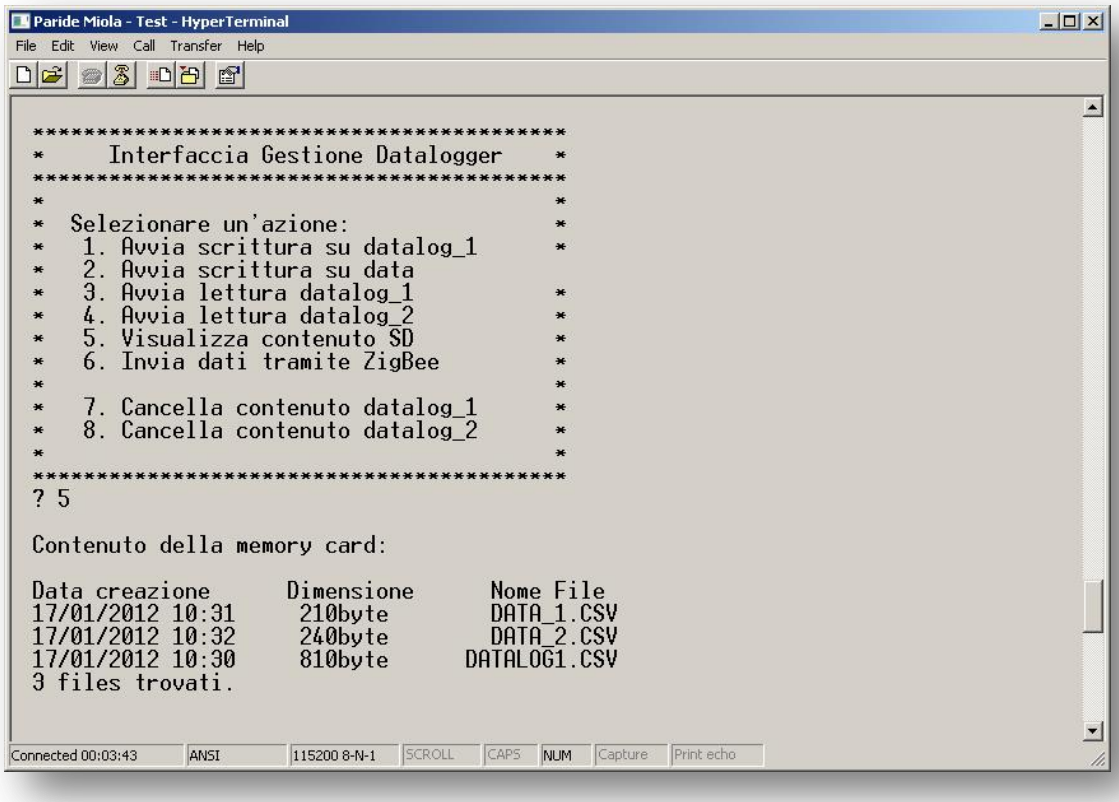


Fig. 13 Output della funzione dir_sd()

```
//*************************************************************************/
//_____Dir_____________________________________________________________/
//*************************************************************************/
void dir_sd()
{
  #ifdef USE_DIR_LED
  digitalWrite(30, HIGH);           //LED2 on
  #endif

  CDC.println("5");
  CDC.println(" ");
```

```
  if (mount(8))
  {

    CDC.println("Contenuto della memory card:");
    CDC.println(" ");

    char b;
    unsigned int files, dirs;
    SD_Dir dir;                    //directory object
    SD_FileInfo Finfo;             //file info
    SD_Error res;

    res = SD.openDir(&dir, "/");
    files = dirs = 0;
    CDC.println("Data creazione      Dimensione      Nome File");
    for (;;)
    {
      if ((SD.readDir(&dir, &Finfo) != 0) || !Finfo.fname[0])
      {
        break;                     //break in caso di errore o EOF
      }

      if (SD.isDir(Finfo))
      {
        dirs++;                    //conto le directory presenti
      }
      else
      {
        files++;                   //conto i file presenti
      }

      CDC.printf("%02u/%02u/%u %02u:%02u ", Finfo.fdate & 31,
           (Finfo.fdate >> 5) & 15,(Finfo.fdate >> 9) + 1980,
(Finfo.ftime >> 11),
           (Finfo.ftime >> 5) & 63);
      CDC.printf(" %6ubyte ", Finfo.fsize);
      CDC.printf(" %16s\r\n", Finfo.fname);
    }
    CDC.printf("%d files trovati.\r\n", files);


    unmount();                   //deinizializzo la memory card

  }
  else
  {
    CDC.println("\n\r*** ERRORE memory card non presente o File System
corrotto ***\n\r");
  }

  #ifdef USE_DIR_LED
  digitalWrite(30, LOW);         //LED2 on
  #endif

}
```

4.4.3 delete_from_sd(filename)

Per completezza è stata scritta anche una semplice funzione per cancellare i vari file di log creati sulla scheda SD. Avviando l'interfaccia utente (fig. 12) è sufficiente richiamare la routine passando come parametro il nome del file da cancellare, per eliminare il relativo file CSV dalla scheda di memoria.

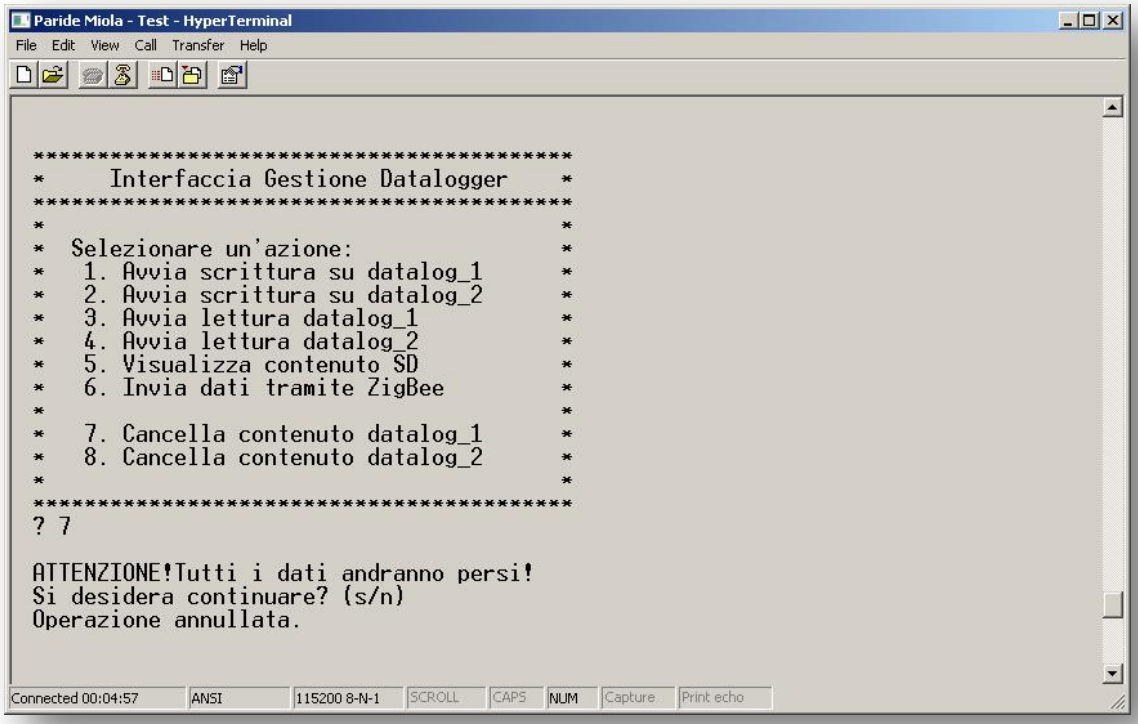


Fig. 12 Output della funzione delete_from_sd(filename)

```
//*********************************************************************/
//_____Erase_____________________________________________________________/
//*********************************************************************/
void delete_from_sd(char filename[])
{
   CDC.println(" ");
   CDC.println("ATTENZIONE!Tutti i dati andranno persi!");
   CDC.print("Si desidera continuare? (s/n)");

   menu_ok = 1;
   while(menu_ok)
   {
      input_menu = CDC.getKey();
      switch (input_menu)
      {
      case 's':
```

```
        menu_ok = 0;
        SD.mount(8);
        SD.remove(filename);
        SD.unmount();
        CDC.printf("\r\nIl file %s e' stato eliminato con successo.\r\n",
filename);
        break;
      case 'n':
        menu_ok = 0;
        CDC.println("\r\nOperazione annullata.");
        break;
      }
   }

}
```

4.4.4 create_dummy_data(filename)

Questa routine si occupa di generare una stringa formattata e pronta per essere salvata su memory card. In questo modo è stato possibile separare la funzione di generazione da quella di scrittura, rendendo il più generale ed universale possibile, la funzione write_to_sd(filename, *data).

```
//*********************************************************************/
//_____Dummy_data_____________________________________________________/
//*********************************************************************/
void create_dummy_data(char filename[])
{
  CDC.println(" ");

//---------------------------------------------------------------------
// INTERROGO I SENSORI IN MODO CHE IL TIMESTAMP SIA COERENTE
//   es. double valore_sensore = SPI.read(PIN);
//---------------------------------------------------------------------

  get_rtcc();          //interrogo l'RTC

  //__costruisco_la_stringa_da_salvare_____________________________
  //nel formato "SENSOR_ID,dd/mm/yyyy,hh:mm:ss,##.##"
  //separo i dati con una virgola per creare un file CSV compatibile
  //es. "ID0,21/12/2012,12:00:00,37.00"

  char buffer[16] = "";   //temporanea per la costruzione della stringa
  char stringa[WRITE_BUFFER_SIZE] = ""; //----- buffer in uscita -----

  //inserisco l'id del sensore
  sprintf(buffer, "ID%d,", 0);
  strcat (stringa, buffer);          //stringa=ID0,

  //inserisco la data dd/mm/yyyy,
  sprintf(buffer, "%02d/%02d/%04d,", data.mday, data.mon, data.year +
2000);
  strcat (stringa, buffer);          //stringa=ID0,dd/mm/yyyy,

  //inserisco l'orario HH:MM:SS,
  sprintf(buffer, "%02d:%02d:%02d,", orario.hour, orario.min,
orario.sec);
  strcat (stringa, buffer);          //stringa=ID0,dd/mm/yyyy,HH:MM:SS,

  //aggiungo il valore letto dal sensore
  sprintf(buffer, "%d.%02d\n", 37, 00);
  strcat (stringa, buffer);        //ID0,ddd,dd-mmm-yyyy,HH:MM:SS,37.00"

//---------------------------------------------------------------------
//A QUESTO PUNTO HO LA STRINGA FORMATTATA PER ESSERE SCRITTA SULLA SD
//---------------------------------------------------------------------

  write_to_sd(filename, &stringa);    //scrivo il contenuto di stringa
                                      //sul file data_1.csv/data_2.csv
}
```

4.4.5 zigbee_send_test()

Questa funzione raccoglie semplicemente le routine utilizzate per generare una stringa di prova da trasmettere via radio.

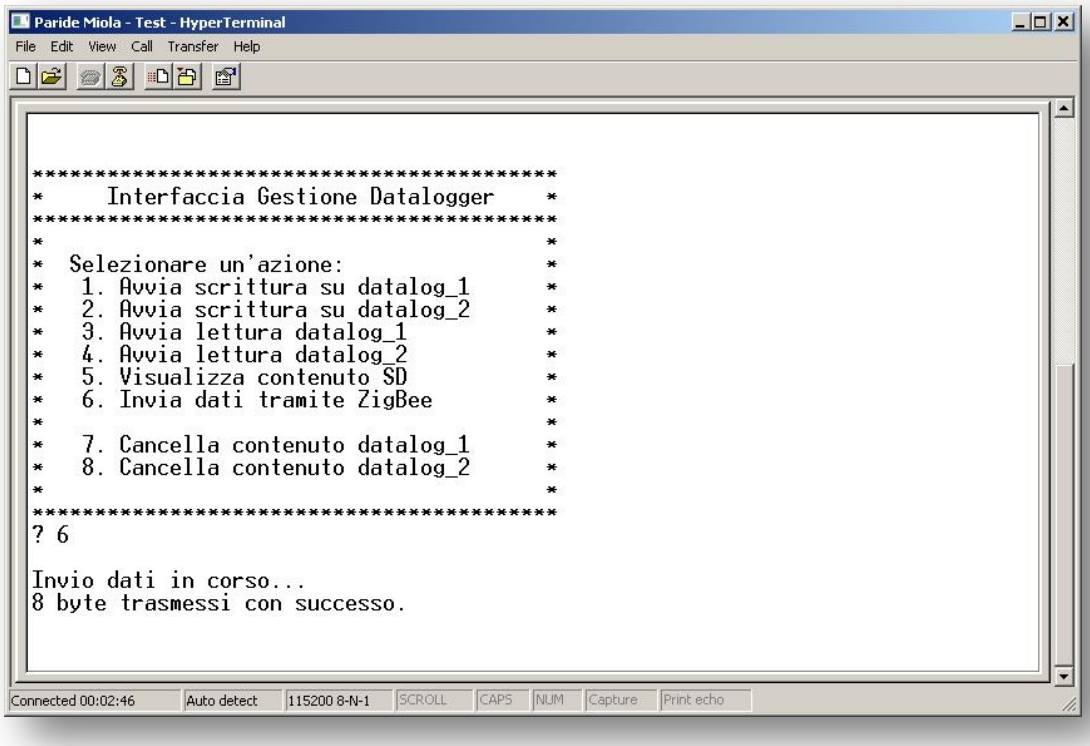


Fig. 14 Output della funzione zigbee_send_test()

```
//*************************************************************************/
//______ZigBee_____________________________________________________________/
//*************************************************************************/
void zigbee_send_test()
{
  CDC.println("\n\rInvio dati in corso...");
  #ifdef USE_SEND_LED
    digitalWrite(30, HIGH);      //LED2 on
  #endif
  #ifdef USE_ZIG_BEE
    ZIG.init(ZIG_CHANNEL, ZIG_NETWORK_ID, BASE_ADDRESS);
    char test[16] = "";
    sprintf(test, "Test n.%d", n);
    ZIG.send(DEST_ADDRESS, test, strlen(test));
    n++;
  #endif
  CDC.printf("%d byte trasmessi con successo.\n\r", strlen(test));
  #ifdef USE_SEND_LED
    digitalWrite(30, LOW);       //LED2 off
  #endif
}
```

4.5 Main Loop

Il main è stato strutturato sotto forma di un unico loop, in modo da ripetere più volte l'utilizzo delle varie funzioni che compongono il programma e poter vedere sul terminale i risultati che queste producono.

```
//*************************************************************************/
//_____Main_____________________________________________________________/
//*************************************************************************/
void loop()
{
  CDC.println(" ");
  CDC.println(" ");
  CDC.println("*****************************************");
  CDC.println("*     Interfaccia Gestione Datalogger    *");
  CDC.println("*****************************************");
  CDC.println("*                                       *");
  CDC.println("*  Selezionare un'azione:               *");
  CDC.println("*   1. Avvia scrittura su datalog_1     *");
  CDC.println("*   2. Avvia scrittura su datalog_2     *");
  CDC.println("*   3. Avvia lettura datalog_1          *");
  CDC.println("*   4. Avvia lettura datalog_2          *");
  CDC.println("*   5. Visualizza contenuto SD          *");
  CDC.println("*   6. Invia dati tramite ZigBee        *");
  CDC.println("*                                       *");
  CDC.println("*   7. Cancella contenuto datalog_1     *");
  CDC.println("*   8. Cancella contenuto datalog_2     *");
  CDC.println("*                                       *");
  CDC.println("*****************************************");
  CDC.print("? ");

  menu_ok = 1;
  while(menu_ok){
  input_menu = CDC.getKey();
    switch (input_menu)
    {
    case '1':
      menu_ok = 0;
      CDC.println("1");
      create_dummy_data("data_1.csv");
      break;
    case '2':
      menu_ok = 0;
      CDC.println("2");
      create_dummy_data("data_2.csv");
      break;
    case '3':
      menu_ok = 0;
      CDC.println("3");
      read_sd("data_1.csv");           //leggo il file data_1.csv
      break;
    case '4':
      menu_ok = 0;
      CDC.println("4");
      read_sd("data_2.csv");           //leggo il file data_1.csv
```

```
        break;
      case '5':
        menu_ok = 0;
        dir_sd();
        break;
      case '6':
        menu_ok = 0;
        CDC.println("6");
        zigbee_send_test();
        delay(1000);        //il delay mi evita che si sovrappongano
                            //richieste di invio sul modulo radio
        break;
      case '7':
        menu_ok = 0;
        CDC.println("7");
        delete_from_sd("data_1.csv");
        break;
      case '8':
        menu_ok = 0;
        CDC.println("8");
        delete_from_sd("data_2.csv");
        break;
      }
   }

   delay(500);

}
```

4.6 Scrittura su Memory Card

La funzione necessita unicamente di due parametri per essere richiamata: il nome del file su cui si intende scrivere i dati ed un puntatore al buffer di scrittura. Il nome del file va passato, sotto forma di stringa, con il path completo e comprensivo di estensione, ad esempio "cartella/file.csv".

Il buffer di elementi da scrivere deve essere in formato char e può avere dimensione arbitraria [23], la funzione write_to_sd infatti provvede da sola a calcolare l'effettivo numero di byte che dovranno essere scritti sulla memory card e a sequenziarli correttamente durante le operazioni di scrittura.

Da un punto di vista strettamente concettuale in termini di programmazione, sarebbe stato più corretto permettere alla funzione write_to_sd di ritornare un valore che specificasse se la scrittura fosse andata a buon fine. Tuttavia, a causa di alcuni bug presenti nell'IDE X.2, non è possibile, per l'utente, dichiarare funzioni diverse dal tipo void, pertanto si è deciso di integrare i messaggi di errore direttamente nell'output sul terminale.

Come è possibile vedere in figura 12, ogni operazione di scrittura termina con un messaggio di stato che segnala l'eventuale successo o presenza di errori.

La funzione è stata scritta in modo da segnalare e distinguere errori dovuti a problemi della scheda microSD (scheda assente, File System non compatibile, memory card corrotta) da quelli di scrittura sul file, dovuti a blocchi compromessi o problemi di alimentazione.

Per avere un debug visivo delle operazioni, il led giallo presente sulla scheda, denominato LED2, è stato programmato per accendersi durante le fasi di scrittura sulla memory card.

Il secondo led montato sulla scheda di sviluppo, denominato LED1, risulta essere collegato al piedino numero 13 della PORTD del PIC. Tale pin, come è possibile constatare osservando lo schema elettrico in figura A1 allegata in appendice, è condiviso con il segnale SCK[24] presente sul lettore di mermory card.

[23] Il numero massimo di byte è limitato dalla dimensione massima del tipo integer fissato a 32767, tuttavia non andrebbe creato un buffer più grande di 128byte per evitare problemi di paging della memoria RAM.

[24] SCK o SCLK corrisponde al segnale di sincronismo Sync Clock

Le routine di accesso alla scheda SD utilizzano questo canale per inviare un breve impulso di sincronismo al lettore di memory card, con il risultato che anche il led verde, presente di default sulla scheda, si accede per tutta la durata dell'impulso. In questo modo l'utente ha una conferma visiva del fatto che si sta accedendo alla memory card.

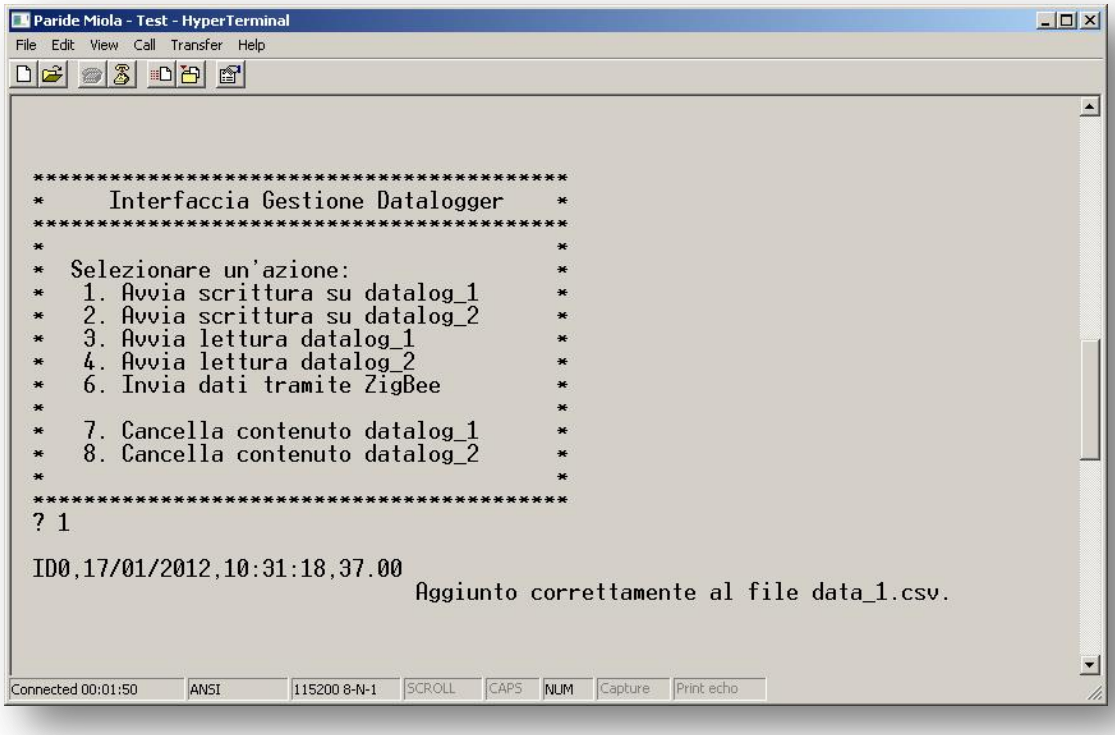


*Fig. 15 Output della funzione write_to_sd(filename, *data)*

```
//*********************************************************************/
//_____Scrittura_____________________________________________________/
//*********************************************************************/
void write_to_sd(char filename[], int *buffer_pointer)
{
  #ifdef USE_WRITE_LED
    digitalWrite(30, HIGH);      //LED2 on
  #endif

  int  byte_written;             //contatore byte scritti su memory card

  //calcolo la dimensione EFFETTIVA del buffer in uscita
  int buffer_length = strlen(buffer_pointer);

  SD.mount(8);                   //inizializzo la memorycard
  mount_error = SD.open(&file, filename, FA_OPEN_ALWAYS | FA_READ |
FA_WRITE);
```

```
  if(!mount_error)
  {
     file_size = file.fsize;      //leggo la dimensione del file esistente
     SD.seek(&file,file_size);    //imposto il puntatore di scrittura
                                  //alla fine del file (append)

     //avvio la scrittura su file e salvo l'esito in dummy_error
     dummy_error = SD.write(&file, buffer_pointer, buffer_length,
&byte_written);
     SD.flush(&file);             //svuoto la cache

     //controllo se ho scritto tutti i byte
     if(buffer_length == byte_written)
     {
        //TUTTO OK
        CDC.printf("%s ", buffer_pointer);
        CDC.printf("Aggiunto correttamente al file %s.\n\r", filename);
     }
     else
     {
        //ERRORE NELLA SCRITTURA
        CDC.printf("\n\r*** ERRORE di scrittura sul file %s ***\n\r",
filename);
     }

     SD.close(&file);               //chiudo il file
     SD.unmount();                  //deinizializzo la memorycard
  }
  else
  {
     //ERRORE NEL MOUNT
     CDC.printf("\n\r*** ERRORE (w) %s non trovato o memory card non
presente ***\n\r", filename);
  }

  #ifdef USE_WRITE_LED
     digitalWrite(30, LOW);       //LED2 off
  #endif
}
```

4.7 Lettura da Memory Card

La funzione di lettura della memory card richiede unicamente un parametro per essere utilizzata, ovvero il nome del file da aprire e leggere comprensivo di path ed estensione. Se il file è presente sulla memory card il suo contenuto viene inviato direttamente tramite USB sul terminale del computer, in caso contrario viene segnalato un errore di lettura.

La dimensione del buffer di lettura, impostabile tramite la definizione READ_BUFFER_SIZE, influisce sulla velocità di lettura dalla memory card ed è direttamente collegato alla disponibilità di memoria RAM per il microcontrollore. Purtroppo una mancanza dell'IDE è un visualizzatore di risorse occupate: risulta pertanto difficile calcolare con precisione quanta RAM resta effettivamente disponibile per l'utente, senza incorrere in pericolosi errori di overflow.

Come la funzione write_to_sd, anche in fase di lettura è possibile riconoscere errori di mount o lettura parziale, tramite degli avvisi sul terminale.

Per avere un debug visivo delle operazioni, il led giallo, denominato LED2 sulla scheda di sviluppo, è stato programmato per accendersi durante le fasi di lettura dalla memory card.

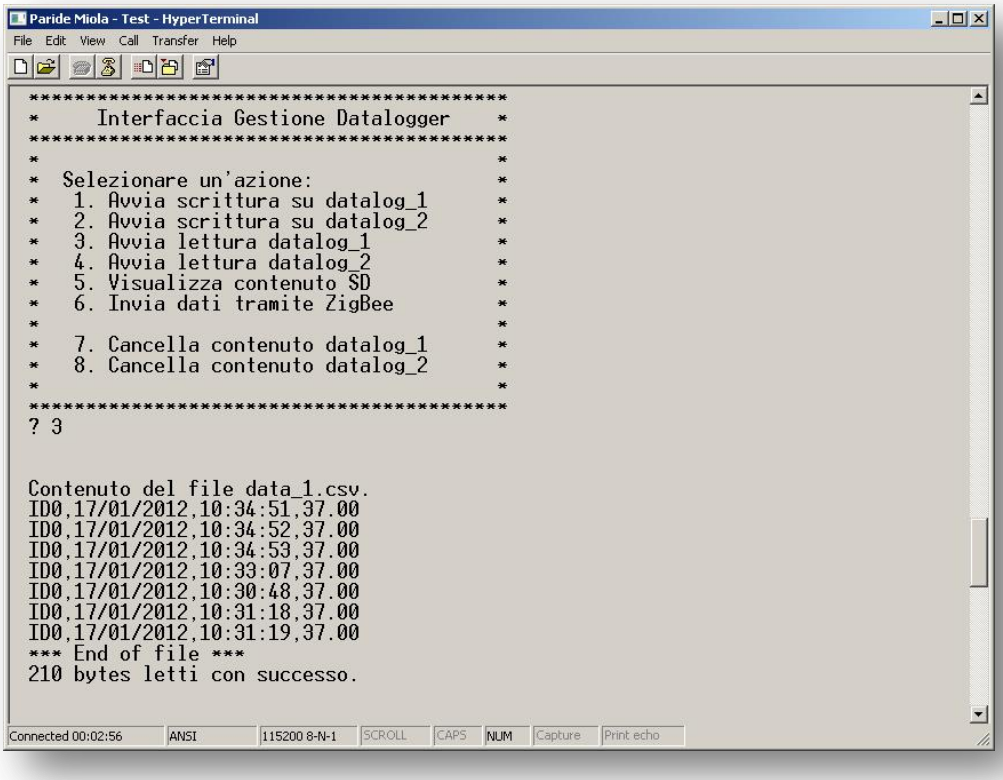


*Fig. 16 Output della funzione write_to_sd(filename, *data)*

```
//*********************************************************************/
//_____Lettura_____________________________________________________/
//*********************************************************************/
void read_sd(char filename[])
{
  #ifdef USE_READ_LED
    digitalWrite(30, HIGH);      //LED2 on
  #endif
  CDC.println(" ");
  int  i;                      // -__-'
  int  byte_read;              //byte letti da memory card (dipende dal FS)
  int  byte_counter;           //contatore byte letti

  SD.mount(8);                 //inizializzo la memorycard
  mount_error = SD.open(&file, filename, FA_OPEN_EXISTING | FA_READ |
FA_WRITE);

//Poichè voglio leggere il file dall'inizio non mi serve impostare la
//posizione del puntatore di lettura/scrittura in quanto, avendo
//reinizializzato il tutto, il puntatore parte nuovamente da 0

  if(!mount_error)
  {

    CDC.printf("\n\rContenuto del file %s.\n\r", filename);
    byte_counter = 0;
    do
    {
      dummy_error = SD.read(&file, read_buffer, READ_BUFFER_SIZE,
&byte_read);

      byte_counter = byte_counter + byte_read;//conta finale byte letti
      for(i = 0; i < byte_read; i++)
      {
        CDC.printf("%c", read_buffer[i]);   //stampo ogni byte letto
        if(read_buffer[i] == '\n')          //contenuto nel read_buffer
        {
          CDC.printf("\r");                 //separo i vari pacchetti
        }
      }
    }while(byte_read == READ_BUFFER_SIZE);     //leggo a blocchi di
                                               //READ_BUFFER_SIZE byte

    CDC.println("*** End of file ***");
    CDC.printf("%d bytes letti con successo.\n\r", byte_counter);

    SD.close(&file);               //Chiudo il file
    SD.unmount();                  //deinizializzo la memorycard
  }
  else
  {
    //ERRORE NEL MOUNT
    CDC.printf("\n\r*** ERRORE (r): %s non trovato o memory card non
presente ***\n\r", filename);
  }
  #ifdef USE_READ_LED
    digitalWrite(30, LOW);       //LED2 off
  #endif
}
```

5. Hardware Wireless

L'hardware scelto per gestire la parte di comunicazione wireless è basato su moduli radio MRF24J40MB-I/RM prodotti dalla Microchip® , vedi figura 17.

Nel dettaglio, questi moduli operano sulla frequenza di 2.4Ghz offrendo un *data rate* massimo di 250Kbps. I moduli supportano diversi protocolli wireless, tra cui il protocollo ZigBee, MIWI e MiWi P2P per la creazione di PAN[25].

Nell'applicazione specifica, è stato scelto il protocollo ZigBee, per l'efficienza nella trasmissione a bassa potenza e la semplicità dello stack e codice di gestione.

Fig. 17 Modulo radio MRF24J40MB-I/RM

Per testare l'effettivo funzionamento delle librerie wireless si è realizzato un ricevitore basato su una seconda scheda di sviluppo PIC32-PINGUINO-OTG ed anche essa equipaggiata con un modulo radio MRF24J40MB-I/RM.

Collegando questo ad un secondo computer è possibile realizzare un esempio di ponte radio, in cui un device fa da master (in questo caso la scheda di datalogging) ed un device fa da slave (il ricevitore).

[25] Personal Area Network

5.1 Pinout Shield ZigBee

Per semplificare il montaggio dei moduli radio, è stato realizzato un piccolo PCB per creare uno *shield* ZigBee che potesse alloggiare comodamente i moduli radio.

Dato che i moduli comunicano tramite SPI, si è scelto di collegare questo *shield* direttamente sul connettore di espansione UEXT presente sulla scheda di sviluppo.

Su questo connettore sono infatti presenti tutti i segnali utili per poter dialogare direttamente con i moduli radio senza dover ricorrere ad ulteriori pin; in questo modo si è evitato di occupare gli header presenti sulla scheda per lasciarli disponibili a *shields* standard.

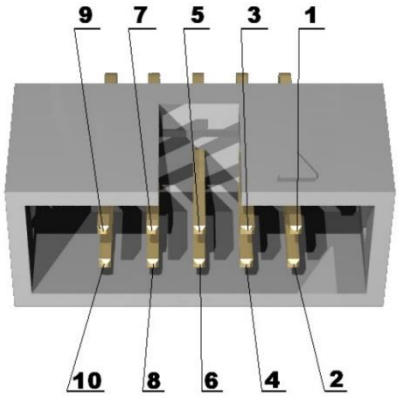


Fig. 18 Pinout connettore UEXT

Pin #	Segnale
1	+3.3V
2	GND
3	TX2
4	RX2
5	A5 (SCL1)
6	A4 (SDA1)
7	D12 (MISO)
8	D11 (MOSI)
9	D13 (SCK / LED1)
10	UEXT_#CS

Tab. 3 Corrispondenza segnali/pin

Nella figura sottostante è possibile osservare qual è il pinout del modulo radio MRF24J40MB-I/RM. Dallo schema si vene che non sono presenti pin a cui collegare un'antenna esterna, in quanto questa è integrata direttamente nella parte superiore del circuito stampato del modulo.

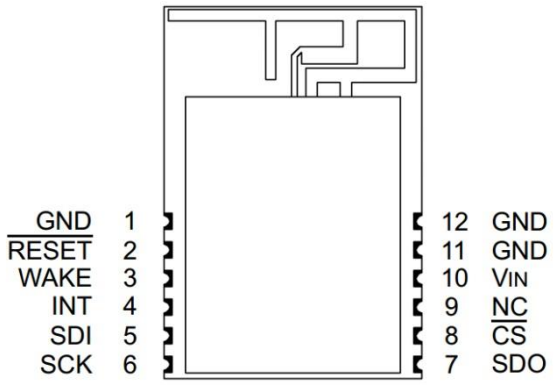


Fig. 19 Pinout modulo MRF24J40MB-I/RM

Nella tabella 4 vengono riassunte le corrispondenze tra i segnali presenti sul modulo MRF e quelli presenti sul connettore UEXT della scheda di sviluppo.

PIN # MRF	Segnale MRF	PIN # UEXT	Segnale UEXT
1	GND	**2**	GND
2	RESET	**3**	TX2
3	WAKE	**NC**	NC
4	INT	**5**	A5 (SCL1)
5	SDI	**8**	D11 (MOSI)
6	SCK	**9**	D13 (SCK / LED1)
7	SDO	**7**	D12 (MISO)
8	CS	**10**	UEXT_#CS
9	NC	**NC**	NC
10	VIN	**1**	+3.3V
11	GND	**2**	GND
12	GND	**2**	GND

Tab. 4 Corrispondenza segnali/pin tra UEXT e modulo radio

La figura 20 mostra quali sono i pin utilizzati sul connettore UEXT, evidenziando nelle parentesi tonde i relativi pin del modulo radio.

9. SCK (6)	7.SDO (7)	5. INT (4)	3. RST (2)	1. VIN (10)
10. CS (8)	8. SDI (5)	6. NC	4. NC	2. GND (1, 11, 12)

Fig. 20 Vista frontale connessioni molex UEXT

Si è quindi realizzato un PCB per agevolare il collegamento dei moduli radio direttamente sulle schede di sviluppo, in modo da mantenere i collegamenti ben saldi ed offrire una soluzione priva di cavi. Il PCB mostrato in figura 21, non fa altro che instradare correttamente i segnali tra il molex UEXT ed il modulo radio MRF.

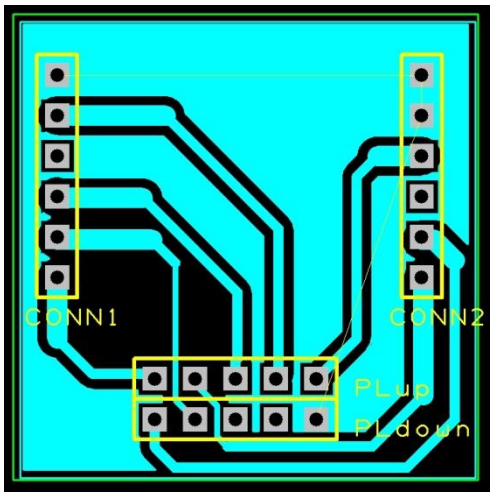


Fig. 19 PCB realizzato per il collegamento del modulo MRF24J40MB-I/RM

Nella figura A3 allegata in appendice, è possibile osservare una foto in dettaglio degli shields radio così realizzati.

6. Protocollo ZigBee

In telecomunicazioni ZigBee è il nome di una specifica per un insieme di protocolli di comunicazione ad alto livello che utilizzano piccole antenne digitali a bassa potenza e basato sullo standard IEEE 802.15.4 per WPAN[26].

6.1 Stack ZigBee

Lo stack è basato su di una recente ricerca nel campo degli algoritmi di routing (Ad-hoc On-demand Distance Vector) che puntano a costruire delle reti ad-hoc di nodi a bassa velocità. Queste reti oltre a prevedere un'architettura point-to-point prevedono anche la realizzazione di reti mesh o cluster.

L'architettura ad alto livello dello stack ZigBee, permette di semplificare notevolmente la complessità del codice necessario per poter utilizzare le funzionalità di ricetrasmissione, rendendo i moduli ZigBee superiori ad altre soluzioni wireless come il Bluetooth.

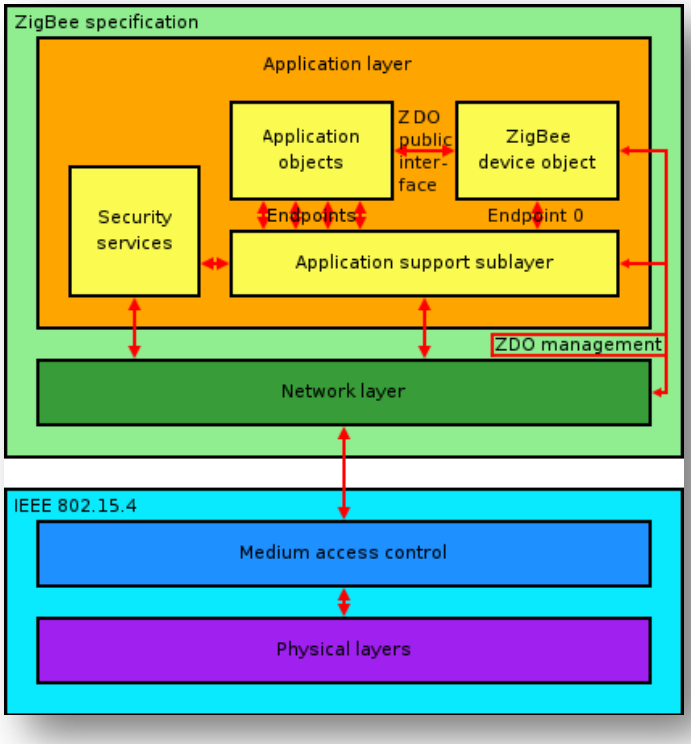


Fig. 22 ZigBee protocol stack

[26] Wireless Personal Area Networks

7. Codice sorgente ricevitore

Il programma si occupa di inoltrare, tramite USB, tutti i pacchetti ricevuti via radio e provenienti dal datalogger.

7.1 Header

L'header relativo al ricevitore è molto sintetico; le librerie presenti nella versione X.2 dell'IDE non hanno mostrato conflitti o problemi di compatibilità, rendendo così lo sviluppo del codice molto semplice.

```
/*
********************************************************************
INFO:
  Terminale di ricezione basato su protocollo ZigBee ed inoltro dei dati
  su porta USB tramite driver CDC.

Developer:     Paride Miola 578216 Univeristà di Padova
Tested on:     Olimex PIC32-PINGUINO-OTG + MRF24J40MB
IDE:           x.2 trunk r305

Based on Jean-Pierre Mandon documentation.

ZigBee Pinout to UEXT connector

********************************************************************
CHANGELOG:
  1. code size: 25716 / 262144 bytes (9% used)

********************************************************************
NOTE:
  1. Creare un array di più di 128byte sembra causare un funzionamento
     imprevedibile dell'applicazione.

********************************************************************
METODI:
  1.

********************************************************************
*/
```

7.2 Impostazioni, variabili e costanti

Le impostazioni disponibili per l'utente sono relative al solo modulo ZigBee: è sufficiente assegnare il canale di trasmissione e ricezione, l'ID definito per la rete radio e l'indirizzo base del ricevitore.

Particolare attenzione va posta nel configurare gli indirizzi dei moduli ZigBee: trattandosi di una comunicazione tra singoli elementi, è importante che questi siano correttamente identificati ed i rispettivi indirizzi assegnati siano univoci; in questo modo è possibile evitare problemi nel routing dei pacchetti scambiati.

Tramite la definizione USE_POLLING è possibile settare il programma di ricezione in due modalità, una di debug continuo in cui viene stampato in output lo stato del buffer in ingresso anche in assenza di dati ed una in cui il polling non è continuo. Questa seconda modalità è molto più utile per applicazioni finali in quanto è possibile interrogare il buffer di ricezione del modulo radio e "scaricare" i dati ricevuti solo se la coda in ingresso è non vuota.

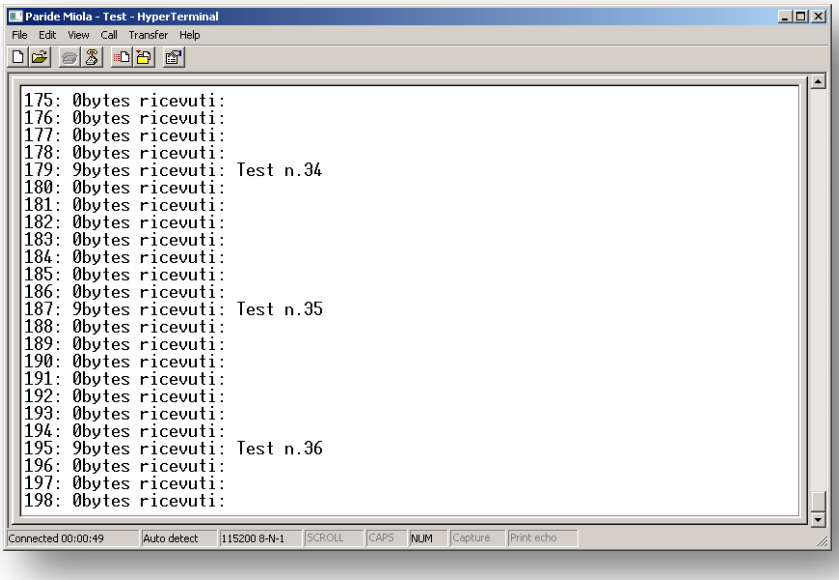


Fig. 23 Esempio di polling continuo

```
//--- IMPOSTAZIONI ZigBee ---
#define ZIG_CHANNEL 20            //il canale va scelto tra 11 e 26
#define ZIG_NETWORK_ID 0x003C     //personal area network 0~65535 (PAN)
#define BASE_ADDRESS 0x0002       //indirizzo modulo ricevente 0~65535
//#define USE_POLLING             //usa modalità continua di debug

//--- VARIABILI GLOBALI ---
#ifdef USE_POLLING
  u8   n;                         //heart beat
#endif
```

7.3 Setup

Anche in questa applicazione l'entry point punta direttamente alla routine di setup come prima operazione. Poiché l'unico hardware da inizializzare è il modulo ZigBee, il codice è molto simile a quello utilizzato nell'applicazione relativa al datalogger. In particolare viene inizializzato il modulo ZigBee utilizzando, ovviamente, lo stesso Network ID, ma un indirizzo (BASE_ADDRESS) differente e coincidente a quello impostato come DEST_ADDRESS nel sorgente del datalogger.

```
//*************************************************************************/
//______Setup_____________________________________________________________/
//*************************************************************************/
void setup()
{
  int i;

//imposto i pin collegati ai led verde e giallo come output
  pinMode(13, OUTPUT);
  pinMode(30, OUTPUT);

  digitalWrite(13, HIGH);      //LED1 on
  digitalWrite(30, HIGH);      //LED2 on

  //Inizializzo il modulo ZigBee (se abilitato)
  ZIG.init(ZIG_CHANNEL, ZIG_NETWORK_ID, BASE_ADDRESS);

  //Pausa iniziale, contente a Windows di montare il dispositivo senza
  //mandare in overload il buffer in ingresso della porta USB
  delay(5000);


  //routine led blink
  for (i = 0; i < 25; i++)  {
       digitalWrite(30, HIGH);
       digitalWrite(13, HIGH);
       delay(100);
       digitalWrite(30, LOW);
       digitalWrite(13, LOW);
       delay(100);
  }

  CDC.println("\n\r*** Premere invio per avviare l'applicazione.");
  while(CDC.getKey() != '\r');      //attendo finchè non ricevo un return
  CDC.println("\n\r*** Applicazione in ascolto...");

}
```

7.4 Main Loop

Il main dell'applicazione è abbastanza semplice, si tratta di un loop il cui funzionamento è impostabile dalla definizione USE_POLLING. Per fare uso completo delle librerie ZigBee sono stati inseriti, nei messaggi di stato, anche i parametri relativi al routing dei pacchetti ricevuti.

Come per le altre funzioni presentate in questa tesi, anche il software di ricezione fa uso dei led di segnalazione; il LED2 viene attivato con una intensità più bassa quando si sta eseguendo un semplice polling del buffer di ricezione ed utilizzando un'intensità più alta quando sono presenti dei dati, in questo modo si ha un debug visivo della comunicazione wireless, come è possibile osservare in figura 24.

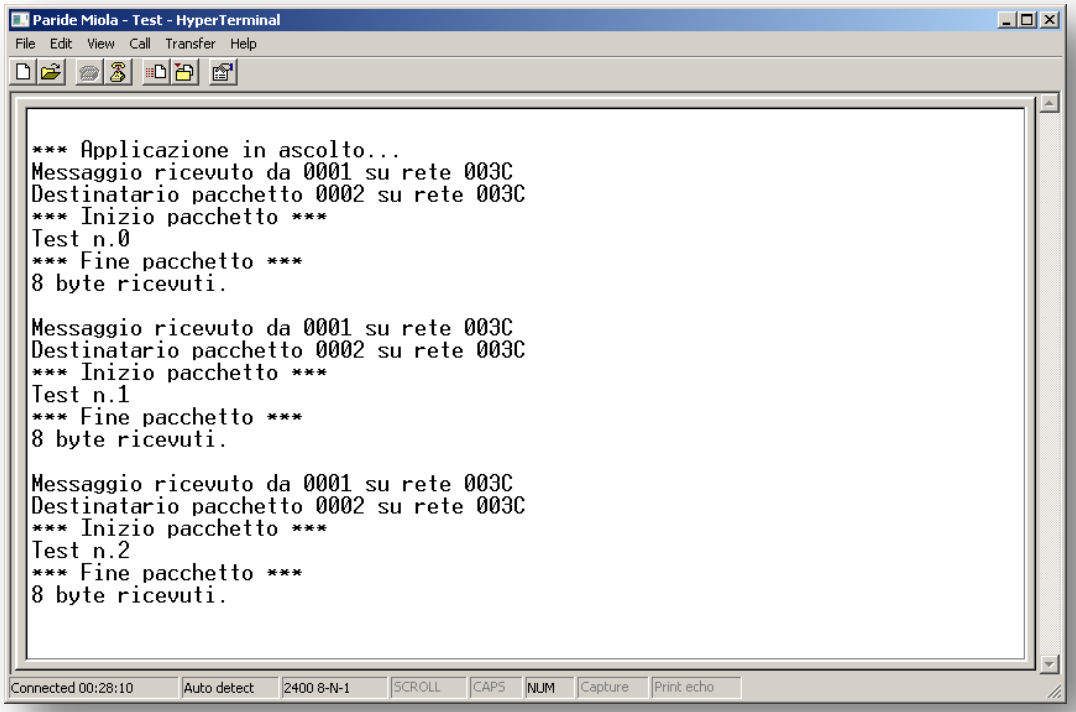


Fig. 24 Modalità di polling non continuo

```
//*********************************************************************/
//_____Main_____________________________________________________________/
//*********************************************************************/
void loop()
{
  digitalWrite(30, HIGH);              //LED2 on
  unsigned char rxdata[32] = "";       //usare più di 128byte provoca un
                                       //comportamento imprevedibile
```

```
  unsigned char in_buffer_length;

  in_buffer_length = ZIG.read(rxdata);  //buffer pieno?

  #ifdef USE_POLLING
     CDC.printf("\r\n%d: ", n);
     n++;
     CDC.printf("%dbytes ricevuti: ", in_buffer_length);
     CDC.printf(rxdata);
     delay(100);
  #endif

  #ifndef USE_POLLING
     if (in_buffer_length > 0)
     {
        CDC.printf("Messaggio ricevuto da %04X ", ZIGsrcadd);
        CDC.printf("su rete %04X\n\r", ZIGsrcpan);
        CDC.printf("Destinatario pacchetto %04X ", ZIGdestadd);
        CDC.printf("su rete %04X\n\r", ZIGdestpan);
        CDC.println("*** Inizio pacchetto ***");
        CDC.printf(rxdata);
        CDC.println("\r\n*** Fine pacchetto ***");
        CDC.printf("%d byte ricevuti.\r\n", in_buffer_length);
        CDC.println(" ");
     }
  #endif

  digitalWrite(30, LOW);                //LED2 off

  delay(100);

}
```

8. Sviluppi ulteriori

Nel capitoli precedenti sono state trattate le routine e le funzioni per stabilire e gestire la comunicazione tramite protocollo ZigBee e USB. Particolare interesse però meritano le librerie di comunicazione con dispositivi mobili. In particolare l'IDE Pinguino integra le librerie IOIO ADB per la comunicazione con dispositivi basati sul sistema operativo Android.

Per completezza si è deciso quindi di stilare una bozza di codice, per offrire un punto di partenza a chiunque volesse abilitare tale funzionalità all'interno del datalogger.

Grazie alla semplicità del linguaggio ed alla completezza delle librerie, sono sufficenti poche righe di codice per poter stabilire una connessione con un terminale ADB installato su un device mobile. In questo modo diventa possibile vedere i dati, registrati dal datalogger, direttamente sullo schermo del proprio tablet o del proprio cellulare.

```
/*
********************************************************************
INFO:
  Applicazione di esempio per la comunicazione verso dispositivi
  Android v1.6 utilizzando il protocollo ADB e le IOIO ADB Library

Developer:     Paride Miola 578216 Univeristà di Padova
Tested on:     Olimex PIC32-PINGUINO-OTG
IDE:           x.2 trunk r305

Based on Jean-Pierre Mandon documentation.

********************************************************************
CHANGELOG:
  1. code size: 46580 / 262144 bytes (17% used)

********************************************************************
NOTE:
  1. Creare un array di più di 128byte sembra causare un funzionamento
  imprevedibile dell'applicazione.

********************************************************************
METODI:
  1.

********************************************************************
*/
```

```
#include <stdbool.h>                    //definizioni boolean datatype


//--- VARIABILI GLOBALI ---
unsigned char ADB_in_buffer[128];      //buffer in ingresso
bool          ADB_opened = 0;          //flag connesione stabilita



//*********************************************************************/
//______Setup_____________________________________________________________/
//*********************************************************************/
void setup()
{
  //inizializzazione comunicazione Android
  Adb.init();
}



//*********************************************************************/
//______Main______________________________________________________________/
//*********************************************************************/
void loop()
{

  //Adb.refresh() va richiamato ciclicamente
  unsigned char connected = Adb.refresh();

  if(connected)                    //dispositivo collegato e riconosciuto
  {
    //apro una connessione TCP con il device
    if(!ADB_opened) ADB_opened = Adb.open();

    if(ADB_opened)                           //connessione stabilita
    {
      Adb.send("Test\n\r", 6);               //invia una stringa di test
      //Adb.send(out_buffer, strlen(out_buffer));  //es array di char
    }
  }

}
```

9. Conclusioni

In questo lavoro di tesi è stato realizzato e testato un codice per l'acquisizione di dati, la memorizzazione e trasmissione sia wired sia wirless. L'hardware utilizzato, basato su un progetto open source, è risultato adeguato alla realizzazione di un sistema di data logger. Il lavoro ha permesso di approfondire tutti gli aspetti relativi alla configurazione ed inizializzazione del microprocessore nonché l'implementazione di funzioni e librerie *ad-hoc*.

I dispositivi realizzati troveranno applicazione all'interno delle attività del Dipartimento nell'ambito dell'energy metering in cui giocherà un ruolo fondamentale sia la velocità di acquisizione sia la possibilità di trasmettere i dati in wireless.

Possibili sviluppi riguardano la possibilità di comunicazione con dispositivi mobili quali tablet, palmari o smartphone.

Appendice

Vengono riportate nel seguito le immagini dei prototipi utilizzati e lo schema elettrico della scheda di sviluppo Olimex PIC32-PINGUINO-OTG.

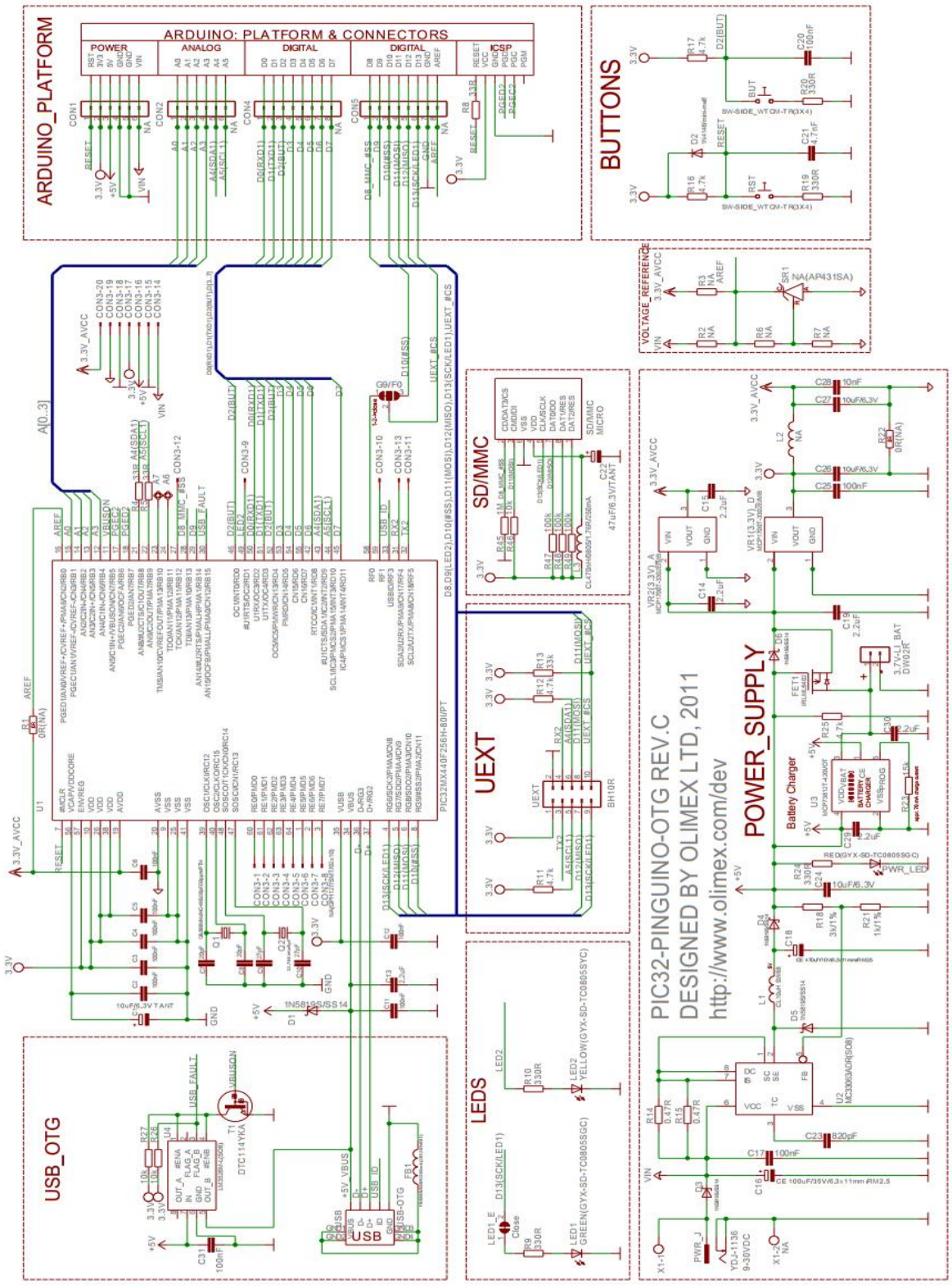


Fig. A1 Schema elettrico scheda di sviluppo PIC32-PINGUINO-OTG

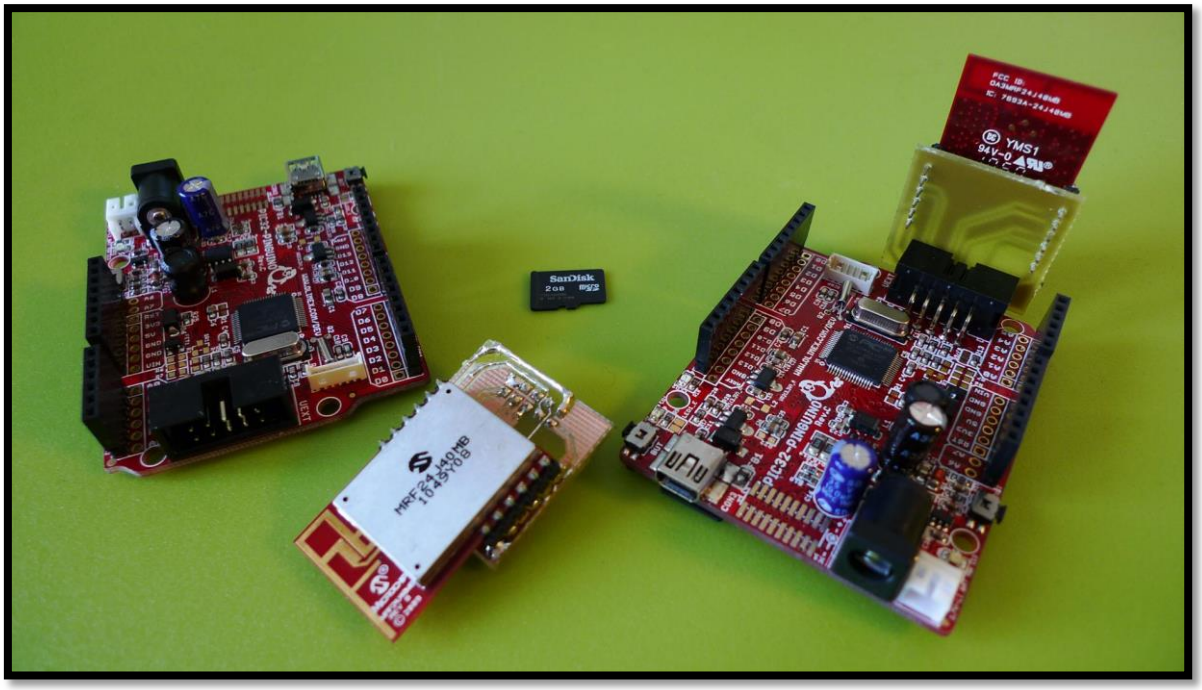


Fig. A2 Foto dei prototipi utilizzati

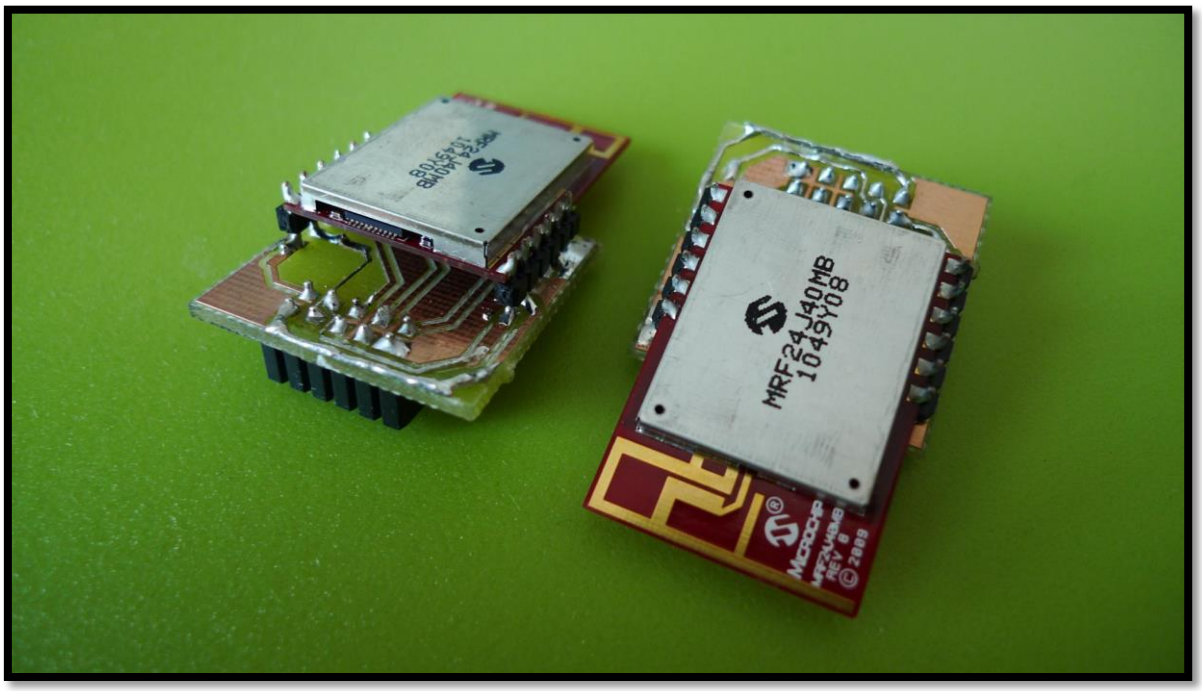


Fig. A3 Dettaglio degli shields ZigBee