Versione N° 1.0

LaurTec

Progettare Sistemi Domotici controllati per mezzo di moduli GSM

Come collegare un modulo GSM ad un microcontrollore ed impartire ad esso dei comandi tramite SMS o squilli

Autore : *Ferrero Vercelli Renato (fer_ver)*

ID: UT0002-IT

INFORMATIVA

Come prescritto dall'art. 1, comma 1, della legge 21 maggio 2004 n.128, l'autore avvisa di aver assolto, per la seguente opera dell'ingegno, a tutti gli obblighi della legge 22 Aprile del 1941 n. 633, sulla tutela del diritto d'autore.

AVVERTENZE

I progetti presentati non hanno la certificazione CE, quindi non possono essere utilizzati per scopi commerciali nella Comunità Economica Europea.

Chiunque decida di far uso delle nozioni riportate nella seguente opera o decida di realizzare i circuiti proposti, è tenuto pertanto a prestare la massima attenzione in osservanza alle normative in vigore sulla sicurezza.

L'autore declina ogni responsabilità per eventuali danni causati a persone, animali o cose derivante dall'utilizzo diretto o indiretto del materiale, dei dispositivi o del software presentati nella seguente opera.
Si fa inoltre presente che quanto riportato viene fornito così com'è, a solo scopo didattico e formativo, senza garanzia alcuna della sua correttezza.

L'autore ringrazia anticipatamente per la segnalazione di ogni errore.

Indice

Introduzione

Sono sempre stato del parere che per imparare "un qualcosa" per hobby occorre avere un obiettivo ben definito. Due anni fa, dopo aver seguito il tutorial "C18 Step by Step" ho realizzato un piccolo antifurto e preso dall'euforia mi sono proposto di automatizzare il campanile del paese con un PIC e di poterlo gestire tramite comandi inviati con il cellulare. Da qui è sorta l'esigenza di capire come funzionavano i moduli GSM.
In questo articolo voglio raccontare e mettere a disposizione di tutti gli utenti di questo bellissimo sito le mie esperienze personali ed i sistemi che ho escogitato per arrivare al mio obiettivo finale.
Il documento non ha la presunzione di essere una guida esaustiva nelle comunicazioni GSM, comandi AT e programmazione PIC. Può essere considerato come un'introduzione all'uso dei moduli GSM da chi ha già seguito il corso "*XC8 Step by Step*" e gli articoli sulla comunicazione seriale pubblicati da Mauro Laurenti, allo stesso tempo servirà anche come promemoria a me stesso in caso di modifiche da effettuare tra qualche anno.
A Mauro devo attribuire la mia iniziativa di scrivere questo documento, con le sue capacità si dedica con passione e professionalità a chi muove i primi passi nel mondo dell'elettronica e nella programmazione...non è una cosa comune.
Ho quindi pensato di mettere a disposizione anch'io le conoscenze acquisite nel corso della realizzazione dell'automatismo del campanile, ed anche un crono-termostato per la gestione della caldaia della chiesa gestiti da comandi via GSM, senza aver la minima presunzione di spiegare con la chiarezza e professionalità l'argomento.

Anche se creati per hobby e non professionalmente, ormai funzionano da svariati mesi ed anche questo mi ha spinto a scrivere questo articolo nel quale non verranno trattati i due sistemi sopracitati nel loro insieme ma verranno raccontate le esperienze avute su come collegare e gestire un modulo GSM ad un microcontrollore.

Nota

Il sistema presentato nell'articolo presenta l'utilizzo di moduli RF. I sistemi sviluppati non sono certificati secondo le normative rilasciate dai relativi organi di competenza. In particolare non possiedono la certificazione CE, FCC. I sistemi presentati devono essere pertanto considerati solo sistemi di sviluppo e valutazione. Potrebbero essere soggetti ad interferenze elettromagnetiche ed essere fonte di disturbo elettromagnetico per altri dispositivi elettronici.

Applicazioni

Le applicazioni che si possono realizzare seguendo questo articolo, sono veramente molteplici e possono contribuire alla progettazione di più svariati automatismi, tra i quali:

- Controllo a distanza di una caldaia.
- Controllo a distanza di un antifurto.
- Inviare o ricevere SMS da un sistema embedded o da PC.
- Funzione apri cancello con cellulare.
- Controllare a distanza un automatismo specifico (vedi ad esempio il campanile).
- Ricevere a distanza informazioni da un sistema (temperatura, allarmi, ecc.)

I moduli GSM

Un modulo GSM può essere considerato come un vero e proprio cellulare senza tastiera e display. Esso comunica con il mondo esterno tramite la porta seriale, nel nostro caso, considerando che deve interagire con un PIC, comunicherà tramite la porta USART. Le istruzioni da svolgere vengono impartite tramite dei comandi AT (AT sta per *ATtention*, cioè *attenzione*). Per inviare un comando occorre trasmettere sulla linea seriale una stringa in formato ASCII contente AT seguita dal comando desiderato e dal carattere di ritorno a capo (nel nostro caso i caratteri speciali \r\n).
Dopo aver alimentato il modulo, occorre, come avviene per un cellulare, attivare o meglio accendere il modulo stesso, ciò avviene a seconda del modulo usato, premendo il tasto sulla scheda oppure inviando un segnale tramite l'input del modulo denominato ON/OFF o PWRKEY. Una volta acceso, il LED presente sul modulo si illuminerà e se tutto ok, dopo alcuni secondi, quando la SIM card verrà registrata presso il proprio gestore, incomincerà a lampeggiare con una pausa di 3 secondi circa.
Da questo momento in poi il modulo è pronto a comunicare con il PIC il quale potrà inviare o ricevere chiamate, come anche SMS. In particolare, sebbene l'articolo mostri come inviare e ricevere delle telefonate ed anche come elaborare gli SMS, non tratta la funzione analogica dei moduli, ovvero la possibilità di collegare un microfono ed un altoparlante al fine di poter inviare e ricevere messaggi vocali. Queste funzioni sebbene non siano trattate, non dovrebbero essere difficili da aggiungere, visto che i moduli hanno delle linee dedicate sia per il microfono che l'altoparlante.

Descrizione dei moduli GSM utilizzati nell'articolo

I moduli utilizzati e descritti sono:

- MOD-GSM della Olimex
- 8100-TDGGSM

Entrambi i moduli presentati hanno un costo di circa 50€. Il MOD-GSM è il modulo che ho utilizzato finora per la realizzazione dei miei progetti in quanto dotato di antenna incorporata e con la possibilità di essere alimentato a 12V. mentre a mio parere il TDGGSM è il modulo più utilizzato e conosciuto sulla rete, soprattutto perché interfacciabile con la piattaforma Arduino.

Il modulo MOD-GSM OLIMEX

Tra le caratteristiche principali del modulo si ricordano:

- Alimentatore incorporato
- Antenna incorporata
- Presenza pulsante ON/OFF sulla scheda
- Auto Baudrate
- Possibilità di essere posizionato al di fuori della scheda madre tramite cavo flat.
- Insensibile all'invio del carattere di fine stringa \0 da parte del PIC.

Specifiche tecniche
Dimensioni…… : 79,2x57,6 mm
Peso……..: 28gr
Alimentazione …….: 9 - 12 VDC
SIM……..: SIM340DZ
Antenna……: incorporata
Banda…….....: quad band 800/900/1800/1900 Mhz
Caricabatterie…….....: incorporato
Pulsante accensione....: incorporato

Lo schema elettrico del modulo è possibile scaricarlo dal sito della Olimex. In Figura 2 è riportato un dettaglio del modulo.

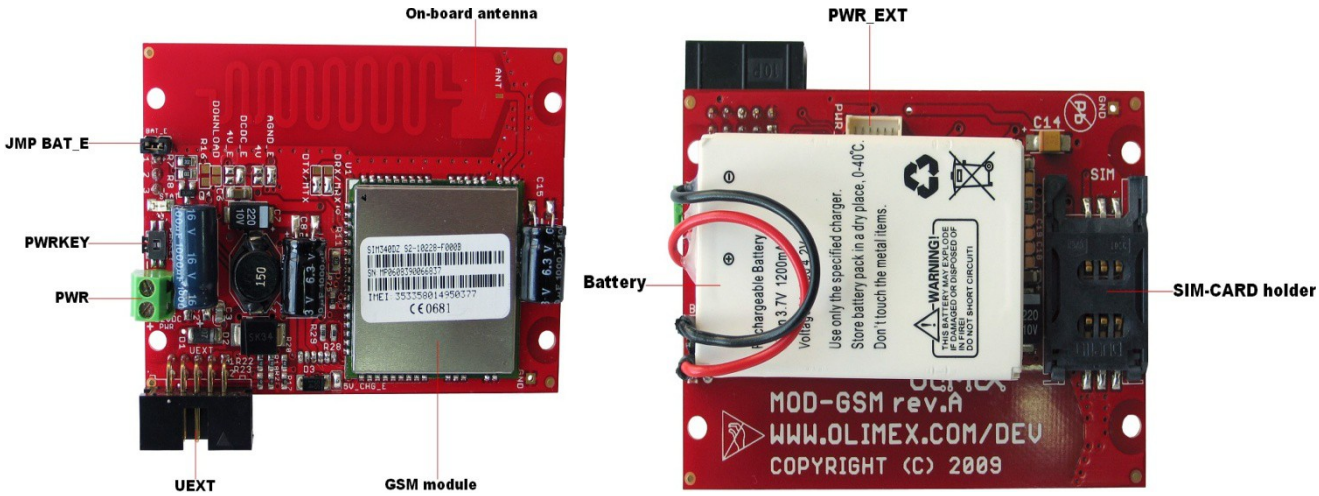


Figura 1: *Modulo MOD-GSM.*

I segnali principali sono accessibili dal connettore IDC nominato UEXT. Il pin 1 è indicato dal triangolo sul connettore.

PIN	Descrizione
1	NC
2	GND
3	RXD
4	TXD
5	RTS
6	CTS
7	DTR
8	DCD
9	RI
10	PWRKEY

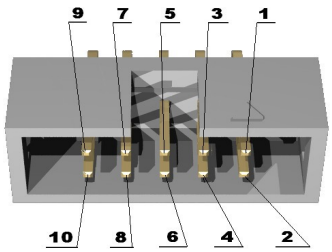


Figura 2: *Connettore del modulo MOD-GSM.*

Nelle mie applicazioni ho utilizzato il PIN 1 come ingresso alimentazione +12V, collegandolo al positivo della morsettiera PWR.

Il modulo 8100 TDGGSM

Tra le caratteristiche principali del modulo si ricordano:

- Dimensioni minori
- Presenza connettore per antenna esterna
- Più segnali disponibili in ingresso/uscita
- Connessione più pratica se fissato direttamente sulla scheda madre.
- Compatibile con Arduino

Specifiche tecniche

Dimensioni……: 51x45 mm
Peso……...: 17gr
Alimentazione …..…: 3,5/ 4,5 VDC
SIM…….....: SIM900
Antenna……..…: esterna tramite connettore
Banda……..…..: quad band 800/900/1800/1900 Mhz
Caricabatterie……...: non presente
Pulsante accensione...: non presente

In Figura 3 è riportato lo schema elettrico della scheda.

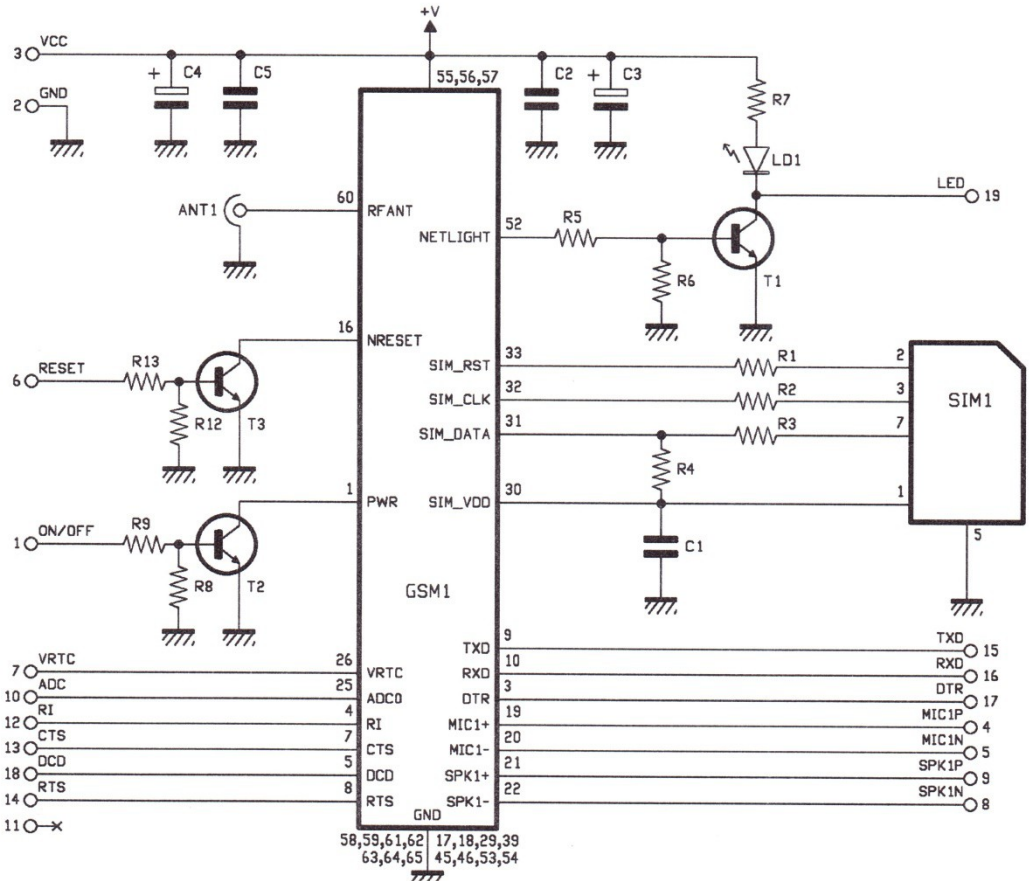


Figura 3: *Schema elettrico del modulo 8100 TDGGSM basato su SIM900.*

Un dettaglio del modulo con montata la SIM card, è riportato in Figura 4.

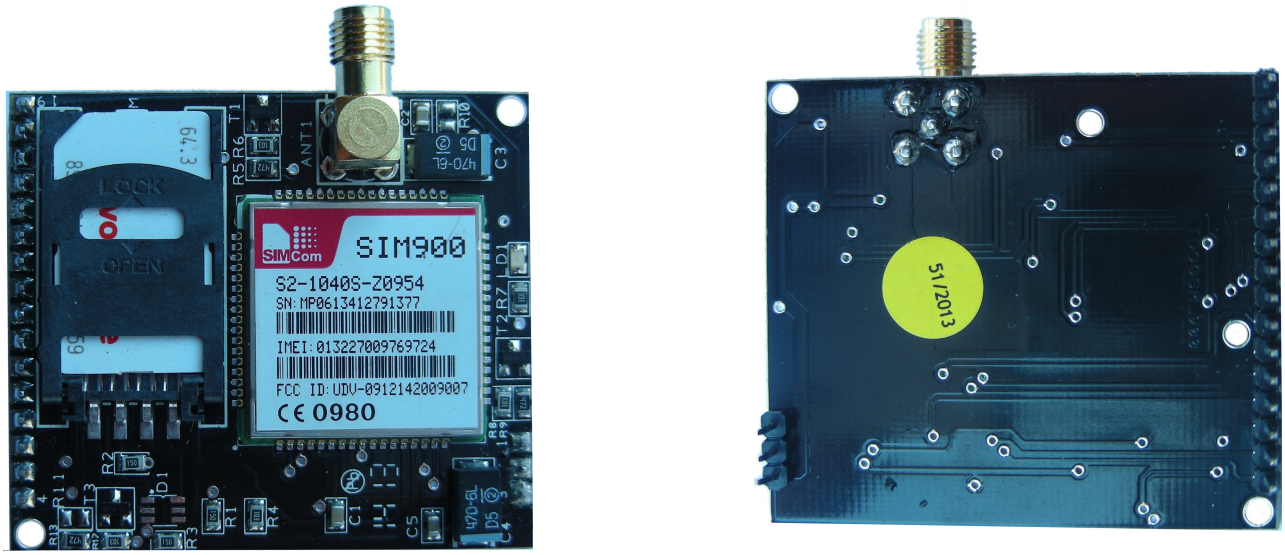


Figura 4: *Modulo 8100 TDGGSM basato su SIM900.*

I dettagli dei collegamenti disponibili sulla scheda sono riportati in Figura 5.

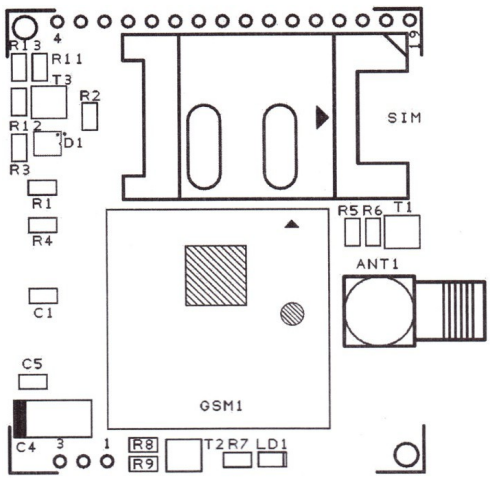


Figura 5: *Modulo 8100 TDGGSM basato su SIM900.*

Analisi del Progetto

Lo schema base per connettere tra di loro un microcontrollore ed un modulo GSM è il seguente:

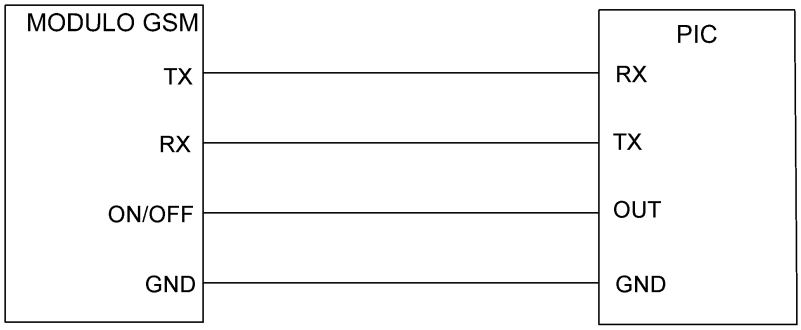


Figura 6: *Schema a blocchi per il collegamento del modulo e il microcontrollore.*

Utilizzando una comunicazione seriale, lo scambio di informazioni avviene tramite tre soli fili, ovvero le linee RX, TX e GND, oltre a queste occorre aggiungerne la linea ON/OFF se si desidera che sia il microcontrollore ad accendere e/o spegnere il modulo GSM. La maggior parte dei moduli contiene un microprocessore alimentato da 3,3 a 4,5V e pertanto, se il microcontrollore da collegare è alimentato a 5V, come nel caso della scheda FREEDOM II, occorre adattare i livelli logici tra i due apparecchi. Lo schema base per ottenere questo adattamento di segnale può essere il seguente:

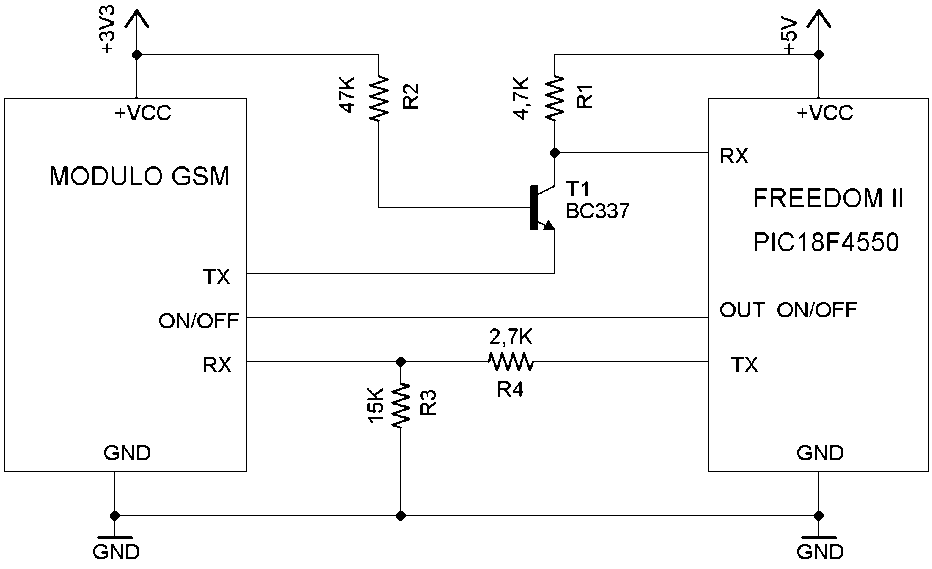


Figura 7: *Schema elettrico con il dettaglio del Level shifter.*

Questo schema l'ho ricavato da quello pubblicato a pagina 7 del file:

```
SIM900_Serial_Port_Application_Note_V1.03
```

scaricabile previa registrazione gratuita dal sito:

```
http://wm.sim.com/producten.aspx?id=1019
```

Sul sito ufficiale della SIMCOM, http://wm.sim.com, produttore delle due SIM utilizzate in questo articolo, sono presenti diversi file inerenti alla SIM900 e la SIM340. Sebbene lo scopo principale di questo articolo sia quello di interfacciare un modulo GSM con un PIC, consiglio vivamente di collegare prima il modulo ad un PC tramite la porta seriale e di utilizzare un programma adatto a gestire la porta RS232 quale ad esempio "*RS232 Terminal*" che si può liberamente scaricare dal sito www.LaurTec.it. In questo modo sarà molto più pratico inviare dei comandi al modulo e verificare visivamente la risposta ricevuta. Proprio per poter collegare tra di loro il modulo ed il PC, occorre adattare nuovamente i livelli logici dei due apparecchi. Per ottenere questo ho utilizzato un integrato MAX232 che, partendo da un'unica alimentazione a 5V, permette il collegamento tra logica TTL o CMOS a 5V e le tensioni definite nel nel protocollo RS232, che possono variare da -12V a + 12V. All'interno del MAX232 sono presenti quattro porte invertenti.

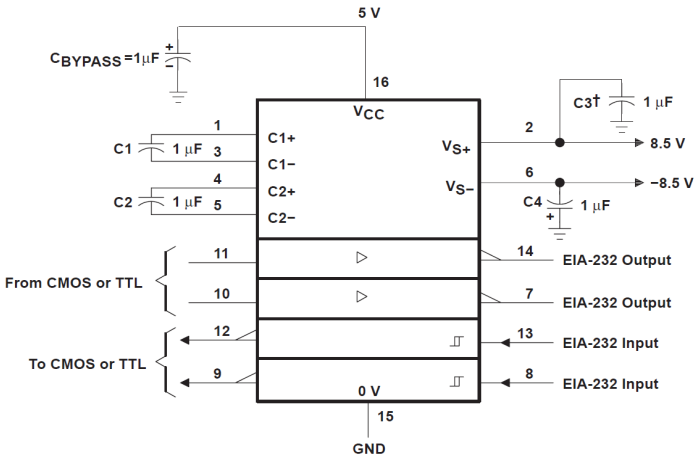


Figura 8: *Schema elettrico per il il Transceiver MAX232 (estratto dal datasheet Texas Instruments).*

Per poter variare rapidamente il collegamento del modulo GSM dal PC alla scheda Feedom II, oppure controllare tramite PC i dati inviati dal PIC o dal modulo, oppure poter collegare tra loro il PIC ed il PC, ho realizzato la scheda d'interfaccia PIC-GSM-PC la cui fotografia è riportata in Figura 9, montando il tutto sulla scheda di espansione Freedom II modello PJ7010. Il dettaglio dello schema elettrico è riportato in Figura 11.

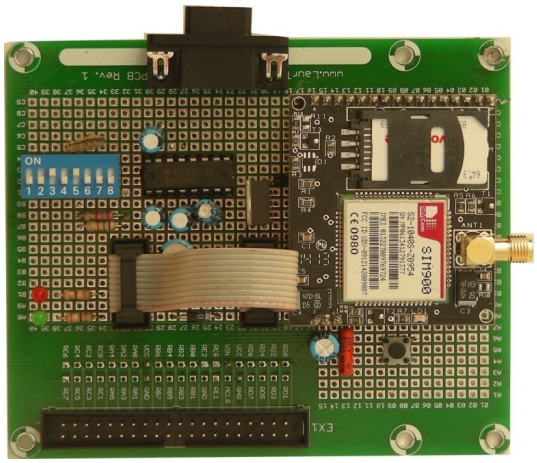


Figura 9: *Montaggio ultimato sulla scheda di sviluppo PJ7010.*

In Figura 10 è mostrato il dettaglio del montaggio finale tra i due moduli e la scheda di sviluppo Freedom II.

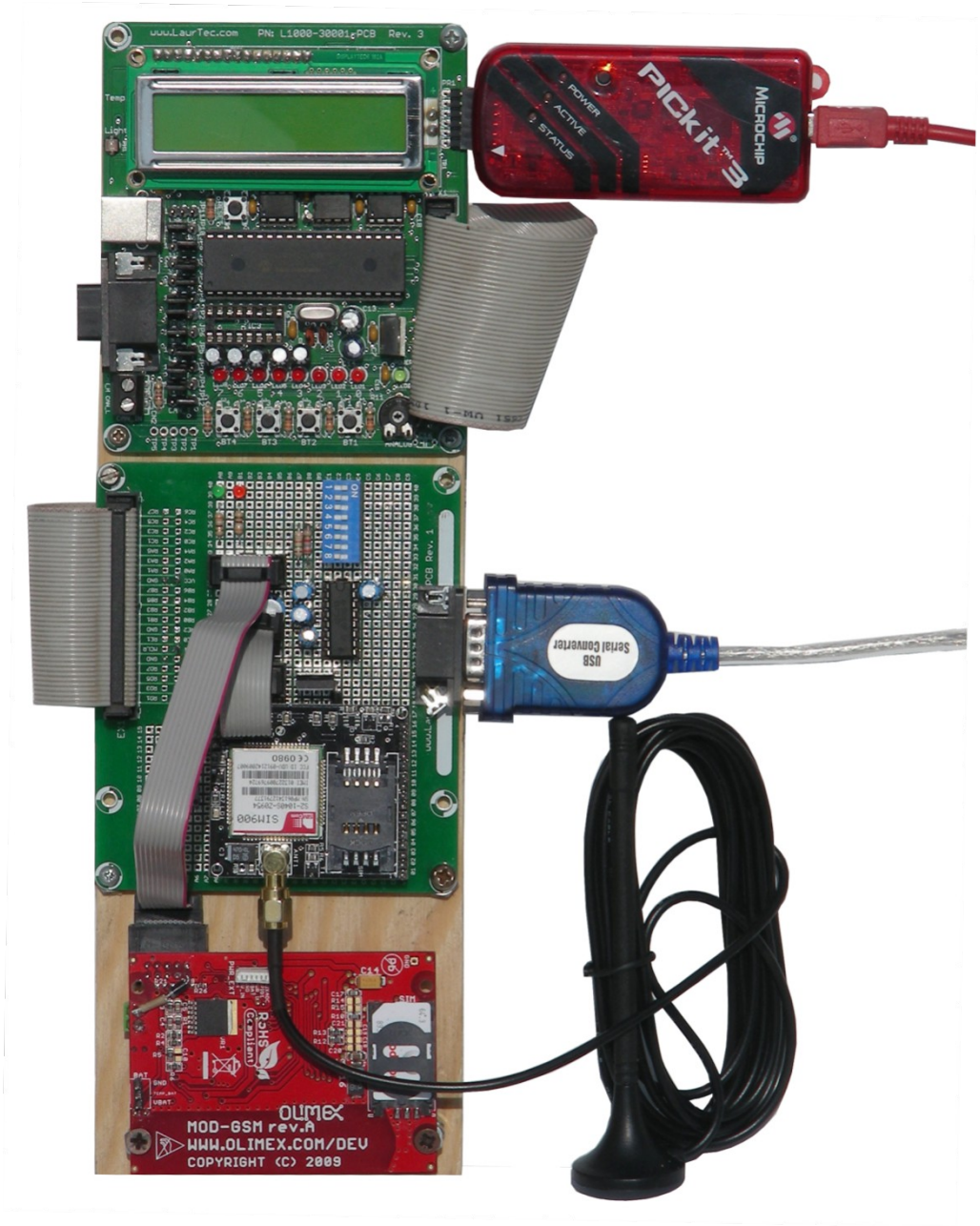


Figura 10: *Montaggio ultimato con l'integrazione dei vari sistemi e Freedom II.*

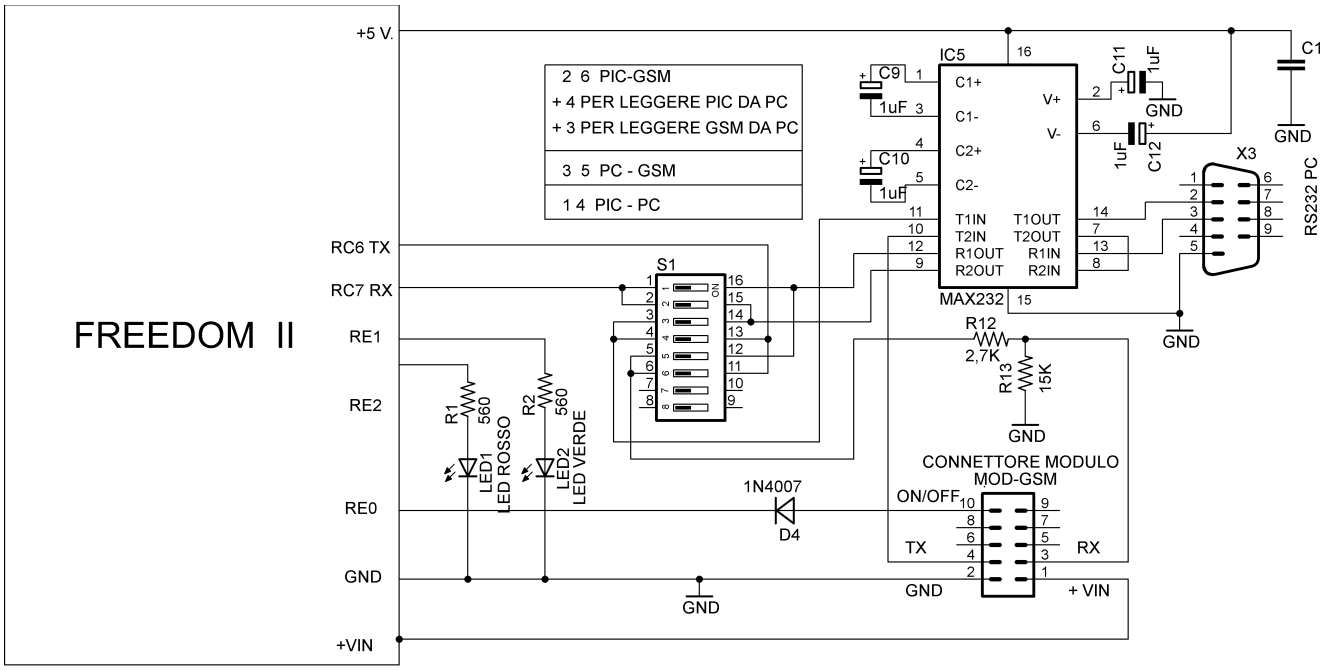


Figura 11: *Schema elettrico collegamento Freedom II- MOD GSM (SIM340)-RS232.*

Seguendo lo schema di Figura 11 si può notare che per adattare i livelli logici tra PIC e GSM sono state utilizzate le due porte inutilizzate dell'integrato MAX232 che fanno capo ai Pin 7-10 e 8-9. Per accendere e spegnere il modulo è stata utilizzata l'uscita del PIC RE0, mentre quelle denominate RE1 e RE2 servono per comandare i due LED, rosso e verde, inseriti sulla scheda.
Per adattare il circuito utilizzato per il MOD-GSM alla TDGGSM utilizzando lo stesso programma firmware ho aggiunto il transistor TR1 per invertire il segnale logico all'uscita di RE0 ed il regolatore di tensione 7805 con in serie alla sua uscita due diodi in modo da ottenere la tensione di 3,8V necessaria ad alimentare la SIM900. Il relativo schema risulta è riportato in Figura 12.

Sebbene la soluzione presentata possa funzionare nella maggior parte dei casi, potrebbe presentare problemi derivanti dal fatto che la corrente richiesta dal modulo SIM900 in modalità burst può richiedere fino a 2000mA mentre il regolatore 7805 può fornire fino ad 1A. Come soluzione ottimale si consiglia di usare un alimentatore locale con convertitore DC-DC modello LM2596-ADJ che fornisce anche la possibilità di regolare direttamente l'alimentazione a 3.8V.

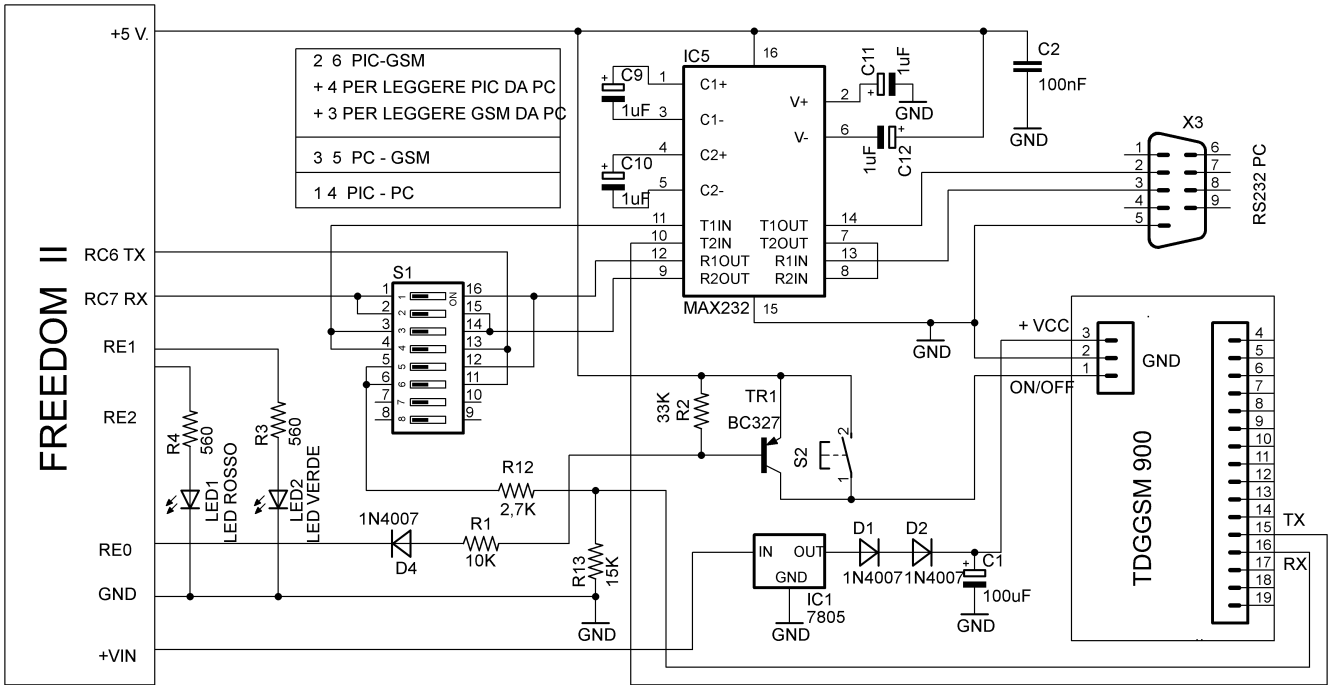
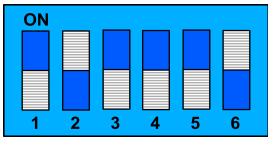


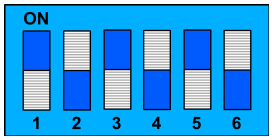
Figura 12: *Schema elettrico collegamento FREEDOM - TDGGSM (SIM900) – RS232.*

Come si può notare, tramite il commutatore minidip S1, è possibile cambiare facilmente i collegamenti tra i vari componenti del circuito. Le configurazioni che si possono ottenere ponendo sulla posizione ON i vari switch sono riportate in Figura 13.

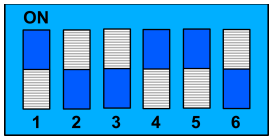
Per poter permettere una corretta comunicazione tra Freedom II e la scheda di sviluppo, bisogna rimuovere il MAX232 dalla Freedom II ed inserirlo nella scheda d'interfaccia PIC-GSM.



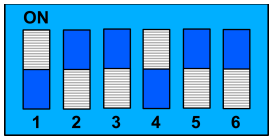
2-6 su ON
In questo modo vengono collegati tra di loro:
PIC e modulo GSM



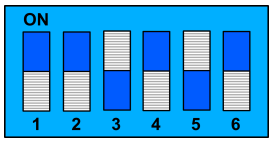
2-4-6 su ON
In questo modo vengono collegati tra di loro:
PIC e modulo GSM
Sul PC si leggono i comandi inviati dal PIC



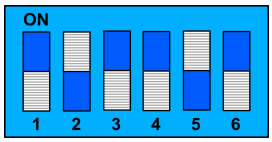
2-3-6 su ON
In questo modo vengono collegati tra di loro:
PIC e modulo GSM
Sul PC si leggono i comandi inviati dal GSM



1-4 su ON
In questo modo vengono collegati tra di loro:
PC e PIC



3-5 su ON
In questo modo vengono collegati tra di loro:
PC e modulo GSM



2-5 su ON
In questo modo il PC invia i comandi al GSM
e il PIC riceve le risposte dal GSM

Figura 13: *Posizione degli interruttori e relative funzioni.*

Comandi AT

Ogni istruzione dei moduli GSM in questione è gestita da dei comandi AT Hayes, originariamente creati e sviluppati per il modem Hayes da 300 baud. La SIM900 e SIM340 supportano, un elenco molto esteso di comandi AT.
In questo articolo, verranno descritti solamente i comandi necessari per gestire un sistema controllato da un PIC tramite squilli o invio di SMS. Per inviare un comando AT occorre inviare in formato ASCII, tramite la porta seriale, i caratteri che identificano il comando stesso, seguito da <CR> o ritorno a capo, oppure dai caratteri speciali \r\n.

Esempio:

```
AT <CR>
```

oppure in C

```
putrsUSART((char *) "AT\r\n")
```

Il modulo invierà quindi una risposta nel seguente formato:

```
<CR><CR><LF> risposta <CR><LF>    =  \r\r\n risposta \r\n.
```

Per default, l'eco locale dei moduli è attivo e perciò il modulo ripeterà in uscita ogni carattere ricevuto. Nel caso si sia inviato il comando AT\r\n senza eco locale attivo (ATE0) la risposta del modulo sarà:

```
\r\r\nOK\r\n
```

ovvero

```
OK
```

Un comando AT inizia sempre con il prefisso AT tranne che per il comando:

A/

il quale ripete il comando, e il comando di escape:

```
+++
```

Nella maggior parte dei casi, i comandi AT sono formati da una sigla iniziale, che definisce il tipo di comando, seguito da un parametro numerico. Per esempio, ATH0 serve a chiudere la linea telefonica. Questi comandi possono essere composti senza il parametro finale, cioè senza il numero, quando si vuole fare riferimento allo zero. Quindi, ATH è esattamente uguale a ATH0.
Altri comandi sono costituiti dalla sigla iniziale AT+ ed un secondo comando come ad esempio

```
AT+COPS
```

che restituisce il nome del gestore a cui è registrata la SIM card. I due moduli utilizzati non sono *Case Sensitive* e quindi si possono inserire i caratteri che compongono il comando come si preferisce, in maiuscolo o minuscolo. Tutti gli esempi di chiamate a cellulare riportati in questo articolo si riferiscono ad un numero scelto casualmente e corrispondente a +391231231231. La Tabella 1 fornisce l'elenco dei comandi AT utilizzati.

Comando AT	Descrizione Comando
AT	Codice Attenzione. Viene richiamata l'attenzione del modulo. Risposta: "OK" in caso di buon funzionamento, altrimenti "ERROR".
ATE0	Disabilita l'eco dei comandi da parte del modulo.
ATE1	Abilita l'eco dei comandi da parte del modulo. In questo caso il modulo ripeterà sul terminale TX della linea seriale ogni carattere ricevuto su quella RX.
AT+CLIP=1	Consente di identificare il numero del chiamante
AT+CLIP=0	Disabilita la visione del numero del chiamante
AT+CSQ	Verifica il livello del segnale di copertura campo Risposta: +CSQ : 29,99 (considerare solo i primi 2 numeri in una scala da 0 a 31) Da 0-9 segnale scarso // 10-19 segnale discreto // 20-31 segnale ottimo 99 = sconosciuto o non calcolabile Per esperienza personale si richiede un valore minimo di 6/7
AT+CGMM	Restituisce il nome del modulo usato. Risposta: SIM_COM_SIM900 oppure : SIMCOM_SIM340D OK
AT+COPS?	Restituisce il nome del gestore a cui si è registrati Risposta: +COPS: 0,0,"vodafone IT" OK
ATH	Interrompe una comunicazione telefonica (riaggancia).
ATD	Effettua una chiamata vocale ATD<numero telefonico;> <CR> (attenzione il numero telefonico è sempre preceduto dal prefisso internazionale ad es. ATD+391231231231;<CR>) Risposta: OK o NO CARRIER
AT+CMGF=1	Imposta il modulo in modalità testo per invio SMS. Risposta: OK oppure ERROR in caso il modulo non supporti la modalità testo o la SIM card non sia pronta
AT+CMGD=n	Cancella dalla SIM card il messaggio SMS numero "n"
AT+CMGS	Invia un SMS direttamente senza memorizzarlo sulla SIM Sintassi: AT+CMGS="numero cell. Compreso pref. internazionale"<CR> Digitare il testo da inviare e concludere con Ctrl+Z.

Tabella 1: *Comandi AT più utilizzati.*

In Tabella 2, sono riportati i comandi non utilizzati nel programma ma che potrebbero essere utili per altre applicazioni. Oltre a quelli presentati ogni modulo fornisce anche ulteriori comandi.

Comando AT	Descrizione Comando
AT+CMEE	Indica al modulo GSM come rispondere in caso di errore: AT+CMEE=0 in caso di errore si ha la risposta "ERROR" AT+CMEE=1 in caso di errore il modulo risponde con il n° di errore AT+CMEE =2 in caso di errore il modulo invia una stringa contente l'errore
AT+ IPR	AT+IPR? Risponde la velocità di trasmissione in baud AT+IPR = 9600 imposta la velocità a 9600 baud
AT+ CPIN	Permette di inserire il pin se richiesto dalla SIM AT+CPIN="1234" .
At+CMGW	Scrive nella SIM il numero del cellulare a cui inviare un SMS ed il testo che deve contenere. Sintassi; CMGW= "numero cellulare compreso +39"<CR> Quindi digitare il testo e concludere premendo Ctrl+z Risposta: +CGMW; n in cui n è il numero che gli è stato assegnato
AT+CMGR =n	Legge il messaggio SMS numero "n" dalla SIM card e lo invia sulla porta seriale. Risposta: +CMGR: "STO UNSENT","+391231231231","" Testo del SMS OK
AT+CMSS= n	Invia il messaggio memorizzato sulla SIM card con "n"
AT+CMGL ="ALL"	Lista tutti gli SMS presenti sulla SIM card
AT+CREG?	Verifica che il modulo sia registrato alla rete Risposta: +CREG ; 0,1 (il modem è registrato)

Tabella 2: *Comandi AT secondari.*

Per ricevere un SMS contente il credito residuo della SIM, inviare, a seconda del gestore utilizzato, un SMS ad uno dei numeri riportati in Tabella 3[1].

Nome Operatore	Numero da Comporre	Testo SMS
Tim	40916	Credito
Vodafone	404	Traffico
Wind	4155	Saldo

Tabella 3: *Numeri di servizio per richiedere il credito ai vari operatori.*

Esempio di SMS in risposta:

```
+CMGR: "REC UNREAD","Vodafone","","14/03/15,16:17:58+04"
Vodafone 15-03-2014, 16:17. Traffico Disponibile Euro 9.61
```

[1] Il numero potrebbe variare nel tempo, per cui far affidamento sempre alle informazioni fornite dal proprio gestore.

Utilizzo degli SMS con i moduli GSM

Il termine SMS (*Short Message Service)* è comunemente usato per indicare l'invio di un messaggio di testo tra due o più utenti di telefonia. Lo standard prevede due diverse tipologie di messaggi: quelli *Point to Point* (SMS/PP), impiegati nella comunicazione da un terminale ad un altro, normalmente utilizzato dagli utenti, e quelli tipo *Cell Broadcast* (SMS/CB), originati normalmente dal gestore e distribuiti a tutti i terminali sotto la sua copertura.

Il messaggio ha una dimensione fissa di 140 byte per un totale di 1120 bit. Considerato che ogni carattere trasmesso occupa solo 7 bit, esso può quindi contenere 160 caratteri di testo (a 7 bit). L'architettura con cui viene gestito l'ottavo bit del byte è alquanto complessa ed esula dall'argomento in questione. le lingue che usano altri caratteri rispetto all'alfabeto latino, per esempio il cinese, il messaggio è limitato a soli 70 caratteri (ognuno di 2 byte, usando il sistema Unicode). Nel caso in cui il testo di un SMS sia più lungo di 160 caratteri vengono automaticamente inoltrati due o più messaggi che vengono riuniti nel giusto ordine, dal terminale ricevente.

I moduli GSM utilizzati possono inviare gli SMS in due modalità selezionabili tramite il comando:

```
AT+CMGF=?
```

Text Mode = AT+CMGF=1

Questo protocollo è basato su caratteri (*Text Mode*) e permette di trasferisce il messaggio al modulo inviando direttamente la stringa contenente il numero da chiamare ed il testo da trasmettere. Sarà poi il processore del modulo che si interesserà di formattare in modo idoneo la trasmissione dei dati. Questo protocollo è più semplice da usare ma permette solo la trasmissione di caratteri.

Esempio di invio SMS:

```
AT+CMGS="+391231231231"<CR>
Testo sms  CTRL+Z        ( CTRL+Z= carattere ASCII 26)
```

PDU Mode = AT+CMGF=0

Il protocollo *PDU Mode* è molto simile al *Text Mode*, con la differenza che il primo delega all'applicazione la responsabilità di costruire il tracciato del messaggio. Questo protocollo permette di costruire delle trasmissioni più complesse, vale a dire che si possono trasmettere dati binari e non solo caratteri.

Esempio di invio SMS:

```
AT+CMGS=42<CR>07915892000000F001000B915892214365F7000021493A283D0795C3F33C88FE06C
DCB6E32885EC6D341EDF27C1E3E97E72E
```

Penso che nelle nostre applicazioni sia molto più semplice inviare gli SMS in Modalità Testo! Comunque per chi non si accontenta ho trovato sul web un articolo che spiega l'architettura degli SMS in PDU Mode scritto da Giovanni Peres al seguente indirizzo:

http://xes.dyndns.org/index.php/codifica-in-pdu-di-un-sms.html

Quando il modulo GSM riceve un SMS non trasmette direttamente sulla porta seriale il messaggio ricevuto bensì lo memorizza sulla SIM card, gli affida un numero per identificarlo e trasmette sulla seriale la seguente stringa:

```
+CMTI: "SM" ,n
```

in cui:

- +CMTI : indica che è arrivato un messaggio
- SM : indica su quale supporto è stato memorizzato, in questo caso SM sta per SIM card.
- n: indica il numero con cui è stato memorizzato.

Per visualizzarlo occorre inviare al modulo il comando:

```
AT+CMGR=n
```

Il modulo trasmetterà sulla porta seriale il messaggio ricevuto.
Il formato di un SMS trasmesso dal modulo GSM sulla porta seriale è il seguente:

```
+CMGR: "REC UNREAD","+391234567890","","14/03/15,17:12:33+04"
Hello world
OK
```

In cui:

"REC UNREAD" significa che il messaggio non è ancora stato letto.
"+391234567890" è il numero del chiamante. Segue data ed ora in cui è stato ricevuto il messaggio ed il testo.

Dal punto di vista pratico, per analizzare quindi il messaggio da parte di un microcontrollore possiamo quindi osservare che dopo la terza (") è presente il numero del chiamante e dopo il comando di ritorno a capo (in ASCII 10) è presente il testo del messaggio.
E' possibile richiedere al modulo di trasmettere tutti i messaggi memorizzarti sulla SIM card tramite il comando:

```
AT+CMGL="ALL"
```

Esperienze con RS 232 Terminal

Come già detto il modo migliore per fare esperienza con i moduli GSM è di connetterli al PC tramite un programma adatto a gestire la porta seriale. Un ottimo programma, ed allo stesso tempo facile da utilizzare, è *RS232 Terminal* scaricabile gratuitamente, previa registrazione, sul sito www.LaurTec.it.

In questo articolo è stata utilizzata la versione 1.4.2. Per connettere la porta seriale del PC ad un modulo GSM è necessario utilizzare un cavo incrociato con connettori maschio-

femmina DB9, ovvero un cavo Null-Modem, come da schema sotto riportato:

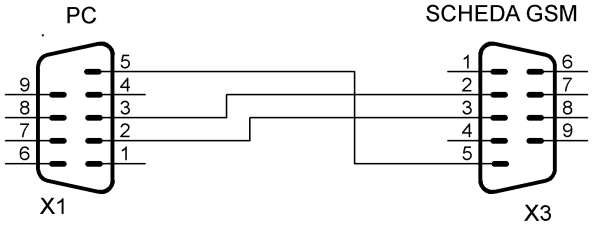


Figura 14: *Schema elettrico del collegamento Null-Modem.*

Se si utilizza la porta USB del PC occorre utilizzare un adattatore USB-RS232, come per esempio miniCOM USB (progetto presentato sul sito www.LaurTec.it).
Ricordo che per collegare le due unità occorre adattare i livelli di segnale e quindi tutto quanto esposto di seguito si riferisce agli schemi sopra riportati FREEDOM- MOD GSM (SIM340)-RS232 oppure FREEDOM - TDGGSM (SIM900)-RS232 a seconda del modulo utilizzato. Per effettuare il collegamento con il PC è necessario mettere sulla posizione ON i mini DIP n. 3 e 5 e lasciare tutti gli altri su OFF. Inserire quindi la scheda SIM card nel modulo GSM.

- Bisogna eliminare preventivamente, con un cellulare, l'eventuale richiesta del PIN da parte della SIM.
- Togliere l'integrato MAX232 dalla Freedom II ed apporlo nella scheda d'interfaccia.

Inserire il connettore DB9 maschio alla porta X3 della scheda d'interfaccia. Unire la Freedom II alla scheda d'interfaccia tramite un connettore a 40 poli con cavo flat (personalmente ne ho utilizzato uno recuperato da un vecchio PC). Alimentare quindi la Freedom II. Aprire con *RS232 Terminal* la porta seriale utilizzando le seguenti impostazioni :

- Baud Rate : 115200
- Data Bits : 8
- Stop Bits : 1
- Parity : None
- Handshake : None

Accendere quindi manualmente il modulo premendo il pulsante ON/OFF o PWRKEY .
Se tutto ok, il LED presente sul modulo si accenderà lampeggiando una volta al secondo; quando, dopo qualche istante, si sarà collegato al gestore il lampeggio avrà una pausa di circa 3 secondi. A questo punto è d'obbligo illustrare la differenza di Auto Baudrate, ovvero l'autoapprendimento della velocità di trasmissione, tra i due moduli GSM.

MOD-GSM (SIM340)

Se non si sono memorizzate precedentemente, sulla memoria non volatile del modulo, delle impostazioni particolari tramite il comando AT&W (non l'ho mai sperimentato!) il modulo MOD-GSM si avvierà in modalità Auto Baudrate (AT+IPR=0), ad una velocità di 115200 baud e invierà la seguente stringa

```
"Call Ready"
```

Per cambiare velocità di trasmissione basterà inviare un nuovo carattere, in maiuscolo o minuscolo, alla velocità desiderata ed automaticamente il modulo si adatterà al nuovo Baudrate (il primo carattere inviato verrà perso).
L'ultima velocità utilizzata dal modulo prima dello spegnimento non verrà memorizzata neppure se impostata manualmente tramite il comando AT+IPR = 9600.

TDGGSM (SIM900)

Questa SIM si comporta in modo decisamente diverso, infatti al momento dell'accensione, se l'Auto Baudrate era stato attivato nell'ultima sessione tramite il comando (AT+IPR=0) o per default, essa invia sulla seriale una serie di caratteri casuali ad esempio: I I I I ? ? ? ? e rimane in attesa di ricevere un carattere, tassativamente in maiuscolo, per auto apprendere la velocità di trasmissione. Quando la nuova velocità di trasmissione è stata appresa l'Auto Baudrate viene escluso e quindi per variare nuovamente la velocità occorre resettare il modulo oppure utilizzare il comando AT+IPR=XXXXX in cui XXXXX corrisponde alla velocità in baud desiderata. Se si utilizza questo comando (AT+IPR=…) la velocità di trasmissione verrà memorizzata sulla memoria non volatile del modulo e quindi mantenuta ad una nuova riaccensione del modulo. In questo caso, ad una nuova accensione, essendo stata memorizzata una velocità di trasmissione, il modulo invierà sulla seriale la seguente "stringa di benvenuto":

```
RDY

+CFUN: 1

+CPIN: READY

Call Ready
```

In considerazione di tutto ciò, se desiderate ricevere tale messaggio, in caso di collegamento tra TDGGSM e *RS232 Terminal* dovete avviare il modulo, far auto apprendere la velocità di trasmissione, ed impostarla a 115200 bit/s tramite il comando AT+IPR=115200 . Dopodiché, spegnere e riaccendere il modulo e controllare che vengano inviate le stringhe sopra descritte.
Se tutto è OK, consiglio di riportare la SIM900 in modalità Auto Baudrate tramite il comando AT+IPR=0, in modo da renderla più idonea ad essere gestita da più applicazioni compresa quella presentata in questo articolo.

Bene, ora che ci siamo tolti il tarlo dell'Auto Baudrate e non ci rimane che prendere confidenza con il nostro modulo. Molti dei comandi che utilizzeremo sono già stati descritti in precedenza ma consideriamolo un ripasso e quindi li ripeterò.

Il comando base è come anticipato quello AT seguito da <CR> al quale il modulo deve rispondere OK. Se visualizziamo solamente la risposta, abilitiamo allora l'eco dei comandi tramite

```
ATE1<CR>
```

in modo che il modulo ritorni tutti i caratteri ricevuti e di conseguenza vengano visualizzati a video. Se tutto ciò avviene, possiamo interrogare il modulo con il comando:

```
AT+CREG?
```

La risposta nel caso il modulo sia registrato in rete è:

```
 +CREG ; 0,1
```

Per controllare il nome del gestore a cui si è registrati si deve scrivere il comando:

```
AT+COPS?
```

Per controllare il livello del segnale GSM si deve scrivere il comando:

```
AT+CSQ
```

Proviamo ora ad osservare l'invio di segnalazioni d'errore da parte del modulo a seconda del comando AT+CMEE impostato. Proviamo ad inviare un comando errato ad esempio

```
ATDFH <CR>
```

la risposta sarà:

```
ERROR
```

Impostiamo ora l'invio di errori in modalità numerica tramite il comando

```
AT+CMEE=1
```

digitiamo nuovamente:

```
ATDFH<CR>
```

questa volta la risposta sarà:

```
+CME ERROR : 27
```

ed ora impostiamo la modalità testo con il comando:

```
AT+CMEE=2
```

riscrivendo un comando errato:

```
ATDFH<CR>
```

la risposta sarà:

```
+CME ERROR: invalid characters in dial string
```

Quale modalità è la migliore? A mio parere dipende dall'applicazione, se si sta utilizzando

RS232 Terminal lascerei quest'ultima mentre se dovessi gestire tali errori con il PIC utilizzerei AT+CMEE=1.

Passiamo ora a qualcosa di concreto, ad esempio facendoci squillare il cellulare (sempre che il credito della SIM inserita nel modulo sia in attivo!
Ricordatevi, purtroppo, che anche i moduli GSM attingono dal credito SIM !

Utilizziamo il comando ATD seguito dal numero che si desidera chiamare, comprensivo di prefisso internazionale e quindi dal carattere ;(punto e virgola) e <CR>

```
ATD+391231231231;<CR>
```

Sorpresa!
Il cellulare squilla!

Digitare il seguente comando per terminare la chiamata:

```
ATH <CR>
```

Proviamo ora a fare l'inverso, cioè proviamo a chiamare il modulo con il nostro cellulare e vediamo cosa succede. Sul video del mio PC sono comparse le seguenti stringe.

```
RING

+CLIP: "+391231231231",145,"",,"",0

RING

+CLIP: "+391231231231",145,"",,"",0

RING

+CLIP: "+391231231231",145,"",,"",0

NO CARRIER
```

Il messaggio ricevuto ad ogni squillo è quindi composto da due stringhe, RING che avvisa che è in corso una chiamata, e +CLIP seguito dal numero del chiamante più altre caratteristiche specifiche che nel nostro caso non ci interessano. Per chi vuole approfondire l'argomento basta andare a leggere le specifiche dei comandi AT del modulo usato nel relativo Datasheet.

Facciamo un passo indietro e torniamo al comando ATD utilizzato per effettuare una chiamata. Ebbene in questo caso mi è rimasta una grande perplessità o lacuna, in quanto il modulo quando riceve il comando ATD per effettuare una chiamata risponde solamente con OK e quindi non si è in grado di capire se il telefono del ricevente sta squillando e quanti squilli ha fatto oppure se non è raggiungibile. Attualmente non ho trovato modi diretti per mezzo dei comandi AT. Una possibile soluzione, che non ho verificato, potrebbe essere quella di monitorare l'uscita audio, verificando la presenza analogica degli squilli e la frequenza degli stessi (libero od occupato).

Ora proviamo con gli SMS, proviamo a vedere se all'interno della SIM sono memorizzati dei messaggi tramite il comando:

```
AT+CMGL="ALL" <CR>
```

Nel caso vi siano degli SMS memorizzati verranno visualizzati tutti.

Proviamo ora a memorizzare sulla SIM un SMS da inviare, per prima cosa specifichiamo che intendiamo inviarlo in modalità testo e quindi digitiamo:

```
AT+CMGF=1 <CR>
```

Il modulo risponderà

```
OK
```

Ora procediamo con il messaggio vero e proprio e quindi digitiamo:

```
AT+CMGW="+391231231231"<CR>
Hello world  <Ctrl+z>
```

Nota
Ctrl+z = in esadecimale 0x1A.

Il modulo risponderà:

```
+CMGW;n
```

dove 'n' è il numero con cui è stato memorizzato

Possiamo ora richiamarlo e visualizzarlo tramite il comando:

```
AT+CMGR=n<CR>
```

oppure spedirlo al destinatario con il comando:

```
AT+CMSS=n<CR>
```

Dopo averlo spedito decidiamo di cancellarlo dalla SIM tramite AT+CMGD=n<CR>.
Vi è un altro modo per inviare un SMS, senza memorizzarlo sulla SIM, questo comando è:

```
AT+CMGS="+391231231231"<CR>
Hello world  <Ctrl+z>
```

E voilà...ecco il messaggio che arriva sul nostro cellulare!

Proviamo ora a spedire un SMS dal nostro cellulare al modulo contenente il testo "Ciao Mauro" . Sorpresa ! Sul video del PC non comparirà il messaggio completo ma la stringa:

```
+CMTI: "SM",n
```

che indica dove il messaggio è stato memorizzato (SM=Sim Card) e con che numero di assegnazione "n". Se vogliamo quindi vedere il messaggio ricevuto dobbiamo richiamarlo tramite:

```
AT+CMGR=n<CR>          il modulo risponderà:
+CMGR: "REC UNREAD","+391231231231","","14/03/15,17:12:33+04"
Ciao Mauro
OK
```

In seguito potremmo decidere di eliminarlo dalla SIM con il comando:

```
AT+CMGD=n<CR>
```

Andiamo un po' più nello specifico. Chiudiamo la porta seriale con *RS232 Terminal*, e tramite il TAB Display scegliamo il data formato *Integer* anziché ASCII, noteremo che anche la spunta dalla sezione "*Special Formatting*" inerente a "*Ignore 0-31 ASCII Codes(except CR an LF)*" viene eliminata in modo che tutti i caratteri ricevuti vengano evidenziati a video. Selezionano invece "*Insert Space between bytes*" in modo da inserire uno spazio tra ogni carattere ricevuto. Inoltre impostiamo la velocità a 9600 baud (da questo momento verrà sempre utilizzata questa velocità, anche nei firmware allegati all'articolo). La videata del PC dovrebbe risultare come riportato in Figura 15.

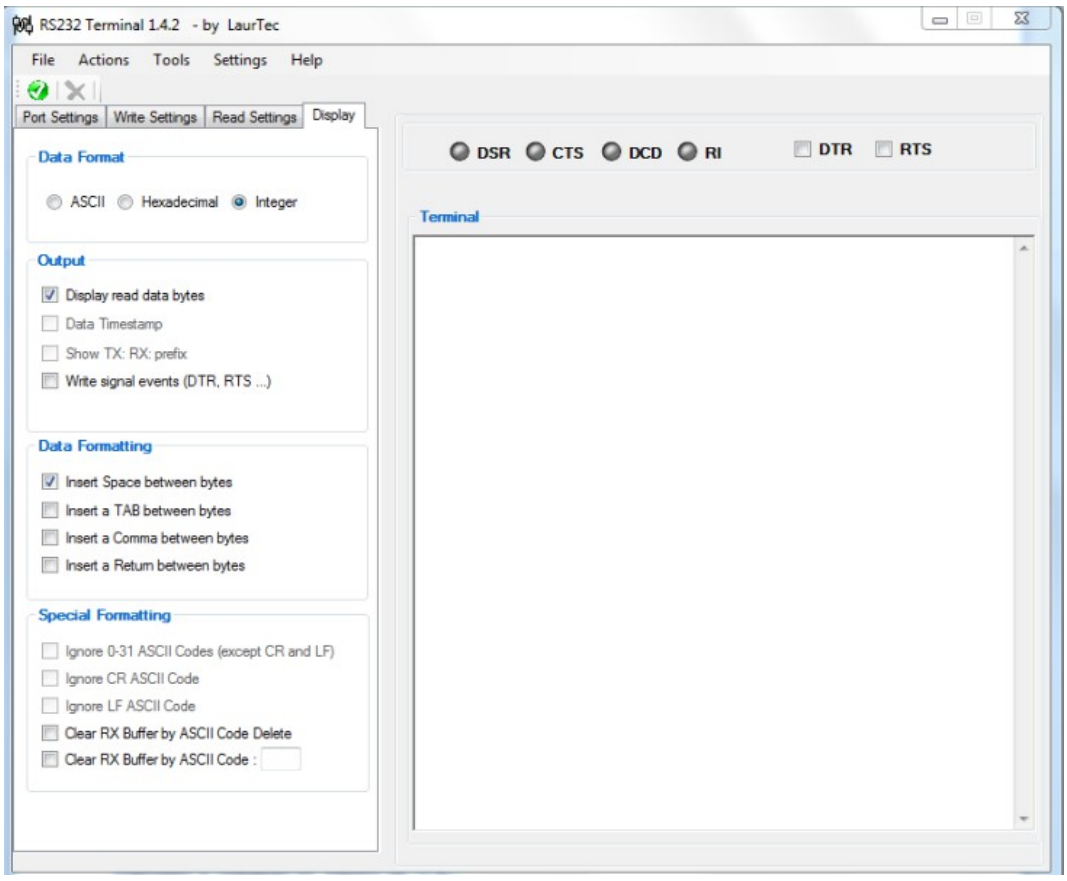


Figura 15: *Schermata di RS232 Terminal con le nuove impostazioni.*

Apriamo nuovamente la comunicazione con la porta seriale. Digitiamo il comando:

```
AT  <CR>
```

Il video dovrebbe mostrare la risposta con la rappresentazione intera.

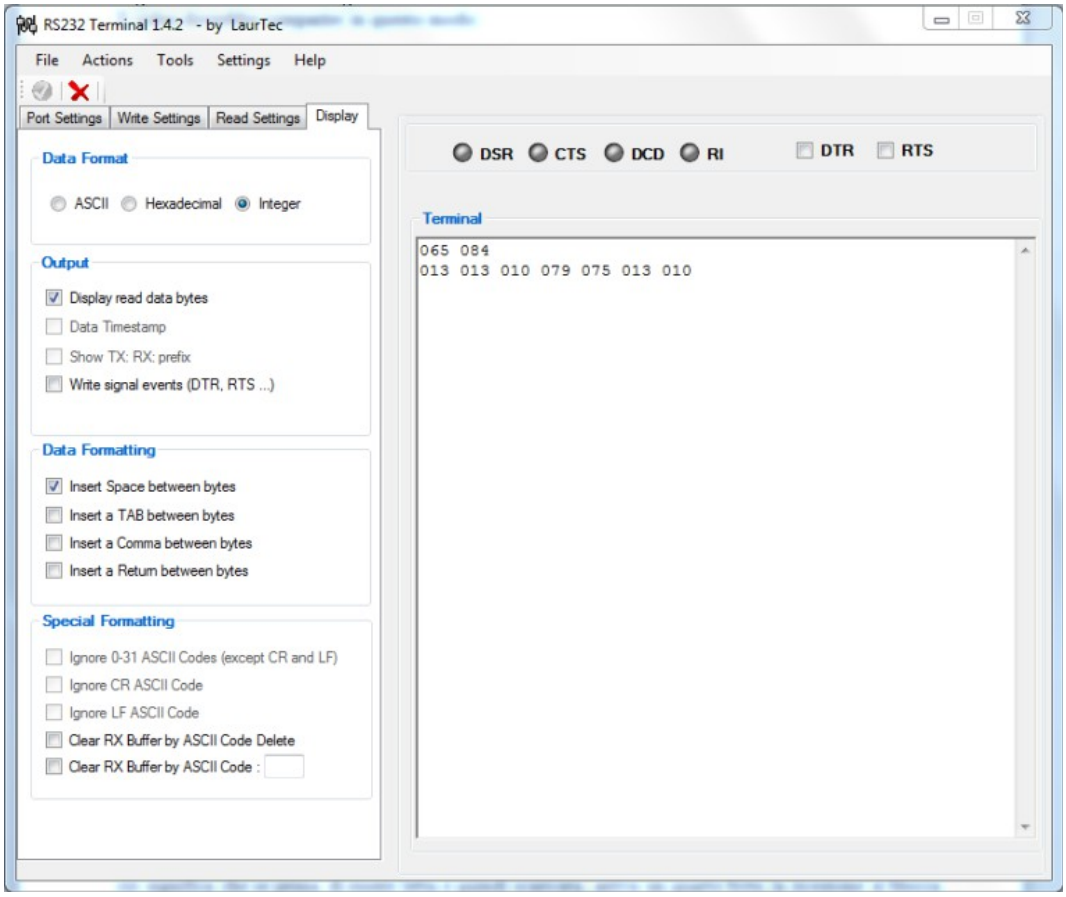


Figura 16: *Schermata di RS232 Terminal con la rappresentazione intera.*

In particolare il codice ASCII 065=A, 84=T per cui AT, mentre la risposta è cosi composta:

```
013 = \r
013 = \r
010 = \n
079 = O
075 = K
013 = \r
010 = \n
```

Per cui: \r\r\nOK\r\n

Dovendo analizzare la risposta del modulo tramite un microcontrollore, dovremmo quindi tenere in considerazione che prima della risposta effettiva vengono inviati 3 caratteri che normalmente risultano invisibili a *RS232 Terminal*, a meno di non selezionare le opportune impostazioni.

Colleghiamo ora il PIC al modulo GSM. Impostiamo quindi i minidip n. 2-4-6 della scheda d'interfaccia GSM su ON. In questo modo abbiamo collegato il PIC con il GSM ed impostato il PC a leggere i comandi inviati dal PIC.
La velocità utilizzata in tutti i programmi sotto riportati è di 9600 bit/s. Ritengo sia una velocità adeguata alla gestione di SMS o squilli in quanto considerato che un SMS ha una lunghezza max di 140byte, con una velocità di 9600 bit/s, si possono gestire circa 7 SMS in un secondo. Personalmente credo che, mettendomi d'impegno, riesco ad inviare un SMS di 20/30 byte ogni 15/20 secondi!
Comunque, quando facevo le prove per il campanile, ho utilizzato una velocità di 115200 bit/s ma a volte la porta USART del PIC andava in blocco. Infatti la porta USART del PIC184550 ha un buffer di 2 byte, per cui se dovessero arrivare tre byte prima che il buffer sia stato letto, la ricezione si blocca e viene attivato il bit di Overrun del registro RCSTA:

```
RCSTAbits.OERR
```

per azzerare l'errore occorre, come indicato nel Datasheet, disabilitare e riattivare nuovamente la ricezione della porta USART tramite le seguenti istruzioni:

```
RCSTAbits.CREN = 0;
RCSTAbits.CREN = 1;
```

In Tabella 4 sono riportati i dettagli del registro RCSTA interno al PIC18F4550.

Proprio per verificare se si è verificato un errore di *overflow* della porta USART, nella stesura dei programmi successivi, nei quali è prevista la lettura del buffer USART, ho inserito il seguente codice:

```
if (RCSTAbits.OERR) {
            Beep(100, 5);
            ClearLCD();
            WriteStringLCD("ERRORE USART GSM");
            goto_line_LCD(2);
            WriteStringLCD("BUFFER OVERFLOW!");
            delay_ms(2500);
            RCSTAbits.CREN = 0;
            RCSTAbits.CREN = 1;
            ClearLCD();
}
```

Procediamo con il collegamento PIC-GSM, accertiamoci quindi che il modulo GSM sia già acceso oppure attiviamolo tramite il pulsante ON/OFF. (Per semplificare l'esercizio la procedura per accendere il modulo tramite PIC e verificarne le impostazione verranno inserite solo nel programma finale).
Chiudiamo la porta seriale tramite *RS232 Terminal*, riselezioniamo la modalità di ricezione in ASCII anziché *Integer* e dopo aver impostato la velocità a 9600 bit/s, riapriamo la comunicazione seriale.

Registro RCSTA: Receive Status and Control Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R/W-0
SPEN	**RX9**	**SREN**	**CREN**	**ADDEN**	**FERR**	**OERR**	**RX9D**
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Leggenda			
R = Readable bit	W = Writable bit	U = Unimplemented bit read as 0	S : Settable bit
-n = Value at POR	1 = Bit is set	0 = Bit is cleared	x = Bit is unknown

Bit 7 **SPEN** : Serial Port Enable Pin
1 : Porta Seriale abilitata
0 : Porta Seriale disabilitata

Bit 6 **RX9 :** 9 bit – Receive enable bit
1 : Ricezione a 9 bit
0 : Ricezione a 8 bit

Bit 5 **SREN :** Single Receive Enable Bit

Modalità Asincrona

Ignorato

Modalità Sincrona

1 : Ricezione singola abilitata
0 : Ricezione singola disabilitata

Bit 4 **CREN** : Continuous Receive Enable bit

Modalità Asincrona

1 : Ricevitore abilitato
0 : Ricevitore disabilitato

Modalità Sincrona

1 : Abilita ricezione continua fino a quando CREN è posto a 0
0 : Ricezione continua disabilitata

Bit 3 **ADDEN** : Address Detect Enable bit

Modalità Asincrona

1: Abilita riconoscimento degli indirizzi
0: Disabilita il rilevamento degli indirizzi

Modalità Sincrona

Ignorato

Bit 2 **FERR** : Framing Error bit
1: Errore di Frame
0: Non è presente alcun errore di Frame

Bit 1 **OERR** : Overrun Error bit
1 : Errore di Overrun
0 : Non è presente alcun errore di Overrun

Bit 0 **RX9D** : 9th bit of Received Data
Valore del nono bit ricevuto

Tabella 4: *Registro RCSTA del PIC18F4550.*

Primo programma di esempio : TestInvioAT

Proseguendo nella nostra esercitazione possiamo ora caricare nel PIC il Firmware realizzato con MPLABX e XC8 e denominato TestInvioAT allegato a questo articolo.
Si presume che si siano già letti i vari tutorial scritti da Mauro Laurenti e quindi tralascio la spiegazione di come scaricare il Firmware nel PIC. Le librerie utilizzate sono le LaurTec 3.2.0 scaricabili sempre dal sito www.LaurTec.it e scompattate nella directory C:\LAURTEC_PIC_libraries_v_3.2.0 .
TestInvioAT è un semplice programma che invia il comando AT (AT\r\n), sulla seriale, alla semplice pressione del tasto BT1 o BT2 della scheda Freedom II.
Avviato il programma, leggeremo sul display la frase:

Il programma sarà quindi pronto ad eseguire i nostri comandi tramite i pulsanti BT1 e BT2. Premiamo uno dei due pulsanti e verifichiamo che su *RS232 Terminal* compaia la scritta AT, indipendentemente dal pulsante premuto.
Tutto OK? Spero proprio di si altrimenti occorre verificare nuovamente tutti i collegamenti e le impostazioni effettuate.
Per il momento il PIC non controllerà la risposta del modulo ma, se desideriamo farlo, potremmo verificarla eventualmente sul PC posizionando momentaneamente su OFF il dip switch n. 4 e su ON quello n. 3, collegando in questo modo la ricezione del PC sull'uscita del GSM; verifichiamo quindi che ad ogni pressione di uno dei due pulsanti sul video compaia la scritta OK.
Per proseguire l'esercizio occorre rimettere i mini DIP sulla posizione iniziale e cioè 2-4-6 su ON. Partendo dal presupposto che tutto quanto funzioni correttamente andiamo ora a vedere la differenza che intercorre tra la pressione di BT1 e BT2 seppur *RS323 Terminal* non abbia evidenziato differenze.
Se analizziamo il file main.c del programma notiamo che l'unica differenza tra la pressione di BT1 e BT2 è che uno invia la stringa desiderata al modulo GSM tramite l'istruzione:

```
putrsUSART((char*) "AT\r\n");
```

mentre BT1 la invia con:

```
InviaStrUSART((char *) "AT\r\n");
```

La funzione `putrsUSART` fa parte della libreria USART del compilatore XC8 mentre il codice di `InviaStrUSART` è stato creato dal sottoscritto ed inserito nel main.c del programma.
A parte la struttura delle due funzioni, la differenza sostanziale tra di loro è che con `putrsUSART` viene inviato anche il carattere di fine stringa \0 mentre ciò non avviene con `InviaStrUSART`.
Dal punto di vista pratico, in seguito alle prove effettuate, la SIM_340 non rileva alcuna differenza tra l'utilizzo delle due funzioni, in pratica non rileva il carattere \0 , infatti nei miei progetti ho sempre utilizzato solo questa funzione.
La SIM_900 rileva invece degli errori quando si utilizza `putrsUSART` per inviare dei comandi composti da più variabili e stringhe. Facciamo un esempio, analizziamo la

ricezione di un SMS. Il modulo ricevendo un SMS invia al PIC la seguente stringa:

```
+CMTI: "SM",n
```

In cui "n" indica il numero per identificare il messaggio sulla SIM(SM) che questo esempio presupponiamo corrisponda a n =3.

Nel programma del PIC, per gestire il messaggio, leggo tale stringa e memorizzo quindi il carattere "3" in una variabile che ho definito `RichiamaSMS`. Per richiamare il messaggio devo inviare il seguente comando AT:

```
AT+CMGR=3\r\n
```

invio quindi al PIC le seguenti istruzioni:

```
PutrsUSART((char*)"AT+CMGR=");
WriteUSART(RichiamaSMS);
delay_ms(10);
PutrsUSART((char*)"\r\n");
```

Se provate con la SIM_340 tutto ok mentre con la SIM_900 viene riscontrato un errore, perché?
Semplicemente perché la SIM_900 rileva il carattere di fine stringa \0 e quindi il comando ricevuto sarà il seguente:

```
AT+CMGR=\03\r\n\0
```

Tutto questo l'ho riscontrato facendo prove pratiche, non escludo che vi sia, tramite dei comandi AT, la possibilità di settare diversamente la SIM_900.
Ho quindi creato la funzione `InviaStrUSART` che invia le stringhe senza il carattere \0, per cui le istruzioni composte da più parti, utilizzate nel programma finale sono le seguenti:

```
InviaStringaUSART((char*)"AT+CMGR=");
WriteUSART(RichiamaSMS);
delay_ms(10);
InviaStringaUSART((char*)"\r\n");
```

Se volete verificare la differenza tra le due funzioni potrete farlo tramite *RS232 Terminal* settando nuovamente il formato di ricezione carattere in *Integer*, infatti in questo formato vengono mostrati tutti i caratteri, anche quelli speciali di formattazione.
Tengo a precisare che nei comandi AT inviati in una singola istruzione come ad esempio:

```
PutrsUSART((char*)"AT+CMGR=3\r\n");
```

il carattere di fine stringa \0 è irrilevante per entrambi i moduli.
La soluzione da me adottata per inviare comandi AT composti da più stringhe non è di sicuro l'unica possibile e probabilmente neppure la migliore, come in molti casi, possono esserci più metodi che portano allo stesso risultato.

Questo esempio ci ha permesso di :

- Verificare la trasmissione dati dalla Freedom II verso il modulo GSM.
- Controllare tali trasmissioni tramite *RS232 Terminal.*
- verificare la risposta del modulo sempre attraverso *RS232 Terminal.*
- Spiegare la differenza tra le funzioni `PutrsUSART` e `InviaStringaUSART`.

Secondo programma di esempio: TestRxUsart

Passiamo ora ad eseguire il secondo programma di Test che serve a controllare che il PIC rilevi correttamente le stringhe di comandi e/o risposte inviate dal modulo GSM, il nome del programma è `TestRxUsart`. Con esso inoltre simuleremo un errore di overflow del buffer di ricezione USART. Iniziamo impostando i minidip switch come mostrato nella Figura 17 in modo tra collegare tra di loro il PIC ed il modulo GSM.

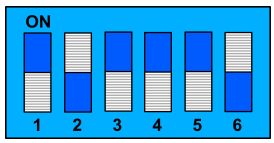


Figura 17: *Impostazione dei minidip switch.*

Scarichiamo a avviamo quindi il programma, controllando che il modulo sia già stato avviato tramite il tasto PWRKEY o ON/OFF . Inizialmente verrà visualizzata la seguente scritta sul display LCD:

In seguito verranno inviati automaticamente alla velocità di 9600 baud due comandi al modulo, il primo:

```
AT <CR>
```

per far auto apprendere la velocità di trasmissione ed il secondo:

```
ATE0 <CR>
```

per eliminare l'eco locale dei caratteri ricevuti da parte del modulo.
Rispetto al programma precedente è stato aggiunto un Interrupt denominato `RxUsart` ad alta priorità collegato alla porta USART del PIC, tale funzione ha un duplice compito, il primo di riempire l'Array `DataUsart[]` con i caratteri ricevuti dal modulo GSM ed il secondo di attivare il `FlagDataUsart` se l'Array supera i 16 caratteri (0-15) oppure se rileva in sequenza d'ingresso, i due caratteri memorizzati in precedenza dalla funzione `AzzeraDataUsart()` nelle due variabili `CarOld` e `CarAtt`; al fine di non considerare i caratteri \r\r\n, inviati dal modulo prima di trasmettere le nuove stringhe. Tale confronto avviene solo dopo l'arrivo del terzo carattere.
In questo esercizio le variabili `CarPrec` e `CarAtt` corrispondono sempre a *\r (ASCII=13)* e *\n(ASCII=10)* e quindi il `FlagDataUsart` verrà attivato (portato ad un valore logico 1) quando, dopo il terzo carattere arriverà in ingresso tale sequenza, oppure se la stringa supera i 16 caratteri (16 è stato scelto per visualizzare in una sola riga del display il valore

contenuto). In pratica quindi, se il PIC riceve dall'USART la seguente stringa:
\r \r \n OK \r \n il `FlagDataUsart` verrà attivato e l'Array `DataRxUsart[ ]` conterrà tale sequenza.
Quando nell'esecuzione del ciclo infinito del main.c verrà trovata attivo `FlagDataUsart`, verranno visualizzati sul display, per 3 secondi, i caratteri contenuti in `DataRxUsart` ad esclusione di quelli aventi un valore decimale inferiore a 31 corrispondenti ai caratteri speciali (esempio \r \n), ed in seguito, richiamando la funzione `AzzeraDataUsart()` verrà azzerato nuovamente il `FlagDataUsart` ed il contenuto di `DataRxUsart[]` in modo da essere pronti a ricevere nuovi caratteri in ingresso.

Per testare il tutto basta premere il pulsante BT1, che invia il comando "AT" al modulo. Sulla seconda riga del display dovrà apparire la scritta OK mentre se si preme BT2 che invia "AT+CGMM\r\n", sul display apparirà il nome del modulo utilizzato.
Vi consiglio di fare anche prove di ricezione con altri comandi, infatti potete collegare il modulo in modo che riceva i comandi dal PC anziché dal PIC e leggere sul display le risposte inviate dal GSM. Per fare questo dovete procedere nel seguente modo: mettere su ON solo i seguenti minidip: 2/5, avviate *RS232 Terminal* aprendo la porta seriale a 9600 bit/s dopo aver selezionato, dalla scheda *Write Settings*, la spunta su *Local Echo* in modo da visualizzare i caratteri digitati.
Consiglio di iniziare inviando un semplice comando quale ad esempio "AT" e verificare che sul display della scheda Freedom II compaia la risposta OK.

Un'altra prova interessante che consiglio di effettuare è di simulare un errore di *overflow* del buffer USART del PIC, ovvero inviamo ad esso più di tre caratteri prima che venga letto tramite l'interrupt. Per fare questo dobbiamo collegare tra di loro il PC e il PIC mettendo su ON solo i minidip switch n. 1 e n. 4, possiamo farlo anche senza riavviare il tutto, ma avendo l'accortezza di porre su OFF tutti i minidip prima di selezionare quelli da posizionare su ON.
Prima di procedere alla simulazione vera e propria proviamo ad inviare dei caratteri al PIC dalla tastiera e verifichiamo che dopo il sedicesimo invio compaia sul display la stringa che abbiamo composto. Possiamo anche visualizzare la stringa prima dell'invio del sedicesimo carattere inviando al PIC i caratteri \r \n tramite la pressione dei tasti "*Return*" e "Ctrl+Return" .
Passiamo ora alla simulazione vera e propria, modifichiamo il programma inserendo nell'interrupt l'istruzione `delay_s(1)` come sotto indicato:

```
_interrupt(high_priority) void RxUsart(void) {

         if (PIR1bits.RCIF == 1) {
        // Controllo che l'interrupt sia stato generato dall'USART
         DataUsartOld = DataUsart[Rx - 1];

       // Leggo il dato dal buffer di ricezione
       DataUsart[Rx] = ReadUSART();

        delay_s(1);

       if (FlagDataUsart == 0) {
            if ((DataUsartOld == CarPrec && DataUsart[Rx] == CarAtt && Rx
> 3)) {

                if (Rx > 3) {
                // esclude avanzamento riga (13 + 10)inviato dal
```

```
                // modulo gsm prima del comando
                FlagDataUsart = 1;

...il programma continua
```

In questo modo abbiamo inserito un ritardo di un secondo nell'esecuzione dell'interrupt e quindi durante questo lasso di tempo il buffer non verrà più scaricato.
Scarichiamo il Firmware appena modificato nel PIC ed avviamo l'esecuzione del programma. Avviamo *RS232 Terminal* abilitando il "*Local Echo*" e apriamo la porta seriale a 9600 bit/s, proviamo quindi a digitare velocemente 4/5 caratteri. Dopo qualche secondo sul display comparirà:

Anche se sembra un errore molto banale, non è assolutamente da trascurare e bisogna tenerne sempre conto, infatti quante volte sul forum di www.LaurTec.it abbiamo avuto segnalazioni che sulla porta seriale si riesce a ricevere solo i primi caratteri inviati e poi si ha il blocco della ricezione?
Bisogna valutare attentamente la velocità di trasmissione, la frequenza del quarzo utilizzato e cercare di realizzare il codice degli interrupt nel modo più breve possibile in modo da non ritardare la successiva lettura del buffer.

Programma di esempio finale: PIC_GSM

Bene, se siete giunti a leggere questa riga vuol dire che siete veramente interessati all'argomento e quindi passiamo ora al programma finale che ci permetterà di controllare i due LED collegati alle uscite RE1 e RE2 della Freedom II, tramite il cellulare!
Prima di scaricare nel PIC il programma occorre modificare l'Array `NumeroUtente[]`.
Questa variabile dovrà contenere il numero di cellulare dell'utente che come descritto in seguito sarà abilitato ad eseguire i comandi e a ricevere degli SMS di segnalazione, dovete cioè inserire il vostro numero di cellulare.
Il programma è progettato per funzionare con un numero utente avente una lunghezza tassativa di 10 numeri più il prefisso internazionale per un totale di 13 caratteri.

Funzionalità dei tasti BT1 e BT2 sulla Freedom II

Il tasto BT1, tramite il relativo `FlagContrNumero`, serve a impostare se si desidera che solo l'utente sia abilitato ad eseguire i comandi tramite squilli o SMS.
In pratica, solo il cellulare del chiamante, avente lo stesso numero memorizzato nella variabile `NumeroUtente[]`, è abilitato ad eseguire i comandi. Di default il Flag è disabilitato per cui se si vuole selezionare tale opzione occorre premere il tasto BT1, per disattivarla basterà premere nuovamente il tasto. Se, con il Flag abilitato, il modulo riceve uno squillo o un SMS da un numero diverso dal `NumeroUtente[]` registrato, il modulo non eseguirà alcun comando ma invierà al numero dell'utente un SMS in cui il testo sarà composto dalla seguente stringa: "E ARRIVATO UN SMS/SQUILLO DAL SEGUENTE NUMERO:+39numero chiamante" in modo da avvisare l'utente che qualcuno ha provato ad accedere al modulo GSM.
Il tasto BT2, tramite il relativo `FlagInviaSms`, serve invece a inibire l'invio di SMS da parte del modulo. Questa opzione è stata inserita per non consumare del credito residuo

dalla SIM durante le prove di funzionamento. Il `FlagInviaSms` di default è disattivo e quindi in caso di richiesta di invio SMS da parte del programma, sul display verrà visualizzata la scritta:

E successivamente comparirà:

Per abilitare l'invio di SMS da parte del modulo basta premere il tasto BT2 mentre, se si preme nuovamente, verrà ripristinata l'opzione precedente.

- **LED verde (output RE1)**
  Il LED verde viene comandato tramite squilli. Se il `FlagContrNumero` è attivo il comando viene eseguito solo se il numero del chiamante corrisponde a quello memorizzato nella variabile `NumUtente[]`. Alcuni secondi dopo l'accensione del LED verrà fatto squillare il cellulare dell'utente a conferma dell'avvenuta attivazione, se lo squillo ha fatto spegnere il LED non verrà effettuata nessuna chiamata di ritorno. Oltre alla chiamata di ritorno vi è un altro modo per sapere se si sta accendendo o spegnendo il LED, basta contare i numeri di squilli prima di ricevere il distacco della linea. Infatti se il LED è spento verranno conteggiati 3 squilli prima di staccare la comunicazione ed attivare il LED mentre quando si sta spegnendo il LED se ne conteranno solo due.

- **LED rosso (output RE2)**
  Il LED rosso viene pilotato attraverso degli SMS. Anche in questo caso vale il discorso fatto precedentemente sul controllo del numero del chiamante, stabilito dalla variabile `FlagContrNumero`. Per attivare i comandi, il contenuto degli SMS deve contenere determinati caratteri che sono stati precedentemente definiti nel Firmware. I comandi predefiniti attuali sono:

  ON accende il LED rosso
  OFF spegne il LED rosso

  Il testo degli SMS può essere inviato indifferentemente in maiuscolo o minuscolo.
  Se si invia un SMS con il codice corretto il modulo invierà al chiamante un messaggio di conferma mentre in caso contrario verrà inviato un SMS di segnalazione di errore. In Figura 18 potete osservare il diagramma di flusso del programma contenuto nel file main.c.

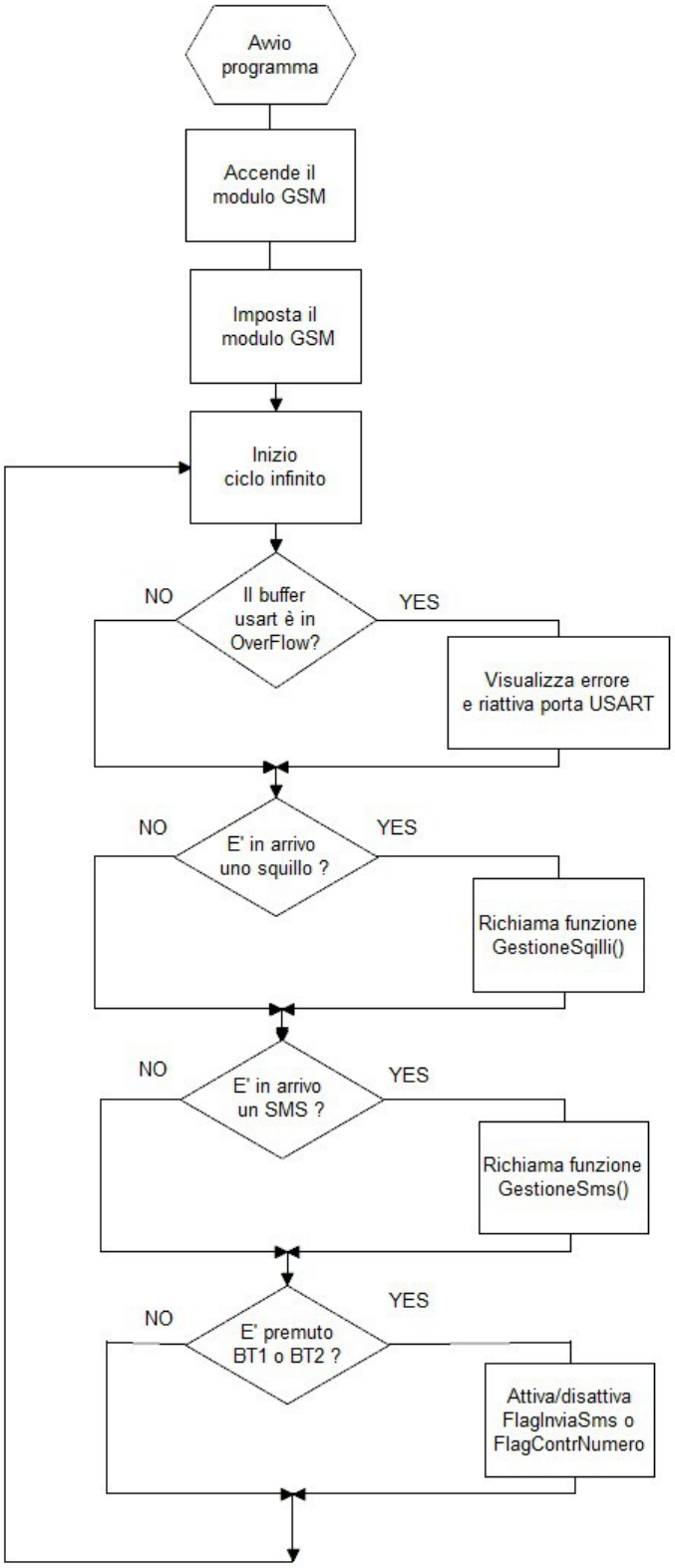


Figura 18: *Diagramma di flusso del programma di esempio.*

Le Funzioni

Nel programma sono presenti le seguenti funzioni:

- VerificaGsmOnOff(void);
- ImpostaGsm(void);
- AzzeraDataUsart(unsigned char, unsigned char);
- ConfrontaDataUsart(char, char*);
- AttendiRispostaUSART(unsigned int);
- GestioneSMS(void);
- GestioneSquilli(void);
- InviaSms(unsigned char, unsigned char);
- InviaSquillo(void);
- InviaStringaUSART(char*);
- Beep(int Durata, unsigned char NumVolte);

VerificaGsmOnOff(void)

Invia un commando AT al modulo e controlla che questi risponda OK, se ciò non avviene significa che il modulo è spento, oppure è in blocco, o è in attesa che venga finita un'istruzione quale ad esempio la fine della composizione del testo di un SMS. Se non viene ricevuto l'OK il PIC attiva l'uscita PWRKEY e riprova nuovamente. Se il modulo era in blocco o in attesa di finire un comando, e quindi non ha risposto OK al comando, con questa operazione il modulo viene spento anziché riacceso ed ecco quindi il motivo per cui il ciclo viene ripetuto per due volte in modo che al "secondo giro" il modulo venga riacceso.

ImpostaGsm(void)

In questa funzione, tramite i relativi comandi AT, il modulo viene impostato:

- Si abilita a identificare il numero del chiamante.
- Invia SMS in modalità testo.
- Cancella dalla SIM gli SMS dal n° 1 al n° 3.
- Restituisce il nome della SIM installata.
- Richiede e restituisce il nome gestore con cui si è registrati.
- Controlla il livello di copertura del segnale GSM ricevuto.

tutte queste segnalazioni vengono visualizzate sul display, ed in caso di malfunzionamento o di risposta non corretta viene indicato che il modulo non funziona correttamente.

AzzeraDataUsart(unsigned char UltimoCar, unsigned char CarAttuale)

Questa funzione serve ad azzerare l'Array `DataRxUsart[]` e memorizza nelle variabili `CarPrec` e `CarAtt` i due caratteri che attiveranno, tramite l'interrupt, il `FlagDataUsart`. Nella maggior parte delle volte tali caratteri corrispondo a 13 e 10 (*Return* e *Line Feed*) cioè i caratteri che il modulo invia dopo una risposta tranne che durante il richiamo di un SMS ricevuto, nel cui caso si attende l'arrivo dei caratteri 'K' e 10 ovvero, considerando che la risposta da analizzare è la seguente:

```
+CMGR: "STO UNSENT","+391231231231",""
Testo del SMS
OK\r\n
```

si attiverà il `FlagDataUsart` allorché i caratteri in arrivo corrisponderanno a 'K' e \r cioè quando verrà intercettato la lettera K, finale di OK, e \r ovvero la fine della stringa ricevuta.

ConfrontaDataUsart(char Inizio, char* str3)

Questa funzione confronta una stringa, che verrà memorizzata nella variabile `str3`, con il contenuto di `DataRxUsart` a partire da un determinato carattere indicato dalla variabile *Inizio*. Ad esempio quando si invia il comando AT+COPS? Il modulo ritornerà il nome del gestore a cui si è registrati in questo modo:

```
AT+COPS?
\r\n+COPS: 0,0,"vodafone"
OK
```

Come di può notare il nome del gestore ha inizio dal 14° carattere di `DataRxUsart` per cui se voglio impostare una variabile che corrisponda ad esempio a 2 se il nome del gestore è "vodafone" utilizzerò il seguente codice:

```
if (ConfrontaDataUsart(14, (char*) "vodafone") != 1) {
            NumGestore = 2;
}
```

AttendiRispostaUSART(unsigned int Tempo)

Quando invio un comando, ad esempio "AT", mi aspetto che il modulo risponda "OK\r\n" entro qualche centesimo di secondo, per cui attendo l'arrivo dei caratteri \r\n per attivare il `FlagDataUsart` ma se il modulo ad esempio è spento non posso attendere all'infinito l'arrivo di questi due caratteri quindi, se si supera il tempo di attesa, indicato in decimi di secondo, contenuto nella variabile `Tempo`, si fa proseguire il programma anche se il modulo non ha inoltrato risposte. Si tratta quindi di un *Time Out* per ricevere la risposta dal modulo.
Nel programma non è stato realizzato ma si potrebbe, nel caso di Time Out, attivare una variabile di errore e segnalare a display che qualcosa non ha funzionato correttamente. Di seguito riporto un esempio di utilizzo della funzione in oggetto:

```
AzzeraDataUsart(13, 10);
putrsUSART((char *) "AT\r\n");
AttendiRispostaUSART(10);

if (ConfrontaDataUsart(2, (char*) "OK") == 0) {

    // se la risposta è OK il modulo è acceso
    // e funziona correttamente

    FlagModuloAcceso = 1;
}
```

In questo caso il programma attende la risposta del modulo per un secondo, dopodiché prosegue.

Vi sono dei comandi in cui occorre attendere la risposta per decine di secondi come ad esempio il comando AT+COPS=? (da non confondersi con AT+COPS?) che restituisce i nomi dei gestori disponibili sulla rete.

unsigned char GestioneSMS(void)

Detta funzione viene richiamata qualora si riceva dal modulo l'istruzione +CMTI, che indica l'arrivo di un SMS, come riportato nel codice seguente:

```
if (FlagDataUsart == 1) {
     if (ConfrontaDataUsart(2, (char*) "+CMTI") == 0) {
          GestioneSMS();
    }
}
```

La funzione `GestioneSMS` per prima cosa richiama il messaggio in causa tramite l'istruzione:

```
AT+CMGR  <CR>
```

alla quale il modulo risponderà inviando una stringa contenente varie informazioni. Quelle a cui noi siamo interessati sono due e precisamente il numero del chiamante ed il testo del messaggio. Come si può osservare dall'esempio sottostante:

```
+CMGR: "STO UNSENT","+391231231231",""
Testo del SMS
OK
```

il numero del chiamante inizia dopo la terza doppia virgoletta e termina prima della quarta, per cui se vogliamo che gli SMS in arrivo attivino un comando solo se il numero del chiamante corrisponde al `NumeroUtente[]` dobbiamo confrontare quest'ultimo con i caratteri contenuti tra la terza e quarta virgoletta..
In questo programma il numero del chiamante viene confrontato solo se la variabile `FlagContrNumero` è attiva.
Nel codice del programma, il numero del chiamante è stato definito con una lunghezza fissa di 13 cifre compreso il codice internazionale (+39 nel caso dell'Italia) per cui questo programma funziona correttamente solo con cellulari aventi un numero di 10 cifre + prefisso internazionale. Il testo del messaggio viene invece memorizzato dopo l'arrivo del carattere 13 (*Return*) per cui tutti i caratteri contenuti dopo di esso verranno memorizzati nell'Array `TestoSMS[]`.
I caratteri, dopo essere stati memorizzati, vengono trasformati in maiuscolo e, in questo programma, vengono confrontati i primi due o tre per verificare se corrispondono a dei valori prefissati per attivare i relativi comandi, ad esempio:

```
if ((TestoSMS[0] == 'O') && (TestoSMS[1] == 'N')) {
        if (LedRosso == 1) {
            InviaSms(4);
        } else {
            LedRosso = 1;
            CodiceOk = 1;
            InviaSms(1);
        }
}
```

Se i caratteri non corrispondono a nessun comando verrà allora inviato al mittente, tramite la funzione `InviaSMS()` un messaggio con la dicitura: "CODICE SMS NON RICONOSCIUTO". Infine il messaggio arrivato, verrà eliminato dalla SIM tramite l'istruzione:

```
AT+CMGD.
```

GestioneSquilli(void)

Quando nel ciclo infinito del main.c si riceve dal modulo la stringa "RING" seguita da "CLIP" come da esempio sotto riportato:

```
RING

+CLIP: "+391231231231",145,"",,"",0
```

significa che è in arrivo una chiamata e quindi viene richiamata la funzione `GestioneSquilli()`.

Se il `FlagContrNumero` è attivo vengono quindi confrontate le 13 cifre a partire dalla posizione 18 di questo Array con quello contenente il numero, precedentemente memorizzato in `NumUtente[]`, abilitato alla gestione squilli.

Se tale confronto ha esito positivo viene quindi incrementata la variabile `ContNumSquilli`; quando tale variabile supera il valore stabilito viene interrotta la comunicazione con il chiamante tramite il comando:

```
AT+ATH
```

e si esegue la relativa operazione. Nel nostro programma, se il LED è spento esso verrà acceso dopo il terzo squillo, mentre in caso contrario, esso verrà spento dopo l'arrivo del secondo squillo. In questo modo, il chiamante ha la possibilità, tramite il conteggio degli squilli eseguiti, di sapere se si sta accendendo o spegnendo il LED. A titolo di esempio, nel programma, al momento dell'accensione del LED viene anche effettuata un chiamata di ritorno al chiamante dopo 5 secondi, in modo da avvisarlo dell'avvenuta esecuzione del comando.

InviaSms(unsigned char TipoSms, unsigned char SceltaNum)

Quando viene richiamata questa funzione il modulo GSM invia un SMS il cui testo è identificato dalla variabile `TipoSms` al numero abbinato all'altra variabile `SceltaNum`. Si tratta quindi di un messaggio che viene inviato dal modulo SMS, tramite il comando:

```
AT+CMGS <CR>
```

In seguito viene controllata la risposta che il modulo deve trasmettere al PIC (+CMGS) dopo aver inviato il messaggio, se tutto ok compare sul display la scritta:

```
SMS INVIATO  CORRETTAMENTE
```

altrimenti si leggerà

```
SMS NON INVIATO CORRETTAMENTE
```

Anche nella lettura della conferma dell'invio SMS (+CMGS) ho constatato una differenza tra i due moduli, infatti nel caso della SIM_340 tale risposta è contenuta a partire dal carattere 2 dell'Array `DataUsart[]` mentre con la SIM_900 parte dal n° 5.
Se il `FlagInviaSms` non è stato attivato, tramite il pulsante BT2, il messaggio non verrà inviato ma sul display comparirà il numero di messaggio che si sarebbe dovuto spedire ed il numero del destinatario.

InviaStringaUSART(char* StrUsart)

Come già spiegato, questa funzione invia sulla porta USART la stringa che gli viene passata tramite la variabile `StrUSart` ad eccezione del carattere di fine stringa \0.

InviaSquillo(void)

Questa funzione effettua una chiamata al numero utente, per una durata di 9 secondi .
La chiamata si effettua con il comando ATD seguito dal numero che si desidera chiamare e con il carattere ; (punto e virgola) come chiusura comando. Per terminare la chiamata basta inviare il comando ATH.

Beep(int Durata, unsigned char NumVolte)

La funzione `Beep` serve per controllare il cicalino della scheda di sviluppo Freedom II, facendolo suonare ripetutamente per `NumVolte` con una durata del suono, determinato in ms, stabilito dalla variabile `Durata`.

Esecuzione del programma finale

Per provare il programma finale dobbiamo mettere i minidip switch n. 2 e n. 6 su ON in modo da collegare il PIC al modulo GSM, è inoltre possibile verificare da *RS232 Terminal* i comandi inviati dal PIC mettendo il minidip switch n. 4 su ON oppure in alternativa quelli inviati dal modulo GSM posizionando il minidip switch n. 3 su ON.
Scarichiamo quindi il Firmware nel PIC ed avviamo l'esecuzione. Prima di tutto verrà verificato se il modulo è acceso inviando il comando "AT" e controllata la risposta "OK", in caso negativo verrà inviato un segnale sull'uscita PWRKEY e dopo un attesa di 10 secondi si riproverà ad interrogare il modulo.

Se tutto è OK verranno inviate le impostazioni al modulo visualizzandole sul display e quindi si entra nel ciclo infinito del main.c controllando se sono in arrivo dei comandi.
Per prima cosa consiglio di fare uno squillo al modulo, accendendo il LED verde. Durante la telefonata sul display verrà visualizzato il numero del chiamante ed il numero di squilli ricevuti e dopo la ricezione del terzo squillo il LED verde si accenderà, a conferma dell'avvenuta accensione il modulo invierà uno squillo di risposta al chiamante. Se tutto ok provate a spegnere il LED richiamando il modulo.
Successivamente tramite il tasto BT1 abilitate il controllo del numero del chiamante e verificatene il funzionamento inviando uno squillo dal numero utente ed uno da un numero diverso.

Se il numero corrisponde si ripeterà il ciclo precedentemente descritto altrimenti non verrà eseguito alcun comando e come già detto verrà inviato un SMS di segnalazione al numero utente.

Passiamo ora al LED rosso! Consiglio di premere nuovamente il tasto BT1 in modo da disabilitare il controllo del numero del chiamante e proviamo quindi ad accenderlo inviando un SMS al modulo con il testo "ON". Quando il modulo riceve il messaggio lo invia al PIC il quale farà comparire sul display il testo contenuto ed il numero del chiamante ed in seguito attiverà il LED. Successivamente, se l'invio degli SMS è abilitato tramite BT2, il microcontrollore invierà un SMS di conferma al chiamante. Per spegnerlo basterà inviare un SMS con il testo "OFF".

Buon divertimento e credo sia molto più divertente ed immediato verificare il funzionamento in modo pratico che non leggendo queste istruzioni.

Analisi finale

Il programma finale è stato scritto a titolo di esempio e quindi può essere sicuramente migliorato, inserendo dei controlli ulteriori, ad esempio controllando se è in arrivo una chiamata del gestore anziché dal chiamante. Si potrebbero inserire altri codici di comandi tramite SMS, ad esempio per richiedere al gestore il credito residuo, inserire una gestione degli errori che controlli ulteriormente le risposte del modulo GSM.

Come detto in apertura questo articolo deve essere considerato come un riassunto di quanto ho imparato, come autodidatta, durante la realizzazione dei miei progetti, non si tratta quindi di un testo stilato da un esperto di programmazione, basta vedere il nome che ho dato alle variabili…quasi tutte in Italiano!
Averle definite così mi sarà di aiuto se tra qualche anno dovrò apporre modifiche al codice. Non nascondo che la stesura stessa di questo articolo ha contribuito a farmi conoscere altri comandi e a correggere alcuni errori che ho fatto nei precedenti progetti realizzati...chissà quanti ce ne sono ancora!
Mi piacerebbe quindi che questo articolo induca chi ne sa più di me a migliorare questo codice o meglio ancora a pubblicare altri codici che siano stati utilizzati per gestire i moduli GSM.

Allo stesso tempo sono anche curioso di sapere se è stato utile a qualcuno per realizzare qualche progetto anche a livello didattico e quindi commentate senza paura!
Non fate domande troppo difficili altrimenti dovremo disturbare il nostro amico Mauro che ha già dedicato un bel po' di tempo alla correzione di questo articolo.

Ringraziamenti

Approfitto di questo spazio per fare alcune considerazioni:
Perché ho dedicato tutto questo tempo alla stesura di questo articolo?
Perché farlo se ho già raggiunto gli obiettivi prefissati ?
Perché farlo anche se non conosco chi leggerà il tutto ?
Perché farlo e rischiare di essere criticato per gli errori contenuti ?
Perché...?
Non importa il perché di tutte queste domande, mi basta sapere che è servito a qualcuno.
Non so se Mauro Laurenti si pone alcune di queste domande quando scrive un tutorial ma, in tal caso, gli basti sapere, che nel suo caso, quel qualcuno sono io, anche se sono sicuro di non essere il solo!

Indice Alfabetico

Bibliografia

[1] www.LaurTec.it: sito ufficiale della scheda FREEDOM II dove è possibile scaricare utili tutorial come ad esempio "XC8 step by step" e le librerie utilizzate negli esempi riportati in questo articolo.
[2]www.microchip.com : sito dove scaricare i datasheet del PIC18F4550.
[3]www.robot-italy.com: sito sul quale è possibile trovare informazioni sul modulo MODGSM.
[4]www.futurashop.it : sito sul quale è possibile trovare informazioni sul modulo TDGGSM
[5] http://wm.sim.com/defaulten.aspx : sito ufficiale della SIMCOM produttrice della SIM340 e SIM900. Sono disponibili diversi file scaricabili previa registrazione gratuita.
[6] http://xes.dyndns.org/index.php/codifica-in-pdu-di-un-sms.html
sito sul quale è presente un articolo scritto da Giovanni Peres inerente gli SMS.

History

Data	Versione	Autore	Revisione	Descrizione Cambiamento
18/11/14	1.0	Ferrero Vercelli Renato	Mauro Laurenti	Versione Originale.