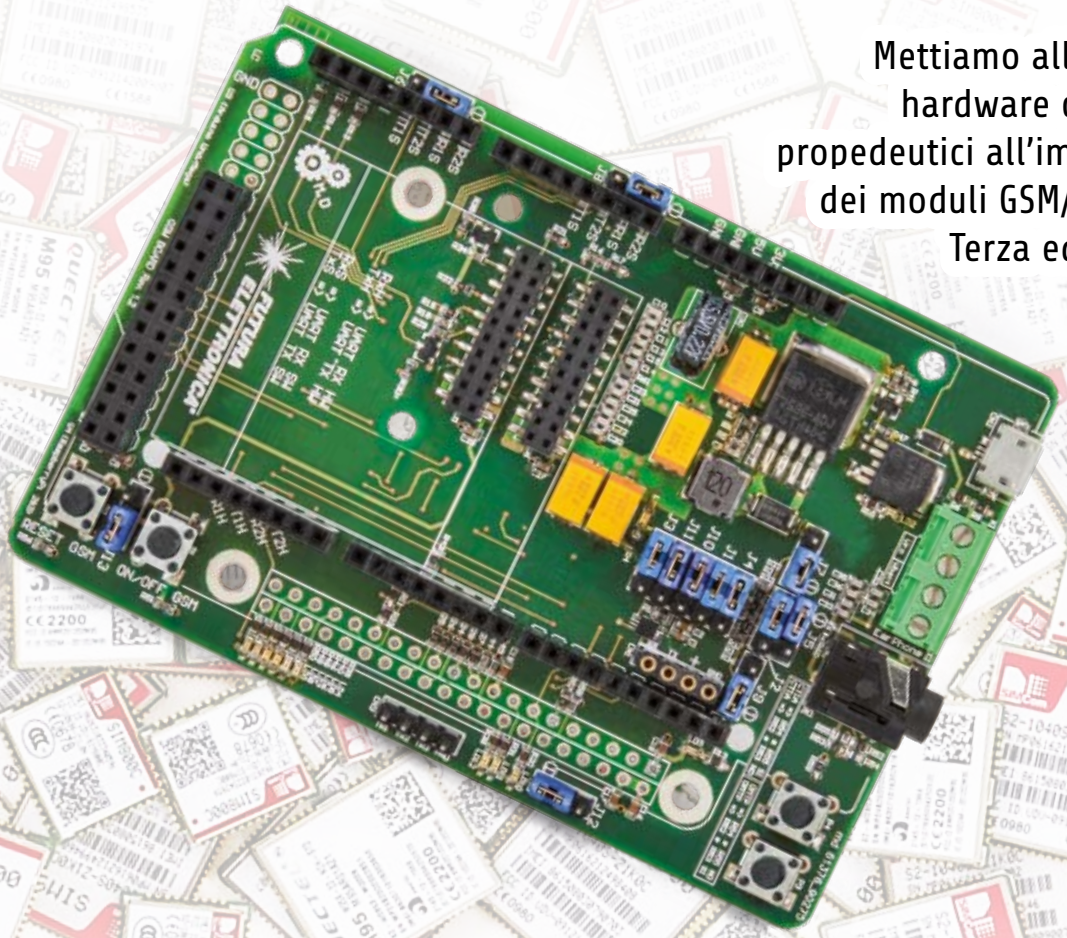
GSM SHIELD UNIVERSALE

di MATTEO DESTRO

Nella precedente puntata abbiamo descritto la libreria e le funzioni messe a disposizione dalla stessa. Ora presenteremo una serie di sketch attraverso i quali vedrete in pratica come funziona la libreria e le sue potenzialità. Gli sketch implementeranno funzioni tra cui ricezione o invio di SMS, chiamate vocali, gestione di alcuni comandi generici, nonché la scrittura in *EEPROM* dei codici PIN, PUK ecc.
La libreria è pensata per svolgere la gran parte del lavoro e lasciare il compito più facile allo sviluppatore, che dovrà semplicemente adattare la libreria alle necessità proprie e delle proprie applicazioni. Ricordiamo che la programmazione dei codici PIN e PUK in *EEPROM* non può essere fatta direttamente dall'IDE Arduino ma solo con apposito sketch di scrittura dei dati in *EEPROM*.
Ciò detto, iniziamo la descrizione degli sketch dalla programmazione in *EEPROM* dei codici.

PROGRAMMAZIONE EEPROM

La prima operazione da compiere per utilizzare il GSM Shield è programmare la *EEPROM* di Arduino con i codici necessari al corretto funzionamento degli altri sketch.



Mettiamo alla prova il nostro hardware con alcuni sketch propedeutici all'impiego sul campo dei moduli GSM/GPRS supportati. Terza ed ultima puntata.

Il file "*GenericCmd_GSM.h*", subito all'inizio del file, contiene l'elenco dei dati da memorizzare nella memoria *EEPROM*.
Allo stato attuale si tratta di una serie di codici numerici che identificano le seguenti voci:

- ***Password generica 1*** → 8 caratteri numerici, spazio occupato 11 byte (comprende le virgolette e il carattere di chiusura della stringa);
- ***Password PH PUK*** → caratteri numerici, spazio occupato 11 byte;
- ***Password PUK2*** → 8 caratteri numerici, spazio occupato 11 byte;
- ***Password SIM PUK*** → 8 caratteri numerici, spazio occupato 11 byte;
- ***Password generica 2*** → 4 caratteri numerici, spazio occupato 7 byte;
- ***Password PH PIN*** → 4 caratteri numerici, spazio occupato 7 byte;
- ***Password PIN2*** → 4 caratteri numerici, spazio occupato 7 byte;
- ***Password SIM PIN*** → 4 caratteri numerici, spazio occupato 7 byte.

Per dire al compilatore che si vuole inserire dei dati in *EEPROM* si utilizza la seguente sintassi:

```
const uint8_t EEMEM LONG_GEN_PSWD[11] = "\"12345678\"";
```

In pratica abbiamo definito una variabile intera senza segno (***uint8_t***) di tipo costante (***const***) che attraverso la direttiva "***EEMEM***" dice al compilatore che i dati fanno parte della *EEPROM*. Notare che nell'esempio riportato non si sta memorizzando una singola variabile, bensì un array di byte di lunghezza 11 contenente una password di tipo numerico.
La locazione di memoria da cui iniziare a scrivere i byte sopra definiti viene decisa dal compilatore, a meno che non venga impostata diversamente da codice e l'indirizzo di partenza di ogni array viene poi recuperato via codice così da permettere l'accesso ai dati sia in lettura che in scrittura.
Il compilatore genera quindi un file con estensione **.epp** che viene memorizzato nella directory:

```
C:\Documents and Settings\YourUserName\LocalSettings\
Temp\arduino_build_xxxxxx
```

La **Fig. 1** mostra il contenuto del file .epp generato durante la compilazione.
Ora, per recuperare gli indirizzi di partenza dei diversi array definiti in EEPROM si deve usare la consueta sintassi in C:

```
&LONG_GEN_PSWD[0]
```

Questo valore viene quindi salvato in un'apposita variabile. Nel nostro caso abbiamo definito una struttura dati per recuperare gli indirizzi di partenza di tutti gli array usati per le password, la quale è:

```
struct
{
uint8_t StartAddPin1Code;
uint8_t StartAddPin2Code;
uint8_t StartAddPhPinCode;
uint8_t StartAddPuk1Code;
uint8_t StartAddPuk2Code;
uint8_t StartAddPhPukCode;
uint8_t StartAddShortPswdCode;
uint8_t StartAddLongPswdCode;
} EepronAdd;
```

Di conseguenza, la sintassi da utilizzare per sfruttare questa struttura dati sarà:

```
Gsm.EepronAdd.StartAddLongPswdCode =&LONG_GEN_PSWD[0];
Gsm.EepronAdd.StartAddPhPukCode    =&PH_PUK_CODE[0];
Gsm.EepronAdd.StartAddPuk2Code     =&PUK2_CODE[0];
Gsm.EepronAdd.StartAddPuk1Code     =&PUK1_CODE[0];
Gsm.EepronAdd.StartAddShortPswdCode=&SHORT_GEN_PSWD[0];
Gsm.EepronAdd.StartAddPhPinCode    =&PH_PIN_CODE[0];
Gsm.EepronAdd.StartAddPin2Code     =&PIN2_CODE[0];
Gsm.EepronAdd.StartAddPin1Code     =&PIN1_CODE[0];
```

Infine, per agevolare questa operazione le righe di codice sopra esposte sono state raggruppate in una funzione di libreria da richiamare durante la sezione di "***Setup***" dello sketch, in particolare abbiamo:

```
Gsm.EepromStartAddSetup();
```

Se tale chiamata a funzione viene omessa il compilatore non genererà il file .epp ignorando il codice di assegnazione dei dati in *EEPROM*.
Ora, come ben sappiamo, lo IDE Arduino programma la sola *Flash* del microcontrollore presente sulla board usata e ignora totalmente la programmazione della *EE-*

```
:10000000223436323922002238383838220022355C
:1000100035353522002231323334220022353430 56
:1000200039343135333220022393939393939393 8E
:1000300022002234343434343434342200223132 35
:08004000333435363738220055
:00000001FF
```

Fig. 1 *- File .epp generato durante la compilazione.*

PROM anche se il compilatore ha generato il file .**epp**. Detto ciò ci sono due possibilità per programmare la *EEPROM*: la prima è sfruttare il tool a riga di comando "***avrdude***"; la seconda è scrivere uno sketch dedicato ed è quello che si è deciso di fare per risolvere il problema. Diamo quindi un'occhiata al semplicissimo sketch "***GSM_SetEeprom***" che ha il solo scopo di programmare la *EEPROM* del microcontrollore. Una volta programmata potremo scaricare nella *Flash* delle schede Arduino i nostri Sketch applicativi per la gestione di SMS, chiamate vocali e altro.
Questo sketch è valido per le schede Arduino Uno, Arduino Mega 2560 e le schede Fishino corrispondenti. Ricordarsi di selezionare nella libreria la scheda Arduino che si vuole utilizzare.
Cominciamo dalla sezione di setup ovvero "***void Setup()***"; la prima cosa da fare è richiamare la funzione per recuperare gli indirizzi di partenza in *EEPROM* in cui sono memorizzate le password, quindi si deve richiamare la funzione:

```
Gsm.EepromStartAddSetup();
```

Dopodiché si richiama la funzione di programmazione della *EEPROM* ovvero:

```
Initialize_PIN_PUK_Eeprom();
```

Infine si attende un tempo di 500ms e poi si inizializza la UART a cui è collegato il monitor seriale in modo da verificare che la *EEPROM* sia stata effettivamente programmata con i dati desiderati. Per inizializzare la UART di debug si deve richiamare la funzione di libreria:

```
Gsm.EnableDisableDebugSerial(true, BAUD_115200);
```

Questa attiva la UART e la inizializza alla velocità di 115200 Baud. Infine attiviamo gli interrupt di libreria ovvero il ***TIMER1*** per la base dei tempi e l'interrupt esterno ***INT0*** o ***INT4*** a seconda che si usi Arduino Uno o Arduino Mega (in questo sketch l'interrupt esterno non viene usato).

```
Isr.EnableLibInterrupt();
```

Si attendono altri 500ms e infine si carica una variabile temporale con un tempo di 15 secondi. Tutte le volte che questo timeout scade il sistema provvede ad eseguire una lettura in *EEPROM* dei dati memorizzati e li invia sul monitor seriale affinché l'utente possa verificarne la correttezza.

```
Isr.TimeOutWait = T_15SEC;
```

La sezione di setup si conclude e il codice passa al main (***void loop()***) dove trova posto una sola funzione che ha lo scopo di leggere i dati in *EEPROM*:

```
Verify_Eeprom();
```

Ora analizziamo velocemente la funzione "***Initialize_PIN_PUK_Eeprom()***" la quale ha il solo ed unico scopo di scrivere i dati in *EEPROM*. Vediamo quindi la sezione di codice per scrivere il codice Pin1 (SIM PIN):

```
Add = Gsm.EepronAdd.StartAddPin1Code;
eeprom_write_byte((uint8_t *)Add++, 0x22);
eeprom_write_byte((uint8_t *)Add++, '4');
eeprom_write_byte((uint8_t *)Add++, '6');
eeprom_write_byte((uint8_t *)Add++, '2');
eeprom_write_byte((uint8_t *)Add++, '9');
eeprom_write_byte((uint8_t *)Add++, 0x22);
eeprom_write_byte((uint8_t *)Add, 0x00);
```

Per prima cosa carichiamo nella variabile puntatore "***Add***" l'indirizzo di partenza in *EEPROM* da dove iniziare a scrivere i dati. In questo caso l'indirizzo di partenza riguarda il codice del PIN1. Seguono sette scritture e ad ogni scrittura il puntatore viene incrementato di uno puntando alla locazione di memoria successiva. Il primo valore memorizzato e il penultimo sono le virgolette. Infatti i codici PIN, PUK ecc solitamente sono sempre racchiusi tra virgolette. Conclude la stringa il carattere di chiusura 0x00. Questa sintassi viene replicata per tutti i codici da salvare in *EEPROM*.
Adesso analizziamo la funzione di verifica dei codici (***Verify_Eeprom()***) la quale, come accennato prima, ogni 15 secondi stampa sul monitor seriale i dati memorizzati in *EEPROM*. Come prima analizziamo solo una porzione di codice:

```
Add = Gsm.EepronAdd.StartAddPin1Code;
sprintf(AddressBuffer, "0x%04X", Add);
Serial.print("Address: ");
Serial.print(AddressBuffer);
Serial.print(" -> PIN1 = ");
Serial.print(char(eeprom_read_byte((uint8_t *)Add++)));
Serial.print(char(eeprom_read_byte((uint8_t *)Add++)));
Serial.print(char(eeprom_read_byte((uint8_t *)Add++)));
Serial.print(char(eeprom_read_byte((uint8_t *)Add++)));
Serial.print(char(eeprom_read_byte((uint8_t *)Add++)));
Serial.print(char(eeprom_read_byte((uint8_t *)Add++)));
Serial.print("\n");
```

Per prima cosa recuperiamo l'indirizzo del codice PIN 1, stampiamo a video la stringa che identifica cosa abbiamo letto e il suo indirizzo di partenza e infine il

contenuto della *EEPROM*. La **Fig. 2** evidenzia quanto letto dalla memoria, dal PIN1 alla password generica da 4 caratteri. Questi dati vengono stampati a video una volta ogni 15 secondi.
Un ultima curiosità riguarda la gestione della variabile temporale "***Isr.TimeOutWait***" usata in questo sketch per decidere ogni quanto stampare i dati letti. Alla base di tutto ciò c'è il ***TIMER1*** gestito e configurato dalla nostra libreria il quale genera un interrupt stabile ogni 2 ms. Questo tempo viene detto base dei tempi e identifica la risoluzione che possono avere le nostre variabili temporali usate dalla libreria (±2 ms).
Ora per il nostro sketch abbiamo bisogno di un tempo di 15 secondi quindi il valore da caricare nella variabile sarà pari a 7.500, tale valore viene decrementato di un'unità ogni qual volta scatta l'interrupt. Quando la variabile giunge a zero significa che sono passati 15 secondi. A questo punto lo sketch stampa a video e ricarica la variabile.

SKETCH INVIO COMANDI GENERICI:
Commentiamo ora lo sketch "*GSM_GenericCommand*" il quale mostra come utilizzare una serie di comandi base implementati nella libreria. Questo sketch è compatibile con le schede Arduino Uno, Arduino Mega 2560 e le schede Fishino omonime. Ricordarsi di selezionare nella libreria la scheda Arduino che si vuole utilizzare.
Lo sketch è formato da un unico file e in testa ad esso troviamo una serie di dichiarazioni in merito alle macchine a stati realizzate per la gestione dei vari task, una per l'invio dei comandi e una per la stampa su monitor seriale delle risposte ricevute dal modulo GSM in merito ai comandi AT inviatogli. Seguono una serie di stringhe, tutte memorizzate nella memoria *Flash* del microcontrollore, necessarie durante la stampa a monitor delle risposte in modo da rendere evidente di cosa si sta parlando.
Iniziamo come al solito la trattazione dalla sezione di Setup (***void setup()***) nella quale dobbiamo inizializzare la libreria affinché possa svolgere alle sue funzioni. Come già fatto nel precedente sketch dobbiamo prima di tutto richiamare la funzione:

```
Gsm.EepromStartAddSetup();
```

Questa inizializza la struttura dati con i puntatori in memoria per il recupero delle password salvate in *EEPROM*, seguono quattro funzioni necessarie a inizializzare e testare gli I/O della scheda Arduino in uso. Ricordiamo che l'hardware della shield prevede una serie di led collegati agli I/O delle schede Arduino utili sia per il debug del codice (Trigger) sia per evidenziare degli eventi significativi come la ricezione di un SMS o altro (l'utente ha la piena libertà nella gestione di questi LED a seconda dell'applicazione che sta realizzando. La libreria mette a disposizione delle opportune funzioni per il loro corretto utilizzo). Quindi abbiamo:

```
Io.SetOutputTrigger();
Io.SetOutputLed();
Io.CheckOutputTrigger();
Io.CheckOutputLed();
```

La prima funzione serve per inizializzare gli I/O di trigger e la seconda inizializza gli I/O con i LED di diagnostica (solo Arduino Mega 2560). Le ultime due funzioni servono per testare gli I/O appena configurati, accendendo e spegnendo in sequenza i vari LED per verificare se funzionano allo start-up.
Il passo successivo prevede l'abilitazione della seriale di debug per la stampa a monitor delle risposte ricevute dal modulo GSM; notate che il codice per stampare a monitor non fa parte della libreria, ma in questo sketch abbiamo scritto un'apposita funzione per adempiere a tale compito (ricordate le stringhe salvate nella Flash spiegate poco fa).
La seriale viene configurata a una velocità di 115200 Baud:

```
Gsm.EnableDisableDebugSerial(true, BAUD_115200);
```

Le altre velocità possibili sono: `BAUD_2400`, `BAUD_4800`, `BAUD_9600`, `BAUD_19200`, `BAUD_38400`, `BAUD_57600`.
Volendo si può impostare anche: `BAUD_230400`,

```
Address: 0x0000 -> PIN1 = "4629"
Address: 0x001C -> PUK1 = "54094153"
Address: 0x0007 -> PIN2 = "8888"
Address: 0x0027 -> PUK2 = "99999999"
Address: 0x000E -> PH_PIN = "5555"
Address: 0x0032 -> PH_PUK = "44444444"
Address: 0x003D -> LONG PASSWORD = "12345678"
Address: 0x0015 -> SHORT PASSWORD = "1234"


Address: 0x0000 -> PIN1 = "4629"
Address: 0x001C -> PUK1 = "54094153"
Address: 0x0007 -> PIN2 = "8888"
Address: 0x0027 -> PUK2 = "99999999"
Address: 0x000E -> PH_PIN = "5555"
Address: 0x0032 -> PH_PUK = "44444444"
Address: 0x003D -> LONG PASSWORD = "12345678"
Address: 0x0015 -> SHORT PASSWORD = "1234"
```

***Fig. 2** - Lettura della memoria EEPROM.*

La tensione giusta per ogni progetto!

Scegli il convertitore di tensione DC-DC che meglio si adatta alle tue esigenze.

CONVERTITORI STEPDOWN DC-DC

Da 5-36V a 1,25-32V - 5A

€ 14,00
cod. YB728

Da 5-23V a 0-16,5V da 2A a 3A

€ 9,90
cod. DCDISPAVDOWN

Da 7-28V a 5V - 3A

€ 2,50
cod. MP1584ENDCDC

Da 4-40V a 1,25-37V - 2A

€ 6,90
cod. STEPDOWNDISP

Da 9-35V a 5V - 5A

€ 6,50
cod. DCDCSTEPDOWN

Da 5-32V a 0,8-30V - 5A

€ 5,90
cod. STEPD5V32

Da 3-40V a 1,5-35V - 3A

€ 4,30
cod. MODULODCDC

CONVERTITORI STEPUP DC-DC

Da 3,5-30V a 4-30V - 2A

€ 6,50
cod. STEPUP30V

Da 0,9-5V a 5V - 0,6A

€ 3,00
cod. STEPUPUSB

Da 3-4,5V a 5V - 1A

€ 4,90
cod. DCDC5V

DISPONIBILE ANCHE NELLE VERSIONI:
da 3-6V a 9V - 1A
cod. DCDC9V € 4,90
Da 3-6V a 12V - 0,4A
cod. DCDC12V € 4,90

CONVERTITORE STEPUP e STEPDOWN DC-DC

Da 4-35V a 1,25-25V - 2A

€ 16,00
cod. STEPUPDOWN

GARANTITI FUTURA ELETTRONICA GARANTITI

Prezzi IVA inclusa.

FUTURA ELETTRONICA®
www.futurashop.it

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

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

`BAUD_250000, BAUD_500000, BAUD_1000000.`
Quelle elencate sono tutte delle costanti definite in libreria, da passare come parametro alla funzione. Al passo successivo abilitiamo gli interrupt:

```
Isr.EnableLibInterrupt();
```

e si imposta la seriale di comunicazione 1 (UART 1) verso il modulo GSM:

```
Gsm.SetBaudRateUart1(true, BAUD_19200);
```

Quest'ultima può essere di tipo hardware o software a seconda della configurazione fatta nel file "***IO_GSM.h***" e dall'impostazione data ai jumper presenti sulla shield. Se si utilizza una seriale software si consiglia di non superare la velocità di 38.400 Baud, meglio se 19.200 Baud. Se invece si usa la seriale hardware, si può tranquillamente impostare velocità fino a 115.200 Baud. Quindi si esegue una pulizia dei buffer usati per la comunicazione seriale per poi dare inizio all'inizializzazione del modulo GSM inviando in sequenza il comando di accensione del modulo e l'inizializzazione della macchina a stati per la gestione dell'UART:

```
Gsm.InitPowerON_GSM();
Gsm.UartFlag.Bit.EnableUartSM = 0;
Gsm.ExecuteUartState();
Gsm.UartFlag.Bit.EnableUartSM = 1;
```

Con le righe di codice appena descritte si conclude la sezione di setup dello sketch. Eseguite queste ultime il codice passa al main, ovvero il consueto "***void loop()***", nel quale ritroviamo il codice di gestione delle macchine a stati per l'inizializzazione del modulo GSM e dell'invio dei comandi AT a regime. Il codice che segue è utilizzabile in tutte le applicazioni in cui si fa uso della nostra libreria e rappresenta la base di partenza. Quindi abbiamo:

```
Gsm.ExecuteUartState();
if (Gsm.GsmFlag.Bit.GsmInitInProgress == 1) {
  Gsm.InitGsmSendCmd();
  Gsm.InitGsmWaitAnswer();
} else {
  Gsm.UartContinuouslyRead();
  Gsm.ProcessUnsolicitedCode();
  Gsm.GsmAnswerStateProcess();
  #ifdef ARDUINO_UNO_REV3
    Io.LedBlink(TRIGGER_2, 25, T_2SEC);
  #endif
  #ifdef ARDUINO_MEGA2560_REV3
    Io.LedBlink(PIN_LED9, 25, T_2SEC);
  #endif
}
```

La prima riga richiama la funzione che gestisce la macchina a stati di invio e ricezione dati tramite la seriale (hardware/software). La gestione della comunicazione seriale necessita di tre stati:
- **IDLE**, dove il sistema rimane in attesa di inviare un comando AT al modulo GSM;
- ***SEND***, necessaria per inviare il comando AT al modulo;
- ***WAIT***, dove si rimane in attesa di una risposta da parte del modulo GSM; questa funzione serve sia durante l'inizializzazione del modulo sia durante l'invio dei comandi AT a regime.

Immediatamente c'è il codice condizionale con cui il sistema verifica se è in corso l'inizializzazione del modulo GSM (Flag `Gsm.GsmFlag.Bit.GsmInitInProgress` a "1" logico) oppure l'invio di comandi AT generici (Flag `Gsm.GsmFlag.Bit.GsmInitInProgress` a "0" logico).
Quando il suddetto flag è a livello "1" vengono chiamate le due funzioni per la gestione della macchina a stati per l'invio dei comandi AT necessari a settare il modulo GSM; la stessa macchina allinea la velocità della UART impostata dall'utente con quella del GSM se le due sono differenti.
Durante l'inizializzazione il sistema verifica anche l'eventualità di inviare il codice PIN alla SIM nel caso questo sia abilitato. Il codice si trova memorizzato nella memoria *EEPROM* come spiegato nel precedente sketch e il puntatore ad esso è stato recuperato con la prima riga di codice di inizializzazione.
Nella seconda parte del codice condizionale c'è la funzione indispensabile per la decodifica delle risposte ricevute dal modulo (`Gsm.GsmAnswerStateProcess()`). Questa è affiancata da altre due funzioni che si occupano di intercettare eventuali dati inviati in autonomia dal modulo GSM i quali potrebbero contenere informazioni utili come ad esempio gli "***Unsolicited Code***" (`Gsm.UartContinuouslyRead()` e `Gsm.ProcessUnsolicitedCode()`).
Allo stato attuale sono gestiti i seguenti: ***CRING***, ***Phonic Call*** e ***CREG***.
Per ultimo abbiamo inserito anche la gestione di uno dei LED di diagnostica usati per evidenziare che il sistema è uscito dalla condizione di inizializzazione ed è ormai a regime. A seconda della scheda Arduino scelta si utilizzerà il LED 9 (Arduino Mega) o il trigger 2 (Arduino Uno).
Durante l'inizializzazione, per evidenziare questa condizione si fa lampeggiare il LED collegato al trigger 3.
Il blocco condizionale appena descritto è indispensabile per il corretto utilizzo della nostra libreria di gestio-

ne dei moduli GSM supportati.
Fuori dal codice condizionale trovano posto tutte le funzioni necessarie alla realizzazione dell'applicazione in esame e svincolate dalla nostra libreria: ad esempio quella per inviare dei comandi AT base in sequenza perpetua (`Send_AT_Cmd()`) e la stampa su monitor delle risposte ricevute dal modulo GSM (`PrintDataReceived()`). Ad ogni invio di un comando il sistema registra la risposta ottenuta e la rende disponibile all'utente. In particolare abbiamo la verifica della registrazione del modulo GSM alla cella tramite proprio operatore (AT+CREG?), verifica della potenza del segnale GSM (AT+CSQ), stato del modulo GSM (AT+CPAS), verifica dell'operatore (AT+COPS?), identificazione del costruttore (AT+GMI), identificazione modello (AT+GMM), revisione software caricata (AT+GMR), numero seriale del prodotto (AT+GSN) e infine lettura di data e ora dal real time clock presente sul modulo GSM (AT+CLCK?). Di seguito quanto viene stampato sul monitor seriale a seguito della risposta ricevuta ai comandi AT inviati, ovvero le risposte ricevute e totalmente decodificate.

```
============ CPIN ============
SIM OK
============ CREG ============
Registered, home network
Location area: "0065"
Cell ID: "16F3"
============ CSQ  ============
Rssi: 11
Ber: 0
============ CPAS ============
Ready (MT allows commands from TA/TE)
============ COPS ============
Automatic mode; <oper> field is ignored
Long format alphanumeric <oper>; up to 16 characters
Operator name: "TELECOM ITALIA
============ GMI  ============
Manufacturer Identification: SIMCOM_Ltd
============ GMM  ============
TA Model Identification: SIMCOM_SIM800C
============ GMR  ============
TA Revision Identification of Software Release:
Revision:1418B04SIM800C32_BT
============ GSN  ============
TA Serial Number Identification (IMEI): 866104027073389
============ CCLK ============
Date: 01/01/04
Time: 00:10:40
GMT: 0
==============================
```

In questo modo si rendono le risposte più leggibili.
Tutte le stringhe stampate sul monitor seriale sono memorizzate nella Flash del microcontrollore e non occupano spazio nella ***SRAM***.
I comandi AT di cui sopra vengono inviati a distanza di 1 secondo uno dall'altro e a gestire la sequenza di invio pensa una macchina a stati realizzata con il consueto costrutto "***switch***" (funzione "`void Send_AT_Cmd(void)`"). Anche la funzione

Listato 1 – GSM_SMSONLY

```
#INCLUDE <AVR/EEPROM.H>
#INCLUDE "UART_GSM.H"
#INCLUDE "IO_GSM.H"
#INCLUDE "ISR_GSM.H"
#INCLUDE "GENERICCMD_GSM.H"
#INCLUDE "SECURITYCMD_GSM.H"
#INCLUDE "PHONEBOOKCMD_GSM.H"
#INCLUDE "SMSCMD_GSM.H"
#INCLUDE "PHONICCALLCMD_GSM.H"
#INCLUDE "GPRSCMD_GSM.H"

SECURITYCMD_GSM   SECURITY;
PHONEBOOKCMD_GSM  PHONEBOOK;
SMSCMD_GSM        SMS;
PHONICCALLCMD_GSM PHONICCALL;
ISR_GSM           ISR;
IO_GSM            IO;
GPRSCMD_GSM       GPRS;

CONST CHAR SMS_PHONE_NUMBER[] PROGMEM = "\"+3934900\"";
CONST CHAR SMS_TEXT[] PROGMEM = "HELLO WORLD";

#DEFINE TRUE  0
#DEFINE FALSE 1

TYPEDEF VOID STATE;
TYPEDEF STATE (*PSTATE)();
//  STATES MACHINE USED TO MANAGE THE INPUTS (P3 AND P4)
PSTATE INPUT_MANAGEMENT;

BOOLEAN SENDSMS;

UINT8_T  DEBOUNCINGTIMEOUT;    //DEBOUNCING TIMEOUT
UINT16_T TIMEOUTP3;            //TIMEOUT TO MANAGE P3
UINT8_T P3_BUTTON = 9;         //INPUT BUTTON P3
UINT8_T P4_BUTTON = 10;        //INPUT BUTTON P4

UNION  DIGINPUTSTATUS {
  UINT8_T  INPUT;
  STRUCT {
    UINT8_T  P3_BUTTON  : 1; //  BIT 0
    UINT8_T  P4_BUTTON  : 1; //  BIT 1
    UINT8_T  FREE       : 6;
  } IN;
} DIGINPUTSTATUS;

UNION  DIGINPUTREADED {
  UINT8_T  INPUT;
  STRUCT {
    UINT8_T  P3_BUTTON  : 1; //  BIT 0
    UINT8_T  P4_BUTTON  : 1; //  BIT 1
    UINT8_T  FREE       : 6;
  } IN;
} DIGINPUTREADED;
UINT8_T DIGINPUTVAR;

VOID SETUP() {
  //MANAGE THE DIGITAL INPUTS (P3 AND P4)
  INPUT_MANAGEMENT = INPUT_IDLE;

  GSM.PSWDEEPROMSTARTADDSETUP();
  SETUPTIMER5();           //INITIALIZE TIMER 5
  SETINPUTPIN();
  IO.SETOUTPUTLED();       //SETS I/O LEDS
  IO.SETOUTPUTTRIGGER();   //SETS I/O TRIGGERS
  IO.CHECKOUTPUTLED();     //CHECKS I/O OUTPUT (LEDS)
  IO.CHECKOUTPUTTRIGGER(); //CHECKS I/O OUTPUT

  GSM.ENABLEDISABLEDEBUGSERIAL(TRUE, BAUD_115200);
  ISR.ENABLELIBINTERRUPT();  //ENABLES INTERRUPT
  GSM.SETBAUDRATEUART1(TRUE, BAUD_19200);
  GSM.CLEARBUFFER();         //CLEAR LIBRARY UART BUFFER

  GSM.INITPOWERON_GSM();     //START GSM INITIALIZATION
```

(Continua)

che si occupa di stampare a monitor le informazione estrapolate lavora sfruttando una macchina a stati con medesimo costrutto (funzione "*void PrintDataReceived(void)*").

SKETCH GESTIONE SMS E CHIAMATA VOCALE

Nel **Listato 1** trovate lo sketch "***GSM_SmsOnly***" in grado di gestire gli SMS. Si tratta un uno sketch molto semplice e dedicato a questa specifica funzione. Lo sketch "***GSM_Sms_PhonicCall***" (che troverete tra i numerosi esempi della libreria) ci permette invece di provare alcune funzioni base riguardanti l'invio/ricezione degli SMS e la possibilità di effettuare una chiamata vocale verso un numero di cellulare fisso, memorizzato nella Flash, oppure di chiamare un numero di telefono qualsiasi tramite apposita stringa di comando inviata da monitor seriale.
Questo sketch è compatibile con le schede Arduino Mega 2560 e Fishino Mega. Ricordatevi di selezionare nella libreria la scheda Arduino che si vuole utilizzare.
Questo sketch è composto da quattro file:

- file "***GSM_Sms_PhonicCall***" contenente tutte le dichiarazioni di variabili e costanti usate nello sketch nonché le funzioni base usate per gestire l'applicazione;
- file "***DigitalInput***" contenente le funzioni per la gestione dei pulsanti P3 e P4 dello shield (debouncing pulsanti e codice per attivazione funzioni di invio SMS o attivazione chiamata vocale a numero di rete fissa memorizzato in Flash);
- file "***DigitalOutput***" contenente la funzioni di gestione degli I/O configurati come uscite (Trigger e LED);
- file "***TimerInt***" contenente il codice di configurazione e gestione del Timer5 usato come base dei tempi per questo sketch.

Cominciamo la trattazione dal file "***GSM_Sms_PhonicCall***" il quale vede in testa al file la configurazione delle variabili e delle costanti usate nello sketch. Tutte le stringhe usate sono memorizzate in Flash.
Nella sezione di Setup ***(void setup()***) ritroviamo le funzioni di inizializzazione già discusse nel precedente sketch con l'aggiunta dell'inizializzazione del Timer 5 usato come base dei tempi:

```
SetupTimer5();
```

cui seguono le due funzioni di inizializzazione degli ingressi per la gestione dei pulsanti P3 e P4 nonché l'inizializzazione degli I/O configurati come

Listato 1 - segue

```
  GSM.UARTFLAG.BIT.ENABLEUARTSM = 0;
  GSM.EXECUTEUARTSTATE();
  GSM.UARTFLAG.BIT.ENABLEUARTSM = 1;
  DELAY(500);
  ISR.TIMEOUTWAIT = T_15SEC;
  SENDSMS = TRUE;
}

VOID LOOP() {
  GSM.EXECUTEUARTSTATE();
  IF (GSM.GSMFLAG.BIT.GSMINITINPROGRESS == 1) {
    GSM.INITGSMSENDCMD();
    GSM.INITGSMWAITANSWER();
  } ELSE {

    GSM.UARTCONTINUOUSLYREAD();
    GSM.PROCESSUNSOLICITEDCODE();
    GSM.GSMANSWERSTATEPROCESS();

    IF (SENDSMS == TRUE) {
      SENDSMS = FALSE;
          GSM.READSTRINGFLASH((UINT8_T  *)SMS_PHONE_NUMBER,
      (UINT8_T *)PHONEBOOK.PHONENUMBER,
       STRLEN(SMS_PHONE_NUMBER));
      GSM.READSTRINGFLASH((UINT8_T *)SMS_TEXT,
      (UINT8_T *)SMS.SMSTEXT, STRLEN(SMS_TEXT));
      SMS.SETCMD_AT_CMGS();
    }
    IO.LEDBLINK(PIN_LED9, 25, T_1SEC);
  }

  DEBOUNCINGINPUT();  //DEBOUNCING (P3 AND P4)
  INPUT_MANAGEMENT(); //MANAGE DIGITAL INPUT
}
```

uscite ovvero i LED di trigger e diagnostica:

```
SetInputPin();
SetOutputPin();
```

Le restanti righe di inizializzazione sono identiche al precedente sketch, anche la seriale di debug, nonché quella di comunicazione verso il modulo GSM sono configurate nello stesso modo ovvero 115.200 Baud per la prima e 19.200 Baud per la seconda.
Procediamo analizzando la funzione di main (***void loop()***) nella quale ritroviamo una serie di funzioni che vengono richiamate in sequenza all'infinito.
Le prime due funzioni gestiscono i pulsanti P3 e P4:

```
DebouncingInput();
Input_Management();
```

La prima si occupa di intercettare la pressione dei tasti P3 e P4 ai quali applica un debouncing di 50 ms. In questo sketch, il tempo di debouncing è impostato a 50 ms ma è possibile aumentarlo o diminuirlo a seconda delle necessità. Tutto sta nel caricare la costante di tempo voluta nella variabile temporale "*DebouncingTimeOut*".
La seconda serve per gestire la macchina a stati riguardante le funzioni correlate a P3 e P4. Se il pulsante P3 è premuto per più di 3 secondi viene inviato un SMS

a un numero di telefono memorizzato in Flash; se P4 è premuto per lo stesso tempo parte una chiamata vocale verso un numero di telefono in Flash. A queste funzioni appena descritte segue:

```
ProcessStateMachineGsm();
```

Qui trova posto il codice di gestione della comunicazione seriale verso il modulo GSM con il consueto codice condizionale che discrimina se si sta eseguendo una inizializzazione del modulo GSM o un invio di comando AT a regime.
Questo codice è identico a quello usato nel precedente sketch tranne per la gestione dei led che dipende ovviamente dall'applicazione. A questo punto seguono tutte le funzioni riguardanti l'applicazione in essere, in particolare abbiamo:

```
ProcessSerialCmd();
ProcessUserAtCmd();
ProcessGenericAtCmd();
```

La prima serve per interpretare i comandi stringa inviati dal monitor seriale. Infatti tramite il monitor seriale è possibile dire all'applicazione a chi inviare un SMS generico, specificandone anche il contenuto, nonché decidere a quale numero inoltrare una chiamata vocale generica.
Ma vediamo meglio come fare: supponiamo di voler inviare un SMS contenente la stringa di testo "***How are you?***" al numero di telefono "***+393490000000***"; il comando stringa da usare sarà il seguente:

```
CmdSms: "+393490000000"#How are you?
```

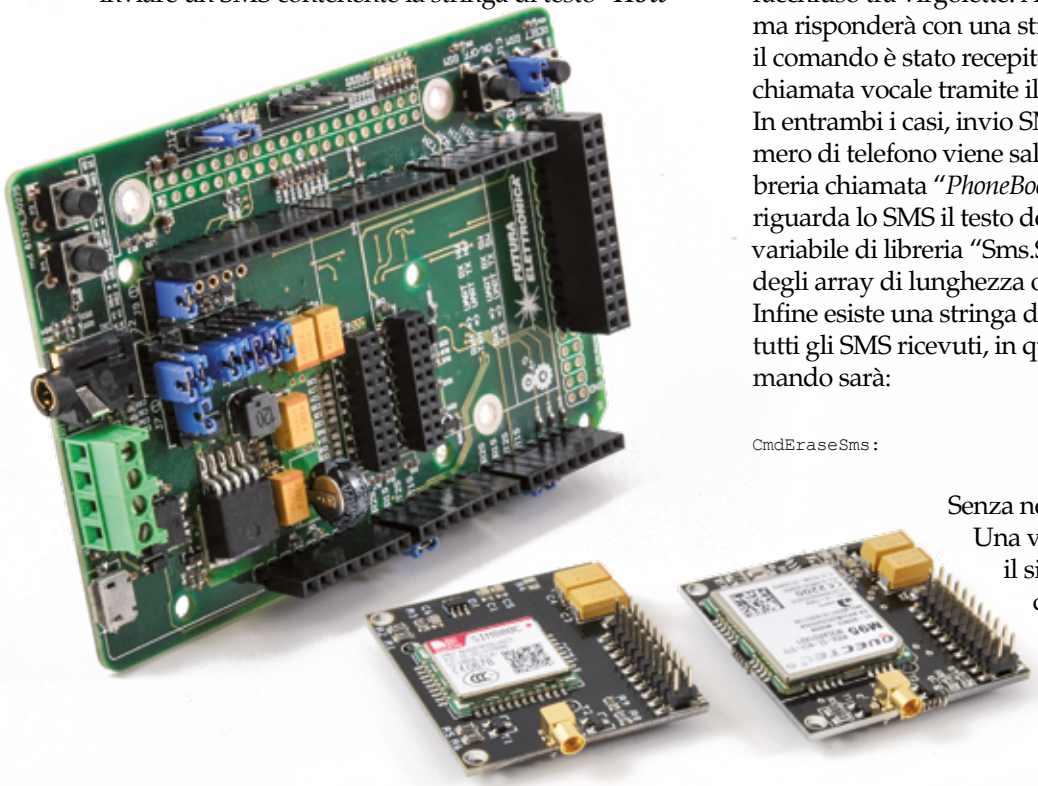
La stringa "***CmdSms:***" identifica che si vuole inviare un SMS, segue il numero di telefono che deve sempre comprendere le virgolette e infine la stringa di testo che si vuole inviare.
Tra il numero di telefono e la stringa di testo deve essere sempre presente il separatore "**#**". Se la stringa di comando è corretta il sistema stampa a video la seguente stringa "*# Command received by user → CmdSms: ……..*" dove al posto dei puntini c'è il numero di telefono e la stringa dello SMS che si sta inviando. Subito dopo avere ricevuto il comando il sistema darà inizio all'invio dello SMS tramite il comando AT+CMGS. Se la spia seriale è attiva si vedrà il comando AT passare sulla seriale con la relativa risposta.
Se invece si vuole eseguire una chiamata vocale verso un numero di telefono a piacere il comando da inviare sarà:

```
CmdCall: +393490000000
```

In questo caso, oltre alla stringa di comando differente, si ha che il solo parametro da passare è il numero di telefono che diversamente da prima non deve essere racchiuso tra virgolette. Anche in questo caso il sistema risponderà con una stringa di testo ad indicare che il comando è stato recepito e subito dopo inizierà la chiamata vocale tramite il comando ATD.
In entrambi i casi, invio SMS o chiamata vocale, il numero di telefono viene salvato in una variabile di libreria chiamata "*PhoneBook.PhoneNumber*". Per quanto riguarda lo SMS il testo dello stesso viene caricato nella variabile di libreria "Sms.SmsText". Entrambi sono degli array di lunghezza opportuna.
Infine esiste una stringa di comando per cancellare tutti gli SMS ricevuti, in questo caso la sintassi del comando sarà:

```
CmdEraseSms:
```

Senza nessun parametro aggiuntivo. Una volta recepito il comando il sistema inizierà un loop di cancellazione degli SMS dalle celle di memoria della SIM a partire dalla cella 1 fino alla cella 30. Il comando di cancellazione è AT+CMGD a cui va



passato il numero della cella da cancellare. Nel caso in cui si invia una stringa di comando errata il sistema ritornerà la stringa di errore:

```
# Command syntax error. Retry
```

Questa è l'ultima stringa di comando inviabile da monitor seriale. Nel caso in cui fosse necessario è possibile aggiungere nuove stringhe di comando aggiungendo il relativo codice.
Passiamo ora ad analizzare la funzione "`ProcessUserAtCmd()`" la quale contiene il codice per l'invio dei comandi AT usati nell'applicazione. Il codice, tramite un costrutto "***switch***" discrimina se deve inviare un SMS o fare una chiamata vocale e tale azione viene selezionata dai pulsanti P3 e P4. Quindi pressione del pulsante P3 per più di 3 secondi attiva l'azione di invio SMS (l'SMS conterrà la stringa di testo: "`FuturaGroup GSM Library R.1.0.`" la quale è memorizzata in Flash. Il numero di telefono a cui sarà inviato lo SMS è anch'esso memorizzato in Flash. Guardando il codice è il numero comprendente anche le virgolette). Se invece si preme il pulsante P4 per più di 3 secondi si attiverà la chiamata vocale verso un numero di telefono sempre memorizzato in Flash (questa volta senza virgolette). Per chiudere una chiamata vocale in corso premere nuovamente il pulsante P4, basta un tocco senza pressione prolungata.
Infine la funzione "`ProcessGenericAtCmd()`" la quale, come il precedente sketch, prevede una serie di comandi AT da inviare in ciclo continuo al modulo GSM. In questo sketch abbiamo i seguenti comandi: stato del modulo GSM (AT+CPAS), verifica della potenza del segnale GSM (AT+CSQ), verifica della registrazione del modulo GSM alla cella tramite proprio operatore (AT+CREG?), verifica necessità codice pin (AT+CPIN?), verifica dell'operatore (AT+COPS?), lettura dello SMS ricevuto (AT+CMGR) solo nel caso di intercettazione di un "***Unsolicited Code***", cancellazione dello SMS ricevuto (AT+CMGD) e se richiesto tramite stringa di comando cancellazione di tutti gli SMS ricevuti (AT+CMGD).
Come ultima funzione è rimasta `ProcessSmsReceived();` Questa, dato un SMS ricevuto, ha il compito di processarlo e se contiene delle stringhe di comando valide esegue dei comandi prestabiliti. Il testo dello SMS ricevuto si trova nella variabile di libreria "`Sms.SmsText`".
Nel caso dello sketch i comandi possono agire sul solo LED 7, presente sulla shield GSM, ed eseguire le seguenti: accendere il LED 7, spegnere il LED 7 oppure fare lampeggiare il LED 7 per 3 secondi.
Quindi inviando un SMS con la dicitura "`Led7On`" si vedrà accendersi il LED 7, viceversa se si invierà la stringa di comando "`Led7Off`" si vedrà spegnersi il LED 7. Invece la stringa "`Led7Blink`" farà lampeggiare il LED 7 per 3 secondi. Il lampeggio avrà un periodo di 250 millisecondi con duty-cycle del 50%. Le stringhe di comando devono essere inviate singolarmente rispettando le maiuscole e le minuscole in quanto il codice è "*case sensitive*".
Per concludere, di seguito riportiamo le costanti stringa salvate nella memoria Flash che identificano rispettivamente il numero di telefono che si vuole usare per inviare lo SMS, il testo da utilizzare e infine il numero di telefono che si vuole usare per la chiamata vocale (pulsante P3 per invio SMS e pulsante P4 per chiamata vocale come descritto prima).

```
//=================================================
const char SMS_PHONE_NUMBER[] PROGMEM = "\"+3934900\"";
const char SMS_TEXT[] PROGMEM = "FuturaGroup GSM Library
R.1.0.";
const char CALL_PHONE_NUMBER[] PROGMEM = "+3934900";
//=================================================
```

Come si può notare il numero di telefono per l'invio dello SMS deve essere sempre racchiuso tra virgolette mentre il numero di telefono per la chiamata vocale no. Il testo da abbinare allo SMS deve avere lunghezza massima di 160 caratteri.

SKETCH GESTIONE COMANDI DI SICUREZZA E RUBRICA TELEFONO

Lo sketch "***GSM_Security_PhoneBook***" ci permette di testare una serie di comandi inerenti la sicurezza e la gestione della rubrica telefonica. Oltre a ciò abbiamo mantenuto la possibilità di gestire una chiamata vocale in uscita. Anche in questo caso lo sketch è composto da più file distinti, come fatto nel precedente, di cui abbiamo già dato una sommaria descrizione.
In questo caso l'unica differenza sta nel file "***GSM_Security_PhoneBook***" il quale deve gestire dei comandi AT differenti rispetto al precedente sketch. Il file "***GSM_Security_PhoneBook***", come del resto il precedente, vede in testa una serie di dichiarazioni di variabili e costanti. Le costanti possono essere di tipo stringa (salvate in ***FLASH***, identificano il testo da visualizzare sul monitor seriale o i comandi stringa da inviare sempre da monitor seriale), sia delle costanti numeriche (identificano gli stati delle macchine a stati usate nello sketch). Oltre a ciò sono anche definite le variabili globali usate in questo sketch.
La sezione di Setup (***void setup()***) ritrova una serie di funzioni di inizializzazione necessarie alla configurazione della libreria, come si può osservare si richiamano le medesime funzioni già usate nel precedente. Anche in questo caso il monitor seriale è impostato a una velocità di 115.200 Baud e la seriale di comunicazione

verso il modulo GSM a 19.200 Baud.
Analizziamo ora la funzione main (***void loop()***) nella quale ritroviamo una serie di funzioni che vengono richiamate in sequenza all'infinito. Come nel precedente abbiamo due funzioni per la gestione del debouncing dei due pulsanti P3 e P4 dove in questo caso si usa solo il pulsante P4 per innescare la chiamata vocale a un numero di telefono salvato nella memoria Flash:

```
DebouncingInput();
Input_Management();
```

Alle funzioni appena descritte segue questa:

```
ProcessStateMachineGsm();
```

dove si trova il codice di gestione della comunicazione seriale verso il modulo GSM con il consueto codice condizionale che discrimina se si sta eseguendo una inizializzazione del modulo GSM o un invio di comando AT a regime. Questo codice è identico a quello del precedente sketch tranne per la gestione dei LED, che dipende dall'applicazione.
Seguono le funzioni:

```
ProcessSerialCmd();
ProcessUserAtCmd();
ProcessGenericAtCmd();
```

La prima serve per interpretare i comandi stringa inviati dal monitor seriale che sono diversi dal precedente sketch (più avanti diamo una dettagliata descrizione). La funzione "`ProcessUserAtCmd()`", tramite il solito costrutto "***switch***", serve per discriminare quali comandi AT si devono inviare al modulo GSM. Tra questi abbiamo la chiamata vocale verso un numero di telefono fisso (pressione pulsante P4) o verso un numero di telefono a piacere tramite comando seriale. In più ci sono tutti gli altri comandi AT gestiti dallo sketch e richiamabili solo tramite stringa di comando inviata da monitor seriale.
Infine la funzione "`ProcessGenericAtCmd()`" la quale, come il precedente sketch, prevede una serie di comandi AT da inviare in ciclo continuo al modulo GSM. In questo sketch abbiamo i seguenti: stato del modulo GSM (AT+CPAS), verifica della potenza del segnale GSM (AT+CSQ), verifica della registrazione del modulo GSM alla cella tramite proprio operatore (AT+CREG?), verifica necessità codice pin (AT+CPIN?), verifica dell'operatore (AT+COPS?).
Vediamo ora le stringhe di comando che si possono inviare dal monitor seriale; la prima stringa disponibile l'abbiamo già incontrata nel precedente sketch ovvero la possibilità di inoltrare una chiamata vocale verso un numero di telefono qualsiasi. La stringa di comando è la stessa, quindi vi invitiamo a rileggere la descrizione del precedente sketch.
Per quanto riguarda le nuove stringhe di comando partiamo dalla gestione del comando AT+CLCK per eseguire il lock/unlock di alcuni servizi messi a disposizione dal modulo GSM.
Le stringhe di comando disponibili sono:

```
CmdFacLock: <fac code>
CmdFacUnLock: <fac code>
CmdFacCheck: <fac code>
```

La prima serve per mettere il blocco (Lock) a un servizio, per funzionare il comando stringa richiede un codice che identifica il servizio su cui si vuole agire.
La **Tabella 1** espone la corrispondenza tra codice e servizio. Il codice inviato verrà poi passato come parametro alla funzione di libreria per l'invio del comando AT+CLCK. Il comando AT+CLCK prevede anche una password per eseguire il blocco del servizio, tale password viene letta dalla memoria EEPROM, ricordate quanto detto con il primo sketch analizzato?
Se invece si vuole sbloccare un servizio si deve usare la seconda stringa di comando con il rispettivo codice. Anche in questo caso la password verrà letta dalla memoria EEPROM.
Infine abbiamo una stringa di comando per interrogare il modulo GSM e vedere se il servizio è bloccato oppure no. In questo caso non serve nessuna password, come risposta il modulo GSM ritornerà "***0***" se il servizio non è bloccato e "***1***" se il servizio è bloccato. Queste informazioni sono facilmente reperibili dalla struttura dati "`Security.SecurityFlag`" e più in dettaglio dal flag "`Security.SecurityFlag.Bit.ClckStatus`".
Qui di seguito vedete l'invio del comando AT per eseguire lo sblocco del servizio riguardate il codice PIN della SIM.

```
AT+CLCK="SC",0,"4629"
OK
```

Invece di seguito riportiamo il codice che attiva il blocco della SIM tramite codice PIN.

```
AT+CLCK="SC",1,"4629"
OK
```

Infine qui di seguito vedete il codice che esegue un test sullo stato del servizio inerente alla SIM.

```
AT+CLCK="SC",2
```

SCEGLI LA TUA STAMPANTE 3D

Affidati a chi le progetta e le sviluppa

STAMPANTE 3D 40x40x40 cm in KIT

stampante 3D FDM (Fused deposition modeling)
capace di produrre stampe di 40x40x40 cm (64.000 cm³)

€ 999,00
Cod. **3D4040**

- ✔ Struttura interamente in alluminio
- ✔ Area di stampa: X 40 cm, Y 40 cm, Z 40 cm
- ✔ Diametro filamento: 1,75 mm
- ✔ Tipo di filamento: ABS, PLA, NYLON ed altri ancora
- ✔ Diametro ugello fornito: 0,4 mm
- ✔ Diametro ugelli opzionali: da 0,3 mm a 0,8 mm
- ✔ Velocità di stampa massima: 300 mm/s (in funzione dell'oggetto da stampare)
- ✔ Piatto di stampa fisso: vetro temperato da 6 mm
- ✔ Riscaldatore per piatto di stampa: 40 x 40 cm – 12V/240 W con adesivo 3M (opzionale)
- ✔ Controllabile da PC o da modulo LCD (opzionale)
- ✔ Alimentazione tramite modulo switching 220 VAC/12 VDC 350 W
- ✔ Istruzioni di montaggi in italiano con illustrazioni

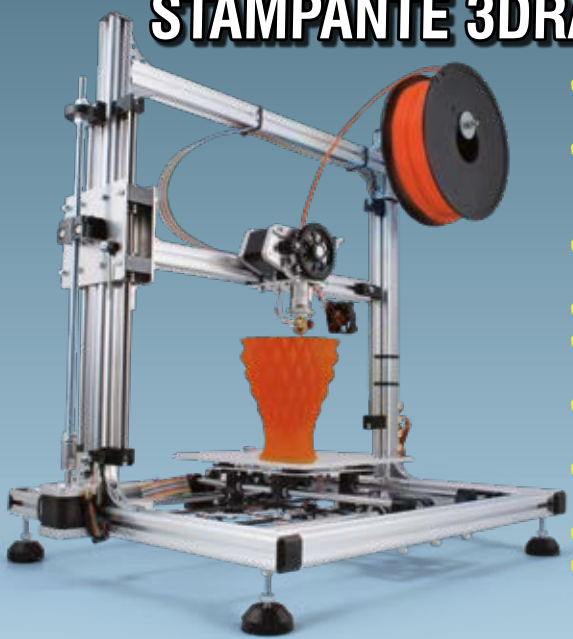
Versione montata
cod. 3D4040/M
€ 1.299,00 IVA inclusa.

STAMPANTE 3DRAG IN KIT

DESIGNED BY FUTURA ELETTRONICA

3DRAG

VASTA GAMMA DI ACCESSORI!
SCOPRILI TUTTI SU
www.futurashop.it



- ✔ Struttura in alluminio per combinare rigidità e leggerezza
- ✔ Assemblaggio semplificato con giunti metallici e parti in ABS stampate a iniezione
- ✔ Piano di stampa con movimentazione X/Y
- ✔ Estrusore movimentato sul solo asse Z
- ✔ Volume di stampa massimo: 20 x 20 x 20 cm
- ✔ Risoluzione: X e Y - 0,015 mm; Z - 0,39 micron
- ✔ Estrusore con ugello da 0,5 mm per filo da 3 mm in PLA e ABS
- ✔ Velocità di stampa tipica: 120 mm/s
- ✔ Piatto riscaldato

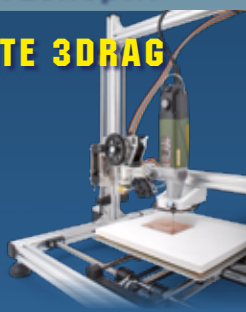
Versione montata
cod. 3DRAG/M
€ 699,00 IVA inclusa.

€ 499,00
Cod. **3DRAG/K**

E CON LA STAMPANTE 3DRAG PUOI ANCHE...

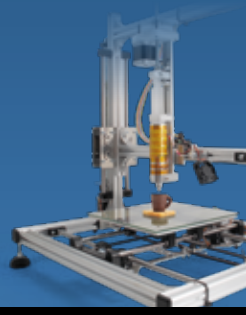
REALIZZARE CIRCUITI STAMPATI MEDIANTE INCISIONE

Minitrapano professionale con 34 accessori
Cod. PROXXON1 - € 114,00



STAMPARE CON IL CIOCCOLATO

Set estrusore Choco 3DRAG - in kit
Cod. 3DCHOCO - € 178,00



```
+CLCK: 1
```

La **Fig. 3** evidenzia il monitor seriale con le stringhe di comando inviate, ponete attenzione alla sintassi dei comandi i quali sono tutti "*case sensitive*". In caso contrario il sistema ritornerà una stringa di errore.
Un'altra stringa di comando implementata riguarda la possibilità di cambiare la password ai servizi tramite il comando AT+CPWD. Quindi abbiamo la seguente stringa inviata:

```
CmdFacSetPwd: <fac code>, <new password>
```

Questa stringa di comando richiede due parametri: il primo è il codice del servizio (vedi **Tabella 1**) mentre il secondo è la nuova password numerica la quale deve essere racchiusa tra virgolette. Di seguito, il comando inviato per cambiare la password, che in questo caso corrisponde al PIN della SIM.

```
AT+CPWD="SC","4629","1234"
OK
```

Il comando AT prevede di inviare sia il codice PIN corrente che il nuovo PIN. Il codice PIN corrente viene letto dalla memoria EEPROM. Se il comando AT va a buon fine, il vecchio codice PIN memorizzato in EEPROM viene sostituito da quello nuovo; nella **Fig. 4** trovate la relativa stringa di comando inviata dal monitor seriale.
Per ultimo analizziamo le stringhe di comando per gestire la rubrica, in questo caso abbiamo:

```
CmdSelPhonebookMem: <MEM code>
CmdQueryPhonebookMem:
CmdReadPhonebookMem: <MEM add>
CmdWritePhonebookMem: <MEM add>, <PhoneNumber>, <Phone-
NumberType>, <Text>
CmdErasePhonebookMem: <MEM add>
```

La prima stringa di comando serve per selezionare la rubrica su cui lavorare; anche in questo caso abbiamo un codice che identifica la rubrica (**Tabella 2**).
Nei nostri esempi lavoreremo con il codice 6 (SM) che identifica la rubrica della SIM, selezionabile con l'invio del comando AT+CPBS.
La seconda stringa serve per verificare quante locazioni di memoria della rubrica selezionata sono disponibili e quante occupate; la porzione seguente di codice mostra che la rubrica della SIM dispone di 250 locazioni di memoria e 5 sono occupate.

```
AT+CPBS?
```

***Tabella 1** - Corrispondenza codice/servizio*

A0	OI	OX	AI	IR	FD	SC	PN	PU	PP
0	1	2	3	4	8	9	10	11	12

```
+CPBS:"SM",5,250
OK
```

La terza stringa serve per leggere la locazione di memoria desiderata e necessita, come parametro, dell'indice desiderato; il comando AT+CPBR ritorna il numero di telefono memorizzato all'indirizzo dato, la tipologia del numero e una breve descrizione in formato stringa.
La porzione di codice seguente mostra i dati ritornati dalla lettura della locazione tre della rubrica.

```
AT+CPBR=3
+CPBR:3,"+393xxxxxxx",145,"Borisxxxxxx"
OK
```

La quarta stringa serve per salvare in rubrica un nuovo numero di telefono, il comando necessita di quattro parametri distinti ovvero indirizzo in rubrica in cui salvare il nuovo numero, il numero di telefono racchiuso tra virgolette, la tipologia del numero e infine un testo descrittivo anch'esso racchiuso tra virgolette.
La **Fig. 16** mostra quanto detto, il comando AT+CPBW salva nella locazione di memoria sei un nuovo numero di telefono.

```
AT+CPBW=6,"+3934xxxxxx",129,"Angxxxxx"
OK
```

Se si seleziona una locazione già occupata questa verrà sovrascritta. L'ultima stringa di comando serve a cancellare una locazione di memoria della rubrica per rimuovere un numero di telefono.
Qui di seguito trovate il comando AT+CPBW con il solo parametro di indice che esegue la cancellazione della locazione.

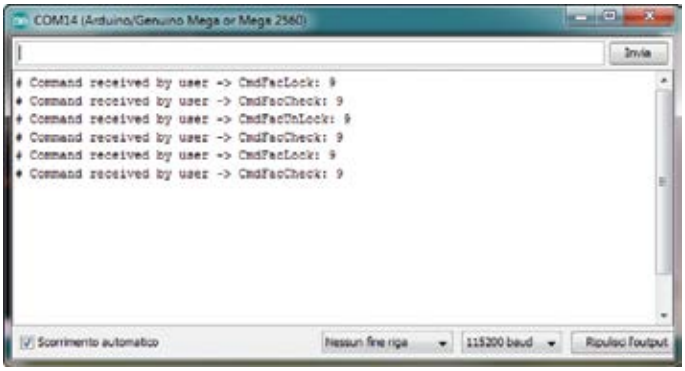


***Fig. 3** - Stringhe di comando su monitor seriale.*

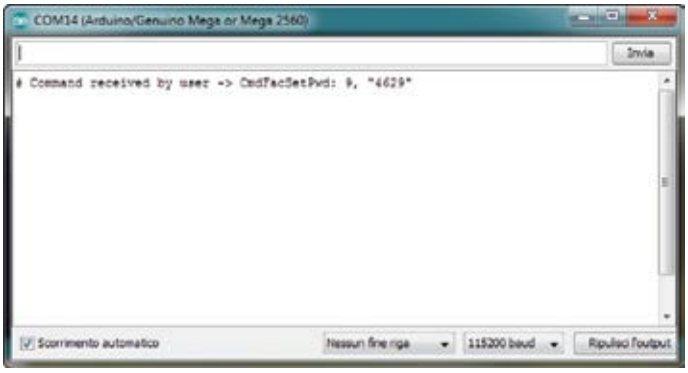


***Fig. 4** - Stringa di comando relativa alla sostituzione del PIN, visualizzata sul monitor seriale.*

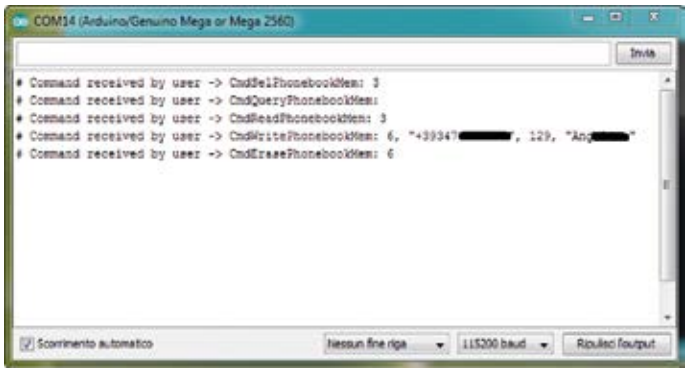


***Fig. 5** - Stringa di comando relativa alla memorizzazione di un numero telefonico, visualizzata sul monitor seriale.*

```
AT+CPBW=6<CR><LF>
<CR><LF>
OK<CR><LF>
```

La **Fig. 5** evidenzia i comandi stringa appena spiegati, passati dal monitor seriale di Arduino.

CONCLUSIONI

Con quest'ultimo sketch abbiamo concluso la carrellata degli esempi. Vogliamo dare ancora qualche consiglio sull'utilizzo della libreria e in particolare sul file "*Io_GSM.h*" nel quale ritroviamo tutte le definizioni necessarie a configurare il funzionamento della libreria in base al modulo GSM che si vuole utilizzare e in base allo hardware a disposizione. Quindi ricordatevi sempre di:

- selezionare uno dei moduli GSM supportati;
- selezionare quale scheda Arduino si intende usare tra quelle supportate;
- selezionare la revisione hardware, la quale deve corrispondere alla revisione hardware stampata sul PCB;
- selezionare tra UART1 hardware o software (la scelta dipende dalla configurazione dei jumper, per la quale rimandiamo alla tabella presente nel file);
- abilitare le sezioni di gestione delle risposte da parte del modulo GSM, rammentando che sono disponibili diverse sezioni a seconda delle categorie di comandi AT; in questo caso si possono abilitare più sezioni contemporaneamente, quindi abbiamo:
  - abilitazione sezione di gestione risposte a comandi AT generici (`ENABLE_ANSWER_GENERIC_AT_CMD_STATE`);
  - abilitazione sezione di gestione risposte a comandi AT inerenti la sicurezza (`ENABLE_ANSWER_SECURITY_AT_CMD_STATE`);
  - abilitazione sezione di gestione risposte a comandi AT inerenti la rubrica (`ENABLE_ANSWER_PHONEBOOK_AT_CMD_STATE`);
  - abilitazione sezione di gestione risposte a comandi AT inerenti gli SMS (`ENABLE_ANSWER_SMS_AT_CMD_STATE`);
  - abilitazione sezione di gestione risposte a comandi AT inerenti le chiamate vocali (`ENABLE_ANSWER_PHONIC_CALL_AT_CMD_STATE`).

Se si omette di abilitare una sezione inerente ai comandi AT inviati, il sistema non riuscirà a gestire le risposte e rimarrà in attesa. Se necessario, abilitare/disabilitare le sezioni di debug.
Ricordate che esiste una definizione globale che disabilita tutto il codice di debug nella libreria e sotto-definizioni per attivare codice di debug mirato.

***Tabella 2** - Identificativi della rubrica.*

DC	EN	FD	MC	ON	RC	SM	LA	ME	BN	SD	VM	LD
0	1	2	3	4	5	6	7	8	9	10	11	12

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