ISSN 2039-7941
Anno 2016_Numero 356

rapporti tecnici INGV

Software per la sincronizzazione dei sistemi di acquisizione attraverso l'uso del GPS: implementazione hardware per architettura PC/104

356

Istituto Nazionale di Geofisica e Vulcanologia

Segreteria di Redazione
Francesca Di Stefano - Referente
Rossella Celi
Tel. +39 06 51860068
redazionecen@ingv.it

in collaborazione con:
Barbara Angioni (RM1)

REGISTRAZIONE AL TRIBUNALE DI ROMA N.173 | 2014, 23 LUGLIO

Rappresentante legale: Carlo DOGLIONI
Sede: Via di Vigna Murata, 605 | Roma

ISSN 2039-7941
Anno 2016_Numero 356

rapporti tecnici

INGV

SOFTWARE PER LA SINCRONIZZAZIONE DEI SISTEMI DI ACQUISIZIONE ATTRAVERSO L'USO DEL GPS: IMPLEMENTAZIONE HARDWARE PER ARCHITETTURA PC/104

Antonino Sicali, Pasqualino Cappuccio, Alfio Amantia

INGV (Istituto Nazionale di Geofisica e Vulcanologia, Sezione di Catania - Osservatorio Etneo)

356

Indice

Introduzione

Dal 1998 la rete magnetica dell'Etna provvede all'acquisizione del campo magnetico totale a una frequenza di campionamento di 0.2 Hz, una misura ogni 5 secondi. Inizialmente il sistema provvedeva alla sincronizzazione delle misure acquisite attraverso la linea di trasmissione [Del Negro et al., 1998]. Un'idea molto semplice da attuare a costo zero, che può nascondere comunque diversi problemi. Durante i periodi di avaria del sistema di trasmissione, che possono accadere più spesso di quando si creda, l'acquisizione può subire una forte deriva temporale a causa della base dei tempi, generalmente un *RTC* (Real Time Clock), imprecisa. Utilizzando particolari tecnologie, come i *GSM* (Global System for Mobile Communications), in luoghi con copertura non totale, si potrebbe esporre il sistema anche a blackout temporanei. Sul M.te Etna accade molto spesso che non ci sia la possibilità di sincronizzare l'acquisizione attraverso la linea di trasmissione. Inoltre, in alcuni siti non esiste un collegamento telemetrico utilizzabile per lo scopo poiché le misure sono acquisite solamente in locale. In altre situazioni la sincronizzazione attraverso i sistemi di trasmissione potrebbe essere addirittura insufficiente e imprecisa, a causa della loro velocità troppo bassa e assolutamente non confrontabile con la frequenza di campionamento. Il sistema di sincronizzazione che utilizza una linea di trasmissione per sincronizzare una base dei tempi si basa sul concetto che il ritardo nella ricezione dei pacchetti rimanga costante o almeno sia prevedibile. Alcuni sistemi come *NTP* (Network Time Protocol) che viaggiano sulle reti tipo Ethernet, la cui velocità è molto superiore alla precisione richiesta dalla maggior parte dei sistemi di acquisizione utilizzati nella geofisica, potrebbero funzionare discretamente. Altri, utilizzanti una semplice connessione *GSM* potrebbero avere non pochi problemi, in quanto, i pacchetti non viaggiano sempre alla medesima velocità. Con i sistemi tipo *GSM* è molto difficile raggiungere una precisione accettabile. La sincronizzazione via *GSM* è perfetta per stazioni che acquisiscono un campione ogni cinque secondi, ma limita qualsiasi incremento della frequenza di campionamento. Per ovviare ai limiti delle trasmissioni *GSM* e cercare di diminuire l'errore, si sono adottati nel tempo molti accorgimenti. Ridurre la lunghezza dei pacchetti inviati e l'intervallo tra il pacchetto di test e di sincronismo, aiuta a minimizzare l'errore. Lo scopo è di evitare fluttuazioni di velocità nel canale di trasmissione, terminando le operazioni di sincronizzazione nel minor tempo possibile. Nonostante ciò, per migliorare i risultati della sincronizzazione e ovviare a tutti quei problemi derivanti dal sistema fino allora adottato, si è deciso di ricercare un modello alternativo.

Figura 1. Magnetometro GSM90 (versione del 1997) commercializzato dalla Gem Systems, ed ancora installato in alcuni siti sul M.te Etna.

1. L'importanza della sincronizzazione temporale per l'acquisizione

Nel 1999, a circa un anno dall'immissione in servizio del sistema di acquisizione *Mag-Net* [Del Negro et al., 1998], si è sentita l'esigenza di migliorare il tipo di sincronizzazione. Inizialmente in *Mag-Net* era impiegato un sistema di sincronizzazione che utilizzava la linea di trasmissione per acquisire l'orario universale (UTC). Gli strumenti scientifici di allora, dei magnetometri a effetto Overhauser [Overhauser, 1953] prodotti dalla Gem Systems (Figura 1), a differenza di quelli attuali, non avevano un sistema di sincronizzazione interno. Perdevano facilmente l'orario a ogni mancanza di corrente. La soluzione scelta, prevedeva d'immagazzinare l'orario in un sistema esterno, un *RTC* (Real Time Clock) e di fornirlo al magnetometro ogni volta che fosse necessario. Entrambi gli orologi, quello interno al magnetometro e quello del sistema *PC/104*, non erano molto precisi e dovevano essere sincronizzati almeno una volta in un giorno. Questo perché, nonostante i due orologi utilizzino un cristallo di quarzo per generare la frequenza base, questi hanno problemi di deriva, causata dalle oscillazioni termiche ambientali. La frequenza nominale del quarzo è semplicemente alterata dalle oscillazioni termiche ambientali [Wikipedia, Quartz clock]. In siti, come il M.te Etna, afflitti da enormi escursioni termiche, non solo stagionali, ma anche giornaliere (Figura 2), la situazione si complica ulteriormente. Come illustrato nella figura 2, in alcuni siti di misura le variazioni termiche possono raggiungere anche molti gradi centigradi in un solo giorno. Il problema della deriva termica, noto fin dalla prima progettazione di *Mag-Net*, è stato risolto attraverso una sincronizzazione continua. Si è progettato per lo scopo un software che utilizza la linea di trasmissione per acquisire l'orario *UTC*. L'orario era acquisito, attraverso Internet, dall'Istituto Elettrotecnico "Galileo Ferraris", oggi divenuto Istituto Nazionale di Ricerca Metrologica (INRiM). Purtroppo un sistema del genere non è molto preciso poiché risente delle fluttuazioni di velocità sulla linea di trasmissione. La velocità per le trasmissioni *GSM*, soprattutto alla fine del secolo scorso, non era costante e c'erano periodi in cui addirittura si annullava. Le variazioni di velocità non erano dannose per il trasferimento dei dati, ma creavano problemi per il meccanismo di sincronizzazione. Per risolvere questi problemi, si può intervenire compensando le variazioni di velocità, ma ciò nonostante rimane sempre il problema delle avarie di lungo periodo cui è afflitto il sistema di trasmissione. I periodi lunghi di blackout del sistema di trasmissione, generati generalmente in seguito a un'avaria hardware, possono creare una notevole deriva nell'orologio di sistema.

Per ovviare a tutti questi problemi, derivanti dall'utilizzo della linea di trasmissione si è deciso di utilizzare un servizio broadcast sempre disponibile. Le due tecnologie più conosciute allora, erano il *GPS* e il *DCF77*. Nel 1999 il *GPS* (Global Position System) era agli inizi, mentre il *DCF77* collaudato e largamente utilizzato [Wikipedia, DCF77]. Nonostante sia più diffuso, possedendo un raggio di diffusione di soli 2000 km, l'utilizzo del *DCF77* è limitato a una precisa regione geografica. Tutto questo perché si basa su un trasmettitore radio posto in Germania, a Mainflingen (Francoforte). Di contro il *GPS* è a diffusione globale poiché basato su una rete di satelliti artificiali in orbita. La globalità del *GPS* e le premesse promettenti fecero ricadere la scelta sul nuovo sistema, allora ancora inesplorato e nuovissimo, oggi molto diffuso e utilizzato nelle più svariate applicazioni. Oggi, molte applicazioni, come la sincronizzazione temporale, sono basate interamente sul *GPS* che è largamente utilizzato per le sue caratteristiche di stabilità e diffusione. Ovviamente anche il *GPS* potrebbe avere problemi di blackout temporanei, dovuti soprattutto a un segnale scadente, causato dalla presenza di alberi, dal maltempo o in generale dalle caratteristiche geografiche del sito. Particolare non trascurabile è che il *GPS* ha bisogno di una sua antenna specifica per funzionare. L'inserimento di un ulteriore antenna potrebbe esporre maggiormente il sistema alla fulminazione, come già è successo per alcuni siti. Possono aiutare l'adozione di scaricatori di tipo I e II sulla linea, a tutela del buon funzionamento dei sistemi e della protezione dalle extracorrenti da fulmine. I problemi di fulminazione causati da un'antenna esterna possono comunque essere generati anche dai sistemi di trasmissione. Si tratterebbe di un problema nuovo solo per quei sistemi che non sono già dotati di un'antenna esterna addetta alle trasmissioni. Il problema potrebbe essere non trascurabile, per quei siti in cui non esistono già antenne esterne in trasmissione, in quanto la trasmissione è garantita attraverso un meccanismo diverso, più protetto come per esempio la fibra ottica. In questi casi l'antenna *GPS*, unica appendice esterna, potrebbe creare problemi ai sistemi durante episodi di fulminazione, se non sufficientemente protetta. Con le dovute precauzioni, l'inserimento di un *GPS* apporta solo vantaggi e permette soprattutto di aumentare la precisione, riducendo l'occasione di forti periodi di deriva temporale causati dai blackout nelle trasmissioni. In alcune applicazioni di acquisizione ad alta frequenza la precisione diviene importante, e un sistema *GPS* fa la differenza [Greco et al., 2008]. La precisione teorica del tempo *GPS* è di 14ns, nella realtà si possono raggiungere tranquillamente valori tra 25 e 100 ns [Wikipedia, Global_Positioning_System].

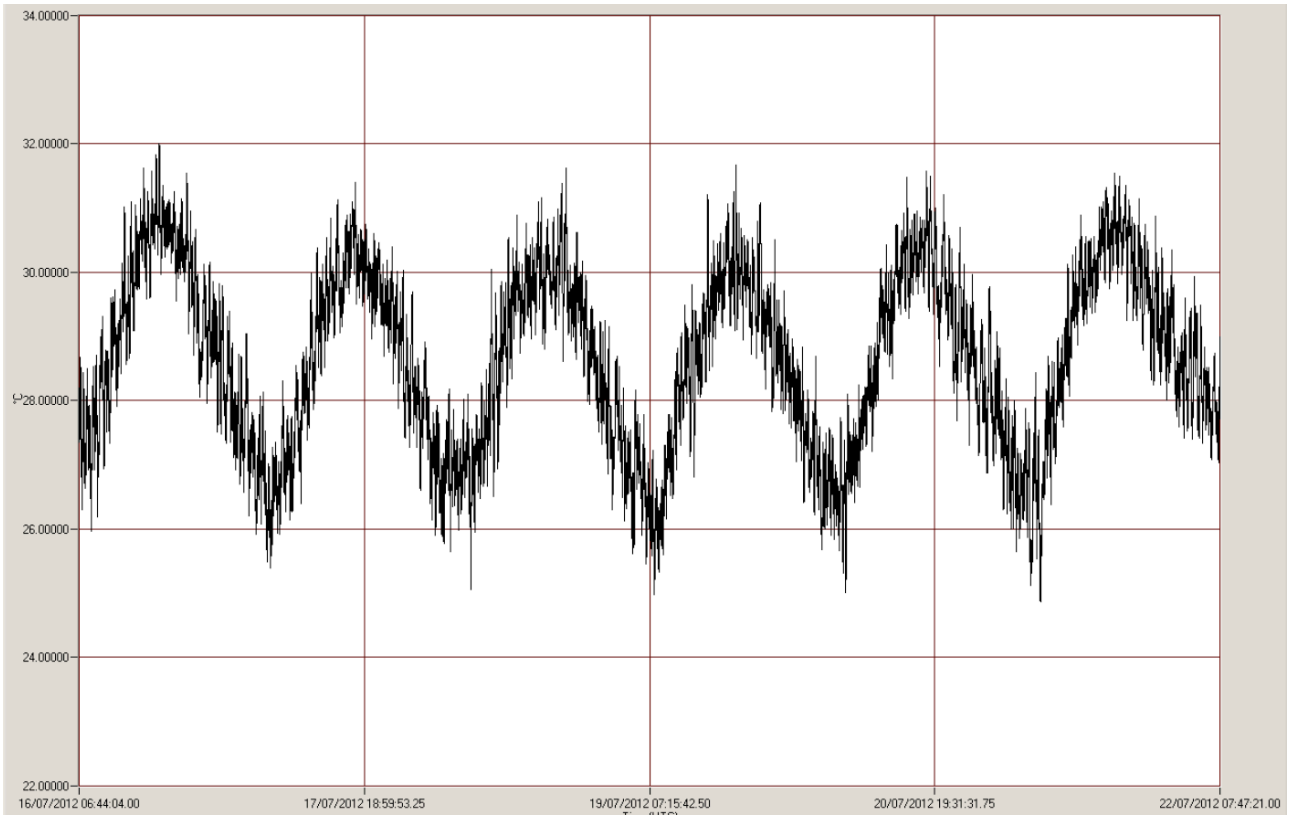

Figura 2. Escursioni termiche in piena estate alla stazione magnetica di riferimento (Cesarò - Nebrodi).

2. Utilizzo dei ricevitori GPS

Inizialmente si è pensato di utilizzare il *DCF77* per la sincronizzazione dell'orario. Il M.te Etna è al limite dell'area di copertura del *DCF77*. Non essendo un sistema universale, il *DCF77*, risolveva solo temporaneamente il problema, limitando l'uso di *Mag-Net* al solo territorio coperto dal trasmettitore di Francoforte. Si è pensato, in alternativa, di sfruttare l'orologio atomico di cui ogni satellite della costellazione *GPS* fa uso.

Un ricevitore *GPS* può ricevere molti satelliti contemporaneamente. Non utilizzandolo per vincolare la posizione nello spazio e usufruendo solamente dell'orario (con una precisione molto limitata) è sufficiente attendere la ricezione anche di un unico satellite. Naturalmente la quantità di satelliti necessari dipende dal tipo di precisione da raggiungere, che va solitamente indicata come scarto quadratico medio rispetto al valore *UTC/USNO* [Wikipedia, Global Positioning System] e quindi dall'applicazione che si vuole implementare. Per tutte quelle applicazioni che non richiedono un'elevata precisione, sarebbe sufficiente attendere solamente un satellite. Per questo è stato inserito nel software la possibilità di specificare il numero di satelliti da attendere. Ciò permette al sistema di velocizzare alquanto le operazioni di sincronizzazione, svincolandosi dall'attendere per un periodo troppo lungo altri satelliti, e avere la sicurezza che la sincronizzazione vada a buon fine. Tutto questo perché, alcune volte, le condizioni sono così particolarmente sfavorevoli che a stento si riesce a ricevere un unico satellite. I ricevitori *GPS* sono solitamente dotati di un'interfaccia seriale. Quelli usati finora della Ashtech (ora acquisita dalla Trimble) sono dotati di una interfaccia seriale asincrona utilizzante livelli logici *TTL* (0-5 volt). I livelli logici *TTL*, non *RS232C*, sono giustificati poiché il ricevitore *GPS* fa parte del mercato *OEM* (Original Equipment Manufacturer) ed è generalmente installato localmente alla strumentazione principale. L'installazione locale permette di escludere eventuali disturbi sulla linea di comunicazione. Inizialmente quel tipo d'interfaccia ha creato qualche problema, poiché non molto diffusa nei sistemi industriali. In seguito, la progettazione della nuova scheda ha permesso di ridurre la complessità circuitale. Questo perché alcuni *chip*, soprattutto *line driver* di conversione dei livelli logici, non erano più necessari per l'interfacciamento del ricevitore. La comunicazione con la scheda avviene attraverso i messaggi *NMEA*, tipica dei ricevitori *GPS*. I messaggi *NMEA* sono comuni nei *GPS* come i comandi *AT Hayes* nei modem. Divenuto oramai uno standard per le periferiche *GPS*, i messaggi *NMEA*, permettono di gestire completamente il ricevitore *GPS*. Esistono

messaggi perfino per spegnerlo o accenderlo. I messaggi *NMEA* sono organizzati in classi e ciascuna classe si occupa di gestire una particolare informazione. Il software di gestione del *GPS* utilizza solo una piccolissima parte dei messaggi com'è mostrato in tabella 1.

Comando	Descrizione
$PASHQ,ZDA	Acquisisce l'orario dal ricevitore GPS.
$PASHQ,SAT	Ritorna le informazioni sui satelliti acquisiti, tra cui il numero di satelliti.
$PASHS,NME,ALL,A,OFF	Arresta il flusso dei messaggi NMEA, inviato dopo lo startup, dal GPS.
$PASHS,PWR,ON	Fa uscire il GPS dallo standby.
$PASHS,PWR,OFF	Manda il GPS in standby.

Tabella 1. Messagi NMEA utilizzati dal codice con relativo formato e significato.

3. Software di gestione dei ricevitori GPS

Il software di gestione di *Mag-Net* [Del Negro et al., 2002] si basa su un modello *client-server*. Nel modello *client-server* esistono due parti separate anche fisicamente che interagiscono tra loro attraverso un canale di comunicazione. La parte dislocata sul territorio, il *server*, fornisce dei servizi ai *client*, presente nel centro di acquisizione dati. Il software di acquisizione (il *server*) è organizzato secondo un modello modulare (paradigma di programmazione a oggetti) che permette di raggiungere livelli di stabilità ottimi [Sicali et al., 2016]. Il software, scritto in *C++*, è composto di un insieme di oggetti (classi), collegate tra loro nel modo più essenziale possibile. La struttura a oggetti permette di limitare l'interazione e l'interdipendenza delle diverse parti. Sono dei mattoncini completamente autosufficienti che, nonostante siano molto complessi e articolati al loro interno, ricevono e rimandano solo poche informazioni essenziali, garantendo il buon funzionamento del sistema. Tutto il software, nessuna parte esclusa, è organizzato in comparti perfettamente stagni. Tutti i comparti permettono al sistema di rimanere stabile, ed agli errori di non propagarsi. Per ciascuna funzionalità del sistema esiste un oggetto: le linee di comunicazione seriali ne hanno uno, così come il modem, il magnetometro e ogni altra periferica presente. Una folta schiera di oggetti, che sono stati collegati elegantemente tra loro come avviene in un semplice e chiaro diagramma di flusso. Per facilitare la lettura e la comprensione del codice a chiunque, la progettazione del software di gestione ha dovuto rispettare alcune regole basilari. Si è cercato di rispettare sempre la stessa interfaccia verso l'esterno e lo stesso tipo di organizzazione interna. I punti di contatti tra i diversi blocchi si riducono a poche unità, e solitamente sono l'inizializzazione e l'accesso al ciclo d'*idle* (*multitask* cooperativo). Rispettando un certo rigore nella progettazione e nella stesura del codice, si può, a fronte di una modifica anche sostanziale, ridurre al minimo l'impatto sulla stabilità del software. La nuova porzione del software, che si occuperà della sincronizzazione delle misure acquisite, rispetta la medesima interfaccia.

La nuova porzione di software, una classe di *C++*, s'integra perfettamente con quello già in uso. Come succede per gli altri dispositivi del sistema *Mag-Net*, il nuovo oggetto crea un'associazione simbolica con il dispositivo (in questo caso il ricevitore *GPS*), che sarà gestito attraverso un'interfaccia semplice e ordinata. L'effetto sarà di nascondere la complessità del ricevitore all'utente e al software di gestione principale. Tutto ciò è molto simile a quello che succede in un moderno *device driver,* in cui i segnali hardware, i protocolli e gli accessi a basso livello sono nascosti all'utilizzatore. La comunicazione con l'esterno è uniformata e il software di gestione vede il ricevitore *GPS* come un semplice orologio. La classe *C++* che permette d'implementare l'oggetto è stata chiamata *GpsLink* e fornisce al restante software, su richiesta, il tempo universale (UTC). Come ogni altra parte del sistema *Mag-Net*, anche il nuovo oggetto ha un funzionamento asincrono. Il funzionamento asincrono, non bloccante, contrapposto al *polling* e al funzionamento sincrono, permette di ricevere l'orario del *GPS* evitando che tutti gli altri servizi vengano ostacolati [Sicali et al., 2016]. Quando la procedura di acquisizione è completata e l'orario disponibile, è fornito allo strumento di misura e a tutti gli altri orologi del sistema. La procedura che gestisce la scheda *GPS*, grazie alla programmazione strutturata e al paradigma a oggetti, è molto lineare e può essere descritta da un semplice diagramma di flusso (Figura 3). Le funzioni che fanno parte dell'interfaccia, quelle più importanti, sono definizioni pubbliche nella dichiarazione nella classe *GpsLink*. La funzione che mantiene in *multitask* il processo del *GPS* è stata chiamata *GpsLink::process*, nome standard usato nel sistema *Mag-Net*.

La sincronizzazione può essere manuale, su richiesta dell'utente, o automatica ogni *'n'* ore. Le richieste automatiche sono configurate durante l'inizializzazione della classe attraverso la funzione *GpsLink::init*. Durante l'inizializzazione può essere fornito semplicemente il numero di sincronizzazioni giornaliere fino a un massimo di 24 oppure specificare direttamente gli orari di sincronizzazione. Una volta inizializzata la classe *GpsLink,* la procedura di sincronizzazione può essere avviata in manuale o in automatico. Una volta avviata la procedura, sarà compito dell'oggetto portarla a termine, gestendo ogni aspetto della comunicazione con il ricevitore.

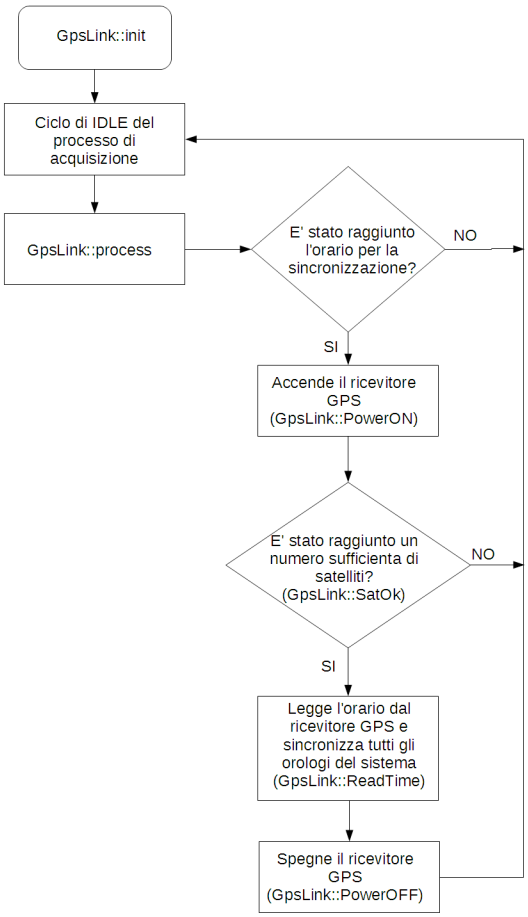

Figura 3. Diagramma di flusso dell'oggetto *GpsLink*.

3.1 Descrizione del codice

Il diagramma di flusso riportato nella figura 3 illustra il funzionamento della classe. La classe, come ripetuto più volte, possiede due gruppi di funzioni: quelle dell'interfaccia e tutte le altre di gestione interna. L'oggetto possiede uno stato che permette di recuperare le operazioni durante il funzionamento asincrono del codice (*multitask* cooperativo). La procedura di sincronizzazione si articola su diverse fasi: accensione del ricevitore, attesa dei satelliti, acquisizione dell'orario e aggiornamento dell'orologio di sistema, chiamato anche *RTC* (Real Time Clock). Durante ciascuna fase il software comunica con il ricevitore attraverso l'uso dei messaggi *NMEA*. Per accendere il ricevitore (uscita dallo *standby*) è spedito un messaggio. Un messaggio è usato per acquisire lo stato dei satelliti, per acquisire l'orario e infine per spegnere il ricevitore, ingresso nello *standby* (Tabella 1).

La funzione *GpsLink::init* permette d'inizializzare la classe durante l'avvio del sistema. La funzione riceve come parametri il numero di sincronizzazioni giornaliere *n* e l'oggetto che gestisce l'interfaccia di comunicazione seriale con il *GPS*. Completata l'inizializzazione della classe, il *token* (controllo) è trasferito al ciclo di *idle* principale. Il *token* viene trasferito alla classe, dal ciclo d'*idle* principale, attraverso la funzione *GpsLink::process.* Questa verifica preliminarmente se l'ora corrente è tra quelle indicate per la sincronizzazione e avvia eventualmente la comunicazione con il ricevitore GPS. Il *token* viene rilasciato comunque dopo ogni singola operazione. Avviata la procedura, poiché raggiunta una delle ore prefissate, è acceso il ricevitore *GPS* attraverso la funzione *GpsLink::PowerON.* La funzione *GpsLink::PowerON* può

semplicemente far uscire il ricevitore dallo standby attraverso l'invio di un messaggio NMEA oppure, abilitando la porzione di codice opportuna, anche accendere fisicamente il dispositivo. Ciò può essere ottenuto utilizzando un interruttore elettronico come quello riportato nella figura 6. Per abilitare l'utilizzo dell'interruttore hardware si deve solo de-commentare la definizione *HARDARE_SWITCH* nell'*header file gps.h* riportato nell'appendice A1. Dopo l'accensione del ricevitore *GPS*, attraverso la funzione *GpsLink::SatOk,* viene richiesto lo stato dei satelliti (almanacco). Attraverso la definizione *MIN_SAT*, riportata nell'*header file gps.h* (Appendice A1), può essere specificato il numero di satelliti richiesto. Raggiunto tale numero, è acquisito l'orario attraverso la funzione *GpsLink::ReadTime* e successivamente impostato l'orologio di sistema. L'aggiornamento dell'orario di sistema avviene attraverso le due *system call DOS* (Disk Operating System) *settime* e *setdate*. Tali funzioni dovranno essere sostituire con altre, nel caso in cui il sistema operativo utilizzato non sia di tipo *DOS*. Come ultima operazione, prima di rilasciare definitivamente il *token,* è mandato in *standby* il ricevitore, ovvero è spento fisicamente se si utilizza un interruttore elettronico come quello mostrato nella figura 6.

3.2 Messagi NMEA

I messaggi *NMEA* (National Marine Electronics Association) sono studiati per lo scambio di dati da particolari dispositivi, utilizzati soprattutto nella nautica. Dispositivi che usano i messaggi *NMEA* possono essere ecoscandagli, sonar, anemometri, girobussole, piloti automatici e ricevitori GPS [Wikipedia, NMEA 0183]. I messaggi viaggiano su una seriale asincrona alla velocità di 4800 *bauds*, con una parola di 8 bit, senza parità e con un *bit* di stop. Non è necessario nessun sistema di *handshake*. Ogni messaggio NMEA è composto di un'intestazione che ne specifica il funzionamento, da una successione di dati e da un codice di controllo (*checksum*). Iniziano con un carattere *$* e finiscono con i due codici *CR* (*carrier return*) e *LF* (*line feed*). Un messaggio *NMEA* tipico è:

$PASHQ,ZDA<CR><LF>

Quando tale messaggio (comando) è spedito al ricevitore GPS, questo risponde con un messaggio simile a questo:

$GPZDA,221554.00,29,09,2004,+00,00*4C<CR><LF>

Indicante la data e l'orario acquisito dal ricevitore GPS.

Gli unici messaggi NMEA che sono utilizzati nel codice, per la completa gestione del ricevitore *GPS*, sono riportati nella tabella 1. Per accendere il ricevitore si può spedire il messaggio "$PASHS,PWR,ON<CR><LF>" mentre per spegnerlo si può utilizzare il messaggio *"$PASHS,PWR,OFF<CR><LF>"*. Si può notare come ogni messaggio *NMEA* inizia sempre con il carattere $ e finisce con il terminatore di linea stile *DOS* (*<CR><LF>*). Iniziare un messaggio con un carattere e finirlo con un terminatore sono un ottimo modo per ridurre l'incidenza dei disturbi di linea. Molto simile ai comandi *AT Hayes* dei modem. Difatti il termine *AT,* con cui iniziano sempre i comandi diretti ai modem, significa *ATtention*, e serve a indicare che solo alcuni caratteri, quelli seguenti, sono dei comandi validi. Tutti i caratteri prima dell'*AT*, o del *$,* e dopo il terminatore di linea vengono semplicemente ignorati.

Il codice dopo l'asterisco rappresenta il *checksum*. Solitamente i messaggi *NMEA* percorrono molte decine di metri attraverso la seriale *RS232C* e possono essere soggetti a errori. Tali errori in alcune applicazioni, come i piloti automatici, sono di vitale importanza. Nel nostro caso il *checksum* è quasi inutile, poiché l'applicazione non è di vitale importanza e il ricevitore comunica attraverso una linea, locale e ben schermata, poco sensibile ai disturbi. Nel codice è stata comunque inserita una funzione di verifica delle stringhe *NMEA*, se l'applicazione lo richiede. La funzione *GpsLink::VerifyString* calcola il *checksum* della stringa e lo confronta con il valore inviato dal ricevitore, presente alla fine del messaggio, dopo l'asterisco.

4. Giusto compromesso tra precisione e affidabilità della sincronizzazione

Chi ha usato un ricevitore *GPS* sa benissimo che acquisire un certo numero di satelliti non è un'operazione istantanea. Per accumulare un discreto numero di satelliti e aumentare la precisione bisogna aspettare diversi minuti. Molte volte, a causa di un segnale molto deteriorato, non si riesce a raggiungere il numero di satelliti desiderato. In questi casi, se l'applicazione è troppo rigida e non ridimensiona la richiesta di satelliti, succede che non riesce ad avviarsi. Per questo, considerando che la frequenza di acquisizione sulle stazioni magnetiche è stata sempre maggiore o uguale a 1 secondo, si è scelto di utilizzare un solo satellite. La precisione raggiunta sarà in questo caso di 100 ns (UTC/USNO ±100ns rms). Tale funzionalità si abilita nel codice lasciando commentata la definizione *MIN_SAT* nell'*header file gps.h* (Appendice A1).

Tale decisione permette, in condizioni sfavorevoli, di procedere comunque alla sincronizzazione. Quando sono richieste maggiori precisioni, poiché la frequenza di campionamento è molto alta, si può specificare un numero minimo di satelliti abilitando la definizione *MIN_SAT*. Sarebbe opportuno, non esagerare con le richieste, poiché in alcune zone particolari, dove il segnale è scadente, potrebbe essere difficile raggiungere il numero di satelliti richiesto. Alcune strumentazioni scientifiche provviste di sistema di sincronizzazione interno, che esagerano con il numero di satelliti da ricercare, non riescono ad acquisire a causa del segnale GPS scadente, aspettando un tempo indefinito che arrivi l'orario corretto. Tutto ciò accade, nonostante abbiano una frequenza di campionamento irrisoria (0.1-10 secondi). Tale strumentazione scientifica si rende inutilizzabile nonostante ci sono le condizioni per farla funzionare.

5. Sistema hardware integrato per l'architettura PC/104

La prima applicazione del software risale al 1999 e si trattava di un aggiornamento al sistema *Mag-Net,* ancora oggi è impiegato nella rete magnetica dell'Etna e dell'isola di Stromboli. L'hardware su cui è stata installata la prima versione del software era un *PC/104*. Non trovando schede *GPS* pronte all'uso, è stato necessario progettare una nuova scheda che rispettasse in pieno le specifiche *PC/104* e potesse essere installata nel sistema senza creare instabilità.

La progettazione è stata necessaria poiché lo standard *PC/104*, creato per le applicazioni industriali, si adatta poco ai sistemi di acquisizione in geofisica. Tale tecnologia è stata scelta principalmente per permettere un eventuale aggiornamento hardware. Tuttavia, nonostante i sistemi basati su standard *PC/104* siano aperti e facilmente estendibili, nella pratica sono dipendenti troppo dalla componentistica di mercato. Il più delle volte non si riesce a trovare ciò di cui si ha bisogno e i sistemi divengono difficilmente modificabili. Si è costretti ad attingere da un paniere in cui sono presenti solo schede, di qualità eccellente, ma per un utilizzo esclusivamente industriale. Tali schede non sempre possiedono caratteristiche compatibili con i sistemi per l'acquisizione geofisica [Sicali et al., 2016]. Lo standard *PC/104,* che doveva garantire una lunga vita al sistema *Mag-Net*, ha generato invece un grosso problema durante l'espansione dello stesso negli anni.

Aggiungere una scheda allo *stack PC/104* non sarebbe stato un problema poiché lo standard prevede un massimo di 6 schede. Lo standard PC/104 prevede specifiche di tipo elettrico, elettronico e informatico. Tra le specifiche meccaniche del formato *PC/104* troviamo le dimensioni della scheda, circa 10x10 cm, e la presenza di due connettori in una determinata posizione. Tali connettori possiedono rispettivamente 64 e 40 contatti, sono passanti per permettere interconnessione delle diverse schede e sono usati per le connessioni al *BUS ISA*. La scheda impiegherà solo il connettore da 64 contatti se la sola porzione di *BUS* utilizzato è quella a 8 bit. Utilizzerà anche la parte di 40 bit se impiegherà anche la porzione a 16 bit del *BUS*. Si dovrà fare molta attenzione alla tipologia di circuiti integrati utilizzati, in quando il bus *PC/104* non supporta le correnti normali di un bus *ISA* tradizionale. Le correnti del *BUS PC/104* sono di soli 4 mA, rispetto ai classici 24 mA di un classico PC.

La versione originale del 1998 di *Mag-Net*, è composta di due sole schede *PC/104*: la motherboard e l'interfaccia di collegamento per la memoria di massa. La memoria di massa inizialmente era in formato *PCMCIA* (Peripheral Component Microchannel Interconnect Architecture), in seguito le schede *PCMCIA* sono state rimpiazzate con le più recenti *CF* (Compact Flash). Solo recentemente sono comparsi sul mercato altri supporti di memoria, come le *SD* (Secure Digital), che possiedono caratteristiche tali da poter essere impiegate in sicurezza nelle applicazioni geofisiche. Le schede di memoria, da impiegare nei sistemi di acquisizione, devono avere soprattutto un intervallo di temperatura esteso, normalmente -40 +85 °C [Sicali et al., 2016]. La scheda madre possedeva solo poche periferiche, di cui due seriali *RS232C* e una porta parallela bidirezionale. Per alcune applicazioni industriali sarebbero state più che sufficienti, ma per un sistema di acquisizione, sicuramente poche. Con due sole porte seriali e una parallela si gestiscono a malapena i dispositivi esterni necessari al sistema per funzionare: un modem, un magnetometro e un tastierino/display. Con tutte le porte d'*input-output* (I/O) già impegnate, inserire un dispositivo per la sincronizzazione dei dati non è stato semplice. La progettazione di una nuova scheda *PC/104* indipendente si è resa necessaria. Inserire una nuova scheda allo *stack PC/104* è stato molto più semplice che utilizzare le porte di comunicazione presenti attraverso tecniche di multiplexing. Considerando che il ricevitore *OEM* impiegato utilizzava livelli *TTL* per comunicare, sarebbe stato necessario, comunque, inserire dell'elettronica di contorno per adattare i segnali *RS232C* (±12 V) delle classiche porte seriali. L'inserimento di una nuova scheda nel sistema non avrebbe comportato problemi in termini di stabilità, a patto di rispettare una rigorosa progettazione hardware e software. Rispettato pienamente lo standard *PC/104* e il modello software adottato da *Mag-Net* si sono evitati problemi seri d'instabilità nel sistema.

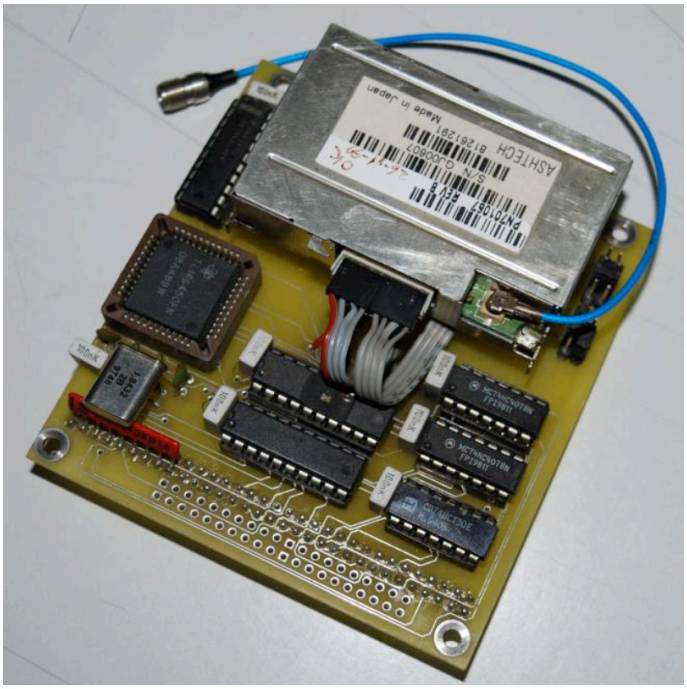

Figura 4. Scheda ricevitore *GPS* in formato *PC/104*. La scheda ha dimensioni (come da specifiche *PC/104*) di circa 10x10 cm.

5.1 Descrizione generale dell'elettronica di supporto al ricevitore GPS

In commercio esistevano molte schede in formato *PC/104* che eseguivano le più diverse funzionalità. C'erano schede impiegabili per la comunicazione, l'archiviazione e la visualizzazione, ma nessuna che supportasse un ricevitore *GPS*. Si è deciso quindi, di realizzare una nuova scheda e di affrontare i problemi derivanti da una progettazione, allo scopo di ammodernare il sistema. Aggiungere una periferica al sistema, collegandosi al *BUS PC/104*, ancora oggi, resta il modo più invasivo possibile di operare, nonostante il sistema *PC/104* fosse stato scelto per essere esteso in modo semplice, poiché aperto e modulare. Nella realtà dei fatti si stava comportando alla pari di una *blackbox* [Sicali et al., 2016], poiché non era disponibile sul mercato una scheda apposita. L'aggiornamento è stato eseguito utilizzando periferiche connesse direttamente al *BUS* di sistema, operazione ancora oggi piuttosto complessa. Tale operazione può rendere un sistema instabile se non eseguita correttamente. La complessità risiede nel dover soddisfare determinate specifiche e caratteristiche, e non è un'operazione immediata come collegare un dispositivo a una porta di comunicazione esterna, seriale o parallela che sia. L'idea d'inserire un dispositivo nel cuore del sistema, il bus *ISA*, era piuttosto radicale e difficile da assimilare. I *local bus* [Wikipedia, Local bus], come possono essere stati i *BUS ISA,* per i sistemi di fine anni '80 e inizio '90, sono il centro di tutto il sistema. Se mal gestita, l'operazione poteva diventare fallimentare. D'altra parte era un'operazione necessaria poiché non erano disponibili abbastanza porte di comunicazione, già pronte all'uso, per collegare il ricevitore *GPS*. Oggi basta una porta *USB* per inserire molte periferiche in parallelo, allora il canale di comunicazione più usato era la *RS232C* che poteva gestire una comunicazione per volta. Standard come *RS485*, molto simile all'attuale *USB*, non erano molto diffusi, soprattutto nei prodotti *consumer*.

Il ricevitore *GPS* utilizza una seriale asincrona, molto simile alla *RS232C*, con livelli *TTL*. Per interfacciare il *BUS PC/104* al ricevitore bisogna interporre un dispositivo che trasforma i dati paralleli del *BUS ISA* in seriali. Il dispositivo che si occupa di ciò è l'*UART* (Universal Asynchronous Receiver/Transmitter). Per aiutare l'*UART* nello scopo, sono stati aggiunti pochi altri dispositivi, soprattutto *line driver*, usati per l'interfacciamento con il *BUS*. Tutta l'elettronica impiegata è in tecnologia *MOS,* rigorosamente a basso consumo per non influenzare il bilancio energetico. Il risultato, molto compatto come tutte le schede *PC/104*, è mostrato nella figura 4, mentre lo schema a blocchi del sistema hardware è riportato nella figura 5.

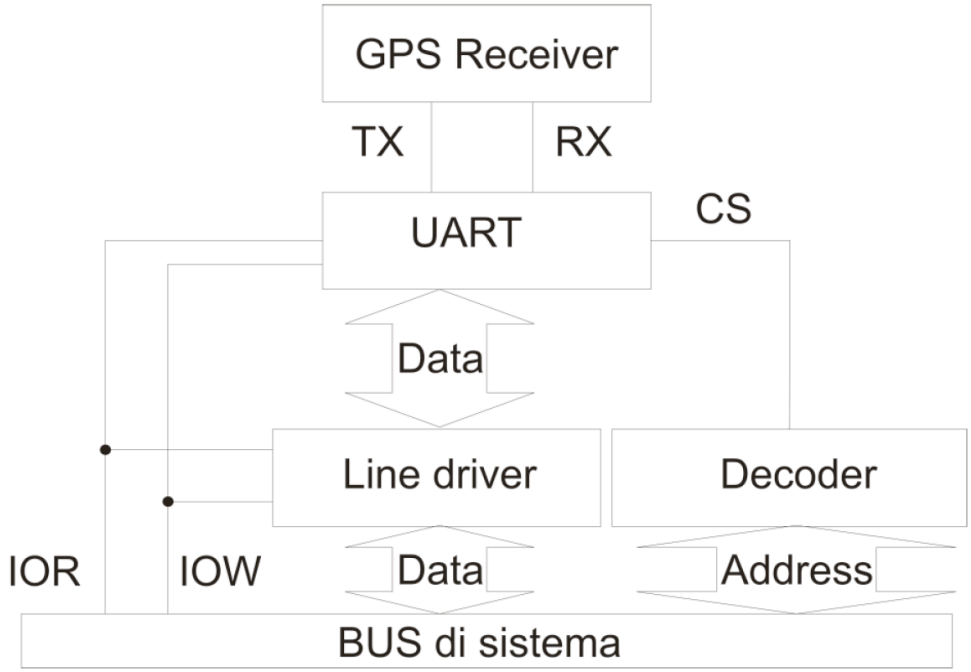

Figura 5. Schema a blocchi di un sistema d'interfacciamento per architetture a *BUS* tipo Von Neumann della scheda *PC/104*.

Come *UART* si è utilizzato il classico 16C450. Come tutti i circuiti integrati con logica *TTL*, accetta livelli di tensione tra 0 e 5 volt [Wikipedia, Transistor-transitor logic]. L'*UART*, come molti dispositivi impiegati nei sistemi con architettura di von Neumann [Wikipedia, Architettura di von Neumann], possiede un *BUS* di dati, un *BUS* degli indirizzi e un *chip select* (CS). Un *UART* classico, come quello impiegato, non funziona mai come *BUS* master e le comunicazioni sono sempre iniziate e finite dalla *CPU*. La comunicazione avviene utilizzando il bus di sistema, dopo che il chip è stato abilitato attraverso il *CS* dall'elettronica d'indirizzamento, su richiesta della *CPU*. L'*UART* possiede un *BUS* per i dati di soli 8 bit, quindi verrà usata solo una parte del bus *ISA* che lo standard *PC/104* prevede. L'*UART* non può essere connesso direttamente al *BUS*, soprattutto per evitare di sovraccaricarlo. Lo standard *PC/104* prevede che le correnti siano di soli 4 mA, rispetto ai classici 24 mA di un *personal computer*. Correnti superiori potrebbero creare instabilità o addirittura danneggiare il *BUS*. Per evitare problemi conviene inserire un *buffer* per il trasferimento dei dati. Tale buffer sarebbe comunque necessario utilizzando dispositivi non realizzati in logica *three-state* [Wikipedia, Three state logic]. Il buffer utilizzato è un dispositivo bidirezionale e permette che i dati fluiscano in entrambe le direzioni. Secondo le richieste della *CPU* e rispettando lo stato dei segnali *IOR* e *IOW* del bus *ISA* [Wikipedia, Industry Standard Architecture] i dati possono fluire in entrambe le direzioni.

5.2 Accesso alle periferiche nell'architettura ISA/PC104

Come anticipato, l'*UART* non può funzionare da bus master ma solo da slave: è la *CPU* a fare da arbitro nelle comunicazioni. Il *BUS ISA* prevede che anche altri dispositivi, oltre la *CPU*, funzionino da bus master. Questi dispositivi, che funzionano da *bus master*, possono iniziare le comunicazioni, occupando il *BUS* e traferendo dati a un altro dispositivo. Un esempio tipico di dispositivi che funzionano da bus master sono quelli che utilizzano il *DMA* (Direct Memory Access). Questi dispositivi trasferiscono dati, indipendentemente dalla *CPU*, direttamente alla memoria centrale (*RAM*). Per tutti i dispositivi che non funzionano da bus master, ma che hanno la necessità di trasferire urgentemente dati, esiste il meccanismo degli *interrupt*. L'uso degli *interrupt* è consigliato, non solo nelle comunicazioni di urgenza, ma anche per evitare di perdere cicli di *CPU* in attività di *polling*. *L'interrupt* è simile come funzionalità a un campanello, che il dispositivo può utilizzare quando è pronto, per richiedere alla *CPU* un trasferimento di dati. Indipendentemente dalle modalità di accesso al BUS, ogni dispositivo, master o slave che sia, riceverà un indirizzo che lo identificherà univocamente nel sistema. Esistono due modi per identificare un dispositivo

(mappatura): utilizzando le porte d'*input-output* (I/O) o attraverso le locazioni di memoria centrale. Un dispositivo che utilizza una mappatura con porte può essere interrogato solo attraverso le due sole istruzioni, *in* e *out*. Un dispositivo mappato in memoria può essere interrogato attraverso tutte le altre istruzioni. I due meccanismi sono totalmente equivalenti dal punto di vista dell'hardware, la differenza è solo formale. Ciascun dispositivo, e quindi anche l'*UART*, riceverà un indirizzo base, ovvero un intervallo di porte d'*input-output*. L'indirizzo nel caso del bus *ISA* è di 24 bit. Oltre all'indirizzo, a ciascun dispositivo può essere assegnato un *interrupt* hardware. L'*interrupt* hardware che prende il nome di *IRQ* (Interrupt Request Query) viene utilizzato per eseguire operazioni in modo asincrono, evitando la tecnica del polling e soprattutto la perdita di dati. L'*IRQ* è usato dall'*UART* ogni volta che c'è la necessità di trasferire urgentemente un dato.

Nello standard *ISA*, cui il *PC/104* fa riferimento, per le porte seriali sono riservati gli *interrupt* 3 e 4. Gli *UART* sono solitamente mappati nello spazio delle porte e sono interrogabili attraverso le istruzioni *in* e *out*. Gli indirizzi base assegnati agli *UART* sono 2E8h, 3E8h, 2F8h, 3F8h. Un *UART* comunque non ha l'obbligo di occupare tali risorse, ciò nonostante è vivamente consigliato usarle, per rendere un sistema compatibile con lo *standard ISA*. Inoltre si eviteranno malfunzionamenti dovuti a eventuali sovrapposizioni con altre risorse del sistema.

5.3 Sistemi di accesso sincrono (polling) e asincrono attraverso gli interrupt

Il meccanismo di accesso alla scheda è molto semplice: all'arrivo di un dato o al completamento di una trasmissione l'*UART* asserisce l'*interrupt* assegnatogli e si pone in attesa. In seguito la *CPU* lo contatta per prelevare o trasferire il dato. L'utilizzo degli interrupt non è una scelta obbligata ma consigliata. Per accedere a un dispositivo che non funziona da bus master esistono solo due modi: il *polling* e gli *interrupt*. Durante le operazioni in *polling* la *CPU* ha il compito d'interrogare continuamente il dispositivo. La tecnica del *polling* potenzialmente può creare perdita dei dati. Oggi le *CPU* hanno oramai velocità altissime, quindi è impensabile che una trasmissione a 9600 baud possa creare una perdita di dati. In effetti, i problemi iniziano se la velocità di trasmissione inizia a essere confrontabile con la frequenza del *polling*, ovvero diviene più alta. Nel caso in cui la frequenza del *polling* sia più alta di quella di trasmissione si invece una ha perdita di risorse, questo perché la *CPU* impiega molti cicli per il mantenere il *polling*. Riassumendo: quando si usa il *polling*, la frequenza di scansione non può essere più bassa o confrontabile a quella di trasmissione poiché si ha una perdita di dati. Non può essere neanche troppo alta poiché si ha una perdita di risorse, ne segue che non conviene assolutamente usare la tecnica del *polling*, quindi l'uso degli *interrupt* è vivamente consigliato, soprattutto per tutti quei sistemi che non si possono permettere una perdita inutile di dati e di risorse [Sicali et al., 2016]. Il *polling* si potrà usare solo in casi molto rari in cui le operazioni si esauriscono velocemente. Fanno parte di tale insieme tutte le operazioni eseguite sui dispositivi che non prevedono l'uso degli *interrupt*.

5.4 Applicazione delle specifiche ISA/PC104

Un sistema come il *PC/104* possiede solitamente un *PIC* (Programmable Interrupt Controller) che permette di estendere l'unica linea d'*interrupt* a disposizione della *CPU*. In un sistema con *PIC*, grazie alla condivisione, si possono avere oltre 8 linee d'*interrupt*, quindi più che sufficienti per qualsiasi applicazione. Per evitare comunque problemi di sovrapposizione tra gli *interrupt* di diversi dispositivi, bisogna osservare delle regole. Lo standard prevede che gli *UART* usino gli *interrupt* 3 e 4. Ciò nonostante, se la situazione lo richiede, si possono usare anche altre linee d'*interrupt*. Per questa eventualità si è deciso d'inserire la possibilità per la scheda di usare anche gli *interrupt* 6 e 7.

La configurazione in *Mag-Net*, della scheda, prevede l'uso dell'indirizzo base 3E8h e l'utilizzo dell'*IRQ4*. Tali risorse, assegnate agli *UART* nel modello *ISA*, non creano conflitti con le altre 2 seriali della motherboard *PC/104*. Attraverso tali risorse il software di gestione può accedere all'*UART* e al ricevitore *GPS*. D'altra parte il sistema hardware del *PC/104* la riconosce come periferica standard e può essere usata anche da altre applicazioni senza la necessità di un *device driver* aggiuntivo. La configurazione della scheda può essere comunque cambiata attraverso dei *jumper*. Il software potrà far riferimento alla scheda attraverso una propria configurazione o leggendo l'area dati *BIOS*, che ogni sistema *ISA* possiede, all'indirizzo di memoria 0000:0400.

Nel realizzare la scheda si sono dovute rispettare anche delle specifiche meccaniche, oltre a quelle elettroniche e informatiche. Finita la progettazione elettronica e informatica, il risultato doveva essere ospitato su una scheda (circuito stampato) conforme alle specifiche *PC/104*. Lo standard *PC/104* prevede che le schede abbiano delle dimensioni opportune, circa 10x10 cm, [Wikipedia, PC/104] e che a una determinata posizione ci siano due connettori di 64+40 contatti. In effetti, per la scheda ne basta solamente uno, poiché non è utilizzata la porzione del *BUS* a 16 bit. Non avendo la necessità di propagare segnali alle schede superiori il connettore può anche essere non passante. Su una scheda non terminale, per lo *stack* di schede *PC/104*, i connettori dovranno invece essere passanti.

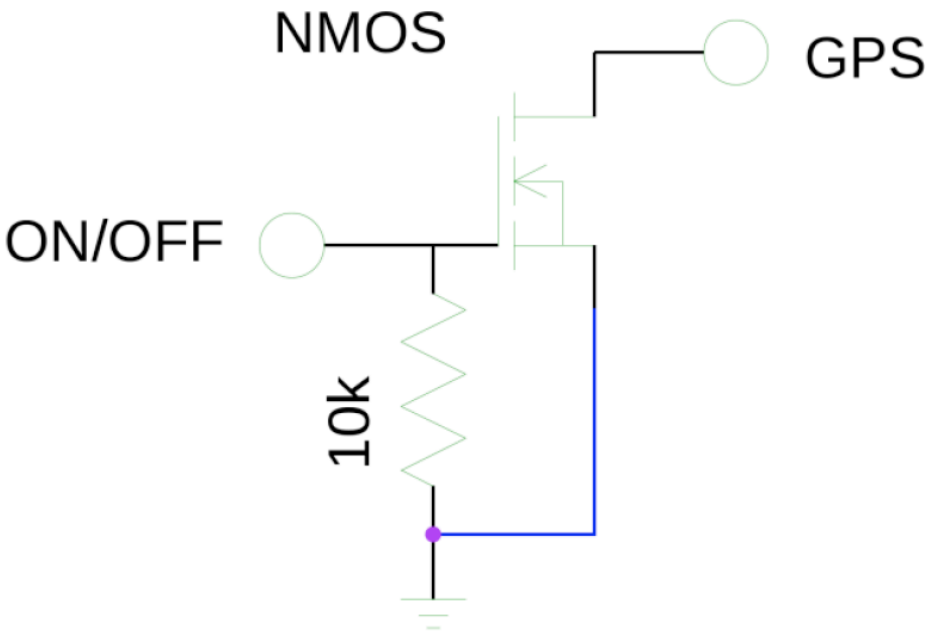

Figura 6. Schema elettrico interruttore elettronico utilizzato per resettare il ricevitore *GPS*.

6. Impatto dell'evoluzione dell'hardware sul software

Negli anni il ricevitore *GPS* impiegato è cambiato più volte, perché divenuto obsoleto. Grazie all'uso dei messaggi *NMEA* il software è stato riutilizzato con ciascun nuovo ricevitore. Solo recentemente, per evitare problemi di stallo e ridurre i consumi medi, si è dotato l'hardware di un interruttore elettronico [Sicali et al., 2016], il cui schema elettrico è riportato nella figura 6. Negli anni non si sono introdotte sostanziali modifiche al software.

La scheda *PC/104*, progettata molti anni or sono, è ancora utilizzata. Solo grazie ai nuovissimi *PC/104*, utilizzanti una *CPU* Vortex x86, la scheda è stata completamente rimpiazzata. I nuovi *PC/104* hanno 6 seriali *on board* e possono gestire molte periferiche. L'interfaccia seriale esterna rimane comunque utilizzabile per tutte quelle architetture che non hanno abbastanza seriali a disposizione sulla scheda madre. Il software è stato utilizzato negli anni, nonostante l'hardware sia mutato ampiamente, non subendo grandi variazioni da quando fu realizzato nel 1999. Il software con i dovuti accorgimenti è stato infine utilizzato anche nell'implementazione del sistema *IDRA* (Interactive Device for Remote Acquisition) [Sicali et al., 2016].

Conclusioni

La sincronizzazione dei dati acquisiti è importante per l'analisi degli stessi. Il *GPS* come strumento di sincronizzazione si è rivelato ottimo e universale. La programmazione a oggetti e l'uso dei messaggi *NMEA* hanno permesso di contenere le modifiche al software, nonostante l'hardware si evolvesse. La stabilità si è mantenuta a livelli alti e gli effetti di eventuali errori, introdotti nel software, limitati.

Il software è stato studiato per essere pienamente integrato nel sistema *Mag-Net*, ma è possibile riutilizzarlo per altri scopi e ambiti. L'utilizzo del *C++*, oramai diffusissimo, permette di ricompilare facilmente il codice sorgente in diversi ambienti. Ciò lo rende, allo stesso tempo, indipendente dall'hardware e impiegabile nella maggior parte dei sistemi industriali e di acquisizione.

Ringraziamenti

Ancora una volta volevamo ringraziare tutta la Segreteria di Redazione del CEN che si è dimostrata molto veloce ed efficiente. Personalmente ringraziamo la dott.ssa Rossella Celi per la cordialità e la disponibilità nel tenere i rapporti con gli autori e l'Editor, dott. Umberto Sciacca, per i suggerimenti costruttivi.

Bibliografia

Del Negro C., Di Bella A., Ferrucci F., Napoli R., Sicali A., (1998). *Automated System for Magnetic Surveillance of Active Volcanoes.* In: Final Report Tekvolc: Technique and Method Innovation in Geophysical Research, Monitoring and Early Warning at Active Volcanoes. Commission of European Communities Environment Programme, Contract ENV4 CT95 0251.

Del Negro C., Di Bella A., Napoli R., Sicali A., (1999). *Development of an automated console prototype for control of remote magnetic sensors In Interim Report Tomave: Electromagnetic and potential field integrated tomographies applied to volcanic environments.* Commission of European Communities Environment Programme, Contract ENV4 CT98 0697.

Del Negro C., Napoli R., Sicali A., (2002). *Automated system for magnetic monitoring of active volcanoes*. Bull. Volcanol., 64, 94-99.

Greco F., Carmisciano C., Del Negro C., Loretti I., Sicali A., Stefanelli P., (2008). *Seismic-induced accelerations detected by two coupled gravity meters in continuous recording with a high sampling rate at Etna volcano.* Annals of Geophysics, vol.51, n.1.

Sicali A., Amantia A., Cappucci, P., (2016). *Linee guida e criticità nella progettazione di sistemi per l'acquisizione di dati geofisici in prossimità di vulcani attivi.* Rapporti Tecnici INGV n°. 347, ISSN 2039-7941.

Overhauser A.W., (1953). *Polarization of Nuclei in Metals*. Phys. Rev. 92, 411.

Sitografia

Wikipedia, *Global Positioning System*, https://en.wikipedia.org/wiki/Global_Positioning_System#Accuracy
Wikipedia, *NMEA 0183*, https://en.wikipedia.org/wiki/NMEA_0183
Wikipedia, *Quartz clock*, https://en.wikipedia.org/wiki/Quartz_clock
Wikipedia, *DCF77*, https://en.wikipedia.org/wiki/DCF77
Wikipedia, *Transistor-transistor logic*, https://en.wikipedia.org/wiki/Transistor%E2%80%93transistor_logic
Wikipedia, *Three state logic*, https://en.wikipedia.org/wiki/Three-state_logic
Wikipedia, *Architettura di von Neumann*, https://it.wikipedia.org/wiki/Architettura_di_von_Neumann
Wikipedia, *Industry Standard Architecture*, https://it.wikipedia.org/wiki/Industry_Standard_Architecture
Wikipedia, *PC/104*, https://en.wikipedia.org/wiki/PC/104
Wikipedia, *Local bus*, https://en.wikipedia.org/wiki/Local_bus

Appendice A1. Codice sorgente software di gestione della scheda GPS: gps.h

```
#ifndef _GPS_H
#define _GPS_H

#include "serial.h"
#include <time.h>

#define NOSTRING       3
#define NOSAT          4
#define GPS_POWEROFF   5

//#define MIN_SAT      4
//#define HARDWARE_SWITCH

extern short int GetDays(struct date d);

class GpsLink {
                SerialPort *com;
                unsigned short times;
                char hour[25];
                BOOL timesignal;
                SimbolikTimer Tsat;
                enum {maxsat=15};
                enum {sattimeout=30};
                enum {maxrecord=1024};
                int nsat;
                int abssat;
                char *record[maxrecord];
                char GpsString[5000];

                void SendString(char *s);
                BOOL ConvertTime(char *buffer,struct time *t,struct date *d);
                BOOL ConvertSat(char *buffer);
                BOOL VerifyString(char *s);
        public:
                GpsLink() : Tsat(10,YES) {}
                void ReadTime(struct date *d,struct time *t);
                BOOL Init(int n,SerialPort *c,int *hours=NULL);
                BOOL processor(void);
                void PowerON(void);
                void PowerOFF(void);
                void StopNME(void);
                BOOL SatOk(void);
                #ifdef _HARDWARE_SWITCH
                void Reset(void);
                void TurnON(void);
                void TurnOFF(void);
                #endif
              };

#endif
```

Appendice A2. Codice sorgente software di gestione della scheda GPS: gps.cpp

```
#include <windows.h>
#include "serial.h"
#include "gps.h"
#include "service.h"
#include "hardware.h"
#include <conio.h>

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                      Funzioni di inizializzazione                  //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Inizializza la Classe
*/
BOOL GpsLink::Init(int n,SerialPort *c,int *hours)
{
        int i=0;
        int h=0;

        times=0;
        com=c;
        timesignal=0;
        nsat=0;
        abssat=0;
        if (n < 1 || n > 24) return FALSE;

        if (hours) {
                for (i=0;i<n;i++) hour[i]=hours[i];
                return TRUE;
        }

        do {
                if (h > 23) break;
                hour[i]=h;
                h += (24/n);
                i++;
                hour[i]=-1;
            } while(i < n);

       i=0;
       while(hour[i] != -1) {
                        printf("[h=%d]",hour[i]);
                        i++;
                             }

        return TRUE;
}

/*
        Ferma il flusso dei messaggi NMEA che alcuni ricevitori
        Spediscono in continuo subito dopo l'accensione
*/
void GpsLink::StopNME(void)
{
        SimbolikTimer t(10,YES);
        unsigned char c;
```

```
        printf("[STOP NME]");
        while (t != YES);
        SendString("$PASHS,NME,ALL,A,OFF");
        while ((*com > c) != NO);
}


////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                 Funzioni di multitask senza prelazione             //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Questa funzione implementa un multitask di tipo cooperative.
        Deve essere richiamata nel ciclo principale del processo
*/
BOOL GpsLink::processor(void)
{
        struct date d;
        struct time t;

        gettime(&t);

        if (hour[times] == -1)
                if (hour[times-1] == t.ti_hour) return NO;
                                                else
                                                {
                        struct time t;
                        struct date d;
                        getdate(&d);
                        gettime(&t);
                        times=0;
                        printf("[prossima sincronizzazione %d",hour[times]);
                                                }

        if (t.ti_hour == hour[times]) {
                if (timesignal == NO) {
                        nsat=0;
                        abssat=0;
                        PowerON();
                        timesignal=YES;
                        return NO;
                }
                else
                {
                        switch(SatOk()) {
                                case NO:
                                        return NO;
                                case NOSAT:
                                        times++;
                                        timesignal=NO;
                                        return GPS_POWEROFF;
                        }
                        timesignal=NO;
                }
                ReadTime(&d,&t);
                settime(&t);
                setdate(&d);
```

```
                times++;
                PowerOFF();
                return YES;
        }
        return NO;
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//              Funzioni di accesso all'orario del ricevitore         //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Estrae l'orario dalle informazioni ricevute dal ricevitore GPS
*/
BOOL GpsLink::ConvertTime(char *buffer,struct time *t,struct date *d)
{
        int i,j=0;
        unsigned char *p=buffer;
        unsigned char pnull[]="";

        if (!VerifyString(buffer)) return FALSE;

        for (i=0;i<maxrecord;i++) {
                                record[i]=pnull;
                                }
        i=0;

        printf(buffer);

        while(buffer[i] != 0) {
                                if (buffer[i] == ',') {
                                                buffer[i]=0;
                                                record[j]=p;
                                                j++;
                                                p=buffer+i+1;
                                                if (j == (maxrecord-1)) break;
                                                      }
                                i++;
                               }
        record[j]=p;
        printf("[%s]",record[0]);
        if (strcmp(record[0],"GPZDA") != 0) {
                                        printf("[err]");
                                        return FALSE;
                                            }
        printf("ok string");

        t->ti_min=((record[1][2]-'0')*10)+(record[1][3]-'0');
        t->ti_hour=((record[1][0]-'0')*10)+(record[1][1]-'0');
        t->ti_hund=((record[1][7]-'0')*10)+(record[1][8]-'0');
        t->ti_sec=((record[1][4]-'0')*10)+(record[1][5]-'0');
        d->da_year=((record[4][0]-'0')*1000)+
                   ((record[4][1]-'0')*100) +
                   ((record[4][2]-'0')*10)  +
                   (record[4][3]-'0');
        d->da_mon =((record[3][0]-'0')*10)+
                   (record[3][1]-'0');
        d->da_day =((record[2][0]-'0')*10)+
```

```
                    (record[2][1]-'0');

        printf("%2.2hd/%2.2hd/%4.4hd %2.2hd:%2.2hd:%2.2hd-%2.2hd",
                        d->da_day,d->da_mon,d->da_year,
                        t->ti_hour,t->ti_min,t->ti_sec,t->ti_hund);


        if (t->ti_hour > 23) return FALSE;
        if (t->ti_sec > 59) return FALSE;
        if (t->ti_min > 59) return FALSE;
        if (d->da_mon < 1 || d->da_mon > 12) return FALSE;
        if (d->da_day < 1 || d->da_day > GetDays(*d)) return FALSE;

        printf("[%d]",t->ti_sec);
        return TRUE;
}

/*
        Richiede le informazioni sull'orario al ricevitore GPS
*/
void GpsLink::ReadTime(struct date *d,struct time *t)
{
        int i=0;
        unsigned char c;
        unsigned short k;

        SimbolikTimer t1(5,TRUE);

        while(TRUE) {
        printf("\n[re send in]\n");
        SendString("$PASHQ,ZDA");
        printf("\nre send\n");
        t1.Restart();
        while(TRUE) {
                k=t1;
                if (k == 1) break;
                if (*com > c) {
                        printf("%c",c);
                        if (c == '$')   { i=0; }
                                        else
                                        { GpsString[i]=c; i++; }
                        if (c == 0x0D) {
                                GpsString[i-1]=0;
                                printf("scan\n");
                                if (ConvertTime(GpsString,t,d) == TRUE) return;
                                break;
                        }
                }
        }
   }
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//        Funzioni di accesso alle informazioni sui satellitit        //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////


/*
```

```
        Estrae le informazioni riguardanti i satelliti
*/
BOOL GpsLink::ConvertSat(char *buffer)
{
        int i,j=0;
        int nsat;
        unsigned char *p=buffer;
        unsigned char pnull[]="";

        if (!VerifyString(buffer)) return FALSE;

        for (i=0;i<maxrecord;i++) record[i]=pnull;

        i=0;
        printf(buffer);
        printlcd(buffer,abssat);

        while(buffer[i] != 0) {
                                if (buffer[i] == ',') {
                                                buffer[i]=0;
                                                record[j]=p;
                                                j++;
                                                p=buffer+i+1;
                                                if (j == (maxrecord-1)) break;
                                                      }
                                i++;
                               }
        record[j]=p;
        printf("\n<%s %s %d>\n\n\n",record[1],record[2],strlen(record[2]));
        if (strcmp(record[1],"SAT")) return NOSTRING;
        #ifndef MIN_SAT
                if (strcmp(record[2],"") &&
                    strcmp(record[2],"*1E") &&
                    strcmp(record[2],"00*1E")) return TRUE;
        #else
                if (sscanf(record[2],"%d",&nsat) != 1) return FALSE;
                if (nsat < MIN_SAT) return FALSE;
        #endif
        return FALSE;
}

/*
        Richiede informazioni sui satelliti
*/
BOOL GpsLink::SatOk(void)
{
        int i=0;
        unsigned char c;
        int r;
        SimbolikTimer t1(2,YES);
        unsigned short k;

        if (Tsat == 0) return NO;

        while(TRUE) {
        printf("\nre send\n");
        Tsat.Restart();
        SendString("$PASHQ,SAT");
        if (abssat >= sattimeout) {
                                abssat=0;
                                nsat=0;
                                return NOSAT;
```

```
                                }
        if (nsat >= maxsat) nsat=0;
        printf("[abs=%d]",abssat);
        nsat++;
        abssat++;
        t1.Restart();
        #ifdef HARDWARE_SWITCH
        int reset=TRUE;
        #endif
        while(TRUE) {
        k=t1;
        if (k == 1) break;
        if (*com > c) {
                        printf("%c",c);
                        if (c == '$') { i=0; }
                                      else
                                      { GpsString[i]=c; i++; }
                        if (c == 0x0D) {
                                       #ifdef HARDWARE_SWITCH
                                       reset=FALSE;
                                       #endif
                                       GpsString[i-1]=0;
                                       printf("scan\n");
                                       r=ConvertSat(GpsString);
                                       if (r == TRUE) {
                                                      nsat=0;
                                                      abssat=0;
                                                     }
                                       if (r != NOSTRING) {
                                                      printf("[ret %d]",r);
                                                      return r;
                                                         }
                                       break;
                                      }
                     }
                       }
        #ifdef HARDWARE_SWITCH
        if (reset) Reset();
        #endif
                    }
        return NO;
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//            Funzioni di Accensione/Spegnimento (o stadby)           //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Fa uscire il ricevitore GPS dallo standby
*/
void GpsLink::PowerON(void)
{
        char s[64];

        SimbolikTimer t(5,YES);
        printf("[PON]");
        #ifdef HARDWARE_SWITCH
        TurnON();
```

```
        #else
        SendString("$PASHS,PWR,ON");
        #endif
        printf("[PRST]");
        SendString("$PASHS,RST");
        StopNME();
}

/*
        Pone in standby il ricevitore GPS
*/
void GpsLink::PowerOFF(void)
{
        printf("[POFF]");
        #ifdef HARDWARE_SWITCH
        TurnOFF();
        #else
        SendString("$PASHS,PWR,OFF");
        #endif
 }

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                      Funzioni di Supporto                          //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Spedisce una stringa al ricevitore GPS,
        necessaria solo se le API del sistema software
        non prevede già una tale funzione
*/
void GpsLink::SendString(char *s)
{
        unsigned short i=0;

        while(s[i] != 0) {
                                *com < s[i];
                                i++;
                        }
        *com < 0x0D;
        *com < 0x0A;
}

/*
        Verifica la bontà della stringa ricevuta attraverso il checksum
*/
BOOL GpsLink::VerifyString(char *s)
{
        unsigned short n=0;
        unsigned short i=0;
        unsigned short cs=0;
        char tmp[3];

        while(s[i] && s[i] != '*') i++;

        if (!s[i]) return false;

        n=i;
```

```
        for (i=0;i<n;i++) cs[i]  ^= s[i];

        sprintf(tmp,"%2.2hx",cs);
        return !strcmp(tmp,s+i+1);
}

#ifdef HARDWARE_SWITCH
/*
        Spegne fisicamente il ricevitore togliendo l'alimentazione
*/
void GpsLink::TurnOFF(void)
{
        unsigned short int base;
        unsigned char c;

        base=peek(0x0040,0x0008);


        printf_LCD("GPS TURN OFF");


        c=inport(base+2);
        c |= 0x02;
        outportb(base+2,c);
}

/*
        Accende fisicamente il ricevitore riallacciando l'alimentazione
*/
void GpsLink::TurnON(void)
{
        unsigned short int base;
        unsigned char c;

        base=peek(0x0040,0x0008);


        printf_LCD("GPS TURN ON");


        c=inport(base+2);
        c &= ~0x02;
        outportb(base+2,c);
}

/*
        Resetta fisicamente il ricevitore togliendo l'alimentazione e
        riallacciandola
*/
void GpsLink::Reset(void)
{
        SimbolikTimer t(10,YES);


        TurnOFF();

        t.Restart();
        while(t != YES);


        TurnON();
```

```
        t.Restart();
        while(t != YES);
}
#endif
```

ISSN 1590-2595

Quaderni di Geofisica

http://istituto.ingv.it/l-ingv/produzione-scientifica/quaderni-di-geofisica/

I Quaderni di Geofisica coprono tutti i campi disciplinari sviluppati all'interno dell'INGV, dando particolare risalto alla pubblicazione di dati, misure, osservazioni e loro elaborazioni anche preliminari, che per tipologia e dettaglio necessitano di una rapida diffusione nella comunità scientifica nazionale ed internazionale. La pubblicazione on-line fornisce accesso immediato a tutti i possibili utenti. L'Editorial Board multidisciplinare garantisce i requisiti di qualità per la pubblicazione dei contributi.

ISSN 2039-7941

rapporti tecnici INGV

http://istituto.ingv.it/l-ingv/produzione-scientifica/rapporti-tecnici-ingv/

I Rapporti Tecnici INGV pubblicano contributi, sia in italiano che in inglese, di tipo tecnologico e di rilevante interesse tecnico-scientifico per gli ambiti disciplinari propri dell'INGV. La collana Rapporti Tecnici INGV pubblica esclusivamente on-line per garantire agli autori rapidità di diffusione e agli utenti accesso immediato ai dati pubblicati. L'Editorial Board multidisciplinare garantisce i requisiti di qualità per la pubblicazione dei contributi.

ISSN 2039-6651

Miscellanea INGV

http://istituto.ingv.it/l-ingv/produzione-scientifica/miscellanea-ingv/

La collana Miscellanea INGV nasce con l'intento di favorire la pubblicazione di contributi scientifici riguardanti le attività svolte dall'INGV (sismologia, vulcanologia, geologia, geomagnetismo, geochimica, aeronomia e innovazione tecnologica). In particolare, la collana Miscellanea INGV raccoglie reports di progetti scientifici, proceedings di convegni, manuali, monografie di rilevante interesse, raccolte di articoli ecc..

Coordinamento editoriale e impaginazione
Centro Editoriale Nazionale | INGV

Progetto grafico e redazionale
Daniela Riposati | Laboratorio Grafica e Immagini | INGV

Via di Vigna Murata, 605
00143 Roma
Tel. +39 06518601 Fax +39 065041181

http://www.ingv.it

Istituto Nazionale di Geofisica e Vulcanologia