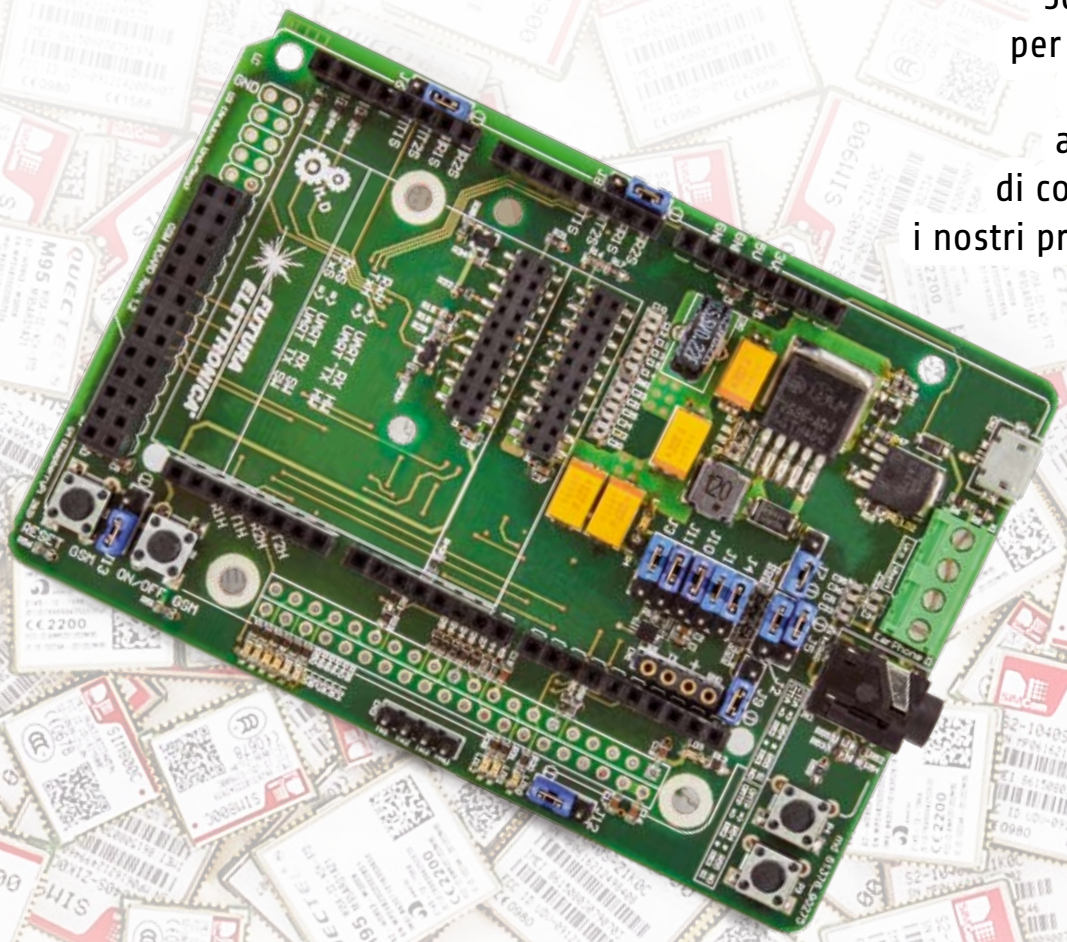
GSM SHIELD UNIVERSALE

di MATTEO DESTRO

Nella precedente puntata abbiamo introdotto l'hardware dello shield ed esposto le caratteristiche tecniche e le possibilità di utilizzo, oltre che accennato alla libreria per Arduino; qui completiamo il discorso spiegando nei dettagli il funzionamento della stessa, che, ricordiamo, supporta le schede Arduino Uno, Arduino Mega 2560, Fishino Uno e Fishino Mega 2560. La libreria è composta da un file principale più dei file satellite contenenti il codice necessario per la gestione di una serie di funzioni di invio comandi AT verso il modulo GSM installato nello shield. Il file principale è **GenericCmd_GSM.cpp** e contiene il codice di infrastruttura che tiene interconnesse le funzioni presenti negli altri file di libreria. Con una serie di direttive è possibile quindi attivare/disattivare delle parti di codice nonché selezionare la scheda Arduino desiderata e il relativo modulo GSM da associare.
I moduli attualmente supportati dalla libreria sono:

- SIM800C (SIMCOM);
- SIM900 (SIMCOM);
- SIM928A (SIMCOM);
- G510 (FIBOCOM);
- M95 (QUECTEL).

Scopriamo la libreria per gestire vari moduli GSM/GPRS low-cost allo scopo di dotare di connettività cellulare i nostri progetti con Arduino. Seconda puntata.



Ricordiamo che il SIM928A prevede la gestione di due seriali. Per tutti i moduli attualmente sono supportate solo le funzioni di gestione SMS e chiamata fonica, mentre le funzioni di gestione del GPRS per connessione Internet e l'invio/ricezione di dati sono in fase di sviluppo. In futuro verrà sviluppata anche la parte di gestione del sistema GPS.
Per meglio comprendere alcuni termini utilizzati da qui in avanti va ricordato che i moduli GSM possono essere definiti:

- ME (Mobile Equipment);
- MS (Mobile Station);
- MT (Mobile Termination);
- TA (Terminal Adapter);
- DCE (Data Communication Equipment).

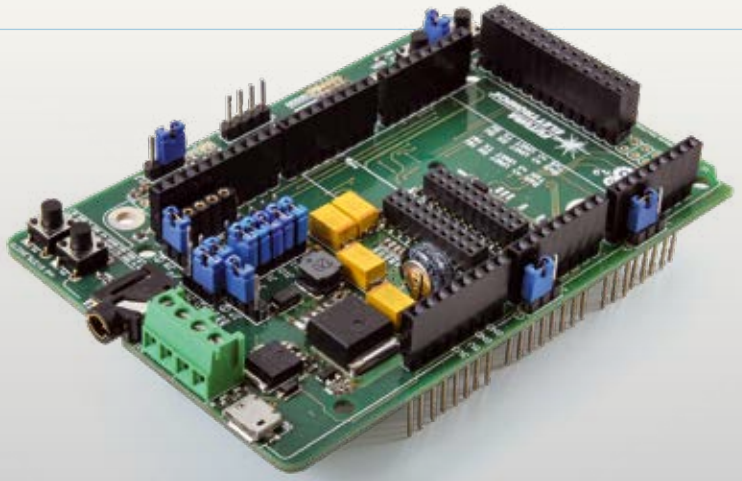
I dispositivi che controllano i moduli GSM, inviando appositi comandi AT, possono essere:

- TE (Terminal Equipment);
- DTE (Data Terminal Equipment), termine equivalente all'applicazione che gira su un sistema embedded e si occupa di gestire il modulo GSM.

LA LIBRERIA IN DETTAGLIO

La libreria è pensata per l'invio di un comando AT tramite un'apposita funzione e l'attesa della relativa risposta da parte del modulo GSM. Ciò è reso possibile da una serie di macchine a stati che prendono in carico sia l'invio del comando AT, sia la gestione della risposta da parte del modulo. Attualmente sono stati selezionati alcuni comandi AT di uso comune, che abbiamo implementato nella libreria.
Per poterla utilizzare è necessario copiare tutti i file scaricabili dal sito *https://github.com/open-electronics/GSM_Library_Arduino* nella cartella "*C:\Program Files (x86)\Arduino\libraries/GSM2*". Una volta che tutti i file sono stati copiati, la libreria deve essere inizializzata da parte dell'utente a seconda dell'hardware utilizzato e del modulo GSM selezionato. Sono state inoltre sviluppate apposite funzioni per attivare il processo di inizializzazione del modulo GSM (accensione modulo, e invio comandi AT di configurazione) il quale è totalmente automatizzato, compresa la richiesta del codice PIN (se richiesto) e invio dello stesso al modulo GSM. Notate che i codici PIN, PUK ecc. sono memorizzati nella EEPROM di Arduino, quindi, per come è strutturato un sistema Arduino, ci vorrà un apposito sketch per caricare i dati in EEPROM. L'attuale IDE Arduino, purtroppo, non permette la programmazione della EEPROM durante la compilazione del progetto.
Guardando il file associato "**GenericCmd_GSM.h**" (apribile con un editor di testo come Notepad++) troviamo in testa una serie di costanti, tutte memorizzate in FLASH tramite la direttiva "**PROGMEM**", le quali identificano una serie di comandi AT utilizzati sia per la inizializzazione del modulo GSM sia per uso comune all'interno di uno sketch. I comandi AT utilizzati nella libreria vengono *sempre* memorizzati in flash per non occupare memoria SRAM inutilmente. Infatti se si fosse usato l'approccio di definire una stringa per ogni comando AT da inviare non avremmo avuto abbastanza memoria SRAM per lo sviluppo della libreria e relativi sketch.
Sotto ai comandi AT memorizzati troviamo una serie di direttive che identificano delle costanti utilizzate nella gestione dei relativi comandi AT. Sono anche definiti i codici dei caratteri ASCII e i codici degli STEP delle macchine a stati utilizzate per l'invio dei comandi AT sia durante l'inizializzazione che a regime. Infine abbiamo definito una serie di codici di errore che potrebbero ritornare dal modulo GSM, ovvero i codici di errore CME e CMS.
In fondo a tutte le direttive descritte ritroviamo le dichiarazioni delle variabili usate e delle relative funzioni sia che esse siano pubbliche che private.
Al file appena descritto se ne abbina uno altrettanto importante, "**Io_GSM.h**", che contiene le dichiarazioni per la configurazione hardware dello shield, più la possibilità di abilitare/disabilitare le varie macchine a stati o il codice aggiuntivo di debug.
La prima cosa che si nota aprendo il file è la presenza di una tabella (in cui sono evidenziate le possibili combinazioni ottenibili con i jumper presenti sulla scheda (corrisponde alla **Tabella 1** che trovate nella prima puntata). A seconda di come questi vengono posizionati si potrà selezionare, per esempio, quale interfaccia di comunicazione utilizzare, ovvero UART hardware o software con o senza spia seriale. È anche possibile la comunicazione diretta col PC, bypassando Arduino.
Immediatamente dopo iniziano le direttive per la selezione del modulo GSM (tra quelli supportati) da utilizzare; ne può essere impostato uno solo alla volta. Per selezionare il modulo si deve decommentare la riga corrispondente lasciando le altre:

```
#define SIMCOM_SIM800C // Use this for SIMCOM  SIM800C GSM module
//#define SIMCOM_SIM900 // Use this for SIMCOM  SIM900 GSM module
//#define SIMCOM_SIM928A // Use this for SIMCOM  SIM928A GSM module
//#define FIBOCOM_G510 // Use this for FIBOCOM G510 GSM module
//#define QUECTEL_M95 // Use this for QUECTEL M95 GSM module
```



La riga in grassetto identifica il modulo GSM che si desidera usare, mentre le altre corrispondono ai moduli supportati ma non selezionati. Questo concetto vale per tutte le possibili selezioni di un singolo oggetto da un elenco di possibili scelte.
Seguono le direttive per la selezione della scheda Arduino che si intende usare: Arduino Uno o ArduinoMega 2560 (FishinoUno e FishinoMega 2560).
Segue la selezione della revisione hardware; lo sviluppo era iniziato con la versione 1.0, che ovviamente ha perso di significato. Attualmente si deve selezionare la sola versione R.1.2.
Definito l'hardware, si deve dire al compilatore se usare l'interfaccia UART hardware o software; per ora la seconda UART disponibile, sia hardware che software, non è usata ma resta disponibile per sviluppi futuri.
Infine abbiamo la sezione con l'assegnazione di un nome univoco ai pin di Arduino, la cui prima parte riguarda l'hardware revisione 1.0 (che è obsoleto), mentre subito dopo troviamo le definizioni riguardanti l'hardware revisione 1.2:

- definizione pin per Arduino Uno sfruttando le direttive "**ARDUINO_UNO_REV3**";
- definizione pin Arduino Mega 2560 sfruttando la direttiva "**ARDUINO_MEGA2560_REV3**".

Ricordiamo che è possibile selezionare una sola direttiva. Nell'elenco delle definizioni dei pin troviamo i segnali TX e RX delle UART sia software che hardware, la definizione dei LED collegati ad Arduino, nonché i segnali di trigger.
Conclusa la configurazione della parte hardware si deve passare all'attivazione o disattivazione di alcune sezioni software per la gestione delle risposte ricevute dal modulo GSM ai comandi AT inviati. Come al solito si utilizzano le direttive e in questo caso è possibile avere più sezioni abilitate contemporaneamente. Allo stato attuale abbiamo cinque possibili macchine a stati, che riguardano:

- risposte ai comandi AT generici, come ad esempio il comando AT+CFUN;
- risposte ai comandi AT per la gestione della sicurezza, come ad esempio il comando AT+CPIN;
- risposte ai comandi AT per gestire la rubrica telefonica, come ad esempio il comando AT+CPBS;
- risposte ai comandi AT per la gestione degli SMS, come ad esempio AT+CMGD;
- risposte ai comandi AT per la gestione delle chiamate foniche, come ad esempio il comando ATA.

Quindi abbiamo le seguenti *define*:

```
#define ENABLE_ANSWER_GENERIC_AT_CMD_STATE
#define ENABLE_ANSWER_SECURITY_AT_CMD_STATE
#define ENABLE_ANSWER_PHONEBOOK_AT_CMD_STATE
#define ENABLE_ANSWER_SMS_AT_CMD_STATE
#define ENABLE_ANSWER_PHONIC_CALL_AT_CMD_STATE
//#define ENABLE_ANSWER_GPRS_AT_CMD_STATE
//#define ENABLE_ANSWER_HTTP_AT_CMD_STATE
```

Per ciascuna macchina a stati esiste un codice univoco di identificazione necessario per la corretta gestione delle risposte; i codici e le definizioni vanno di pari passo, quindi quando si creano nuove sezioni di codice per gestire delle risposte ai comandi AT si dovrà sia aggiungere una definizione per l'attivazione/disattivazione del codice, sia assegnare un codice univoco alla nuova sezione. Di seguito riportiamo i codici già assegnati in libreria:

```
#define ANSWER_GENERIC_AT_CMD_STATE 0
#define ANSWER_SECURITY_AT_CMD_STATE 1
#define ANSWER_PHONEBOOK_AT_CMD_STATE 2
#define ANSWER_SMS_AT_CMD_STATE  3
#define ANSWER_PHONIC_CALL_AT_CMD_STATE 4
#define ANSWER_GPRS_AT_CMD_STATE  5
#define ANSWER_HTTP_AT_CMD_STATE  6
```

L'ultima sezione contiene le direttive di debug, che abilitano/disabilitano delle sezioni di codice all'interno della libreria per il debug dei comandi AT inviati e relative risposte, flusso dati sulla seriale selezionata, ON/OFF/RESET del modulo GSM collegato e gestione dei comandi AT di sicurezza.
Le informazioni di debug vengono stampate sul monitor seriale dell'IDE Arduino sfruttando la connessione seriale predefinita, quindi da codice dovete abilitare la seriale di debug alla velocità voluta e impostare di conseguenza il monitor seriale.
Detto ciò in testa abbiamo la direttiva globale "**DEBUG_MODE**" la quale ha la funzione di **MASTER** e se commentata disabilita l'intero codice di debug presente nella libreria. Seguono una serie di direttive **SLAVE** per selezionare cosa si vuole debuggare

VUOI SVILUPPARE LE TUE APPLICAZIONI CON ARDUINO?

Da noi trovi tutto quello che ti serve!

l'ABC di ARDUINO

COD. ABCARDU

€ 9,90

Arduino e le tecniche di programmazione dei microcontrollori Atmel

COD. ATPROMA

€ 15,00

ARDUINO UNO Programmazione avanzata e librerie di sistema

COD. ARDUADVANCED

€ 22,00

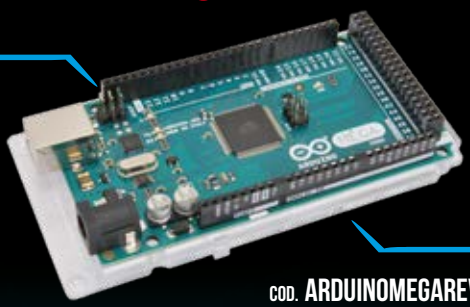
Prezzi IVA inclusa

Arduino Uno Rev3

COD. ARDUINOUNOREV3

€ 24,50

Arduino Mega



COD. ARDUINOMEGAREV3

€ 43,00

Arduino Due

ARDUINODUE

€ 44,00

kit V5 con Arduino Uno REV3

Set contenente tutti i componenti necessari per realizzare gli esperimenti descritti nel libro "L'ABC di Arduino".

COD. ARDUKITV5

€ 59,00

Starter kit per robot con Arduino Uno

Set contenente le parti necessarie per realizzare il Robot descritto nel libro incluso.

€ 155,00

COD. ARDUKITV4

FUTURA ELETTRONICA®

www.futurashop.it

Futura Group srl • via Adige, 11 • 21013 Gallarate (VA)
Tel. 0331/799775 • Fax. 0331/792287

Caratteristiche tecniche di questi prodotti e acquisti on-line su *www.futurashop.it*

***Fig. 1** - Il modulo SIM800C.*

ovviamente con la direttiva **MASTER** attiva. Quindi abbiamo le seguenti direttive:

- "**UART_DEBUG**" per verificare lo stato della connessione seriale, ovvero:
  - *Idle* = in attesa di fare qualcosa;
  - *Send* = invio dati in corso;
  - *Wait* = in attesa di risposta dal modulo GSM;
  - *Overflow* = rilevato Overflow;
- "**GSM_CMD_DEBUG_AT_CMD**" per l'analisi dei dati inviati verso il modulo GSM e relative risposte (la modalità di debug intercetta anche le stringhe ricevute in autonomia dal modulo);
- "**GSM_CMD_DEBUG_ON_OFF**" per l'analisi delle azioni di power ON, power OFF e RESET del modulo GSM;
- "**GSM_SECURITY_DEBUG**" per l'analisi delle funzioni di gestione di PIN e PUK della SIM.

Per ognuna delle direttive elencate esistono delle sezioni di codice sparse per la libreria; a ciascuna è associato del testo aggiuntivo per identificare/capire cosa si sta visualizzando sul monitor seriale. Tutto il testo di debug è memorizzato nella memoria FLASH del microcontrollore in modo da risparmiare la preziosa memoria SRAM. Ricordiamo inoltre che inserire il codice di debug ha un costo in termini di spazio occupato in FLASH di cui bisogna tenere conto soprattutto se si usa Arduino Uno.
In coda al file troviamo le dichiarazioni di variabile e di funzione; in questo caso ci sono solo:

- funzione per la configurazione degli I/O di Arduino per la gestione dei LED (necessario hardware 1.2 e Arduino Mega 2560 (*void* **SetOutputLed***(void)*); è richiamabile da qualsiasi sketch e non è richiesto di passarle parametri;
- funzione per la configurazione degli I/O di Arduino per la gestione dei trigger (*void* **SetOutputTrigger***(void)*); richiamabile da qualsiasi sketch e non è richiesto di passarle parametri;
- funzione per eseguire check dei LED collegati ad Arduino, da usare come test durante l'inizializzazione; necessario hardware 1.2 e Arduino Mega 2560 (*void* **CheckOutputLed***(void)*); la funzione è richiamabile da qualsiasi sketch e non è richiesto di passarle parametri;
- funzione per eseguire check dei trigger collegati ad Arduino, da usare come test durante inizializzazione (*void* **CheckOutputTrigger***(void)*); la funzione è richiamabile da qualsiasi sketch e non è richiesto di passarle parametri;
- funzione per accendere il LED collegato ad Arduino; necessario hardware 1.2 e Arduino Mega 2560 (*void* **LedOn***(uint8_t LedSelected)*); è richiamabile da qualsiasi sketch ma richiede come parametro il numero di I/O cui è collegato il LED (ad esempio **PIN_LED4**);
- funzione per spegnere il LED collegato ad Arduino; necessario hardware 1.2 e Arduino Mega 2560 (*void* **LedOff***(uint8_t LedSelected)*); la funzione richiamabile da qualsiasi sketch e richiede come parametro il numero di I/O cui è collegato il LED (ad esempio **PIN_LED4**);

A queste seguono due funzioni per la gestione dei trigger (*void* **TriggerOn***(uint8_t TriggerSelected) e void* **TriggerOff***(uint8_t TriggerSelected)*); entrambe sono richiamabili da qualsiasi sketch e richiedono come parametro il numero di I/O cui è collegato il trigger: ad esempio **TRIGGER_1** .
Infine è stata definita una funzione per fare lampeggiare un LED tra quelli collegati agli I/O della board Arduino (*void* **LedBlink***(uint8_t LedSelected, uint8_t DutyCycle, uint8_t TimeOutConstant)*); la funzione è richiamabile da qualsiasi sketch e richiede tre parametri:

- I/O cui è collegato il LED (ad esempio **PIN_LED4**);
- duty-cycle del lampeggio (ad esempio 50%);
- costante di tempo, corrispondente il periodo dell'onda quadra che pilota il LED (ad esempio 250ms corrisponde a **T_250MSEC**.

Le funzioni sopra descritte sono quindi richiamabili negli sketch in modo da avere un feedback di situazioni, come ad esempio la ricezione/invio di un SMS, la ricezione di una chiamata vocale ecc. Ad esempio, durante la fase di inizializzazione abbiamo deciso di sfruttare la funzione di lampeggio per indicare che è in corso l'inizializzazione del modulo GSM, il LED smetterà di lampeggiare solo a inizializzazione conclusa. La funzione di lampeggio vale sia per i LED, sia per i trigger.
Dopo questo preambolo torniamo sul file "**GenericCmd_GSM.cpp**" e analizziamo come funziona l'invio di un generico comando AT, quindi non tratteremo quelli dell'inizializzazione per ora, ed eventuale decodifica della risposta da parte del modulo GSM.

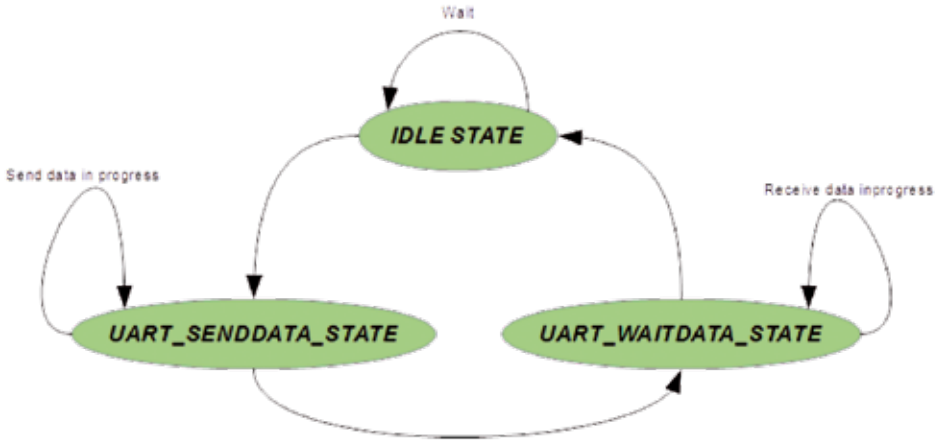


***Fig. 2** - Macchine a stati per la comunicazione su seriale.*

Analizziamo l'invio del comando AT+CFUN, che richiede un parametro che imposta il livello di funzionalità del modulo GSM ovvero:

- 0 = funzionalità minima;
- 1 = funzionalità completa;
- 4 = disabilità stadi RF di trasmissione e ricezione.

La funzione è dichiarata come *uint8_t* **GenericCmd_GSM::SetCmd_AT_CFUN**(*uint8_t* **FUN_Code**) e prevede come parametro il codice da passare al comando AT in esame; essa restituisce 0 se non ha inviato il comando AT richiesto e 1 se il comando è stato inviato con successo.
Prima di inviare il comando AT la funzione:

- azzera il buffer per la trasmissione/ricezione dei dati;
- salva i parametri ricevuti nel caso in cui dovesse ritentare l'invio del comando AT in caso di problemi (tenta l'invio del comando per un massimo di tre volte);
- setta un flag che identifica che sta iniziando il processo di invio comando AT;
- resetta tutti gli altri flag di gestione.

Fatto ciò, la funzione è pronta a comporre il comando AT da inviare quindi legge dalla FLASH il comando AT che deve inviare, in questo caso AT+CFUN. Ricordiamo che tutti i comandi AT usati sono memorizzati nella FLASH, per quelli che necessitano dei parametri questi verranno aggiunti in coda al comando. Letto il comando AT dalla FLASH il sistema provvede ad aggiungere in coda il parametro necessario e aggiorna una variabile con memorizzata la lunghezza del comando AT da inviare (ricordiamo che il comando AT è una stringa). Tale variabile serve alla funzione di gestione della seriale per capire quando ha finito di inviare tutti i byte (char) che formano il comando.
Infine richiama una funzione per l'inizializzazione della macchina a stati per l'invio del comando stesso:

```
Gsm.StartSendData(CMD_IDLE, WAIT_ANSWER_CMD_AT_CFUN,
ANSWER_GENERIC_AT_CMD_STATE);
```

Tale funzione imposta lo stato di **IDLE** per questa categoria di comandi, lo stato di wait per l'attesa della risposta da parte del modulo GSM e, infine, la categoria a cui appartiene il comando, che in questo caso fa parte di quelli generici. Una tipica funzione di invio comando AT è fatta come nel **Listato 1**.
Ora come fa il sistema a dare inizio all'invio del comando AT? Per fare ciò è stata predisposta una apposita funzione *void* Uart_GSM::ExecuteUartState(*void*) la quale viene chiamata continuamente nel ciclo *void* loop() dello sketch (Gsm.ExecuteUartState()). In questa funzione abbiamo una macchina a stati, fatta con un costrutto **switch**, la quale è sempre in IDLE mode fintantoché non si accorge che il flag **UartFlag.Bit.SendData** non è andato a 1 e la variabile contenente la lunghezza del comando AT da inviare sia maggiore di 0. Nel momento in cui queste due condizioni sono verificate il sistema passa nello stato di invio comando ovvero **UART_SENDDATA_STATE**. Non appena viene eseguito questo nuovo stato i dati presenti nel buffer di trasmissione vengono spediti sulla seriale attiva alla velocità impostata, il sistema imposta quindi un nuovo stato di attesa della risposta da parte del modulo GSM ovvero **UART_WAITDATA_STATE**. In questo nuovo stato il sistema rimane in attesa di una risposta da parte del modulo GSM; lo stato si esaurisce se non ci sono più dati disponibili in ricezione o se scade un time-out di ricezione. Conclusa la ricezione viene nuovamente imposto lo stato di **IDLE**. La **Fig. 2** schematizza il funzionamento delle macchine a stati per l'invio/ricezione dei dati su seriale. All'interno delle funzioni che compongono la macchina a stati di gestione della comunicazione UART troviamo la presenza di codice condizionale che viene attivato e compilato solo se sono attive le direttive di cui abbiamo parlato in precedenza, come ad esempio quelle per attivare il codice di

Listato 1

```
uint8_t GenericCmd_GSM::SetCmd_AT_CFUN(uint8_t FUN_Code) {
        if ((Gsm.StateWaitAnswerCmd != CMD_WAIT_IDLE) || (Gsm.UartState != UART_IDLE_STATE) || (Gsm.GsmFlag.Bit.
CringOccurred == 1)) {
                return(0);                //       System Busy
        } else {
                Gsm.ClearBuffer();
                Gsm.BckCmdData[0] = FUN_Code;
                Gsm.GsmFlag.Bit.GsmSendCmdInProgress = 1;
                Gsm.ResetFlags();
                Gsm.ReadStringFLASH((uint8_t *)AT_CFUN, (uint8_t *)Gsm.GSM_Data_Array, strlen(AT_CFUN));
                Gsm.GSM_Data_Array[8] = (FUN_Code + 0x30);
                Gsm.WritePointer = strlen(AT_CFUN);
                Gsm.StartSendData(CMD_IDLE, WAIT_ANSWER_CMD_AT_CFUN, ANSWER_GENERIC_AT_CMD_STATE);
        }
        return(1);                        //       Command sent
}
```

debug della seriale. Attualmente viene usata solo la UART1 e anche per questa ci sono delle direttive per attivare e compilare il codice necessario in caso si usi la UART1 hardware o software.
Altra funzione fondamentale per la gestione delle risposte da parte del modulo GSM ai comandi AT inviati è la seguente:

```
Gsm.GsmAnswerStateProcess();
```

Anch'essa viene richiamata continuamente nel ciclo *void* loop() dello sketch. Questa funzione ha lo scopo di richiamare la giusta funzione per decodificare le risposte ricevute dal modulo GSM. Ci sono, per ora, sette possibili casistiche correlate alle diverse categorie di comandi AT in cui abbiamo suddiviso la libreria. Per ognuna di queste è stata definita una direttiva per attivare il codice corrispondente e quindi renderlo disponibile al compilatore.
Ora la funzione che si occupa di decodificare la risposta ricevuta da parte del modulo GSM al comando inviato prima (AT+CFUN) è:

```
Gsm.GsmGenericWaitAnswer();
```

In testa la funzione esegue una serie di test in modo da avere la certezza che è possibile iniziare la decodifica dei dati ricevuti in merito al comando inviato e che quindi non ci siano ancora delle pendenze da assolvere. Dopodiché si accerta che ci siano dei dati nel buffer eseguendo un test sulla variabile **ReadPointer** la quale indica quanti byte sono stati ricevuti dalla seriale.
Il valore deve essere maggiore di zero affinché la funzione possa dare inizio alla decodifica della risposta.
Il secondo test controlla che all'interno della stringa ricevuta ci sia la dicitura "**OK**" la quale indica che il comando AT è stato ricevuto, capito ed eseguito. A seconda del comando inviato potrebbe esserci solo la dicitura "**OK**" oppure anche una serie di parametri in risposta alla richiesta fatta. Per esempio se il comando AT inviato fosse stato **AT+CREG?** la risposta ricevuta oltre a contenere la dicitura "**OK**" avrebbe anche riportato la seguente possibile stringa: **+CREG: 2,1,"0065","16F3"**.
Detto questo il codice, una volta verificato che nella stringa è presente la dicitura "**OK**", procede con la successiva decodifica di quanto ricevuto sfruttando il costrutto **switch** al quale si deve passare il codice del comando AT inviato.
Ogni comando AT è codificato con un codice univoco come già detto. Nel caso del comando AT+CFUN, il sistema non fa nessun altra operazione in quanto non sono presenti parametri da decodificare nella risposta.
Se invece il comando inviato fosse stato **AT+CREG?** il sistema avrebbe decodificato i dati presenti nella risposta e salvati in una apposita struttura dati chiamata **CREG_Info**. Una delle informazioni che viene estrapolata dalla risposta è lo stato della connessione, che viene salvato nella variabile **CREG_Info.Stat**.

***Fig. 3** - Il modulo SIM900.*

Lo stato della connessione può essere:

- "**0**" = non registrato;
- "**1**" = registrato;
- "**2**" = in cerca di un operatore cui registrarsi;
- "**3**" = registrazione negata;
- "**4**" = sconosciuto;
- "**5**" = registrato in roaming.

Altre due informazioni estrapolate sono la "**location area code**" o più semplicemente "**lac**" e la "**Cell ID**" o più semplicemente "**ci**"; tali informazioni vengono salvate nelle due variabili **CREG_Info. LAC_Info** e **CREG_Info.CI_Info**.
Esistono altre due funzioni base: la prima "ascolta" sulla seriale e se arrivano dei pacchetti dati indipendenti dai comando AT inviati, li intercetta e invia al buffer di ricezione (*Gsm.UartContinuouslyRead();*). Questi dati vengono chiamati "**Unsolicited Result Code**" e sono informazioni che il modulo GSM invia automaticamente in seguito ad eventi occorsi sulla rete GSM, come ad esempio un cambio cella, una disconnessione ecc.
La seconda funzione (*Gsm.ProcessUnsolicitedCode();*) invece si occupa di decodificare i dati ricevuti dalla precedente funzione; attualmente è in grado di decodificare gli "**Unsolicited Result Code**" inerenti a SMS, chiamate vocali e lo stato di connessione alla cella.
Quindi possiamo riassumere che il codice necessario nel ciclo *void* loop() dello sketch è il seguente:

```
Gsm.ExecuteUartState();
if (Gsm.GsmFlag.Bit.GsmInitInProgress == 1) {
 Gsm.InitGsmSendCmd();
 Gsm.InitGsmWaitAnswer();
} else {
 Gsm.UartContinuouslyRead();
 Gsm.ProcessUnsolicitedCode();
 Gsm.GsmAnswerStateProcess();
}
```

Come vedete, ritroviamo le funzioni che abbiamo descritto prima; il costrutto "**if else**" serve per capire se il sistema sta inizializzando il modulo GSM; in tal caso verranno chiamate solo le funzioni:

```
Gsm.InitGsmSendCmd();
Gsm.InitGsmWaitAnswer();
```

La prima invia il comando AT di inizializzazione, la seconda attende e decodifica le risposta da parte del modulo GSM.

L'INIZIALIZZAZIONE DEL MODULO

Non ci resta quindi che descrivere il processo di inizializzazione del modulo GSM: il ciclo di inizializzazione comincia dal *void setup()* dello sketch, richiamando la funzione *Gsm.InitPowerON_GSM()* la quale configura la macchina a stati di invio comandi di inizializzazione e la macchina a stati di invio comandi a regime; quest'ultima viene posta in IDLE mode fino a conclusione dell'inizializzazione.
Il primo step è il **power on** del modulo GSM tramite apposito impulso, cui segue l'attesa della risposta; questo perché il modulo GSM quando viene acceso risponde sempre, se non diversamente programmato, con una stringa indicante che l'accensione è avvenuta con successo. Solitamente rispondono con una stringa tipo "**RDY**", moduli SIMCOM e QUECTEL, oppure con "**AT command ready**" per quanto riguarda FIBOCOM. Se il modulo risponde così, significa che l'accensione è andata a buon fine e che si può passare allo step successivo, altrimenti può essere successo quanto segue:

- il modulo GSM era già acceso e quindi è stato spento; questo si capisce perché il modulo ha inviato la stringa "**NORMAL POWER DOWN**"; (ciò vale per SIMCOM e QUECTEL) e in tal caso il sistema ripropone il comando di power on;
- se non si riceve alcuna risposta può essere che il modulo GSM o è spento (FIBOCOM) oppure necessita di sincronizzare la velocità della seriale tramite autobaud e quindi si deve inviare il comando AT fintantoché non avviene la sincronizzazione;

In entrambi i casi il sistema è in grado di reagire.
Una volta acceso il modulo GSM il sistema invia tutta una serie di comandi AT di inizializzazione; di seguito la sequenza di comandi inviata:

- **AT&F0** = riporta i parametri ai valori di fabbrica;
- **ATE0** = disabilita "**Echo mode**";
- **ATQ0** = imposta il sistema per ricevere i "**result code**" da parte del modulo GSM; ad esempio il codice "**OK**" dopo l'invio di un comando andato a buon fine oppure "**ERROR**" per un comando errato o non capito (consigliamo di avere questa funzione attiva);
- **ATV1** = imposta il formato della risposta inviata dal modulo GSM; nel nostro caso impostiamo il "**verbose code**" (sfruttato nella nostra libreria) in modo da ricevere delle stringhe di testo e non dei semplici codici numerici (ad esempio "**OK**", "**CONNECT**", "**ERROR**", "**BUSY**");
- **AT+CMEE** = imposta il formato che devono avere i codici di errore riportati dal modulo GSM; in

***Fig. 4** - Il modulo SIM928A.*

tal caso la nostra libreria è impostata per ricevere i codici di errore nel solo formato numerico, che verranno decodificati dalla libreria (è possibile disabilitare questa funzione, ma è sconsigliato, oppure impostarla per ricevere gli errori in "**verbose <err>**");

- **AT+IPR** = la velocità seriale NON è più fissata a 57600 ma è impostato l'autobaud;
- **AT+ICF** = imposta la comunicazione seriale con 8 bit dati, 0 bit di parità e 1 bit di stop;
- **AT+IFC** = disabilita il "**data flow control**";
- **AT+CPIN?** = verifica se è necessario inserire il codice PIN per sbloccare la SIM;
- **AT+SPIC** = se la SIM richiede il codice PIN, il sistema prima di inviarlo verifica quanti tentativi sono ancora rimasti, giacché inviare tre volte il PIN errato ne disabilita il funzionamento finché non si inserisce il codice PUK; se rimangono meno di due tentativi non invia il PIN;
- **AT+CPIN** = invia alla SIM il PIN memorizzato nella EEPROM del microcontrollore; se il PIN inviato alla SIM è errato il modulo GSM ritorna il codice di errore 16 e l'algoritmo di inizializzazione intercetta questa anomalia interrompendo l'inizializzazione stessa;
- **AT+CLVL** = imposta il volume della uscita spk;
- **AT+CMIC** = imposta il guadagno dell'ingresso microfonico;
- **AT+CRC** = abilita il parametro CRC (**Cellular Result Code**) nel formato esteso, così quando si riceve una chiamata vocale il modulo GSM invierà sulla seriale il codice "**+CRING: VOICE**" al posto della sola stringa "**RING**";
- **AT+CLIP** = abilita il parametro CLI (**Calling Line Identity**) cosicché durante una chiamata vocale in ingresso viene visualizzato il numero di telefono del chiamante, cosa utile in applicazioni con discriminazione dei numeri di telefono;
- **AT+CMGF** = imposta il formato degli SMS; la libreria usa il formato Text Mode e non supporta il PDU mode;
- **AT+CSCS** = imposta il set di caratteri "**IRA**" ovvero *International Reference Alphabet*;
- **AT+CNMI** = imposta come, tramite un **unsolicited result code**, deve essere indicata la ricezione di un SMS da parte dell'ME; nel nostro caso:

- <mode> è impostato a 1, quindi se il link di comunicazione è occupato l'**unsolicited result code** verrà scartato e non inviato, mentre in caso contrario verrà immediatamente inviato al TE;
- <mt> è impostato a 1, perciò quando si riceve un SMS questo viene memorizzato nell'area di memoria preposta, vedere comando AT+CPMS, quindi l'**unsolicited result code** inviato conterrà anche l'informazione della memoria usata per memorizzare l'SMS;
- <bm> è impostato a "0";
- <ds> è impostato a "0";
- <bfr> è impostato a "0".

- **AT+CPMS** = imposta le aree di memoria in cui devono essere salvati gli SMS inviati e ricevuti; la libreria lavora con la memoria "**SM**" (*SIM message storage*);
- **AT+CSMP** = imposta il periodo di validità degli SMS inviati o memorizzati nelle aree di memoria;
- **AT+CREG** = abilita la registrazione alla rete e la possibilità di recuperare le informazioni sulla cella cui si è collegati sia tramite **unsolicited result code** che attraverso la lettura diretta.

Notate che all'esecuzione del comando AT+SPIC, se il numero di tentativi a disposizione per l'invio del codice PIN è inferiore a 2 il processo si interrompe, perché non ci sono automatismi per l'invio del codice PUK; questo va introdotto manualmente, ad esempio sfruttando la connessione seriale verso il PC oppure usando un cellulare.
Quando l'inizializzazione giunge a termine il sistema resetta il flag che indica *inizializzazione in corso* (*Gsm.GsmFlag.Bit.GsmInitInProgress*) abilitando quindi il sistema all'invio dei comandi AT per la gestione della propria applicazione.
Quando viene inviato un comando AT al modulo GSM, sia durante l'inizializzazione che durante l'invio di un comando AT generico, c'è la possibilità che questo non venga compreso dal modulo.
In base alla risposta ricevuta o non ricevuta dal modulo GSM, ritenta fino a un massimo di tre volte l'invio del comando AT; se dopo i tre tentativi di invio il comando non viene recepito allora il sistema provvede a resettare il modulo stesso e questo comporta la re-inizializzazione del modulo GSM.
Nel caso in cui il modulo GSM ritorni dei codici di errore questi verranno decodificati dalla funzione:

```
Gsm.ProcessGsmError();
```

Per ognuno dei codici di errore gestiti esiste un corrispettivo flag di segnalazione ad indicarne l'avvenuta rilevazione, sarà poi cura dell'utente gestire i flag in modo coerente con la propria applicazione. Attualmente esistono quattro categorie di codici di errori identificabili come:

- **Generici**: si trovano sotto la voce *Gsm.GsmFlag* oppure sotto la voce *Gsm.SimFlag*;
- **Sicurezza**: si trovano sotto la voce *Security.SecurityFlag*;
- **Agenda**: si trovano sotto la voce *PhoneBook.PhoneBookFlag*;
- **Messaggi**: si trovano sotto la voce *Sms.SmsFlag*.

Il file "**GenericCmd_GSM.cpp**" oltre alla sezione di inizializzazione presenta tutta una serie di funzioni per l'invio di una serie di comandi AT selezionati con l'intento di agevolare lo sviluppo di applicazioni "**GSM oriented**". Allo stato attuale non copriamo tutte le possibili situazioni ma con il tempo pensiamo di riuscire a raggiungere un buon compromesso tra funzioni implementate e codice di gestione necessario, soprattutto per la decodifica delle risposte ricevute dal modulo GSM. Tra le funzioni attualmente supportate dal file "**GenericCmd_GSM.cpp**", abbiamo:

- Comando AT generico, alla funzione deve essere passato come parametro la lunghezza della stringa di comando. È ancora in fase di sviluppo, soprattutto per quanto riguarda la gestione delle risposte (*Gsm.SetCmd(uint8_t Lenght)*).
- Comando AT per spegnimento modulo GSM via software, che differisce tra SimCom, Quectel o FIBOCOM; nel caso del Quectel necessita anche di un parametro che se "**0**" significa che si deve fare un power down urgente e quindi come risposta al comando viene inviato solo "**OK**", mentre se vale "**1**" significa che si deve fare un power down normale e quindi come risposta si riceve la stringa "**NORMAL POWER DOWN**" (*Gsm.SetCmd_SwPowerDown(uint8_t Type)*).
- Comando per spegnimento modulo GSM da hardware (tramite impulso di opportuna durata) ossia *Gsm.SetCmd_SetCmd_HwPowerDown(void);*).
- Comando AT per impostazione funzionalità (*Gsm.SetCmd_AT_CFUN(uint8_t FUN_Code)*).
- Comando AT per lettura o impostazione data e ora; prevede un parametro di tipo booleano per indicare se si sta leggendo la data e l'ora o se la si sta impostando. Se "**true**" esegue una lettura se "**false**" una scrittura. La data e l'ora sono formattate in una stringa sia per quanto riguarda la lettura che la scrittura. La formattazione della stringa è "**yy/MM/dd,hh:mm:ss±zz**" ovvero anno, mese, giorno, ore, minuti, secondi e il time zone che può andare da -48 a +48. Il time zone indica la differenza, espressa in quarti di ora, tra l'ora e la data locale rispetto al GMT. I fusi orari sono definiti relativamente al GMT, con un numero intero positivo se in ritardo o negativo se in anticipo (*Gsm.SetCmd_AT_CCLK(bool Query)*). La libreria prevede una struttura dati che grazie a una "**union**" a 64bit permette di settare o leggere i valori di data e ora ed è così composta:

```
° struct {
union Flag {
uint64_t  ClockLong;
struct {
uint8_t Year :7; // Year:    00  -  99
uint8_t Month :4; // Month:   01  -  12
uint8_t Day :5; // Day:     01  -  31
uint8_t Hours :5; // Hour:    00  -  23
uint8_t Minutes :6; // Minutes:  00  -  59
uint8_t Seconds :6; // Minutes:  00  -  59
int8_t  GMT :7; // GMT: -47 .. +48
uint8_t Free :24;
} Bit;
} Clock;
} Clock_Info;
```

Quindi l'utente in caso di impostazione di data e ora deve prima compilare i campi della struttura dati e poi richiamare la funzione *Gsm.SetCmd_AT_CCLK(bool Query)*. Viceversa se eseguirà una lettura si ritroverà questi campi compilati con i dati letti dal modulo GSM.

- Comando AT per leggere il codice costruttore: esiste in due formati tra i quali varia la stringa di comando a seconda della versione V25 o 3GPP, e quindi necessita di un parametro per identificare quale formato usare: "**0**" formato V25, "**1**" formato 3GPP. Nel file corrispondente .h sono definite le costanti per questo tipo di comando (*Gsm.SetCmd_ATQ_GMI(uint8_t FormatAt)*). I dati letti sono salvati tramite la struttura dati *ME_Info*
- Comando AT per leggere il codice del modello di modulo GSM. Valgono le considerazioni fatte per la versione di comando che si vuole usare (*Gsm.SetCmd_ATQ_GMM(uint8_t FormatAt)*).
- Comando AT per leggere la revisione software caricata nel modulo GSM. Valgono le considerazioni fatte per la versione di comando da usare (*Gsm.SetCmd_ATQ_GMR(uint8_t FormatAt)*).
- Comando AT per leggere il codice IMEI (Numero seriale di identificazione). Valgono le considera-

zioni fatte per la versione di comando da usare (*Gsm.SetCmd_ATQ_GSN(uint8_t FormatAt)*).

- Comando AT per leggere le informazioni inerenti alla registrazione del modulo GSM alla rete (*Gsm.SetCmd_ATQ_CREG(volid)*). Il comando AT riceve come risposta fino a tre parametri:
  - <n> = riporta il codice con cui è stato configurato il comando ovvero "**0**" indica che è stata disabilitata la registrazione alla rete, "**1**" che è stata abilitata la registrazione alla rete e attivati gli **unsolicited result code** per quanto riguarda lo stato di connessione <stat> e infine "**2**" cioè abilitata la registrazione alla rete e attivati gli **unsolicited result code** per quanto riguarda lo stato di connessione <stat> e le informazioni di locazione **<lac>** e **<ci>**; durante l'inizializzazione del modulo viene inviato il comando AT+CREG=2 in modo da avere tutte le informazioni possibili sullo stato di registrazione alla rete GSM;
  - <stat> = indica lo stato della registrazione alla rete GSM. I possibili valori sono: "**0**" non registrato alla rete e nessuna ricerca di altro operatore, "**1**" registrato alla rete GSM con proprio operatore, "**3**" registrazione negata, "**4**" sconosciuto e infine "**5**" registrato alla rete ma in roaming;
  - <lac> = codice esadecimale in formato stringa che identifica la locazione;
  - <ci> = codice esadecimale in formato stringa che identifica il codice della cella cui si è connessi.

  Queste informazioni sono rese disponibili dalla struttura dati *CREG_Info* che anch'essa, grazie a una "**union**", ci permette di raggruppare più informazioni assieme e renderle disponibili in modo agevole all'utente.
- Comando AT per leggere la potenza del segnale GSM (*Gsm.SetCmd_ATQ_CSQ(volid)*) il quale risponde con i seguenti due parametri:
  - <rssi> = se ritorna valori come "**0**" e "**1**" vuol dire che il segnale GSM è molto debole o assente. Valori compresi tra "**2**" e "**30**" riportano una situazione del segnale che va da debole a buona. Se invece riporta come valore "**31**" significa che il segnale è ottimo. Se invece ritorna "**99**" identifica una condizione sconosciuta o non misurabile. Ai numeri suddetti sono associati dei valori negativi in dBm, dove meno di -101dBm significa segnale molto debole, valori compresi tra -100dBm e -91dBm identificano segnale debole, mentre tra -91dBm e -81dBm il segnale è medio e da -80dBm in su il segnale è buono oppure ottimo.
  - <ber> = questo parametro identifica un valore percentuale compreso tra "**0**" e "**7**" ed è detto "**bit error rating**". Se ritorna il valore "**99**" identifica una condizione sconosciuta o non misurabile.

  Queste informazioni sono registrate nella struttura dati *CSQ_Info* accessibile all'utente.
- Comando AT per leggere il nome dell'operatore il quale risponde con i seguenti due parametri:
  - <mode> = indica la modalità di funzionamento. Quindi "**0**" modalità automatica, "**1**" modalità manuale, "**2**" de-registrazione manuale dalla rete, "**3**" imposta il solo parametro <*format*> e infine "**4**" modalità automatica e manuale attive. Se la modalità manuale fallisce interviene automaticamente la modalità automatica:
  - <format> = questo parametro imposta il formato che deve avere il campo <*oper*>. Quindi se "**0**" formato alfanumerico lungo fino a un massimo di 16 caratteri, se "**1**" formato alfanumerico corto e infine se "**2**" formato numerico;
  - <mode> = questo parametro riporta l'operatore a cui si è connessi.

  Le informazioni acquisite vengono salvate una parte nella struttura dati *SimFlag* in particolare sotto le voci *OperatorSelMode* e *OperatorSelFormat* mentre il nome dell'operatore viene salvato nello array *OperatorName*.
- Comando AT per verificare lo stato del modulo GSM; in altre parole serve per vedere se è occupato ad eseguire una chiamata fonica o altro. Questo potrebbe risultare utile prima di inviare determinati comandi AT (*Gsm.SetCmd_ATQ_CPAS(volid)*). La risposta è un parametro detto <*pas*> che può assumere i seguenti valori numerici: "**0**" pronto a ricevere qualsiasi comandi AT, "**2**" stato sconosciuto (non è detto che il modulo GSM (**ME**) risponda ad un eventuale comando), "**3**" ringing attivo (qualcuno sta chiamando o si è ricevuto un SMS) e infine "**4**" chiamata vocale in corso. I valori sopra citati vengono salvati nella struttura dati *SimFlag* e in particolare sotto la voce *PhoneActivityStatus*.
- Comando AT per impostare il livello del volume dello speaker, il comando richiede un parametro che indica il livello di volume che si vuole impostare. I valori consentiti vanno da 0 a 100 per i modelli SimCom e Quectel mentre da 0 a 6 per il modello FIBOCOM. In fase di inizializzazione i valori vengono impostati a 90 per i modelli SimCom e Quectel e a 6 per FIBOCOM. (*Gsm.SetCmd_AT_CLVL(uint8_t FormatAt)*)
- Infine abbiamo un comando AT per impostare il livello del guadagno del microfono collegato al modulo GSM. Il comando richiede due parametri ovvero il canale e il livello di guadagno, questo per quanto riguarda i moduli SimCom e Quectel.

Futura Energy

Fai il pieno di energia con le batterie alcaline di *FUTURA ELETTRONICA!*

BATTERIE ALCALINE
LR6 AA -1,5V
Confezione da 24 pezzi

OGGI ANCORA PIÙ CONVENIENTI NELLE CONFEZIONI DA 24 PEZZI!

GARANTITE FUTURA ELETTRONICA® GARANTITE

BATTERIE ALCALINE
LR3 AAA - 1,5V
Confezione da 24 pezzi

€ 7,50
Cod. **24LR6AA**

€ 6,50
Cod. **24LR03AAA**

Prezzi IVA inclusa.

BATTERIE ALCALINE LR6 AA 1,5V

Blister da 4 pezzi

€ 1,50
Cod. **LR6-AA-4**

BATTERIE ALCALINE LR3 AAA - 1,5V

Blister da 4 pezzi

€ 1,50
Cod. **LR03-AAA-4**

BATTERIA ALCALINA 6LR61 - 9V

Blister da 1 pezzo

€ 1,80
Cod. **6LR61-9V-1**

BATTERIE ALCALINE 23A - 12V

Blister da 5 pezzi

€ 2,50
Cod. **12V-23A-5**

BATTERIE ALCALINE LR14 C - 1,5V

Blister da 2 pezzi

€ 2,30
Cod. **LR14-C-2**

BATTERIE ALCALINE LR20 D - 1,5V

Blister da 2 pezzi

€ 3,20
Cod. **LR20-D-2**

Invece FIBOCOM ha il solo livello di guadagno. Quindi abbiamo che:
- <channel> : Questo parametro (**SIMCOM**) può assumere i valori: "**0**" canale audio principale, "**1**" canale audio ausiliare, "**2**" canale audio principale in modalità "**hand free**" e "**3**" canale audio ausiliare in modalità "**hand free**". Per quanto riguarda Quectel i valori possibili sono "**0**" microfono normale, "**1**" microfono cuffie e "**2**" microfono altoparlante. Per quanto riguarda FIBOCOM questo parametro perde di significato
- <gainlevel> : Questo parametro indica il livello di guadagno del microfono e per tutte e tre le famiglie di moduli GSM i valori possibili vanno da 0 a 15.

La funzione da usare è quindi *Gsm.SetCmd_AT_CMIC(uint8_t Channel, uint8_t GainLevel)*).

COMANDI DI GESTIONE SMS

Abbiamo quindi esaurito i comandi AT disponibili nel file "**GenericCmd_GSM.cpp**" e passiamo ad analizzare gli altri file di libreria, iniziando con "**SmsCmd_GSM.cpp**" per la gestione degli SMS.

- Comando AT per selezionare dove memorizzare gli SMS: può essere sia usato per leggere la configurazione attuale sia per impostare le tre aree di memoria; in quest'ultimo caso richiede tre parametri. La funzione che richiama tale comando necessita anche di un quarto parametro che serve per distinguere se si vuole leggere la configurazione piuttosto che scrivere dei parametri (*Sms.SetCmd_AT_CPMS(uint8_t TypeOfMem1, uint8_t TypeOfMem2, uint8_t TypeOfMem3, bool Query)*).
  Abbiamo quindi:
  - <TypeOfMem1> = area di memoria 1, che viene usata per la lettura o la cancellazione dei messaggi SMS (le tre aree sono <SM>, residente sulla SIM, <ME>, residente nel modulo GSM/GPRS ed <MT> che indica tutte le aree di memoria associate a un dispositivo GSM/GPRS, vale a dire che se sono disponibili entrambe le SM ed ME, con MT ci si riferisce ad entrambe;
  - <TypeOfMem2> = area di memoria 2; viene usata quando si inviano SMS precedentemente memorizzati attraverso il comando AT+CMSS; la memorizzazione da inviare avviene con il comando AT+CMGW e le tre possibili aree di memoria sono sempre SM ed ME;
  - <TypeOfMem3> = area di memoria 3, utilizzata per salvare i nuovi SMS ricevuti; dipende dal comando **AT+CNMI** inviato durante la fase di inizializzazione del modulo GSM; le aree di memoria sono quelle spiegate sopra.
- Comando AT per cancellare gli SMS (*Sms.SetCmd_AT_CMGD(uint8_t Index, uint8_t DelFlag)*); esso necessita di due parametri:
  - <Index> = locazione di memoria che si vuole cancellare;
  - <DelFlag> = comando da eseguire, ossia:
    - "0" cancella l'SMS alla locazione specificata con il precedente parametro;
    - "1" cancella dalla memoria specificata tutti i messaggi letti con il comando AT+CPMS;
    - "2" cancella tutti i messaggi già letti o inviati ma non quelli da leggere o da inviare;
    - "3" cancella tutti i messaggi già letti, inviati e anche quelli ancora da inviare;
  - "4" cancella tutti i messaggi indistintamente.
- Comando AT per leggere gli SMS ricevuti (*Sms.SetCmd_AT_CMGR(uint8_t Index, uint8_t Mode)*); anche questo richiede due parametri:
  - <Index> = locazione di memoria che si desidera leggere;
  - <Mode> = serve per modificare o meno lo stato del messaggio letto, ovvero se il valore passato è "**0**" viene cambiato lo stato del messaggio letto che passa da "**REC UNREAD**" a "**REC READ**", mentre se si passa il valore "**1**" lo stato del messaggio letto non viene cambiato.

  La risposta viene elaborata al fine di raccogliere informazioni sull'SMS ricevuto e soprattutto il testo del messaggio. Le altre informazioni sono il numero di telefono da cui è arrivato e lo stato del messaggio ovvero "**REC UNREAD**" (ancora da leggere) o "**REC READ**" (già letto).
- Comando AT per inviare un SMS a un dato numero di telefono (*Sms.SetCmd_AT_CMGS(void)*); non bisogna passare parametri perché sia il numero di telefono che il testo del messaggio da

***Fig. 5** - Il modulo G510 della FIBOCOM.*

inviare sono contenuti in due array che devono essere inizializzati prima di richiamare il comando di invio. Il numero di telefono si trova nell'array "**PhoneBook.PhoneNumber**" mentre il testo del messaggio si trova nell'array "**Sms.SmsText**". L'invio di un messaggio di testo avviene in due fasi distinte: nella prima fase si invia al modulo GSM il numero di telefono a cui si vuole inviare il messaggio. Una volta che il modulo ritorna il simbolo "**>**" significa che si può procedere con l'invio della stringa contenente il messaggio da inviare. Le due fasi sono gestite automaticamente dalla libreria; dobbiamo solo preoccuparci di inizializzare i due array.

- Comando AT per scrivere un messaggio da inviare nella memoria tra quelle scelte con il comando AT+CPMS (*Sms.SetCmd_AT_CMGW(void)*). Questo comando si comporta esattamente come il precedente salvo che lo SMS non viene inviato ma memorizzato nella memoria dell'**ME**. Quindi l'utente deve preoccuparsi di inizializzare l'array con il numero di telefono e l'array con il testo da salvare. Gli array sono gli stessi del precedente comando.
- Comando AT per inviare un messaggio salvato nella memoria selezionata con il comando precedente (*Sms.SetCmd_AT_CMSS(uint8_t Index)*). In questo caso si deve passare come parametro la locazione di memoria in cui è memorizzato.

Notate che quando si esegue il comando in lettura il modulo GSM ritorna i seguenti parametri:

- <Mem1> = tipo di memoria; SM, ME o MT;
- <Used1> = locazioni di memoria occupate;
- <Total1> = locazioni di memoria totali a disposizione; siccome queste sono tre, i parametri sopra descritti sono ripetuti anche per la memoria 2 e 3. I valori letti vengono salvati in una apposita struttura dati, una per ogni memoria. Ad esempio per la memoria 1 abbiamo la struttura dati "**Sms_Mem1_Storage**" la quale mette a disposizione le voci "**Type**", "**Used**" e "**Total**".

Con questa carrellata abbiamo esaurito i comandi AT disponibili per il file di libreria "**SmsCmd_GSM.cpp**", ricordiamo che esiste l'equivalente file .h contenente le dichiarazioni delle funzioni, variabili e strutture dati nonché i comandi AT salvati in FLASH e usati dalla libreria. Ricordate di attivare la sezione di gestione delle risposte da parte del modulo GSM (**ME**) tramite la direttiva "**ANSWER_SMS_AT_CMD_STATE**" presente nel file "**Io_GSM.h**"

COMANDI DI GESTIONE CHIAMATE VOCALI

Passiamo a file di libreria che si occupa della gestione delle chiamate sia in ingresso che in uscita. Il file è "**PhonicCallCmd_GSM.cpp**" e mette a disposizione i comandi seguenti.

- Comando AT per rispondere (*PhonicCall.SetCmd_ATA(void)*). Non richiede il passaggio di parametri e il sistema si accorge che è in arrivo una chiamata perché intervengono due fattori distinti. Il primo è l'interrupt generato dal segnale **RI** collegato allo **INT0** sulla scheda Arduino Uno o **INT4** sulla scheda Arduino Mega 2560. Il segnale **RI** è attivo a 0 logico e quindi l'interrupt è stato impostato per essere attivato quando il segnale passa dal livello logico alto (IDLE) al livello logico basso o in altre parole sfruttando il fronte di discesa del segnale.
  Durante la gestione dell'interrupt si attiva la ricezione seriale per catturare la stringa inviata come **unsolicited result code** la quale viene processata dalla funzione **Gsm.ProcessUnsolicitedCode();** come già discusso in precedenza. Viene quindi settato un flag della struttura dati **Gsm.GsmFlag** e in particolare il flag **Gsm.GsmFlag.Bit.IncomingCall**. Sarà vostra cura verificare tale bit e decidere se rispondere o no alla chiamata fonica in ingresso.
- Comando AT per chiudere la chiamata in corso (*PhonicCall.SetCmd_ATH(uint8_t* **Index***)*). Questo comando AT richiede un parametro se il modulo è un SIM900, SIM928A o M95. Il parametro in esame è definito come <n> e solitamente si passa il valore "**0**". Ci sono però altri casi in cui può essere necessario passare dei valori diversi da "**0**" ma che noi attualmente non usiamo con la nostra libreria. Se volete approfondire consultate il datasheet del modulo GSM che implementa e supporta questi codici aggiuntivi.
- Comando AT per iniziare una chiamata fonica in uscita (*PhonicCall.SetCmd_ATD(void)*), questo comando non necessita di parametri in quanto il numero di telefono da passare al comando viene

***Fig. 6** - Il modulo M95 della Quectel.*

caricato nell'apposito array "**PhoneBook.PhoneNumber**". Prima di richiamare questo comando si consiglia di verificare che il modulo GSM (**ME**) non sia già occupato in altra chiamata fonica sfruttando il comando AT+CPAS già discusso in precedenza.

- Comando AT per iniziare una chiamata fonica in uscita sfruttando un numero di telefono presente nella rubrica telefonica (*PhonicCall.SetCmd_ATD_PhoneNumberMemory(uint8_t* **n**)). Questo comando richiede un parametro che indichi quale locazione di memoria contiene il numero di telefono da utilizzare.
- Comando AT per iniziare una chiamata fonica in uscita sfruttando numero di telefono della precedente chiamata fonica (*PhonicCall.SetCmd_ATDL(void)*), nessun parametro richiesto per questo comando.

A proposito del file di libreria "**PhonicCallCmd_GSM.cpp**" ricordiamo che esiste l'equivalente file .h contenente le dichiarazioni delle funzioni, variabili e strutture dati nonché i comandi AT salvati in FLASH e usati dalla libreria. Ricordatevi di attivare la sezione di gestione delle risposte da parte del modulo GSM tramite la direttiva "**ANSWER_PHONIC_CALL_AT_CMD_STATE**" presente nel file "**Io_GSM.h**".

COMANDI DI GESTIONE RUBRICA TELEFONICA

Affrontiamo ora la libreria che si occupa della gestione della rubrica telefonica. Il file associato è "**PhoneBookCmd_GSM.cpp**" il quale mette a disposizione i comandi di seguito descritti.

- Comando AT per selezionare la memoria da usare per la rubrica telefonica, è simile al comando usato per selezionare le memorie da utilizzare per la memorizzazione degli SMS. Il comando prevede un parametro per indicare su quale tipologia di memoria si vuole lavorare (PhoneBookCmd_GSM.*SetCmd_AT_CPBS(uint8_t* **TypeOfPhoneBook.** *bool* **Query**)):
  <storage> = stringa dati che indica il tipo di memoria da utilizzare, che può essere:
  - <DC> = (**ME**), contenente l'elenco delle chiamate effettuate; non è applicabile il comando AT+CPBW;
  - <EN> = (**SIM**), memoria contenente i numeri di emergenza; su questo tipo di memoria non è applicabile il comando AT+CPBW;
  - <FD> = (**SIM**) memoria che contiene i numeri di telefono che l'utente può chiamare; serve per evitare che si possano fare chiamate a numeri indesiderati, quindi tutti i numeri di telefono che non sono in questo elenco non possono essere chiamati (sono esclusi i numeri di emergenza e le chiamate in entrata);
  - <MC> = (**MT**) contiene l'elenco dei numeri di telefono delle chiamate perse; su questo tipo di memoria non è applicabile il comando AT+CPBW;
  - <ON> = (**SIM**) contiene l'elenco dei numeri di telefono propri;
  - <RC> = (**MT**) contiene l'elenco dei numeri di telefono delle chiamate ricevute; su questo tipo di memoria non è applicabile il comando AT+CPBW;
  - <SM> = (**SIM**) è la rubrica telefonica della SIM e vi si possono salvare i numeri di telefono desiderati nella propria applicazione;
  - <LA> = ultimo numero di qualsiasi lista;
  - <ME> = (**ME**) è la rubrica telefonica del modulo GSM ed è possibile salvarvi i numeri di telefono desiderati per la propria applicazione;
  - <BN> = (**SIM**) rubrica numeri esclusi;
  - <SD> = (**SIM**) rubrica numeri di servizio;
  - <VM> = (**SIM**) rubrica numeri casella vocale;
  - <LD> = (**SIM**) ultima rubrica telefonica.

Nella nostra applicazione usiamo la memoria **SM** per memorizzare i numeri di telefono preferiti. Attualmente non è supportata la possibilità di gestire la memoria **FD** in quanto richiede un parametro aggiuntivo, in altre parole una password di accesso detta anche PIN2.
Il parametro da passare al comando AT è di tipo stringa, mentre alla nostra funzione dobbiamo passare un numero intero senza segno che identifica quale memoria vogliamo usare come decodificato nel file associato .h. Ci sarà poi una funzione che si occuperà di estrapolare la giusta stringa dato il codice assegnato. Se si vuole usare il comando in lettura il parametro query dovrà essere true, così il comando ci restituirà una serie di informazioni inerenti la memoria attualmente in uso ovvero: tipo di memoria, numero di locazioni usate e numero totali di locazioni. I dati letti vengono salvati nella struttura dati **PhoneBook.PhoneBookFlag**.

- Comando AT per leggere la rubrica telefonica alla locazione desiderata; prevede un parametro per indicare su quale tipologia di memoria lavorare (PhoneBookCmd_GSM.*SetCmd_AT_CPBR(uint8_t Index***.** *bool* **Query**)):
  - <index> = indice della rubrica da leggere; tale comando restituisce parametri tra cui:
  - <index> = indice della rubrica;

- <number> = numero di telefono presente nella rubrica all'indice dato;
- <type> = tipologia numero memorizzato; se 161 numero nazionale, se 145 numero internazionale (Usare il simbolo "+" in testa al numero seguito dal codice internazionale);
- <text> = breve descrizione assegnata al numero salvato.

I dati letti vengono salvati nelle tre variabili, di cui le prime due sono degli array, **PhoneText – PhoneNumber – PhoneNumberType**.

- Infine abbiamo il comando AT di scrittura di un numero telefonico nella rubrica selezionata (PhoneBookCmd_GSM.*SetCmd_AT_CPBW(uint8_t Index*.*uint8_t CmdType*. *uint8_t PhoneNumberType)*). I parametri da passare al comando AT sono:
  - <index> : Indice della rubrica su cui si vuole scrivere
  - <number> : Numero di telefono da salvare in rubrica
  - <type> : Tipologia numero, vedi precedente comando AT
  - <text> : Testo da assegnare al numero salvato. Ad esempio nome e cognome

Il numero di telefono e il testo ad esso associato devono essere memorizzati negli appositi array discussi prima cosicché da poterli inserire negli appositi spazi durante la composizione del comando AT.
Alla funzione bisogna quindi passare l'indice su cui si vuole salvare il numero di telefono, la tipologia del numero di telefono nazionale/internazionale e poi un parametro che identifica il tipo di comando che si vuole eseguire ovvero: scrivere il nuovo numero di telefono nella rubrica selezionata oppure cancellare il numero di telefono dalla rubrica selezionata. Nel file .h abbiamo codificato dei codici univoci che identificano il comando di scrittura o di cancellazione. Questo codici devono essere passati alla funzione descritta. Quando si esegue una cancellazione di un numero di telefono dalla rubrica il comando AT prevede come parametro il solo indice.
Abbiamo completato anche la descrizione dei comandi AT disponibili per il file di libreria "**PhoneBookCmd_GSM.cpp**"; anche qui esiste l'equivalente file .h contenente le dichiarazioni delle funzioni, variabili e strutture dati nonché i comandi AT salvati in FLASH e usati dalla libreria. Ricordate di attivare la sezione di gestione delle risposte da parte del modulo GSM tramite la direttiva "**ANSWER_PHONEBOOK_AT_CMD_STATE**" presente nel file "**Io_GSM.h**".

COMANDI DI GESTIONE DELLA SICUREZZA

Chiudiamo con la libreria che si occupa della gestione dei comandi inerenti alla sicurezza, ovvero alla gestione di password, PIN ecc. Il file associato è "**SecurityCmd_GSM.cpp**" e mette a disposizione i comandi seguenti.

- Comando AT per verificare quante volte si è inserito il codice **PIN/PUK** senza successo rispetto al numero di tentativi consentito. Questo comando differisce come sintassi tra i tre diversi costruttori **SIMCOM**, **FIBOCOM** e **QUECTEL**. Abbiamo quindi (*SecurityCall.SetCmd_ATQ_SPIC(void)*) per **SIMCOM**, (*SecurityCall.SetCmd_ATQ_TPIN(void)*) per **FIBOCOM** e (*SecurityCall.SetCmd_ATQ_QRTPIN(void)*) per **QUECTEL**. In tutti i casi non si deve passare nessun parametro al comando AT, si deve soltanto utilizzare il giusto comando AT a seconda del modulo GSM selezionato. Come risposta da parte del modulo GSM (**ME**) si ottengono i tentativi già effettuati per <pin1>, <pin2>, <puk1> e <puk2>.
  Questi valori vengono quindi memorizzati nella struttura dati SecurityFlag, più in dettaglio in SecurityFlag.Bit.PinRetry, SecurityFlag.Bit.Pin-2Retry, SecurityFlag.Bit.PukRetry e SecurityFlag.Bit.Puk2Retry. Sarà poi cura dell'utente utilizzare i valori letti per decidere se tentare o no l'invio del codice **PIN** o **PUK**. Ricordiamo che questo meccanismo viene utilizzato durante l'inizializzazione come già ampiamente spiegato prima
- Comando AT per verificare se è richiesto l'inserimento del codice PIN per sbloccare la SIM (*SecurityCall.SetCmd_ATQ_CPIN(void)*). Il comando non richiede nessun parametro. Come risposta da parte del modulo abbiamo le seguenti:
  - <READY> : **ME** non richiede nessun codice PIN. Nessuna azione da compiere
  - <SIM PIN> : **ME** è in attesa del codice PIN. Quindi prima di inviare il codice PIN si consiglia di usare il comando AT descritto precedentemente per verificare quanti tentativi sono disponibili e poi eventualmente inviare il codice PIN. Si ricorda che i codici PIN e PUK sono salvati in EEPROM
  - <SIM PUK> : **ME** è in attesa del codice PUK
  - <PH SIM PIN> : **ME** è in attesa del codice PIN antifurto
  - <PH SIM PUK> : **ME** è in attesa del codice PUK antifurto
  - <SIM PIN2> : **ME** è in attesa del codice PIN2. Deve essere esplicitamente richiesto tramite il codice di errore CME 17
  - <SIM PUK2> : **ME** è in attesa del codice PUK2.

Deve essere esplicitamente richiesto tramite il codice di errore CME 18

- Comando AT per l'invio del codice PIN o PUK (*SecurityCall.SetCmd_CPIN_PUK(uint8_t TypeOfCmd)*). Questo comando AT si aspetta il codice PIN oppure il codice PUK seguito dal nuovo codice PIN, quindi alla funzione si deve passare il codice dell'azione che si vuole eseguire ovvero solo codice PIN oppure codice PUK più codice PIN. Nel file .h abbiamo codificato le varie situazioni operative. A seconda dell'azione scelta la funzione leggerà dalla memoria EEPROM il codice PIN da aggiungere al comando AT oppure il codice PUK seguito dal codice PIN. Bisogna essere sicuri di avere salvato in EEPROM i codici PIN e PUK tramite apposito sketch. Nella prossima puntata vi mostreremo come fare
- Comando AT per bloccare o sbloccare tramite password le funzionalità di rete disponibili. Il comando AT richiede i seguenti parametri:
  - <fac> : Funzionalità di rete che si vuole bloccare tramite password
  - <mode> : Se "**0**" sblocca, se "**1**" blocca, se "**2**" verifica lo stato della funzionalità
  - <passwd> : Password formato stringa
  - <class> : Classe di appartenenza: "**1**" voce, "**2**" tutti i servizi, "**4**" fax e "**7**" tutte le classi

Alla funzione bisogna passare due parametri i quali serviranno ad andare a comporre il comando AT voluto (*SecurityCall.SetCmd_CLCK(uint8_t TypeOfFac, uint8_t Mode)*). Il primo parametro è un codice che identifica la funzionalità di rete che si vuole proteggere con password. I codici sono codificati nel file .h mentre il secondo parametro stabilisce se si deve eseguire un lock, unlock o una semplice query di lettura. Le possibili funzionalità di rete gestibili sono:

- <AO> : 4-Digit → Blocco di tutte le chiamate in uscita
- <OI> : 4-Digit → Blocco delle chiamate internazionali in uscita
- <OX> : 4-Digit → Blocco delle chiamate internazionali in uscita tranne che per il paese di provenienza
- <AI> : 4-Digit → Blocco di tutte le chiamate in entrata
- <IR> : 4-Digit → Blocco di tutte le chiamate in entrata quando si è in roaming in un altro paese
- <AB> : 4-Digit → Blocco di tutti i servizi
- <AG> : Blocco di tutti i servizi in uscita
- <AC> : Blocco di tutti i servizi in entrata
- <FD> : Blocco memoria SIM card contenente i soli numeri di telefono che si possono chiamare
- <SC> : 4/8-Digit → Blocca SIM come se fosse il PIN
- <PN> : 8-Digit → Network personalization
- <PU> : 8-Digit → Network subset personalization
- <PP> : 8-Digit → Service Provider personalization
- <P2> : 4/8-Digit → SIM PIN 2

Ognuno dei codici sopra indicati è stato codificato con un numero univoco da passare alla funzione come sopra esposto.
Come ultimo abbiamo un comando AT per cambiare la password impostata (*SecurityCall.SetCmd_CPWD(uint8_t TypeOfFac)*). Unico parametro da passare alla funzione è il tipo di funzionalità di rete che si vuole gestire. Vale l'elenco fatto al precedente comando ma solo per: <AO>, <OI>, <OX>, <AI>, <IR>, <AB>, <P2> e, <SC>. Il comando AT si aspetta quindi i seguenti parametri:

- <fac> : Funzionalità di rete su cui si vuole cambiare la password
- <old-passwd> : Vecchia password
- <new-passwd> : Nuova password

CONCLUSIONI

Bene, abbiamo esaurito la spiegazione dei comandi AT attualmente disponibili nella nostra libreria di gestione del modulo sullo shield GSM.
Nella prossima ed ultima puntata vi proporremo e descriveremo una serie di sketch per mostrare come utilizzare la libreria esposta con le schede Arduino e con le corrispondenti Fishino. A presto!

per il MATERIALE

Lo shield universale GSM (cod. WWGSMSHIELD) è in vendita presso Futura Elettronica al prezzo di Euro 54,90. Viene fornito montato ad esclusione degli strip line (compresi) che vanno saldati manualmente. Il prezzo è comprensivo di IVA.
Lo shield si può interfacciare con uno dei seguenti moduli GSM: interfaccia con M95 (cod. FT1128M, Euro 39,00), modulo cellulare miniaturizzato con SIM800 (cod. FT1308M, Euro 29,00) e modulo cellulare GSM/GPS con SIM928A (cod. FT1178M, Euro 59,00).

Il materiale va richiesto a:
Futura Elettronica, Via Adige 11, 21013 Gallarate (VA)
Tel: 0331-799775 • Fax: 0331-792287 - www.futurashop.it