Università degli Studi di Padova
Facoltà di Ingegneria

Corso di Laurea in
Ingegneria Informatica

Terminale di Programmazione TLC

Remote Control TLC

Relatore: Michele Moro
Correlatore: Carlo Magro

Laureanda: Bruna Mazzi 541591 / IF

Anno Accademico 2010/2011

INDICE

SOMMARIO

Delta Erre Safe s.r.l. è un'azienda veronese che si occupa di apparecchiature elettroniche di sicurezza: centrali di rivelazione incendio, gruppi di alimentazione, pannelli ottico acustici, rivelatori gas e antintrusione.
L'obiettivo dello stage svolto è stato quello di sviluppare un terminale di visualizzazione e programmazione (TLC) da utilizzare in abbinamento con due tipi differenti di sensori intelligenti: sensori di rivelazione gas serie GD1xx e rivelatori lineari di fumo DLFB. Il progetto comprendeva anche la realizzazione di un banco di collaudo per il test dei terminali durante la fase di produzione.
L'implementazione è stata quindi suddivisa in quattro grandi fasi:

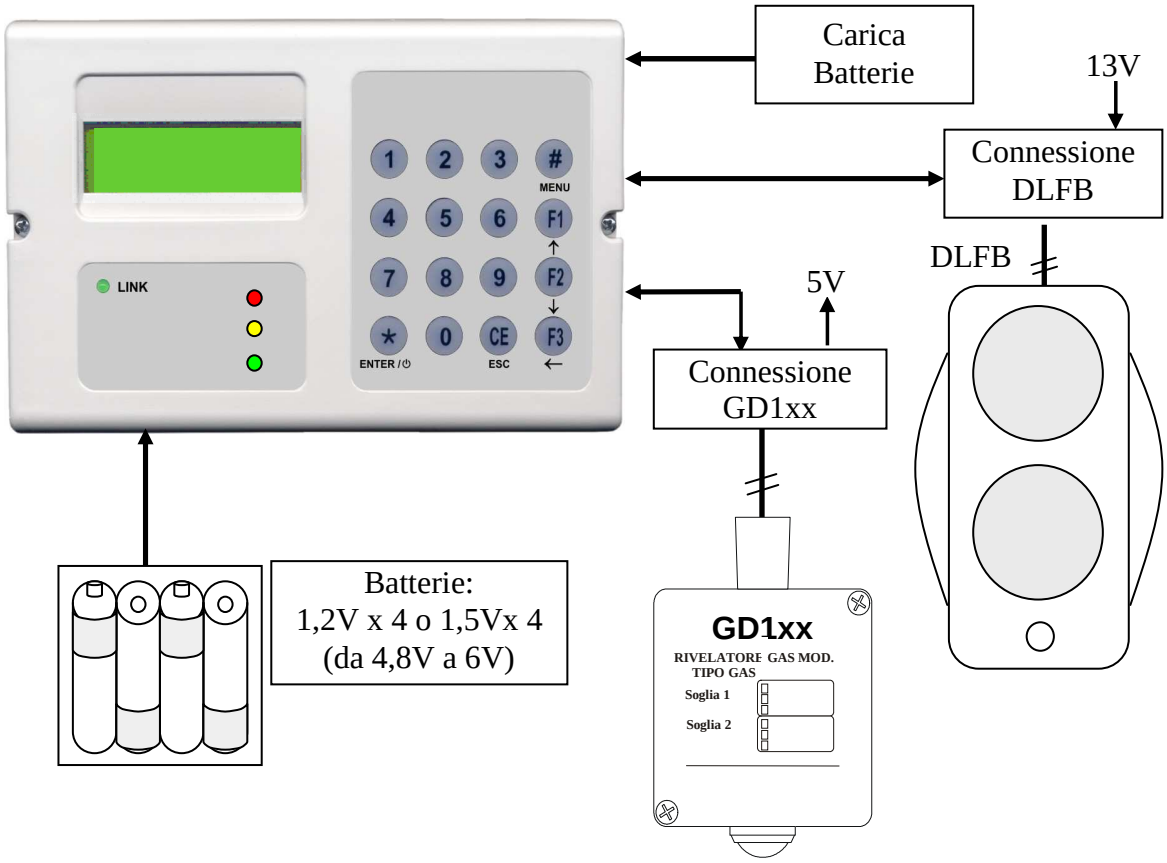
- Realizzazione delle funzionalità base del terminale TLC: gestione tastiera, display, led, alimentazione, rivelazione guasti interni, etc;
- Gestione rivelatori di gas secondo esigenze interne all'azienda;
- Gestione barriere di fumo tramite seriale RS485 secondo le specifiche tecniche e il protocollo di comunicazione definiti dall'azienda richiedente;
- Realizzazione del banco e del software di collaudo per la verifica dei componenti hardware e delle funzionalità software utilizzati durante la produzione.

La realizzazione del terminale TLC si è conclusa con l'approvazione del prodotto finale da parte dell'azienda richiedente ed il dispositivo è attualmente in produzione, applicato sia ai rivelatori di gas sia alle barriere rivelatrici di fumo.
Il banco di collaudo realizzato permette di effettuare un controllo completo, quasi totalmente automatizzato e veloce del terminale. Il software permette il salvataggio dei dati riguardanti il collaudo all'interno del database aziendale, permettendone poi la consultazione in caso di guasti/riparazioni e per valutare le non conformità previste dalla certificazione di qualità dell'azienda.

1 INTRODUZIONE

Il progetto sviluppato durante il tirocinio svolto è nato dall'esigenza, sia da parte della nostra ditta sia dell'azienda produttrice delle barriere di rivelazione fumo DLFB, di avere un dispositivo di interfaccia con i rivelatori che permettesse di eseguire operazioni di manutenzione e taratura sui sensori in modo facile e senza doverli togliere dagli impianti sui quali vengono installati.

L'architettura generale di collegamento tra il terminale ed i rivelatori è la seguente:

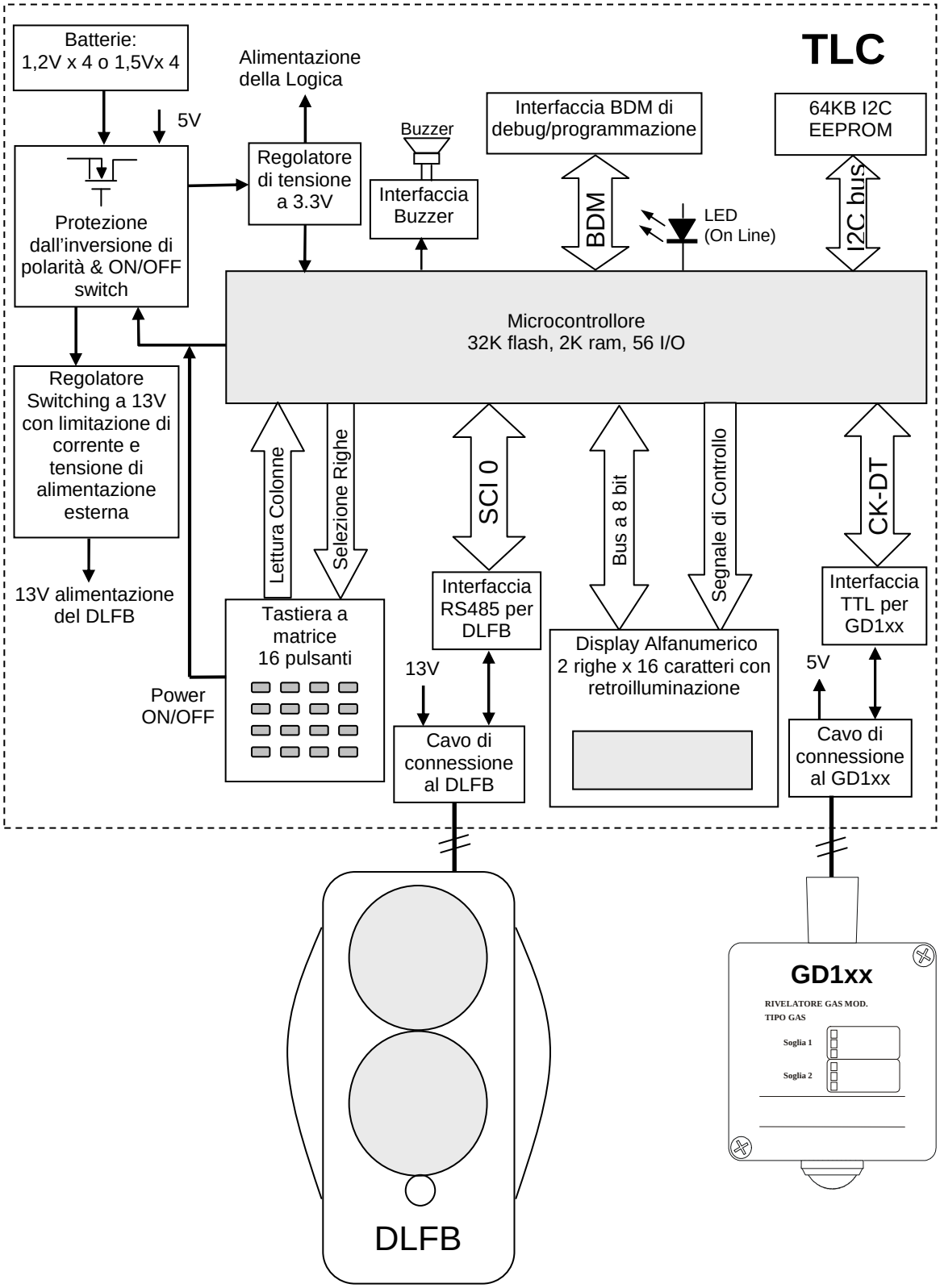


1.1 Il terminale TLC

Questo dispositivo portatile doveva poter provvedere:

- Al dialogo con i rivelatori GD1xx e DLFB tramite specifico collegamento seriale e con un'organizzazione di tipo Master-Slave. Il dialogo sarà gestito per mezzo di protocolli di comunicazione proprietari sviluppati dai costruttori dei rivelatori;
- Alla gestione della parametrizzazione dei rivelatori ed alla lettura ed interpretazione dei valori di misura ricavati dagli stessi;
- Alla comunicazione con l'utilizzatore tramite display e tastiera;
- Alla minimizzazione dei consumi energetici dato l'utilizzo di alimentazione a batterie;

Il terminale è stato sviluppato utilizzando i seguenti componenti hardware:

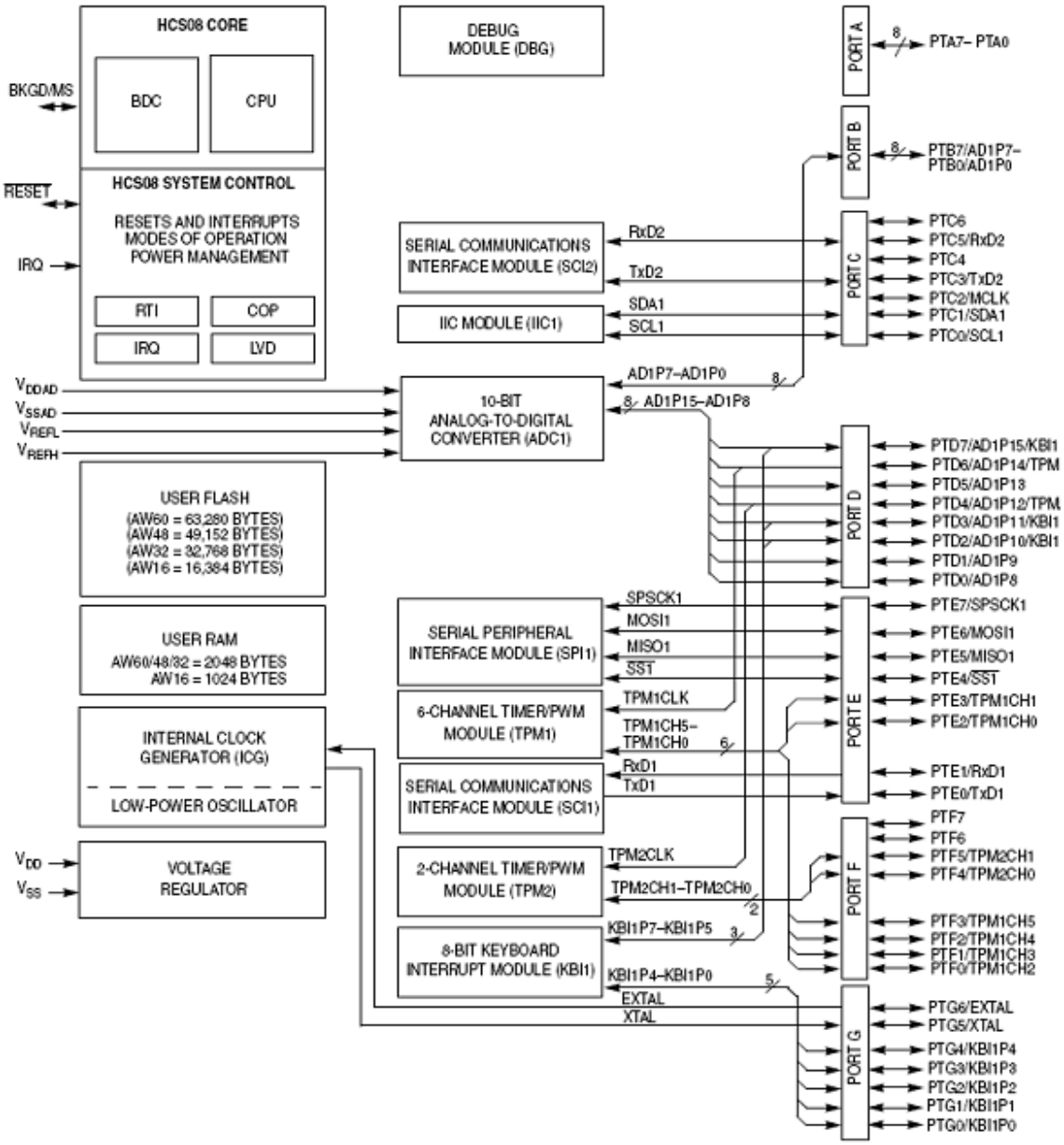


- **Microcontrollore MC9S08AW**

L'MC9S08AW è un microcontrollore ad 8-bit prodotto da Freescale appartenente alla categoria Automotive, che lavora con una tensione tra 2,7V e 5,5V e che gestisce al suo interno:

FLASH (Byte)	32K
RAM (Byte)	2K
Comunicazione Seriale	2 SCI, 1 SPI, 1 IIC
ADC	16 channel 10-bit
Timer	One 16-bit timer/PWM (2-6 channels)
I/O pin	Up to 56
Package	64-Pin

La sua architettura interna è la seguente:



La scelta di questo microcontrollore è stata dettata da motivi di affidabilità, di reperibilità e di gestione (il componente è già utilizzato su altri prodotti) ed anche per l'ottimo rapporto tra prestazioni e costo del componente. Inoltre per la realizzazione del firmware Delta Erre Safe potrà utilizzare tool di sviluppo e ambiente operativo già impiegati per la realizzazione di altri prodotti, con l'evidente vantaggio di ottimizzare i costi e i tempi di realizzazione.

- **Interfaccia TTL**
  L'interfaccia TTL "open collector" viene utilizzata per la comunicazione tramite segnali CK e DT con i rivelatori di gas serie GD1xx. Il controllo dei segnali CK (clock) e DT (I/O data) è gestito direttamente da software tramite routine di interrupt collegate al timer interno del microcontrollore.
  Maggiori dettagli sulla gestione di questa interfaccia verranno forniti nel capitolo 8.

- **Interfaccia RS485**
  L'interfaccia RS485 viene utilizzata per la comunicazione tramite SCI con il rivelatore di fumo DLFB. Il driver utilizzato è di tipo a basso consumo (aspetto fondamentale per l'alimentazione a batteria) e con range di funzionamento esteso relativamente ai livelli di segnale. Maggiori dettagli sulla gestione di questa interfaccia verranno forniti nel capitolo 9.

- **Interfaccia BDM**
  L'interfaccia BDM è specifica per i microcontrollori Freescale ed è utilizzata per le operazioni di debugging e per la programmazione del firmware sul terminale.

- **Memoria E^2PROM**
  Nonostante l'utilizzo di una memoria esterna non sia strettamente indispensabile per questo tipo di dispositivo, si è scelto di inserire una memoria EEPROM da 64K, da utilizzare eventualmente per la gestione dei messaggi in più lingue e per i dati di configurazione. La memoria è di tipo seriale con bus I^2C ed è direttamente collegata al microcontrollore che dispone di una sezione apposita per questa tipologia di bus.

- **Display alfanumerico**
  Il display utilizzato è di tipo alfanumerico, a 16 caratteri per 2 linee, basato su microcontrollore HD44780. E' stato scelto questo tipo di display per ragioni di compatibilità con il contenitore ed inoltre per motivi di costo, di reperibilità e di basso consumo. Il display è interfacciato al microcontrollore tramite bus parallelo a 8 bit e tramite segnali di controllo che consentono le operazioni di scrittura e di comando. Il display è dotato di retroilluminazione a led la cui accensione, controllata da software, potrà essere attivata a tempo tramite un'operazione da tastiera.
  Una volta che il terminale è in comunicazione con il DLFB, la retroilluminazione viene comandata direttamente dal rivelatore di fumo.

- **Tastiera a Matrice 4x4**
  La tastiera a matrice è gestita tramite 4 righe di selezione (output) e 4 segnali di stato colonna (input). I contatti della tastiera sono realizzati direttamente sul circuito stampato tramite apposite piazzole metallizzate con una speciale placcatura

che ne previene l'ossidazione. I tasti sono invece in gomma, con contatti in gomma conduttiva, e sono vincolati tra loro da un supporto comune.
E' stato scelto questo tipo di tastiera poiché non costosa, affidabile, compatibile con il contenitore che si intende utilizzare e semplice da gestire con un numero ridotto di I/O.

- **Led e buzzer**
  Il telecomando TLC-PRM è dotato di alcuni led e di un buzzer piezoelettrico.
  I led saranno accesi / lampeggianti per indicare lo stato del terminale e dei rivelatori ad esso collegati.
  Il buzzer si attiverà con un suono breve per indicare la pressione di un tasto o potrà essere attivato con un suono di durata maggiore per segnalare particolari situazioni (guasti, allarmi, selezioni errate, etc.).

- **Sezione di Alimentazione**
  Il dispositivo può essere alimentato in 3 differenti modalità:
  - Tramite 4 batterie ricaricabili NIMH da 1,2V (4,8 V) o in alternativa tramite quattro pile alcaline da 1,5V (6V);
  - Dal rivelatore di Gas;
  - Collegando il caricabatterie anche senza la presenza delle batterie.

  Il terminale dovrà provvedere, una volta effettuata la connessione, all'alimentazione del rivelatore DLFB qualora lo stesso non disponga di sorgente di alimentazione propria.

- **Switch elettronico di accensione/spegnimento**
  L'accensione e lo spegnimento del dispositivo avvengono tramite uno switch elettronico realizzato con un transistor MOS a canale P che intercetta il positivo di alimentazione generale. L'accensione avviene meccanicamente tramite un tasto dedicato (Enter) ed in seguito è confermata dal microcontrollore che la mantiene attiva per il tempo previsto. Lo spegnimento è gestito dal microcontrollore e può avvenire a tempo, per inattività del dispositivo, o tramite la pressione prolungata (5S) del tasto di accensione/spegnimento (Enter).

- **Regolatore a 3,3V**
  Tramite un regolatore lineare, a basso consumo, la tensione di batteria viene stabilizzata a 3,3V per l'alimentazione del microcontrollore e dei dispositivi logici e di interfaccia ad esso collegati.
  Il regolatore può fornire una corrente massima di 250mA e può lavorare con una tensione di ingresso compresa tra min 3,7V e max.10V.

- **Step-Up Converter**
  E' presente un regolatore switching, basato sul dispositivo MC34063 in configurazione step-up, per la generazione della tensione di 13V necessaria per alimentare il sensore DLFB. Il regolatore è predisposto per fornire in uscita una corrente massima di 50mA è può operare con una tensione di ingresso compresa tra min 3,7V e max.40V.

- **Sensore di gas GD1xx**

I rivelatori di gas della serie GD1xx sono impiegati per rivelare la presenza di sostanze combustibili o tossiche in un'atmosfera costituita principalmente da aria e sono stati concepiti e sviluppati per offrire caratteristiche professionali di sensibilità e stabilità. Il rivelatore è costituito da due schede:
  - Una scheda base interamente controllata da un microprocessore che sovrintende tutte le funzioni di misura del gas da rilevare e di controllo dell'elemento sensibile;
  - Una seconda scheda, gestita da un microprocessore, permette al rivelatore di interfacciarsi sulle linee di rivelazione indirizzate e di trasmettere alla centrale le informazioni di guasto, allarme e preallarme.

- **Sensore di fumo DLFB**

DLFB è un rivelatore lineare di fumo a riflessione analogico-interattivo ideale per sorvegliare ampie superfici ma anche edifici particolarmente alti dove il fumo non può essere rilevato dai rivelatori puntiformi fissati a soffitto.
La particolare copertura da 3 a 100m, il collegamento diretto su loop senza moduli di interfaccia e senza alimentazione esterna e la possibilità di selezionare diverse soglie di intervento, rendono DLFB un rivelatore lineare altamente performante nel mercato di riferimento**.**

Per lo sviluppo del firmware di controllo del prodotto si è scelto di utilizzare un ambiente di sviluppo dedicato e l'interfaccia di emulazione e programmazione ML12 della P&E.
Il linguaggio di programmazione utilizzato per l'implementazione del codice è il C.

È stata fatta la scelta di questo ambiente e questo linguaggio perché sono già stati utilizzati da Delta Erre Safe per lo sviluppo di altri prodotti.

Per il collaudo del prodotto finale si è creato un apposito banco di test a cui viene abbinata un'applicazione software sviluppata in Visual Basic, che provvede ad eseguire numerosi controlli automatici e a guidare l'operatore nei passaggi manuali della procedura di collaudo.

1.2 L’azienda

Delta Erre Safe S.r.l. è un’azienda specializzata nella produzione e commercializzazione di apparecchiature servizi per la rivelazione incendio, gas e antintrusione nel mercato della distribuzione.
Da azienda originariamente specializzata (sin dai primi anni 80) nella realizzazione di prodotti per clienti OEM, diventa leader, per i distributori della sicurezza, nella progettazione, realizzazione e commercializzazione di una gamma completa di prodotti (di produzione propria e del gruppo) e di accessori complementari al settore incendio quali:
- centrali convenzionali, analogiche e di spegnimento
- gruppi di alimentazione
- pannelli ottico acustici
- rivelatori di gas

Nel 2005 Delta Erre è diventata parte qualificata del gruppo francese DEF, portando in dote un patrimonio di professionalità e flessibilità estremamente apprezzato.
Ad oggi Delta Erre realizza il 50% del suo fatturato con l’esportazione e commercializza i suoi prodotti in tutta Europa per mezzo delle filiali di commercializzazione del gruppo DEF.

Il gruppo DEF con sede a Massy (Francia) è divenuto in 50 anni il leader francese della rivelazione incendio, attraverso le sue esperienze nella ricerca e sviluppo, nella produzione, nell’ingegnerizzazione, nell’assistenza tecnica e nella manutenzione.
In qualche cifra, il gruppo DEF rappresenta 29 imprese, 1100 collaboratori, 3 fabbriche in Francia, 1 in Italia e 1 in Cina, 50 ingegneri e tecnici in ricerca e sviluppo e 134 milioni di euro di fatturato nel 2009, frutto di una sapiente strutturazione del gruppo. Delta Erre Safe si inserisce a pieno titolo in questo contesto forte delle certificazioni ISO9001:2008 e NF (quest’ultima indispensabile per operare sul mercato transalpino).

Delta Erre può vantare al suo interno, una rete commerciale presente attivamente su tutto il territorio nazionale, un’efficiente divisione produttiva, una qualificata divisione Ricerca & Sviluppo, una divisione per la qualità e una divisione amministrativa. La sede amministrativa e produttiva di Delta Erre Safe si trova a S. Giovanni Lupatoto (VR).

2 UTILIZZO DEL TERMINALE TLC

2.1 Accensione/ Spegnimento

Per accendere il TLC bisogna premere il tasto * (ENTER) in basso a sinistra sulla tastiera. Al momento dell'accensione sul display verrà visualizzata la versione firmware del prodotto e il livello di carica della batteria (o la presenza del caricabatterie se inserito). Per spegnere il TLC, premere per 2 secondi il tasto ENTER. In mancanza di utilizzo o di comunicazione per più di 2 minuti, il TLC si spegne automaticamente per risparmiare le sue batterie.
Tra accensione e spegnimento (o viceversa), il tasto ENTER deve restare inattivo per almeno 3 secondi. Se, allo spegnimento, il carica batterie è collegato con il terminale, il display visualizza lo stato della ricarica delle batterie.

2.2 Menù TLC

Al momento dell'accensione il TLC comincia ad eseguire dei tentativi di comunicazione con i vari sensori. Se un dispositivo esterno è già collegato al terminale, il TLC riconosce automaticamente il rivelatore e sceglie di conseguenza il protocollo e l'interfaccia da utilizzare e le funzionalità da gestire. Se invece nessun dispositivo risponde alle interrogazioni del terminale, è possibile accedere ad un menù autonomo del TLC.

Questo menù permette di:
- Selezionare il tipo di dispositivo con cui comunicare;
- Selezionare la lingua;
- Visualizzare il livello di carica delle batterie;
- Visualizzare il lotto di produzione del TLC.

Menu Principale	Rivelatore lineare di fumo	Tentativo di comunicazione con il rivelatore lineare di fumo DLFB
	Rivelatore gas	Tentativo di comunicazione con il rivelatore di gas GD1xx
	Lingua	Francese Inglese Italiano Tedesco Olandese
	Livello batterie	Indica il livello di carica delle batterie espresso in %
	Lotto di produzione	Indica il lotto di produzione del TLC

2.3 Gestione delle Batterie

Per ricaricare le batterie, collegare il caricabatterie al TLC attraverso il connettore posto sul lato destro. Per delle batterie con capacità da 2200mAh, il tempo di carica è di 15 ore. Questo tempo è comunque in funzione della capacità delle batterie e della loro carica residua, ed è da considerarsi conclusa quando l'indicatore sul display si ferma. In ogni caso, si consiglia di non ricaricare le batterie oltre le 24h dato che una carica troppo lunga potrebbe danneggiare le batterie.

Il TLC può essere alimentato attraverso:
- Le sole batterie;
- Le batterie mentre il caricabatterie è connesso;
- Il solo caricabatterie (batterie assenti).

Se il TLC è alimentato dalle batterie, all'accensione viene visualizzato il livello di carica delle batterie. Se invece il caricabatterie è connesso al TLC, questo collegamento viene segnalato sul display all'accensione.

Attenzione
L'utilizzo di pile non ricaricabili è vietato con il TLC. L'utilizzo di pile non ricaricabili presenta, qualora l'apparecchio sia connesso alla rete elettrico attraverso il caricabatterie, un rischio di esplosione.

2.4 Modalità di configurazione

Se all'accensione si tiene premuto il testo CE (Esc) insieme al tasto * (Enter), è possibile accedere direttamente ad alcuni parametri di configurazione del TLC. Questa particolare modalità è protetta da una password di 4 numeri per impedire che personale non autorizzato possa impostare questi parametri.

I parametri di configurazione che è possibile impostare sono i seguenti:

INDIRIZZO PARAMETRO	SIGNIFICATO E POSSIBILI VALORI
00	Selezione automatica delle periferiche. Questo parametro può essere impostato ai valori: - **01** : il terminale non esegue la selezione automatica dei sensori collegati ma comincia immediatamente a comunicare con il rivelatore di fumo (DLFB); - **02** : il terminale non esegue la selezione automatica dei sensori collegati ma comincia immediatamente a comunicare con il rivelatore di gas (GD1xx); - **Altri valori** : con tutti gli altri valori il terminale esegue normalmente la selezione automatica dei sensori collegati.

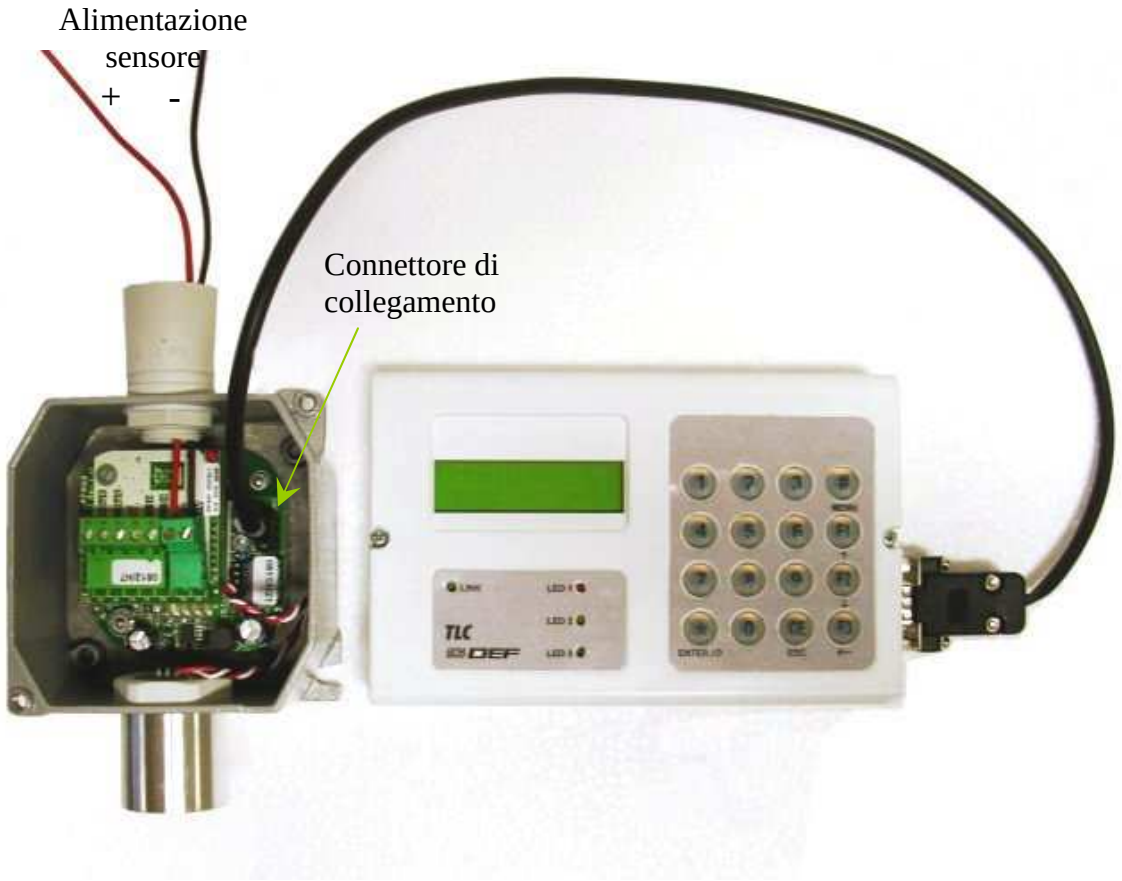
01	Visualizzazione della misura del sensore di gas. Questo parametro può essere impostato ai valori: - **01** : il terminale visualizza il valore della misura in LIE, ppm, percentuale o count a seconda del tipo di sensore di gas collegato. Il valore di questa misura è da considerarsi indicativo, e non come un valore preciso. - **Altri valori** : con tutti gli altri valori il terminale visualizza una stima del valore reale tramite una barra di stato. Inoltre visualizza anche le soglie di pre allarme e allarme per permettere un confronto con l'attuale valore.

Oltre ai parametri sopra indicati, ci sono delle opzioni riservate che l'utilizzatore non può modificare. Tutti questi valori sono scritti nella memoria E^2PROM del terminale.
Dato che le operazioni di lettura/scrittura sono molto dispendiose dal punto di vista del tempo di esecuzione, le opzioni sono lette una sola volta all'accensione e una loro copia viene salvata in RAM, in questo modo è possibile utilizzarle senza dover comunicare con la memoria ad ogni singola operazione.

3 GESTIONE RIVELATORI DI GAS GD1xx

I rivelatori di gas GD1xx prodotti da Delta Erre Safe dispongono di una porta seriale dedicata tramite la quale è possibile il collegamento del dispositivo TLC per la gestione del sensore. Il sensore viene collegato al terminale attraverso l'apposito cavo fornito in dotazione.
Collegando il TLC al rivelatore di gas, esso riceverà alimentazione dallo stesso. Non è quindi indispensabile in questo caso la presenza delle batterie e, qualora presenti non verranno scaricate.

Nota: il rivelatore di gas deve essere sempre alimentato in modo autonomo.



Il collegamento del rivelatore provoca l'accensione automatica del terminale e, viceversa, scollegando il sensore avviene lo spegnimento. Una volta effettuati i collegamenti si avrà:

- Sul display compare la tipologia e la versione SW del sensore di gas connesso;
- Il LED verde «LINK» si accende fisso;

Ricordare di attendere almeno 1 secondo tra 2 successive pressioni dei tasti, in quanto pressioni troppo ravvicinate possono compromettere la buona comunicazione tra il sensore di gas e TLC. In questo caso sul display comparirà "ERRORE DI COMUNICAZIONE".

È possibile premendo il tasto # (Menù) accedere al menù di configurazione per la gestione del rivelatore di gas. Questo menù permette di scegliere il tipo di operazione che si desidera eseguire sul rivelatore di gas connesso.

Menù	Voce principale	Operazione eseguita o selezione successiva
Gestione sensore di gas	Stato del sensore	Visualizza lo stato attuale del sensore
	Misura	Visualizza l'attuale concentrazione di gas misurato
	Test	Allarme
		Preallarme
	Preallarme	Visualizza la soglia
		Modifica soglia
	Allarme	Visualizza la soglia
		Modifica soglia
	Filtro Preallarme	Visualizza la soglia
		Modifica soglia
	Filtro Allarme	Visualizza la soglia
		Modifica soglia
	Ritardo Iniziale	Visualizza il ritardo iniziale (non modificabile)

3.1 Stato del sensore

Questa funzione visualizza l'attuale stato del sensore (Riposo, Allarme, Preallarme e Guasto). Questi diversi stati sono inoltre segnalati dai led presenti sul terminale:

Riposo (led verde) → significa che la misura del livello di gas rivelata dal sensore è più bassa della soglia di preallarme impostata;

Preallarme (led rosso lampeggiante) → significa che la misura del livello di gas è più alta della soglia di preallarme impostata ma più bassa della soglia di allarme prevista;

Allarme (led rosso) → significa che la misura del livello di gas rivelata dal sensore è più alta della soglia di allarme impostata;

Guasto (led giallo) → il terminale segnala un guasto del sensore o un comportamento non corretto.

3.2 Misura

Questa funzione permette di visualizzare il livello attuale di concentrazione del gas rivelato dal sensore. La misura non viene mostrata con il valore effettivo in L.I.E. o ppm, ma graficamente in rapporto ai livelli di allarme (A) e preallarme (P) impostati sul sensore. La misura e la visualizzazione vengono aggiornate ogni 10 secondi. Il valore visualizzato e' un'approssimazione del valore effettivo misurato.

Nota: la taratura delle soglie è in ogni caso effettuata in modo preciso in fabbrica per mezzo di campioni di gas a percentuale predeterminata.

3.3 Test

Questa funzione permette di eseguire il test delle uscite di preallarme e allarme. Per effettuare il test selezionare questa fase dal Menù principale e scegliere l'uscita di allarme o preallarme di cui si desidera effettuare la prova. Il test delle uscite modifica lo stato effettivo del sensore e dura circa 30 secondi.
Nota: L'attivazione delle uscite di preallarme/allarme in test può avvenire con un ritardo di qualche secondo in funzione delle fasi d'elaborazione interne attive su sensore in quel momento.

3.4 Preallarme

Questa funzione permette di visualizzare/impostare la soglia di preallarme (soglia in L.I.E. o ppm secondo il tipo di sensore). La soglia può essere modificata dopo la visualizzazione premendo il tasto Enter, inserendo il nuovo valore e confermando l'impostazione nuovamente con il tasto Enter. Per verifica dopo alcuni istanti il display visualizza il nuovo valore inserito. È possibile terminare l'operazione e ritornare al Menù principale premendo i tasti # (Menù) o F3 (← Backspace).

3.5 Allarme

Questa funzione permette di visualizzare/impostare la soglia d'allarme. La soglia può essere modificata operando come indicato per la soglia di preallarme.
Nota: I valori in L.I.E. o ppm sono approssimati ed e quindi consigliabile, non modificare le impostazioni di fabbrica che vengono effettuate in modo più preciso, per ogni singolo sensore, con percentuali di gas campione.

3.6 Filtro preallarme

Questa funzione permette di impostare il filtro di ritardo attivazione del preallarme. Impostare un valore compreso tra 1 e 9 per ottenere un filtraggio breve (il sensore risponde in modo rapido al preallarme) oppure impostare dei valori multipli di 10 per ottenere un filtraggio corrispondente in secondi (es. impostando 20 si ottiene un ritardo di risposta di circa 20 secondi). Il valore Max impostabile e' 240; l'impostazione di fabbrica e' 15.

3.7 Filtro allarme

Questa funzione permette di impostare il filtro di ritardo attivazione dell'allarme. Impostare un valore compreso tra 1 e 9 per ottenere un filtraggio breve. Impostare dei valori multipli di 10 per ottenere un filtraggio corrispondente in secondi. Il valore Max impostabile e' 240; l'impostazione di fabbrica e' 15.
Nota: E' consigliabile, per evitare ritardi di risposta eccessivi, impostare tempi di filtraggio non superiori a 60.

3.8 Ritardo iniziale

Questa funzione permette di visualizzare il ritardo di attivazione del sensore dopo che lo stesso e' stato alimentato e serve per consentire alla capsula di raggiungere le condizioni di funzionamento di regime. Il ritardo e' impostato in fabbrica in base alla tipologia di gas rivelato. Questo valore non è modificabile.

3.9 Impostazione Celle di memoria

L'utilizzatore può leggere o scrivere (quando possibile) le celle di memoria che caratterizzano il rivelatore di gas tramite il dispositivo TLC.

Questa modalità di gestione è stata creata solo a scopi di debug ad alto livello o di manutenzione del sensore in fabbrica. Normalmente l'utilizzatore finale non ha accesso a questa sessione in quanto i dati modificabili sono sensibili e possono compromettere il funzionamento del sensore stesso.

Per utilizzare questa funzione di configurazione del sensore di gas bisogna accedere ad un particolare menù di configurazione che, per motivi di riservatezza, non viene qui riportato. In questa sezione l'utilizzatore può vedere il valore di tutta la memoria del sensore di gas dall'indirizzo 0 al 255 e può modificare i parametri contenuti nelle celle dalla 0 alla 99 inserendo un valore byte (0-255).

Le celle di memoria, in base all'indirizzo, sono organizzate in tre categorie:
000-099 = celle eeprom/flash riscrivibili
100-239 = celle ram (1-149)
240-255 = celle dati firmware dispositivo (flash non riscrivibile)

Celle dati firmware:

IND. CELLA	SIGNIFICATO
240 - 243	Stringa tipo sensore speciale: è significativa se tipo dispositivo=$50 (cella 250) e se estensione tipo è diversa da quelle predefinite
250	Tipo dispositivo. I tipi previsti sono: $10H = sensore di metano $30H = sensore di CO $40H = sensore di O2 $50H = nuovo tipo sensore. Il tipo effettivo è definito in un'altra cella dedicata in flash
251	Versione
252	Estensione versione
253	Spare

4 GESTIONE RIVELATORI DI FUMO

Al momento della realizzazione del terminale, l'azienda produttrice dei rivelatori DLFB non aveva ancora portato a termine lo sviluppo del sensore. Si è reso quindi necessario creare un tool di simulazione per l'implementazione che rispecchiasse le caratteristiche, i comportamenti e le temporizzazioni fornite nelle specifiche di progetto.

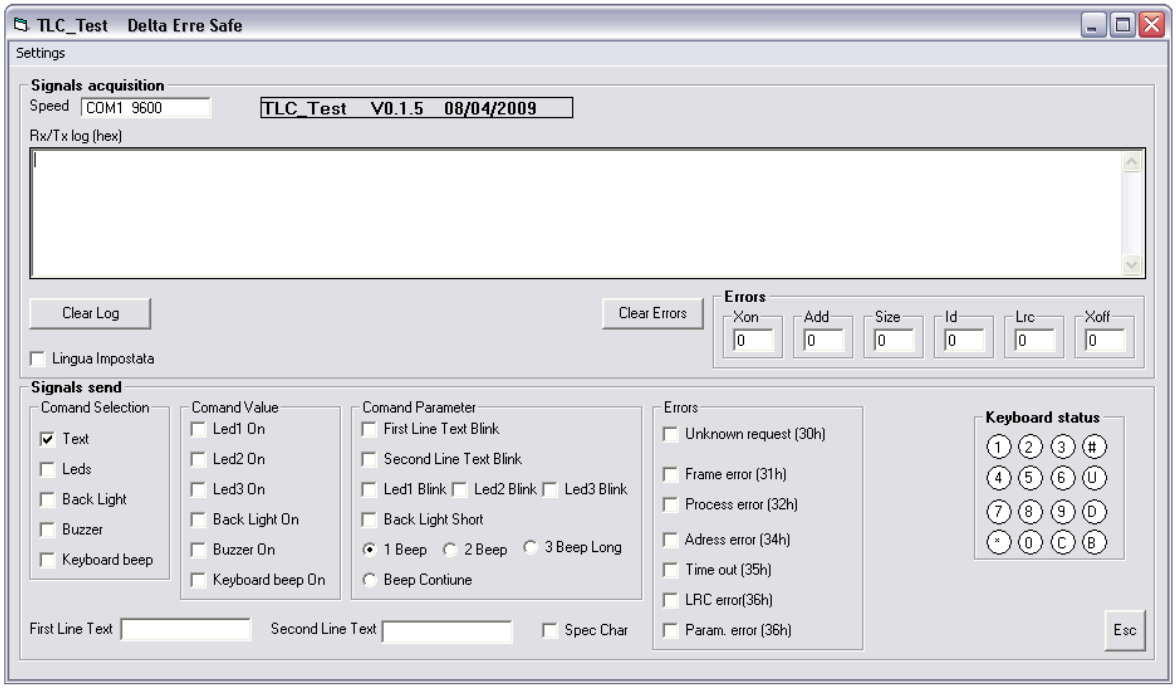
Quando il terminale TLC viene collegato ad un rivelatore di tipo DLFB, nonostante rimanga il master che gestisce la comunicazione, esso si comporta in modo totalmente passivo. Infatti, il TLC fornisce costantemente lo stato della tastiera e cede al DLFB il totale comando:

- dei 3 led (non quello di link);
- del display (compresa la retroilluminazione);
- del buzzer (compreso il suono di conferma quando viene premuto un tasto).

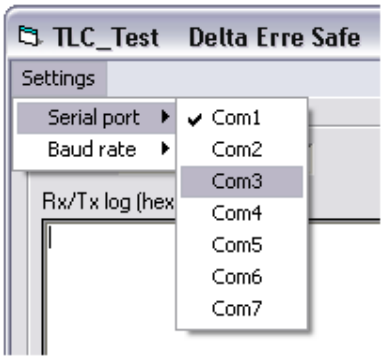
Per maggiori dettagli sulla gestione di questa periferica vedere capitolo 9.

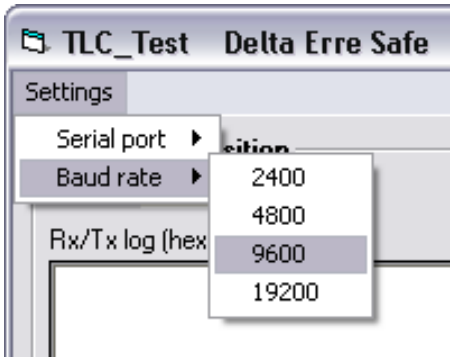
4.1 Tool di simulazione

Attraverso il programma realizzato è possibile simulare il comportamento del DLFB.

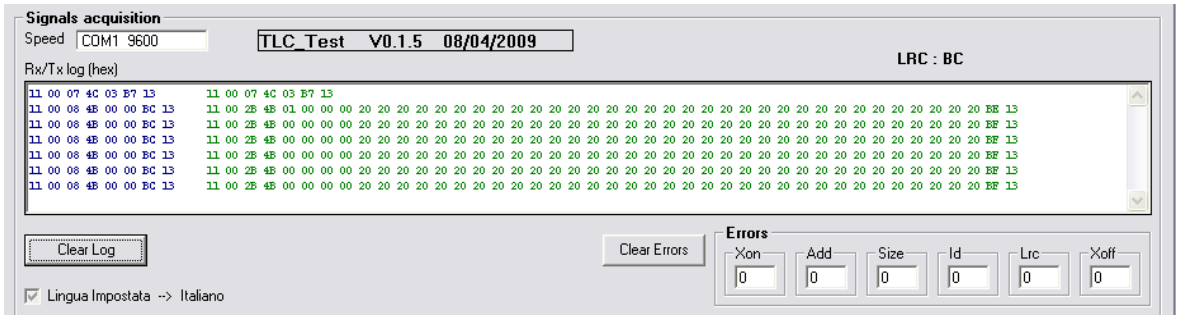


Prima di cominciare la simulazione è necessario impostare la porta seriale sulla quale effettuare la comunicazione e il Baud Rate relativo.



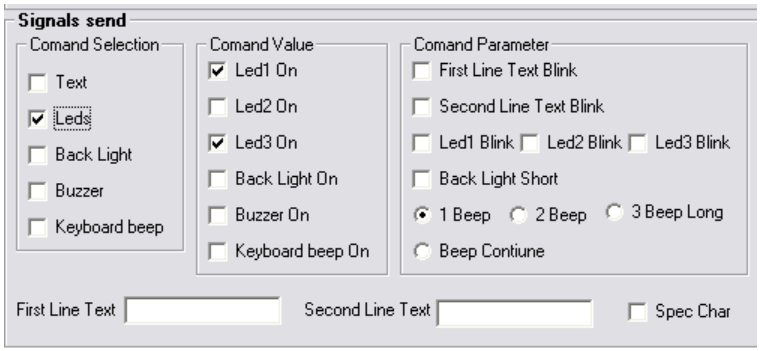


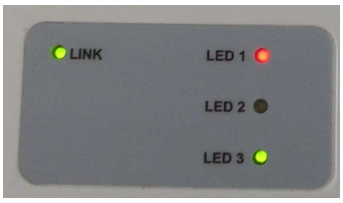
Se a questo punto si collega il TLC al PC, la comunicazione parte automaticamente una volta che il terminale (master) comincia ad inviare i messaggi di polling.



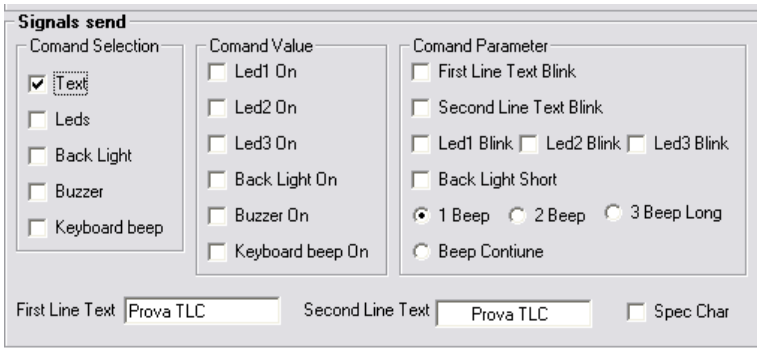
Se la comunicazione avviene in modo corretto, il led verde di Link sul TLC rimane acceso durante tutta la durata della connessione. Tramite questo tool è possibile, come con il dispositivo DLFB:

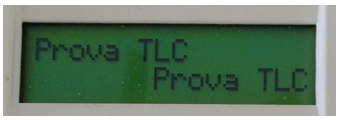
- Comandare l'accensione, lo spegnimento e il lampeggio dei led



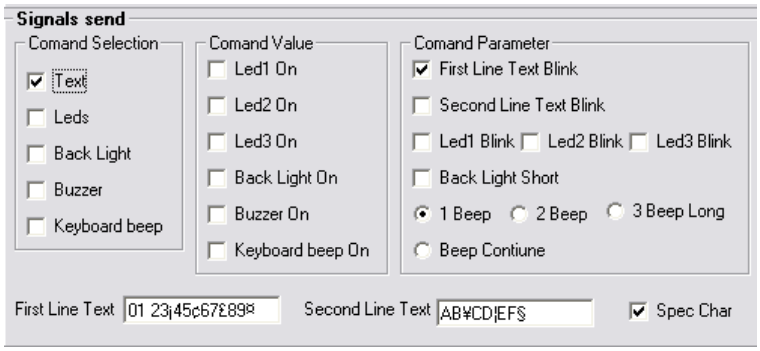


- Visualizzare fino a 32 caratteri sul display

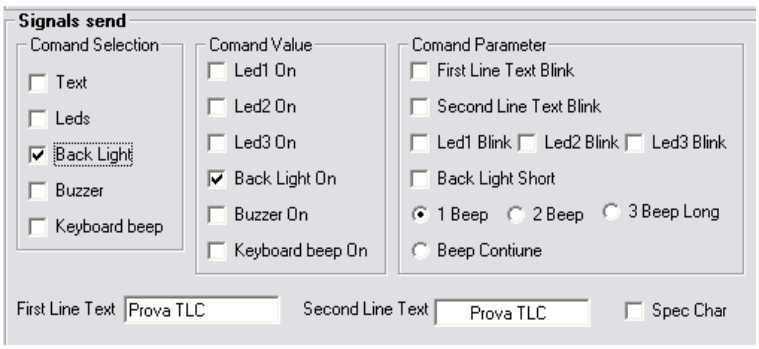




- Visualizzare dei caratteri speciali non compresi nel set del display
- Far lampeggiare le due linee di testo insieme o separatamente



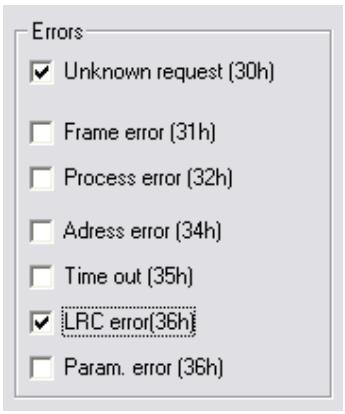
- Spegnere o accendere la retroilluminazione (fissa o per 20 s)



- Accendere, spegnere, far suonare con 1, 2 o 3 beep il cicalino
- Attivare o tacitare il beep dei tasti quando vengono premuti.

Inoltre ad ogni messaggio di interrogazione viene mandato lo stato dei tasto ed è quindi possibile visualizzare se uno o più tasti vengono premuti.

Inoltre è possibile anche simulare il caso di un errore nella comunicazione, selezionando anche il tipo specifico di anomalia riscontrata. In questo caso il led verde di link sul terminale TLC si spegne.



5 COLLAUDO DEL TERMINALE TLC

Prima di eseguire il collaudo del terminale, è necessario:

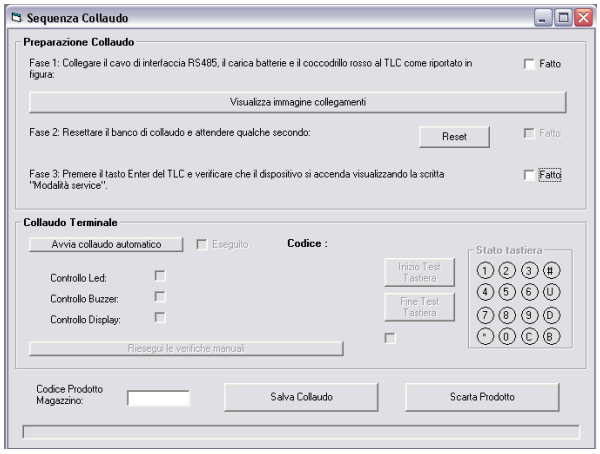
- Verificare che i componenti polarizzati della scheda siano installati correttamente sul circuito stampato;
- Eseguire la programmazione della scheda, trasferendovi il firmware;
- Assiemare tutti i componenti del prodotto finito (scheda stampata, display, tastiera, porta batterie, contenitore).

Una volta eseguite tutte queste fasi preliminari si può cominciare il collaudo vero e proprio del TLC.

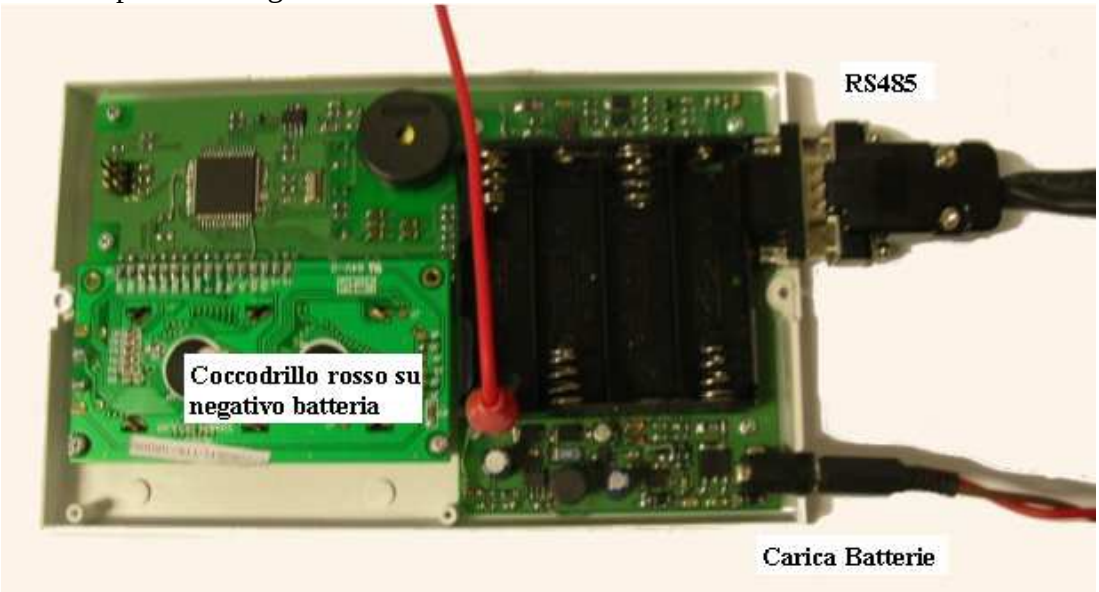
5.1 Fase 1: Preparazione al collaudo

- Lanciare il programma di collaudo TestTLC.exe e impostare la comunicazione con un Baud Rate pari a 9600 baud e la porta COM corrispondente.

- Inserire Nome, Cognome, Ordine di Produzione e Numero del Lotto e selezionare “Sezione Collaudo”.

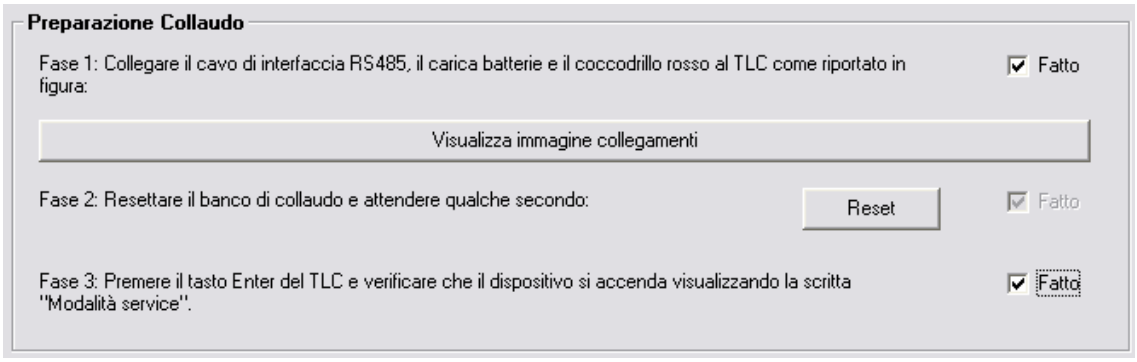


- Collegare il cavo di interfaccia RS485, il carica batterie e il coccodrillo rosso al TLC come riportato in figura:

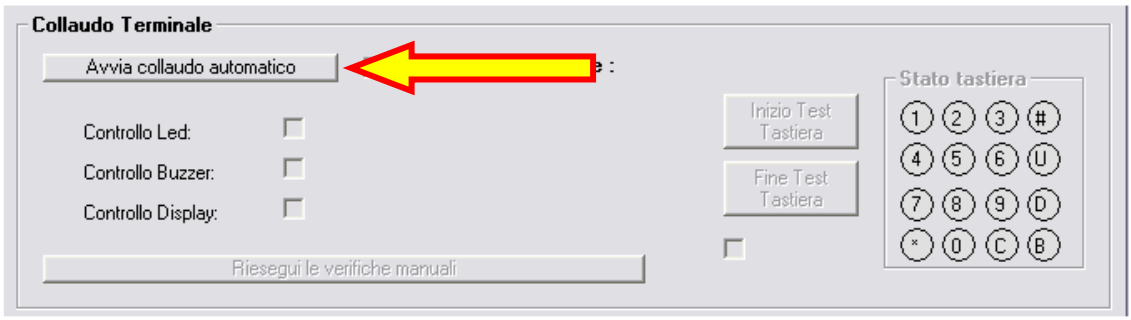


- Resettare il banco di collaudo tramite l'apposito pulsante sull'applicazione.

- Premere il tasto Enter del TLC e verificare che il dispositivo si accenda visualizzando la scritta "Modalità Service".

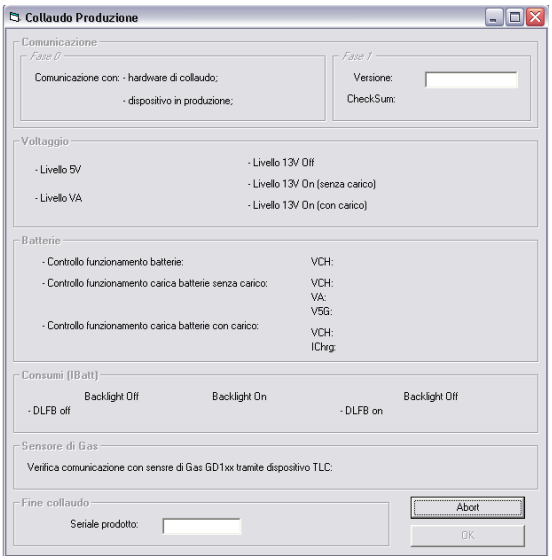


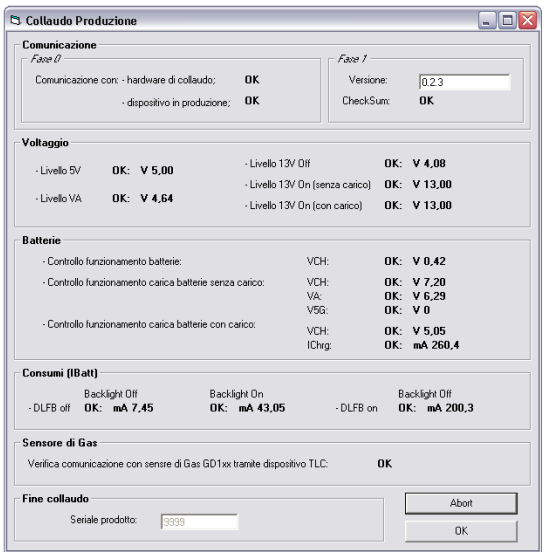
- Avviare le fasi automatiche di verifica.



5.2 Fase 2: Collaudo Automatico

- Attendere che vengano eseguite tutte le fasi automatiche:
  - Verifica della comunicazione con il dispositivo di collaudo e con il terminale;
  - Controllo versione del firmware installata sul TLC e verifica checksum;
  - Controllo che le principali tensioni di riferimento siano dentro i limiti attesi:
    - Tensione 5V;
    - Tensione VA;
    - Tensione 13V;
  - Verifica funzionamento delle batterie;
  - Verifica funzionamento del carica batterie;
  - Controllo che i consumi del dispositivo non superino quelli attesi;
  - Verifica comunicazione con il sensore di gas GD1xx tramite il terminale TLC;
  - Assegnazione del numero seriale;



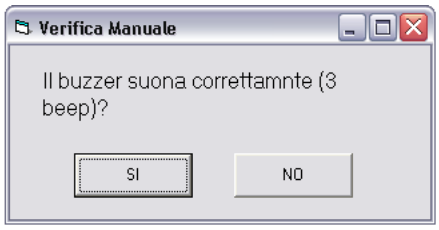


- Al termine delle operazioni premere “Ok” per passare ai controlli manuali se tutti i controlli hanno avuto esito positivo, altrimenti premere “Abort”.

5.3 Fase 3: Collaudi Manuali

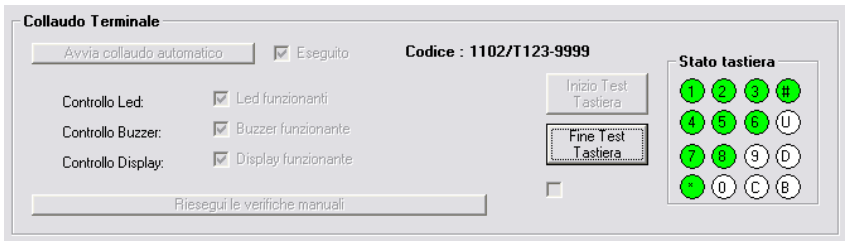
- Verificare visivamente il comportamento in modalità Blink di tutti e 4 i led (Link, Red, Yellow e Green).

- Controllare il comportamento del Buzzer verificando che esegua 3 beep veloci.



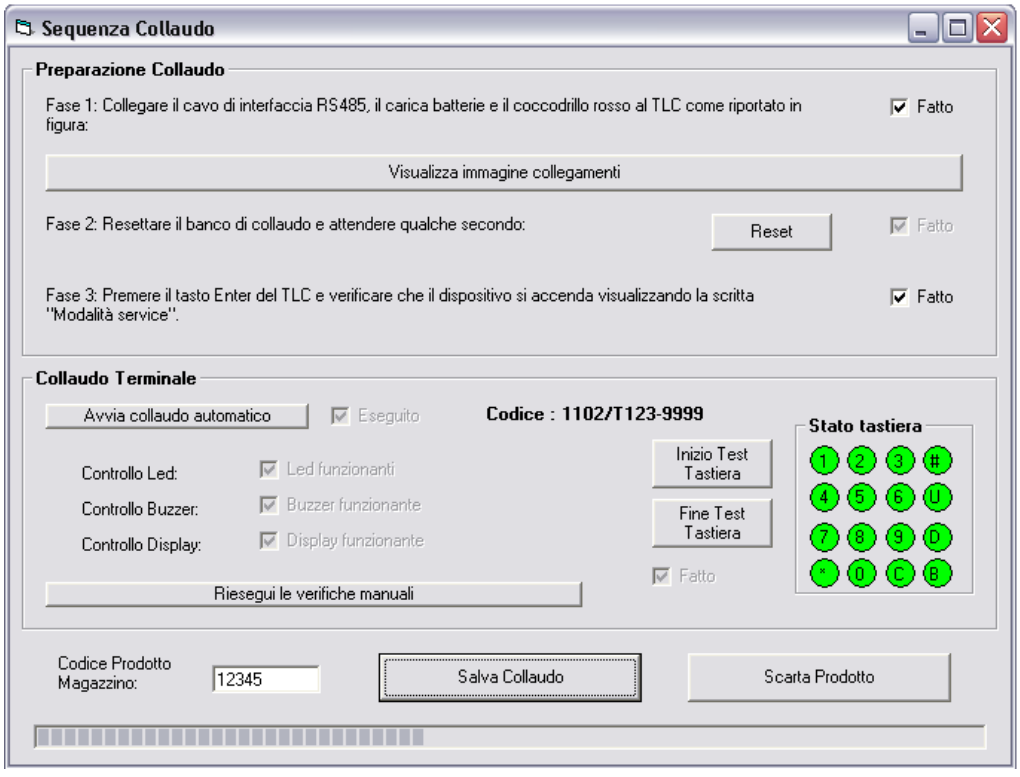
- Verificare che venga stampato sul Display il massaggio " PROVA DISPLAY PROVA DISP BLINK", in cui la seconda linea in modalità Blink.

- Comandare l'inizio del test della tastiera e premere in sequenza i tasti sul TLC. Una volta provati tutti i tasti selezionare "Fine Test Tastiera".



5.4 Fase 4: Registrazione del collaudo

- Selezionare "Salva Collaudo" o "Stampa e Salva Collaudo".



- Controllare che il TLC sia uscito dalla modalità Service (visualizzazione TLC + versione firmware).

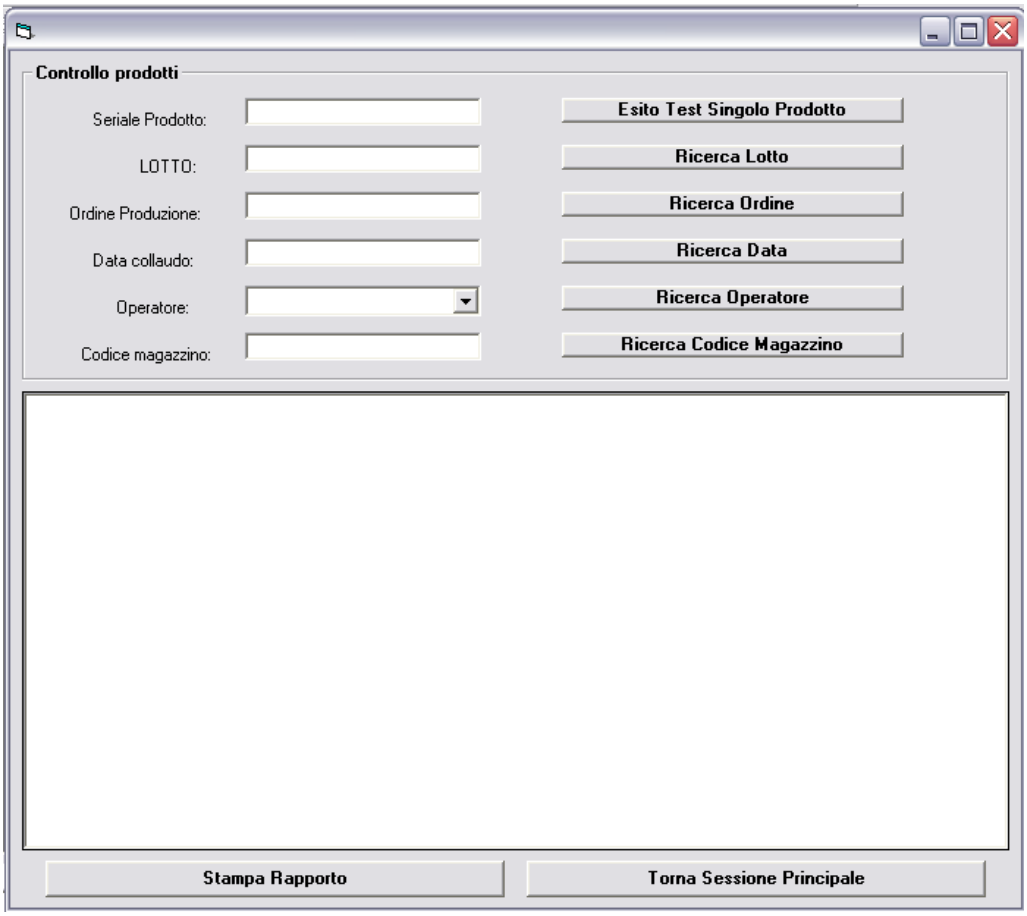
- Quando viene richiesto spegnere il dispositivo tenendo premuto il tasto Enter e segnalare l'avvenuto spegnimento (OK).

- Verificare che il salvataggio dei dati sia andato a buon fine (non compare nessuna segnalazione d'errore).

6 ORGANIZZAZZIONE DATABASE

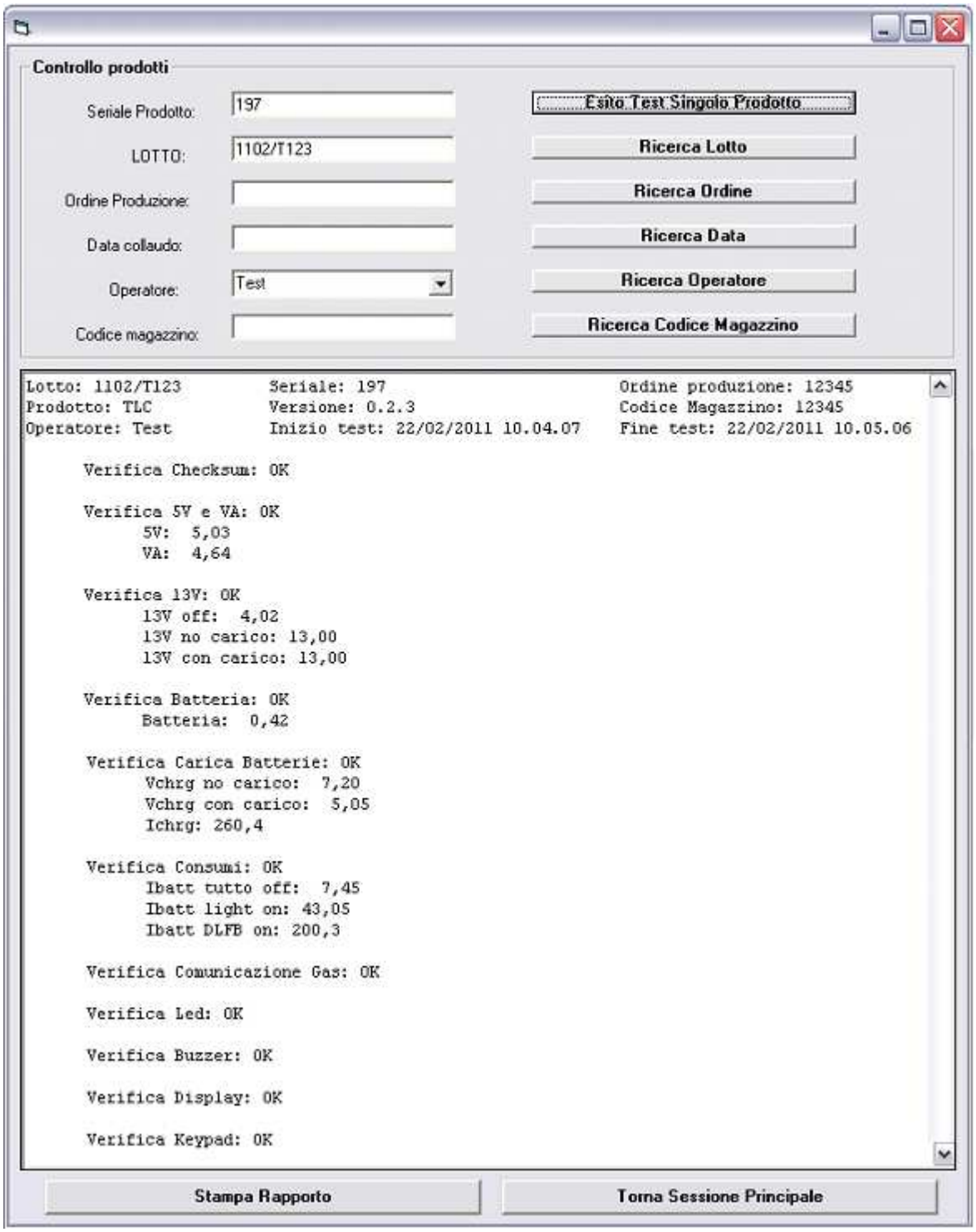
Accendendo alla sezione “Visualizza rapporti” è possibile visualizzare i dati relativi ai collaudi di ogni singolo terminale TLC.

La ricerca dei dati da visualizzare può avvenire attraverso vari criteri di ricerca: Codice Magazzino del prodotto, Operatore che ha eseguito il test, Data del collaudo, Ordine di produzione, Lotto e Seriale del prodotto.



In generale vengono visualizzati tutti i dati generali relativi all’opzione di ricerca selezionata (tutti i collaudi eseguiti da un operatore, tutti quelli fatti in una determinata data

o appartenenti ad un particolare lotto di produzione) e l'esito positivo o negativo dei collaudi. Se invece si inserisce il lotto e il numero seriale del terminale, è possibile visualizzare a tutti i dettagli del collaudo avvenuto.



7 SVILUPPO DELLE FUNZIONALITÀ BASE TLC

Il firmware di controllo è stato organizzato in più moduli sorgenti per la gestione delle funzioni del sistema, divisi in file.h e file.c, eseguiti ciclicamente all'interno di un loop infinito.

Per lo sviluppo si è partiti con la gestione delle funzionalità di base del microcontrollore (inizializzazione del sistema, gestione dei timer, degli interrupt, dei reset, del convertitore analogico/digitale interno, etc), aggiungendo la gestione dei vari componenti del TLC (led, display, buzzer, tastiera, etc) ed infine integrando le parti di comunicazione e gestione dei sensori interfacciati al dispositivo.

7.1 Modulo Main

Le funzioni vengono eseguite con temporizzazioni differenti a seconda delle necessità:

- Ogni 6,7ms:
  - gestione del clock;
  - gestione della comunicazione con il sensore di gas quando richiesta;
  - lettura e conversione dei valori degli ingressi analogici;
  - refresh Watchdog software;
  - gestione stato di wait;

- Ogni 13ms:
  - Gestione della tastiera;
  - Gestione dei timer;
  - Gestione del display;

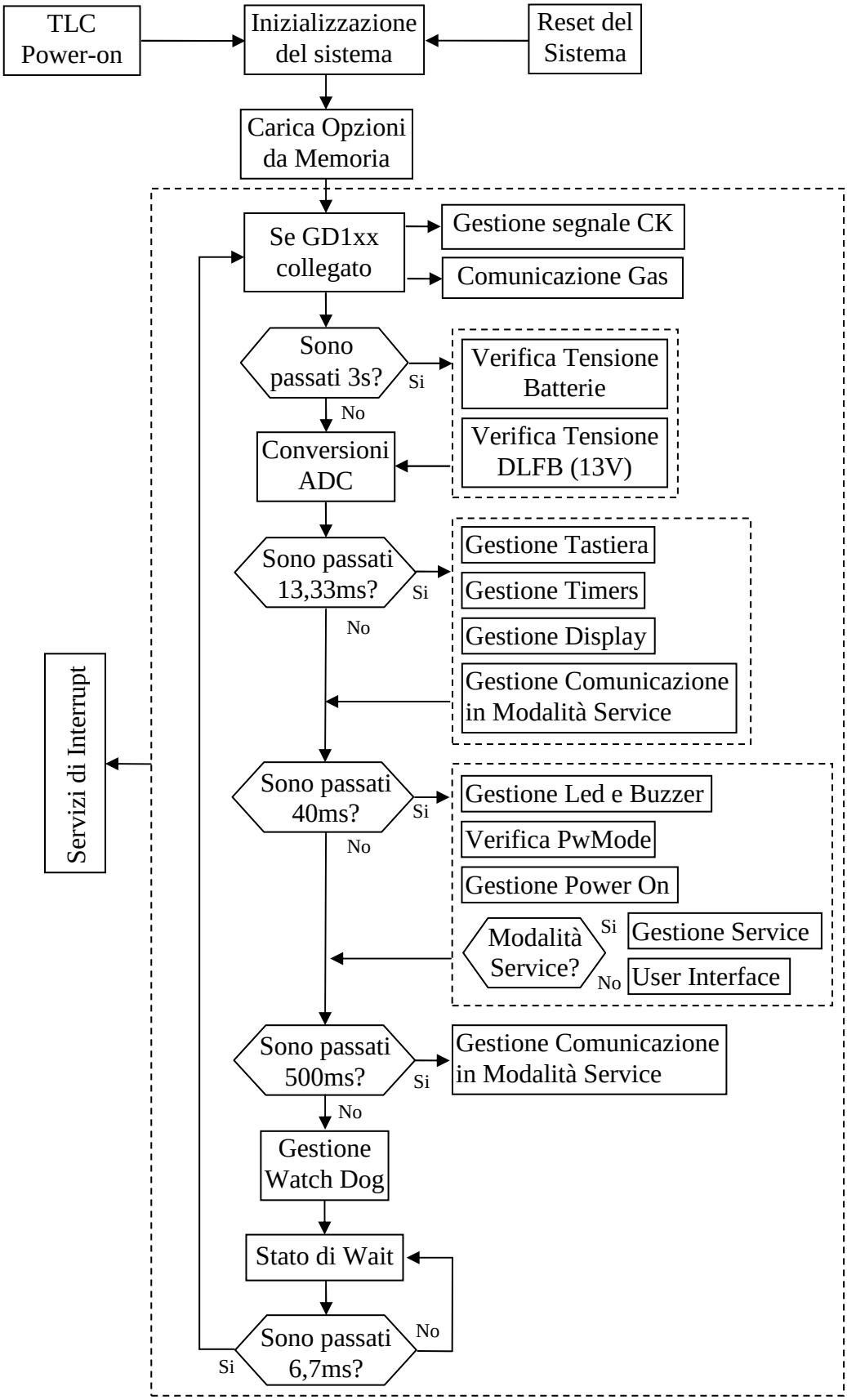
- Ogni 40ms:
  - Gestione dei led e del buzzer;
  - Analisi dello stato di funzionamento;
  - Gestione dell'accensione e dello spegnimento del dispositivo;

- Ogni 150ms/250ms/350ms/500ms:
  - Gestione del DLFB a seconda del modo di funzionamento;

- Ogni 3s:
  - verifica della tensione di batteria;
  - verifica della tensione di alimentazione fornita al DLFB;

Vengono eseguite al di fuori del ciclo le operazioni di:
- inizializzazione del sistema;
- caricamento dei parametri del sistema memorizzati nella EEPROM;
- servizi di interrupt;
- gestione dei reset;

Schema a blocchi



```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Main
*
*    \detail  Main management of all software components
*             Works as superloop cycle with a period of 6,7ms
*/
////////////////////////////////////////////////////////////////////////////////
void main(void) {

  Reset_init();                 // Initialise hardware and variables
  Load_Options();          // Load options array from EEPROM

  for (;;){
    //done every 6,7ms
    presc13++;

    //manage clock F0 every 6,7ms
    if((!stopClkFlg)||(PTFD_PTFD0)){
      if(clkAct) PTFD_PTFD0=(PTFD_PTFD0 == 0 ? 1 : 0); //~PTFD_PTFD0;
      else  PTFD_PTFD0=1;
    } else  stopClkFlg=FALSE;

    //manage gas detector communication if selected
    if(flag_trx_gd1xx!=0 && tim26m.t.timGd1xx==0) TRx_Manag();

    // Do a check of battery voltage and DLFB voltage every 3s
    if(!tim02.t.timCheck){
      tim02.t.timCheck=15;  //15 * 0.2sec = 3 sec
      if(!Check()){
        if(prec_check==check_err){
          if (!tim02.t.timDlyErrCheck){
            tim02.t.timDlyErrCheck=250;  //next error message show after 50s
            fase_check=device;
            device=ERR_CHECK;
            prec_check=check_err;
            viser_phase=0;
          }
        } else{
            tim02.t.timDlyErrCheck=250;  //next error message show after 50s
            fase_check=device;
            device=ERR_CHECK;
            prec_check=check_err;
            viser_phase=0;
        }
      } else  tim02.t.timDlyErrCheck=0;   // keep showing delay to 0 if no any error
    }
    ADConversion();

    if(presc13%2==0){
        //done every 13,33ms
        blink++;

        Kybrd();
        Timers();
        Display();
        if(opzioni[ADDSERVICE]==ADDSERVICE)  SCI1_TX();

        //done every 40ms
        if(TickCounter==3){
            TickCounter=0;
            Ledbuzzer();
```

```
            PwMode();
            PwOn_Manag();
            if(opzioni[ADDSERVICE]!=ADDSERVICE){
              //service mode is active
              ServiceMode();
            } else{
              //User Interface
              UsInt();
            }
        }
        TickCounter++;
      }

    //DLFB communication management, done every 500ms if DLFB is selected
    if((device==DLFB)&&(fase!=ON_OFF)){
      if(!flag_DLFB && !cycDLFB){
        flag_DLFB=TRUE;
        if(periferica==DLFB_TYPE)  cycDLFB=75;   // load counter for 500ms cycle
        else{
          if(impostaTim==0) cycDLFB=75;   // load counter for 500ms cycle
          else{
            if(impostaTim==1) cycDLFB=22;   // load counter for 150ms cycle
            else{
              if(impostaTim==2) cycDLFB=37;   // load counter for 250ms cycle
              else  cycDLFB=52;   // load counter for 350ms cycle
            }
          }
        }
      }
      if(flag_DLFB) Manag_DLFB();
    }
    if(cycDLFB) cycDLFB--;

    __RESET_WATCHDOG();     // feeds the watch dog
    while (TickElapsed == 0){
      EnableInterrupts;     // enable interrupts
      PTDD_PTDD4 = 0;       // CPU is free: on output (for debug purpose)
      wait();
      PTDD_PTDD4 = 1;       // CPU is running: off output (for debug purpose)
    }
    TickElapsed--;
  }
}
```

7.2 Gestione ADC

La lettura dei valori degli ingressi analogici avviene prima attraverso un servizio dedicato di interrupt, che provvede a salvare il valore istantaneo del canale di ingresso in una variabile accumulatore, e poi grazie ad una funzione che, ogni 4 letture istantanee, calcola le medie dei valori.

Le variabili utilizzate per i controlli in tutto il firmware fanno riferimento ai valori medi, in modo da filtrare eventuali disturbi momentanei.

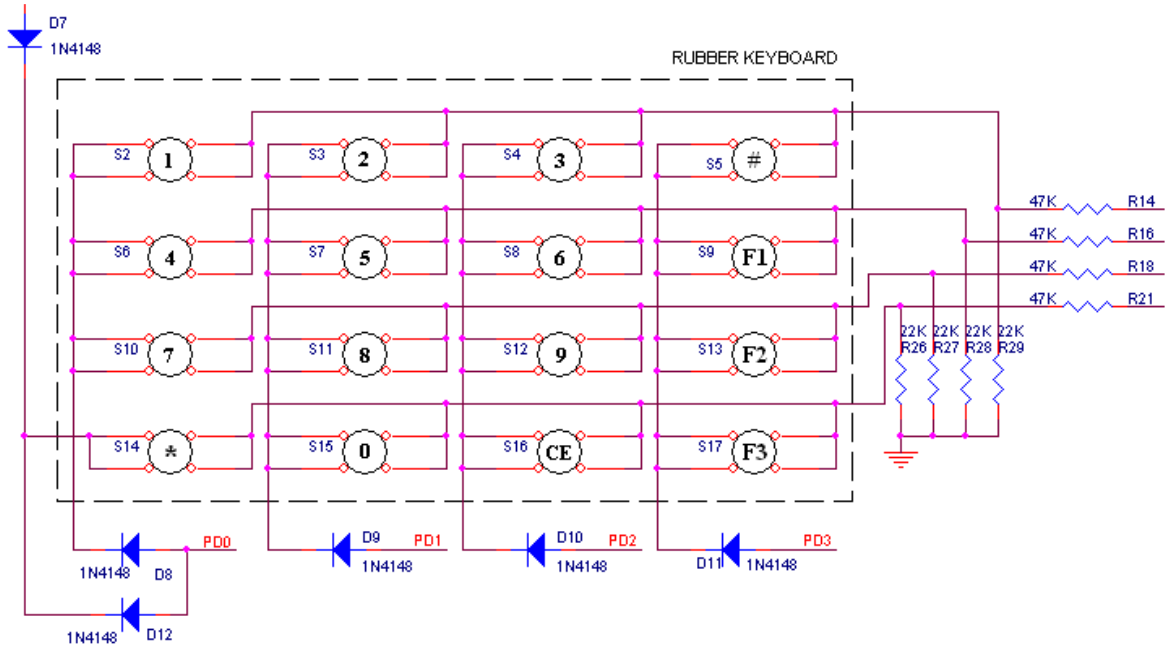
```
//--------------------------- Analog input variables ---------------------------------
struct vADC{
  byte vAnalog[CHANN];                  // istantaneous analog value
  word vAnalogAccu[CHANN];    // values accumulator for average calculation
  byte vAnalogMedia[CHANN];   // average analog value
};
extern volatile struct vADC analogV;
```

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Manage the A/D conversions by AD1
*
*    \detail  This function manages the A/D convertion of all analog channel and save
*             values in appropriate variables.
*/
////////////////////////////////////////////////////////////////////////////////
ISR(AD1_Interrupt){
  byte tmp_val;

  (void)ADC1SC1;
  //get last conversion value and reset interrupt
  tmp_val=ADC1RL;

  if (analogInit>=ADINIT_COUNT){
    if(opzioni[ADDALIM5V]==NO5VALIM){
      if (channel<CHANN){
        //  this special version doesn't analyze the V5G analog input
        if (channel==V5G) tmp_val=0;
        analogV.vAnalog[channel]=tmp_val;
      }
    } else{
        if (channel<CHANN) analogV.vAnalog[channel]=tmp_val;
      }

    channel++;
    if (channel>=CHANN){ // check if conversion sequence is ended
      channel=0;
      (void)ADC1SC1;
      ADC1SC1=0x1F;     // disable A/D module and disable A/D interrupt
      analogInit=0;
    } else
        ADC1SC1=(channel+CHANN_OFFS)|ADC1SC1_AIEN_MASK;   // start new A/D conversion

  }else{
    analogInit++;
    channel=0;
    ADC1SC1=(channel+CHANN_OFFS)|ADC1SC1_AIEN_MASK;   // start new A/D conversion
  }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief     Analogic to Digital conversion
*
*    \detail    This function calculate the average ov A/D conversion and start a
*               new A/D sequence .
*/
////////////////////////////////////////////////////////////////////////////////
void ADConversion(void){
  byte ii;

  if(ADC1SC1==0x1F){
    // calculate average (mean of N_CONV_AVG data) for each value
    for (ii=0; ii<CHANN; ii++)
      analogV.vAnalogAccu[ii]=(analogV.vAnalogAccu[ii]+(word)analogV.vAnalog[ii]);
    convCounter++;
    if (convCounter>=N_CONV_AVG){
      for (ii=0; ii<CHANN; ii++){
        analogV.vAnalogMedia[ii]=(byte)(analogV.vAnalogAccu[ii]/N_CONV_AVG);
        analogV.vAnalogAccu[ii]=0;
      }
```

```
      convCounter=0;
    }
    DisableInterrupts;
    (void)ADC1SC1;            // Reset interrupt request flag
    channel=0;
    analogInit=0;
    ADC1SC1=(channel)|ADC1SC1_AIEN_MASK;   // start new A/D conversion
    EnableInterrupts;

  } else{

      DisableInterrupts;
      (void)ADC1SC1;
      ADC1SC1=0x1F;
      EnableInterrupts;

    }
}
```

7.3 Gestione Tastiera

La tastiera è gestita come una matrice 4x4 in cui ogni tasto viene codificato e decodificato come indicato nella tabella riportata in seguito. Anche in questo caso la lettura avviene con un filtraggio, per rivelare le variazioni di stato tra la pressione e il rilascio del tasto.




```
//------------------ Keyboard management variables ----------------------------------
//keyboard status/debounce array structure define

struct keybStr{
    byte status;
    byte debounce;
};

extern volatile struct keybStr keyb_col[NUM_COL];
```

```
//--------------------Matrix keyboard decode table (keycodes)-------------------------
volatile byte const tabDecKeys[NUM_ROW][NUM_COL]={
    // Column 1 decode table
    1,     // 1
    4,     // 4
    7,     // 7
    14,    // ENTER (*)

    // Column 2 decode table
    2,     // 2
    5,     // 5
    8,     // 8
    10,    // 0

    // Column 3 decode table
    3,     // 3
    6,     // 6
    9,     // 9
    15,    // CE

    // Column 4 decode table
    11,    // MENU' (#)
    12,    // UP    (F1)
    13,    // DWN   (F2)
    16,    // BKSPC (F3)
};

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Keyboard management
*
*    \detail  Check if any key is pressed, does the debounce, saves key codes in the
*             keyb_buf, manage buffer read/write pointer and set beep flag
*/
////////////////////////////////////////////////////////////////////////////////
void Kybrd(void){
  byte ii;
  byte tmp_col;                 // temporary variable for keyboard management
  byte tmp_var;                 // temporary variable for keyboard management
  byte tmp_ix;                  // temporary variable for keyboard management
  byte tmp_keyCod;              // temporary variable for keyboard management

  // Matrix keys
  if (col_sbit == 0){
    col_sel = 0;
    col_sbit = SEL_COL1;
    for (ii = 0; ii < NUM_ROW; ii++){
      keyb_col[ii].status=0;
      keyb_col[ii].debounce=0;
    }
  } else {
      tmp_col = (PTBD & MASK_COLS);   // read row status for the column selected
      if (tmp_col != keyb_col[col_sel].status){
        // column status has changed
        if (tmp_col != keyb_col[col_sel].debounce){
          // it is the first change, prepare variable for debounce
          keyb_col[col_sel].debounce = tmp_col;
        } else {
            // debounce was done
            tmp_var = (keyb_col[col_sel].status ^ tmp_col);    // isolates bits changed
            tmp_var = tmp_var & tmp_col;  // isolates bits changed from 0 to 1
            if (tmp_var != 0){
              // analyse bits to find the new keys pressed
```

```
            bit_mask = 1;
            for (ii=0; ii < 4; ii++){
              if (bit_mask & tmp_var){
                // bit is on, get key code and put it in the buffer
                tmp_keyCod = tabDecKeys[col_sel][ii];    // get key code
                if ((tmp_keyCod > 0)&&(tmp_keyCod <= MAX_KEY_COD)){
                  // put key code in the buffer
                  keyb_buf[keyb_store_ix] = tmp_keyCod;
                  tmp_ix = keyb_store_ix;
                  tmp_ix++;
                  if(tmp_keyCod==K_ENTER && fase==ON_OFF){
                    Reset_init();
                    tmp_ix--;
                  }
                  if (tmp_ix >= MAX_BUF_KEY)  tmp_ix=0;
                  // check if buffer is not full
                  if (tmp_ix != keyb_read_ix)
                    keyb_store_ix = tmp_ix;     // increment store pointer
                  keyM_event=TRUE;    // signal to main key press event
                  if(enable_kbrd_beep==TRUE){
                    //beep
                    if(fase!=ON_OFF)  fdbk_kbrd_beep++;
                  }
                }
              }
              bit_mask = bit_mask << 1;
            }
          }
          // upgrade row status
          keyb_col[col_sel].status = keyb_col[col_sel].debounce;
          key_col_service[col_sel]|=keyb_col[col_sel].status;
        }
      } else {
        // column status has not changed
        keyb_col[col_sel].debounce = keyb_col[col_sel].status;  // debounce = status
        if(col_sel==0){
          if(keyb_col[col_sel].status&0b00001000){
            deviceOff++;
            if(deviceOff==TIM_OFF) SetDevice_OFF();
          } else  deviceOff=0;
        }
        if(col_sel==1){
          if(keyb_col[col_sel].status&0b00001000){
            modConfig1=TRUE;
          } else  modConfig1=FALSE;
        }
        if(col_sel==2){
          if(keyb_col[col_sel].status&0b00001000){
            if (modConfig1) modConfig2++;
            else modConfig2=0;
            if(modConfig2>=TIM_OFF){
              if(device==GD1XX){
                status_menu=CONFIG;
                countTest=0;
              }
            }
          } else  modConfig2=0;
        }
        // select next row if nothing has changed
        col_sel++;
        col_sbit = ((col_sbit << 1) & MASK_ROWS);
        if (col_sbit == 0){
          col_sel = 0;
```

```
            col_sbit = SEL_COL1;
          }
        }
    }
  // upgrade row-LCD column status
  DisableInterrupts;     // make sure to have interrupt disabled
  tmp_var = (PTDD & 0b11110000);   // clean row selection bits
  tmp_var = (tmp_var | col_sbit);  // port D signal status
  PTDD = tmp_var;
  EnableInterrupts;
}
```

7.4 Gestione Timer

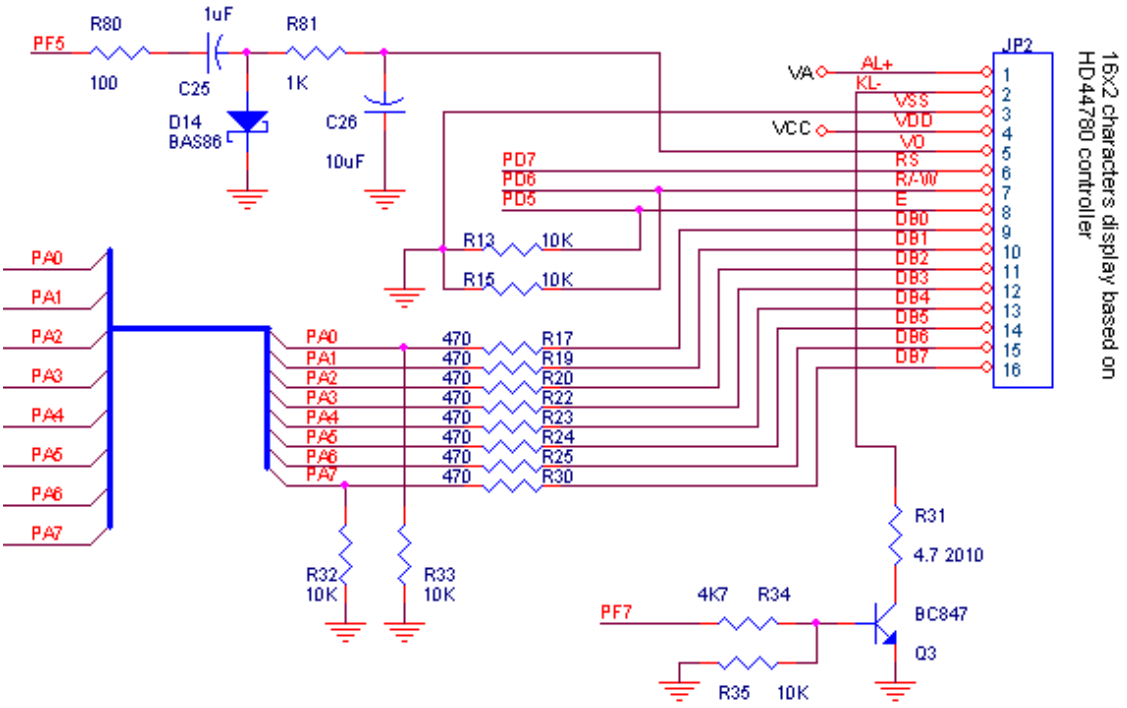
I timer vengo gestiti sia attraverso servizi di Interrupt dedicati, che scandiscono le temporizzazioni, sia attraverso variabili contatore che fanno da base dei tempi per i timer.

```
//----------------- System variables ----------------------------------------------------
extern volatile byte TickElapsed;    // It is incremented from Timer 1 overflow every
                                     // 13.33ms to signal cycle elapsed, and then it
                                     // is decremented from Main before cycle
                                     // execution
extern volatile word TickCounter;    // System cycles counter: it is incremented every
                                     // 13.33ms till 0xFFFF, then it restarts to 0

//------ Timers/counters allocated in local ram and managed from TK_HPController ------
// Eight bits timers with 1S timebase.
// If loaded with a value > 0, are decremented from TK_CK_Timers task every second.
union def_timsec{
   struct {
      byte  tim…;
      …
   } t;
   byte timArray[sizeof(t)];   // timer array definition
};
extern volatile union def_timsec timsec;

// Eight bits timers with 0,2S timebase. If loaded with a value > 0, are decremented
// from TK_CK_Timers task every 0,2 seconds.
union def_tim02{
   struct {
      byte  tim…;
      …
   } t;
   byte timArray[sizeof(t)];   // timer array definition
};
extern volatile union def_tim02 tim02;

// Eight bits timers with 26.66mS timebase (decremented by TK_HPController).
// If loaded with a value > 0, are decremented from TK_HP_Controller task every
// 26.66 milliseconds.
union def_tim26m{
   struct {
      byte  tim…;       // TIMER COMUNICAZIONE GD1xx
      …
   } t;
   byte timArray[sizeof(t)];   // timer array definition
};
extern volatile union def_tim26m tim26m;
```

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Timer1 overflow interrupt
*
*    \detail  The method services the interrupt of the selected peripheral(s)
*/
////////////////////////////////////////////////////////////////////////////////
ISR(T1_OVF_Interrupt){
  TickElapsed++;
  clrReg8Bits(TPM1SC, 0x80);         // Reset interrupt request flag
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Timer Management
*
*    \detail  management of all system software
*/
////////////////////////////////////////////////////////////////////////////////
void Timers(void){
  byte ii;

 // Check if 26ms are elapsed
  if(presc13%4==0){
    // Decrement timers/counters with timebase of 26ms
    for (ii=0; ii < NumElements(tim26m.timArray); ii++)
      if (tim26m.timArray[ii]!=0)
        tim26m.timArray[ii]--;        // decrement 8 bits timers/counters
  }

  // Check if 200ms are elapsed
  presc200++;     //increment prescaler
  if(presc200==15){  // 200 ms are elapsed
    // Decrement timers/counters with timebase of 0,2S
    presc200=0;
    for (ii=0; ii < NumElements(tim02.timArray); ii++){
      if (tim02.timArray[ii]!=0)
        tim02.timArray[ii]--;        // decrement 8 bits timers/counters
    }

    // Check if a second is elapsed
    prescSec++;                           // increment prescaler
    if (prescSec==CYCSEC){  // a second is elapsed
      prescSec=0;
      // Decrement timers/counters with timebase of 1S
      for (ii=0; ii < NumElements(timsec.timArray); ii++){
        if (timsec.timArray[ii]!=0)
          timsec.timArray[ii]--;  // decrement 8 bits timers/counters
      }
    }
  }
}
```

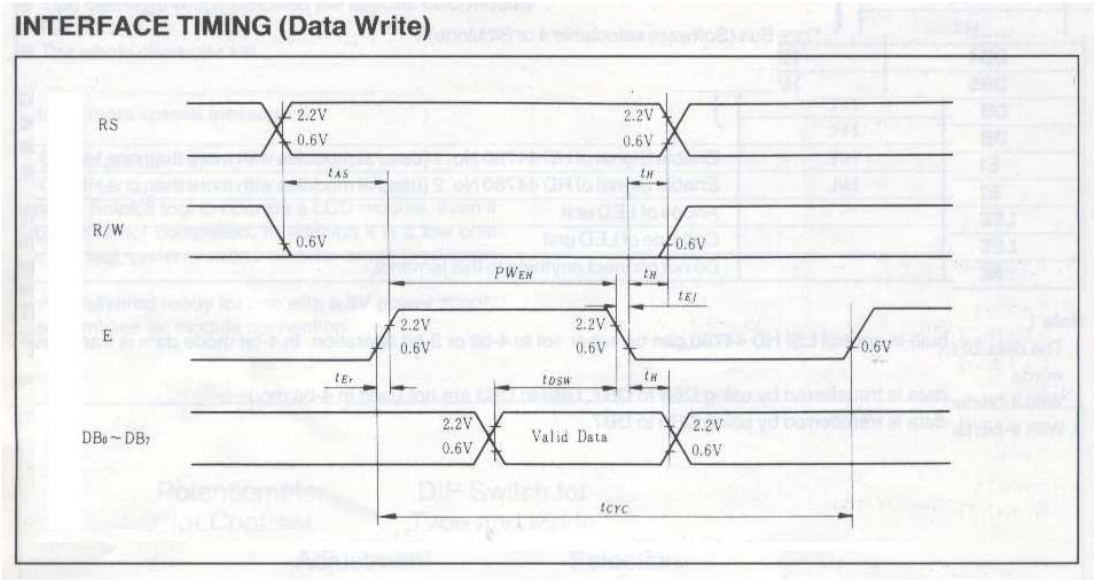
7.5 Gestione Display

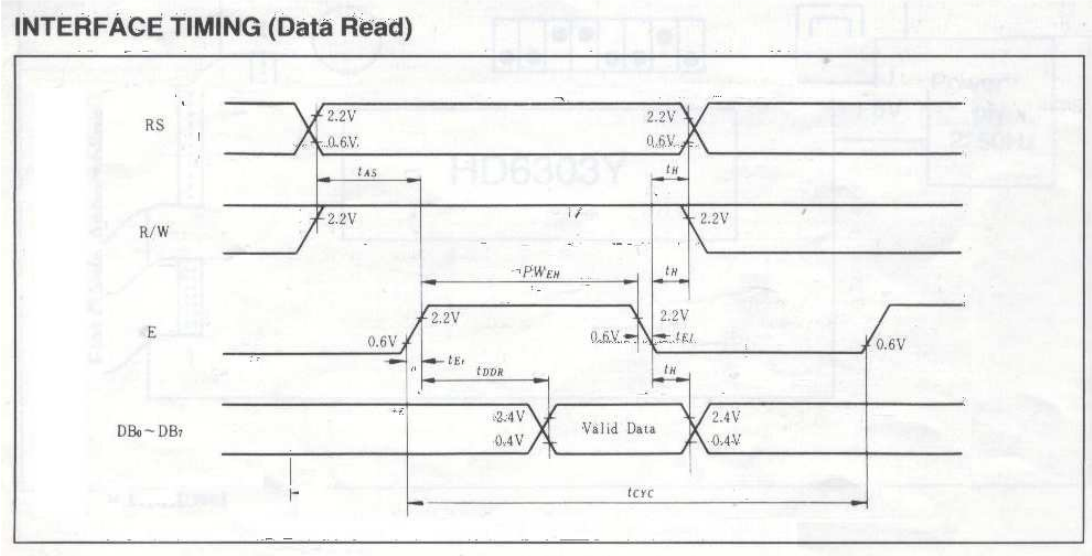
Il display utilizzato è dotato di un suo specifico protocollo di comunicazione, con vari possibili comandi e modalità di visualizzazione, una sequenza di inizializzazione del display e tutte le temporizzazioni relative.



7.5.1 Protocollo di comunicazione

I comandi di Scrittura e Lettura devono seguire le temporizzazioni riportate nei grafici sottostanti:



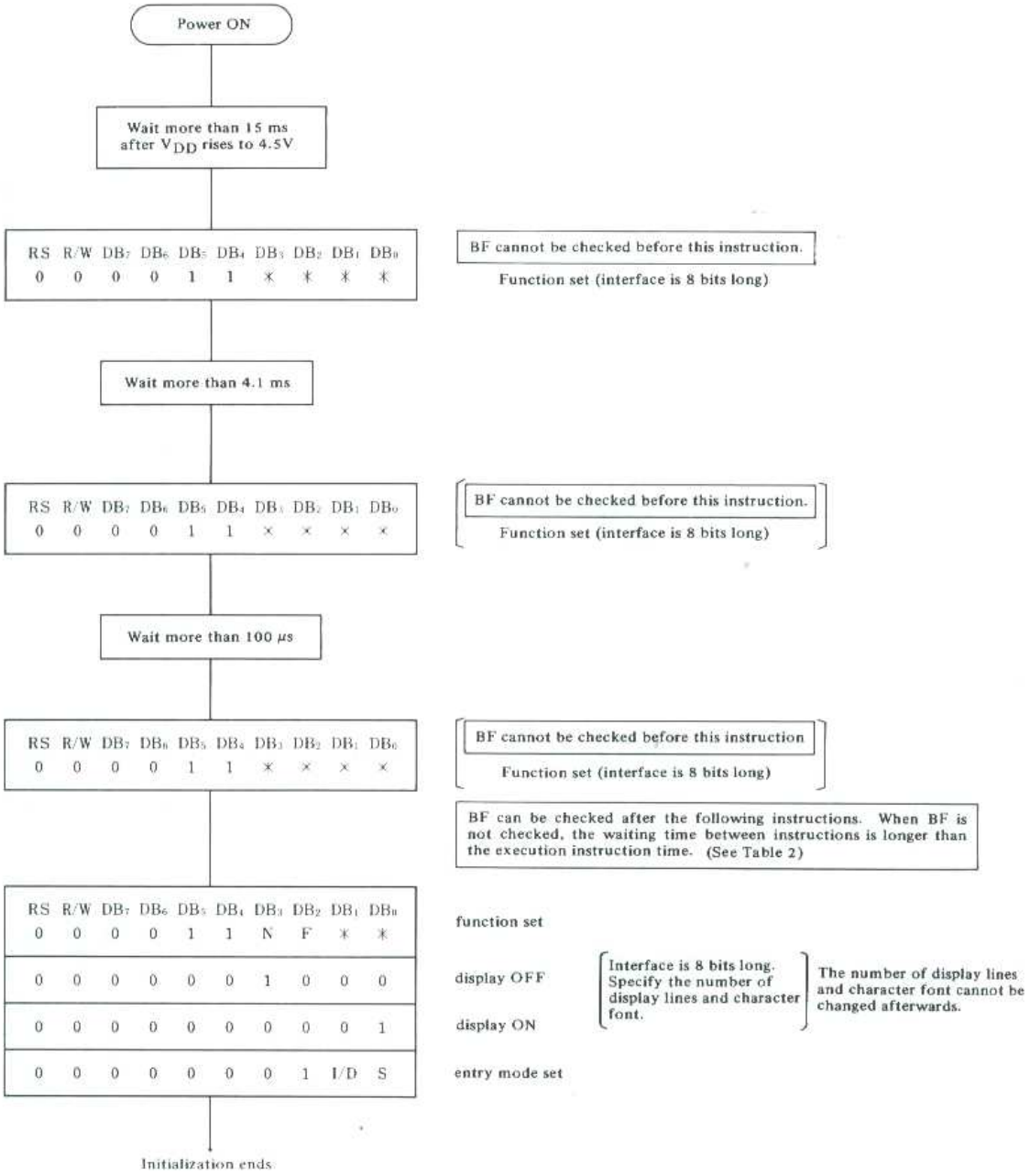


Le istruzioni e i comandi previsti dal protocollo di comunicazione del display sono:

Istruzione	**Codice**										**Descrizione**
	RS	**R/W**	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	
Clear display	0	0	0	0	0	0	0	0	0	1	Pulisce tutto il display e fa tornare il cursore nella posizione di partenza (indirizzo 0).
Return home	0	0	0	0	0	0	0	0	1	*	Fa tornare il cursore nella posizione di partenza (indirizzo 0). Inoltre il display viene spostato nella posizione originale. Il contenuto della DD RAM non cambia.
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Imposta la direzione in cui muovere il cursore e se spostare o no il display. Queste operazioni sono eseguite durante la lettura e scrittura dei dati.
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	Imposta l'accensione e lo spegnimento di tutto il display, del cursore e il lampeggiare del cursore nella posizione.
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	*	*	Muove il cursore e sposta il display senza cambiare il contenuto della DD RAM.
Function set	0	0	0	0	1	DL	N	F	*	*	Imposta la lunghezza dei dati di interfaccia, il

											numero delle righe del display e lo stile del carattere.
Set CG RAM address	0	0	0	1	A_{CG}						Imposta l'indirizzo di CG RAM. I dati di CG RAM sono mandati e ricevuti dopo questa impostazione.
Set DD RAM address	0	0	1	A_{DD}							Imposta l'indirizzo di DD RAM. I dati di DD RAM sono mandati e ricevuti dopo questa impostazione.
Read busy flag & address	0	1	BF	AC							Legge il Flag Busy che indica quando vengono eseguite operazioni interne e legge il contenuto del contatore di indirizzo.
Write data to CG or DD RAM	1	0	Write Data								Scrive un dato in DD RAM o CG RAM
Read data to CG or DD RAM	1	1	Read Data								Legge un dato da DD RAM o CG RAM
	I/D = 1 : incrementa (+1) = 0 : Decrementa (-1) S = 1: Accompagna spostamento display S/C = 1 : Spostamento Display = 0 : Movimento Cursore R/L = 1 : Sposta a Destra = 0 : Sposta a Sinistra DL = 1 : 8 bits = 0 : 4 bits N = 1 : 2 linee = 0 : 1 linea F = 1 : 5x10 punti = 0 : 5x7 punti BF = 1 : operazione interna = 0 : può ricevere istruzioni										DD RAM : Display Data RAM CG RAM : Character generator RAM A_{CG} : CG RAM Address A_{DD} : DD RAM Address, corrisponde all'indirizzo del cursore AC : contatore di indirizzo usato per gli indirizzi della DD RAM e della CG RAM

Il display deve eseguire una precisa sequenza di inizializzazione all'accensione per poter funzionare correttamente:




```
//------------------------- Display ------------------------------------------------
   bool init;
    byte command;
    word param;
    bool refresh;
    bool busy;
    bool dBlink;
    byte cursPos;
    bool cursOn;
    byte fasComSel;
};
extern volatile struct dsply display;
```

```
//--  Display command buffer
#define MAX_BUF_COMDISP     5       // display command buffer size
struct stcComDisp{
    byte command;
    word param;
};
// display command buffer writing pointer
extern volatile byte comDisp_store_ix;
// display command buffer reading pointer
extern volatile byte comDisp_read_ix;
// display command buffer
extern volatile struct stcComDisp comDisp_buf[MAX_BUF_COMDISP];

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Display management
*
*    \detail  The routine is called every 13ms and works executing the following tasks:
*             - execute a display refresh sequence every 420ms or before if requested
*              (every 420ms it is managed also the charachters blink);
*             - execute a special command (CLEAR or SPECIAL CHARACTERS, etc.) sequence
*               if requested;
*             - every 30s execute a sequence to reinitialize the display registers;
*             - manages physically the backlight.
*/
////////////////////////////////////////////////////////////////////////////////
void Display(void){
  if(tim26m.t.timPwr==0){       //wait a delay if power-up occoured
    if(display.init == TRUE){
      InitSeq();
    } else {
      // check if display refresh is active
      if(count_ref!=0){
        VisDisp();
      } else {
        if(display.command!=0){
          CommandSeq();
        } else {
           if (GetComDisp()){
              CommandSeq();
           }else{
              if(display.refresh==TRUE || tmpBlink!=(blink&0b00100000)){
                //refresh and blink evey 420ms
                tmpBlink=(blink&0b00100000);     //refresh of display
                VisDisp();
              } else {
                  if(tim02.t.timInit==0){
                    tim02.t.timInit=TIM_REFRESH;  //periodic re-init display every 15s
                    display.init=TRUE;
                  }
              }
           }
        }
      }
    }
  }
  if(opzioni[ADDBACKLI]!=BACKOFF){
    //manage back light
    if(timsec.t.timBL!=0 | flag_backLight==TRUE)  PTFD_PTFD7=1;  // back light is on
    else  PTFD_PTFD7=0;  // back light is off
  } else  PTFD_PTFD7=0;  // back light is off
}
```

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Write out of data or command
*
*    \detail  This function manage phisical control signals on D port to make possible
*             write operations. First it checks if the last operation is finishes.
*             If the display is not still busy, the function provide to send output
*             signals. It can do write operations of data or of command, on the
*             strenght of the flag com (TRUE=command, FALSE=data).
*/
////////////////////////////////////////////////////////////////////////////////
bool Write(bool com, byte data){
  byte ii;
  byte tmp;
  bool check=FALSE;
  PTDD_PTDD7=0;               //OFF RS
  PTADD=DDIR_IN;              //PTADD= INPUT
  for(ii=0; ii<200; ii++){
    PTDD_PTDD5=0;               //OFF E
    PTDD_PTDD6=1;               // R/W --> read
    PTDD_PTDD5=1;               //ON E
    _asm("nop");
    _asm("nop");
    tmp=(PTAD&0b10000000);        //read busy flag
    if(tmp==0){
      check=TRUE;               //if busy flag=0 --> can accept instruction
      break;
    }
  }
  PTDD_PTDD5=0;               //OFF E
  PTDD_PTDD7=0;               //OFF RS
  PTDD_PTDD6=0;               // R/W --> write
  PTADD=DDIR_OUT;             //PTADD= output
  if(check==TRUE){
    if(!com)  PTDD_PTDD7=1;   //ON RS
    PTAD=data;
    _asm("nop");
    _asm("nop");
    PTDD_PTDD5=1;               //ON E
    _asm("nop");
    _asm("nop");
    PTDD_PTDD5=0;               //OFF E
  }
  return check;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Display inizialization sequence
*
*    \detail  This function manages the initialization sequence of display.
*             The sequence is executed in three phases and while it is active
*             the busy flag is true.
*/
////////////////////////////////////////////////////////////////////////////////
void InitSeq(){
  count_init++;
  display.busy=TRUE;

  //sequence of write to do the inizialization of the display
  switch(count_init){
    case 1: ok_Done=Write(TRUE, 0b00111000);
            break;
```

```
    case 2: ok_Done=Write(TRUE, 0b00111000);
            break;
    case 3: ok_Done=Write(TRUE, 0b00111000);
            ok_Done=Write(TRUE, 0b00111000);
            ok_Done=Write(TRUE, 0b00001100);
            count_init=0;
            display.init=FALSE;   // init was done
            display.busy=FALSE;
            break;
    default: count_init=0;
             break;
  }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Manage special external command
*
*    \detail  Other routine can force the display to do a clear operation or to load
*             special characters. This function provide to execute those special
*             operations. During this operation the display is set busy.
*/
////////////////////////////////////////////////////////////////////////////////
void CommandSeq(){
  byte ii;
  byte tmp;
  byte tmp_char;
  byte *ptrnPointer;

  switch(display.command){

    case CLEAR: //clear display
                ok_Done=Write(TRUE, 0b00000001);  // command clear of display
                for(ii=0; ii<BUF_DISP; ii++){
                  disp_buf[ii]=' ';        // clear the buffer
                  disp_buf_blink[ii]=' ';  // clear the blink buffer
                }
                display.refresh=TRUE;
                display.command=0;
                display.param=0;
                display.fasComSel=0;
                break;

    case CHARACT: //load special characters set into CHR_RAM of display
                  //The carachters set loading is done in 8 cycles
                tmp = (display.fasComSel << 3);
                tmp = (tmp & 0b00111000);
                tmp_char = (BASECHAR | tmp);
                ok_Done=Write(TRUE, tmp_char);
                //param=address of special characters set selected
                ptrnPointer=(byte *)display.param;
                ptrnPointer+=(display.fasComSel * NUM_CHAR);
                // Load one special character
                for(ii=0; ii<N_PATTERNS; ii++){
                  ok_Done=Write(FALSE, *ptrnPointer++);
                }
                display.fasComSel++;
                if (display.fasComSel>=NUM_CHAR){
                  // Special charachters loading done
                  display.command=0;
                  display.param=0;
                  display.fasComSel=0;
                }
```

```
              break;

    default:  display.command=0;     // command not recognized
              display.param=0;
              display.fasComSel=0;
              break;
    }
  }

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Manage of display refresh and of characters blinking
*
*    \detail  Whene this function is called fisrt it manages characters blinking on the
*             strenght of the flag display.dBlink, of the precedent state memBlink and
*             the change of the state of the flag tmpBlink (every 420ms). Then it put
*             out 32 characters in a sequence of 4 steps (8 character for step). During
*             this operation the display is set busy.
*/
////////////////////////////////////////////////////////////////////////////////
void VisDisp(){
  byte ii;
  byte tmp_var;

  display.busy=TRUE;
  if(count_ref==0){
    if(display.dBlink==TRUE){
      if(tmpBlink==0){
        if(memBlink==FALSE){
          memBlink=TRUE;
          for(ii=0; ii<BUF_DISP; ii++){
            if(disp_buf_blink[ii]==TRUE)  tmp_buf[ii]=' ';
            else  tmp_buf[ii]=disp_buf[ii];
          }
        }
      } else{
          if(memBlink==TRUE){
            memBlink=FALSE;
            for(ii=0; ii<BUF_DISP; ii++)  tmp_buf[ii]=disp_buf[ii];
          }
        }
    } else {
        for(ii=0; ii<BUF_DISP; ii++)  tmp_buf[ii]=disp_buf[ii];
      }
  }

  count_ref++;
  switch(count_ref){
    case 1: ok_Done=Write(TRUE, 0b00001100);  //cursor off, display on
            ok_Done=Write(TRUE, 0b10000000);  //return home first line
            for(ii=0; ii<8; ii++) ok_Done=Write(FALSE, tmp_buf[ii]);
            break;

    case 2: for(ii=8; ii<16; ii++)  ok_Done=Write(FALSE, tmp_buf[ii]);
            break;

    case 3: ok_Done=Write(TRUE, 0b11000000);  //return home second line
            for(ii=16; ii<24; ii++) ok_Done=Write(FALSE, tmp_buf[ii]);
            break;

    case 4: for(ii=24; ii<BUF_DISP; ii++) ok_Done=Write(FALSE, tmp_buf[ii]);
            break;
```

```
    case 5: if(display.cursOn && (blink&0b00100000)){
              if(display.cursPos<16)
                tmp_var= 0b10000000 | (0b00001111 & display.cursPos);
              else  tmp_var= 0b11000000 | (0b00001111 & display.cursPos);
              ok_Done=Write(TRUE, tmp_var);
              ok_Done=Write(TRUE, 0b00001110);
            }
            count_ref=0;
            display.busy=FALSE;
            display.refresh=FALSE;
            break;

    default: count_ref=0;
             display.busy=FALSE;
             display.refresh=FALSE;
             break;
  }
}
```

7.5.2 Messaggi di visualizzazione

Se i messaggi da visualizzare sono differenti a seconda della lingua impostata, essi sono organizzati in array con le traduzioni nelle 5 lingue previste. In questo modo basta cambiare il parametro “lingua” nelle impostazioni e menù e messaggi da visualizzare sono già organizzati correttamente.

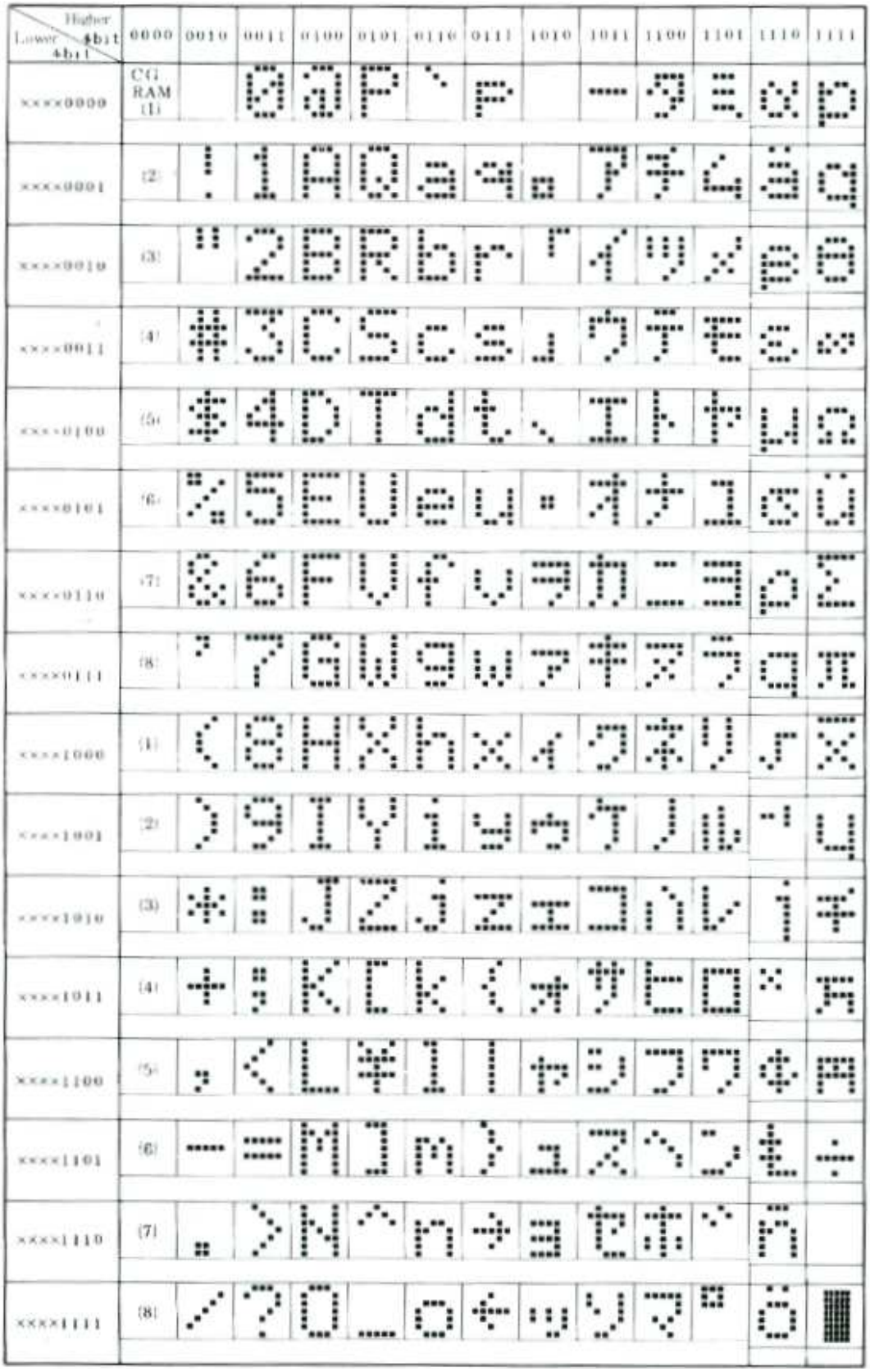
```
const char MAIN_MENU[NUM_LING][NUM_MESS][NUM_CHAR] = {
      //Français
        "Messaggio FR 1 ",
        …

      //English
        "  Messaggio EN 1 ",
        …

      //Italiano
        " Messaggio IT 1  ",
        …

      //Tedesco
        "  Messaggio DE 1  ",
        …

      //Olandese
        "  Messaggio HL 1   ",
        …
};
```

Se invece I messaggi sono uguali indipendentemente dalla lingua selezionata, gli array hanno una dimensione in meno e sono uguali qualsiasi sia l’impostazione della lingua.

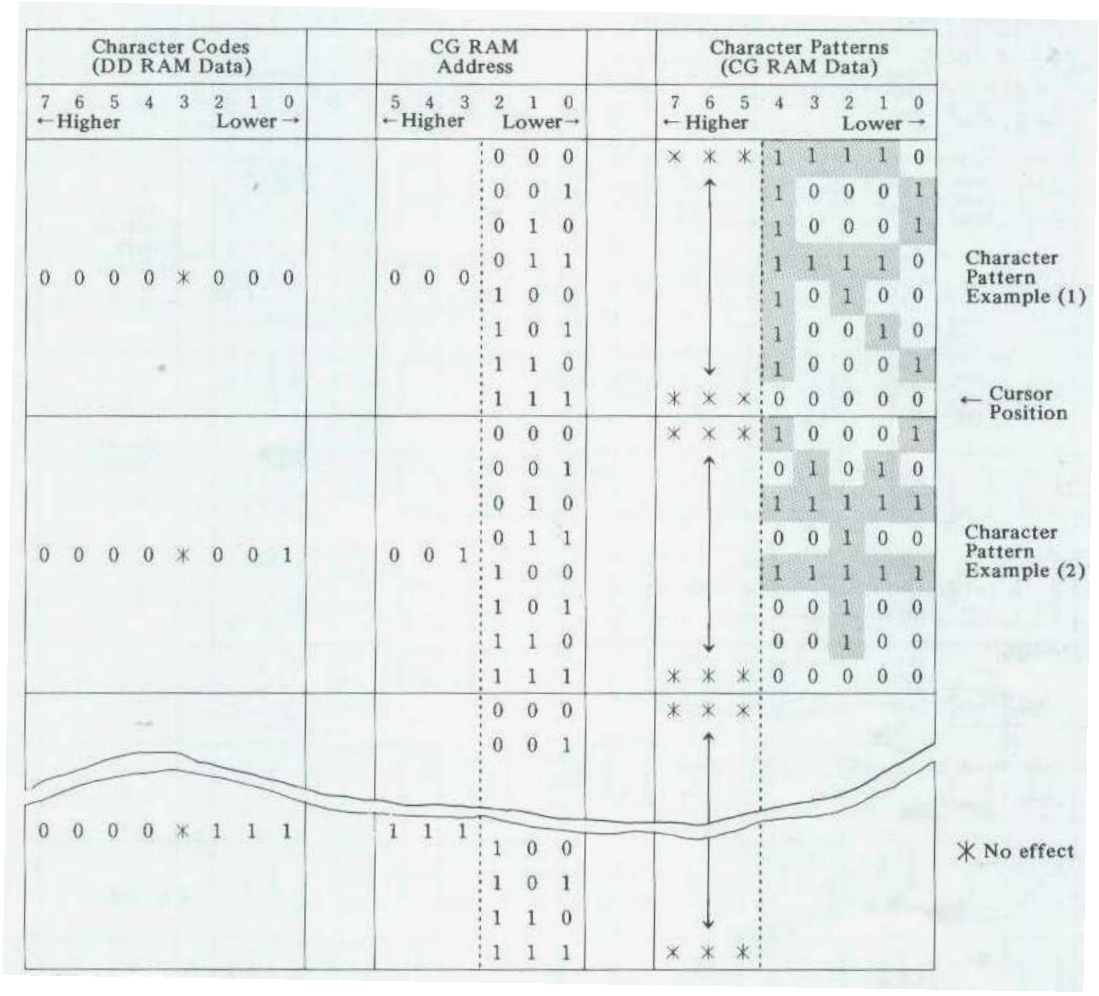
```
const char STRINGS[NUM_STRING][NUM_CHAR] = {
        “ String 1  ",
        …
};
```

7.5.3 Caratteri Speciali

Il display ha al suo interno il set di caratteri indicato nella tabella sottostante:



È possibile generare dei caratteri speciali da inserire nella CG RAM, in modo da ampliare il set a seconda delle necessità specifiche.



Per la gestione interna del TLC si sono resi necessari dei caratteri speciali sia per quanto riguarda la gestione delle batterie (indicatori di carica), sia per i caratteri accentati. I caratteri speciali, che variano a seconda della lingua selezionata, sono organizzati come nelle seguenti tabelle:

TLC BATTERY	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Lev Batt 0	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Lev Batt 1	0x00	0x00	0x00	0x00	0x00	0x00	0xFF	0x00
Lev Batt 2	0x00	0x00	0x00	0x00	0x00	0xFF	0xFF	0x00
Lev Batt 3	0x00	0x00	0x00	0x00	0xFF	0xFF	0xFF	0x00
Lev Batt 4	0x00	0x00	0x00	0xFF	0xFF	0xFF	0xFF	0x00
Lev Batt 5	0x00	0x00	0xFF	0xFF	0xFF	0xFF	0xFF	0x00
Lev Batt 6	0x00	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0x00
Lev Batt 7	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0x00

TLC FRANCAIS	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
è	0x08	0x04	0x16	0x11	0x1F	0x10	0x0E	0x00
è	0x02	0x04	0x16	0x11	0x1F	0x10	0x0E	0x00
à	0x08	0x04	0x16	0x01	0x16	0x11	0x0F	0x00
ç	0x0E	0x10	0x10	0x11	0x0E	0x04	0x0C	0x00
↑	0x00	0x04	0x0E	0x15	0x04	0x04	0x00	0x00
↓	0x00	0x04	0x04	0x15	0x0E	0x04	0x00	0x00
▏	0x18	0x18	0x18	0x18	0x18	0x18	0x18	0x00
▕	0x1B	0x1B	0x1B	0x1B	0x1B	0x1B	0x1B	0x00

TLC DEUTSCH / NEDERLANDS	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
spare								0x00
spare								0x00
spare								0x00
spare								0x00
↑	0x00	0x04	0x0E	0x15	0x04	0x04	0x00	0x00
↓	0x00	0x04	0x04	0x15	0x0E	0x04	0x00	0x00
▏	0x18	0x18	0x18	0x18	0x18	0x18	0x18	0x00
▕	0x1B	0x1B	0x1B	0x1B	0x1B	0x1B	0x1B	0x00

I messaggi relativi al DLFB utilizzano delle tabelle differenti, fornite dal produttore. Al momento sono tutte uguali indipendentemente dalla lingua ma in seguito è prevista una diversificazione a seconda dei tipi di messaggi e di caratteri che si renderanno necessari.

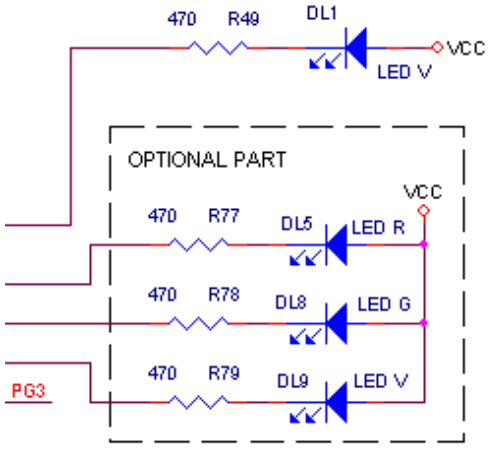
DLFB	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
↑	0x00	0x04	0x0E	0x15	0x04	0x04	0x00	0x00
↓	0x00	0x04	0x04	0x15	0x0E	0x04	0x00	0x00
Quadrato vuoto	0x1F	0x11	0x11	0x11	0x11	0x11	0x1F	0x00
Quadrato pieno	0x1F	0x1F	0x1F	0x1F	0x1F	0x1F	0x1F	0x00
spare	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
spare	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
spare	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
spare	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Special characters selector
*
*    \detail  This function check if special character 'charSetS' is selected and if
*             not then commands the special characters loading into display memory
*/
////////////////////////////////////////////////////////////////////////////////
void SpecialChar(byte charSetS){
  byte *ptrnPointer;

  switch(charSetS){
    case (FRA|DLFB_SET): ptrnPointer=(byte *)CharFraDLFB;
       break;
    case (ENG|DLFB_SET): ptrnPointer=(byte *)CharEngDLFB;
       break;
    case (ITA|DLFB_SET): ptrnPointer=(byte *)CharItaDLFB;
       break;
    case (TED|DLFB_SET): ptrnPointer=(byte *)CharTedDLFB;
       break;
    case (OLA|DLFB_SET): ptrnPointer=(byte *)CharOlaDLFB;
       break;
    case CBATT: ptrnPointer=(byte *)SpecialCharBattery;
       break;
    case TED: ptrnPointer=(byte *)SpecialCharTed;
       break;
    case OLA: ptrnPointer=(byte *)SpecialCharOla;
       break;
    default: ptrnPointer=(byte *)SpecialCharFrans;
       break;
  }

  if (spChAddSelected!=(word)ptrnPointer){
    // Commands special characters loading
    PutComDisp(CHARACT, (word)ptrnPointer);
    ClearBufDis();
    spChAddSelected=(word)ptrnPointer;
    spChsel=charSetS;
  }

}
```

7.6 Gestione led e buzzer

I led di interfaccia sul TLC posso assumere vari stati a seconda del modo di funzionamento:

- Off o On → spento o acceso;
- Blink → lampeggio a diverse frequenze:
  - Blink
    (320 ms On / 320 ms Off);
  - Slow Blink
    (640 ms On / 640 ms Off);
  - Shot Blink
    (100 ms On / 640 ms Off);
  - Rev Blink
    (1 s On / 213 ms Off).

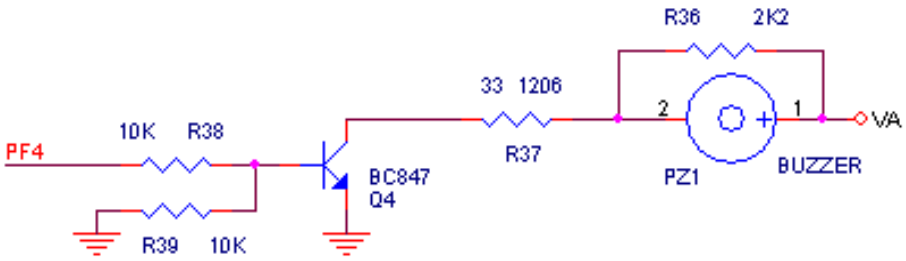


```
////////////////////////////////////////////////////////////////////////////////
/**
*   \brief    Refresh leds status
*
*   \detail   The routine is called every 40ms.
*             The function calculate the actual on/off state of leds and, if it's
*             necessary, it reverses the leds's state.
*/
////////////////////////////////////////////////////////////////////////////////
void Leds(void){
  switch (busyLed_status){
    case OFF: // led off
              statusBusyLed = FALSE;
              break;
    case ON:  // led on
              statusBusyLed = TRUE;
              break;
    case BLINK: // blink 320ms/320ms
                if ((blink & 0b00011000)==0) statusBusyLed = TRUE;
                else  statusBusyLed = FALSE;
                break;
    case SLOWBLINK: // slowblink 640ms/640ms
                    if ((blink & 0b00110000)==0) statusBusyLed = TRUE;
                    else  statusBusyLed = FALSE;
                    break;
    case SHOTBLINK: // shotblink 100sm/640ms
                    if ((blink & 0b00111000)==0)  statusBusyLed = FALSE;
                    else  statusBusyLed = TRUE;
                    break;
    case REVBLINK:  // revblink 1s/213ms
                    if ((blink & 0b00111000)==0) statusBusyLed = TRUE;
                    else  statusBusyLed = FALSE;
                    break;
    default:  statusBusyLed = FALSE;
              break;
  }

  switch (led1_status){
    case OFF: // led off
              statusLed1 = FALSE;
              break;
    case ON:  // led on
              statusLed1 = TRUE;
              break;
    case BLINK: // blink 320ms/320ms
                if ((blink & 0b00011000)==0) statusLed1 = TRUE;
                else  statusLed1 = FALSE;
                break;
    case SLOWBLINK: // slowblink 640ms/640ms
                    if ((blink & 0b00110000)==0) statusLed1 = TRUE;
                    else  statusLed1 = FALSE;
                    break;
    case SHOTBLINK: // shotblink 100sm/640ms
                    if ((blink & 0b00111000)==0)  statusLed1 = FALSE;
                    else  statusLed1 = TRUE;
                    break;
    case REVBLINK:  // revblink 1s/213ms
                    if ((blink & 0b00111000)==0) statusLed1 = TRUE;
                    else  statusLed1 = FALSE;
                    break;
    default:  statusLed1 = FALSE;
              break;
  }
```

```
    switch (led2_status){
    case OFF: // led off
              statusLed2 = FALSE;
              break;
    case ON:  // led on
              statusLed2 = TRUE;
              break;
    case BLINK: // blink 320ms/320ms
                if ((blink & 0b00011000)==0) statusLed2 = TRUE;
                else  statusLed2 = FALSE;
                break;
    case SLOWBLINK: // slowblink 640ms/640ms
                    if ((blink & 0b00110000)==0) statusLed2 = TRUE;
                    else  statusLed2 = FALSE;
                    break;
    case SHOTBLINK: // shotblink 100sm/640ms
                    if ((blink & 0b00111000)==0)  statusLed2 = FALSE;
                    else  statusLed2 = TRUE;
                    break;
    case REVBLINK:  // revblink 1s/213ms
                    if ((blink & 0b00111000)==0) statusLed2 = TRUE;
                    else  statusLed2 = FALSE;
                    break;
    default:  statusLed2 = FALSE;
              break;
  }

  switch (led3_status){
    case OFF: // led off
              statusLed3 = FALSE;
              break;
    case ON:  // led on
              statusLed3 = TRUE;
              break;
    case BLINK: // blink 320ms/320ms
                if ((blink & 0b00011000)==0) statusLed3 = TRUE;
                else  statusLed3 = FALSE;
                break;
    case SLOWBLINK: // slowblink 640ms/640ms
                    if ((blink & 0b00110000)==0) statusLed3 = TRUE;
                    else  statusLed3 = FALSE;
                    break;
    case SHOTBLINK: // shotblink 100sm/640ms
                    if ((blink & 0b00111000)==0)  statusLed3 = FALSE;
                    else  statusLed3 = TRUE;
                    break;
    case REVBLINK:  // revblink 1s/213ms
                    if ((blink & 0b00111000)==0) statusLed3 = TRUE;
                    else  statusLed3 = FALSE;
                    break;
    default:  statusLed3 = FALSE;
              break;
  }

  // upgrade led physical output
  DisableInterrupts;
  PTED_PTED7=!statusBusyLed;
  PTGD_PTGD0=!statusLed1;
  PTGD_PTGD1=!statusLed2;
  PTGD_PTGD2=!statusLed3;
  EnableInterrupts;
}
```

Il buzzer, oltre a gestire il suono per il feedback della tastiera, può anch'esso assumere vari stati:

- Off / On → spento o acceso;
- Long Beep → 1 beep lungoo per segnalazione errore 240 ms
- Beep →1 beep di 40 ms
- 1, 2, o 3 Beep → commandi previsti per il rivelatore DLFB




```
//------------------ Leds and buzzer ------------------------------------------------
typedef union {
  byte Byte[3];
  struct {
    byte status;
    byte numbeep;
  } State;
} buzstr;
extern volatile buzstr buzzer;

////////////////////////////////////////////////////////////////////////////////////////
/**
*   \brief    Manage buzzer activation
*
*   \detail   The routine is called every 40ms.
*             First the function checks if any keys was pressed and it manage that
*             event setting a timer of wait. Else the function calculate the actual
*             on/off state of the buzzer and, if it's necessary, it reverses the
*             buzzer state.
*/
////////////////////////////////////////////////////////////////////////////////////////
void Buzzer(void){
  bool stat_change;
  stat_change=0;
  if(fdbk_kbrd_beep!=0){
    //---- Manage keyboard beep
    if(enable_kbrd_beep==TRUE){
      switch (change){
        case 0: // start sequence
                //buzzer wait for timTCi = 26ms*24 = 1,3s
                tim26m.t.timTCi = 0x30;
                change++;
                if (buzz_stat != BUZZ_OFF){
                  buzz_stat = BUZZ_OFF;
                  TPM2C0SC = BUZZ_OFF;       //off
                }
                break;
        case 1: // buzzer on
                TPM2C0SC = BUZZ_ON;        //on
                change++;
                break;
        case 2: // buzzer off
                TPM2C0SC = BUZZ_OFF;       //off
```

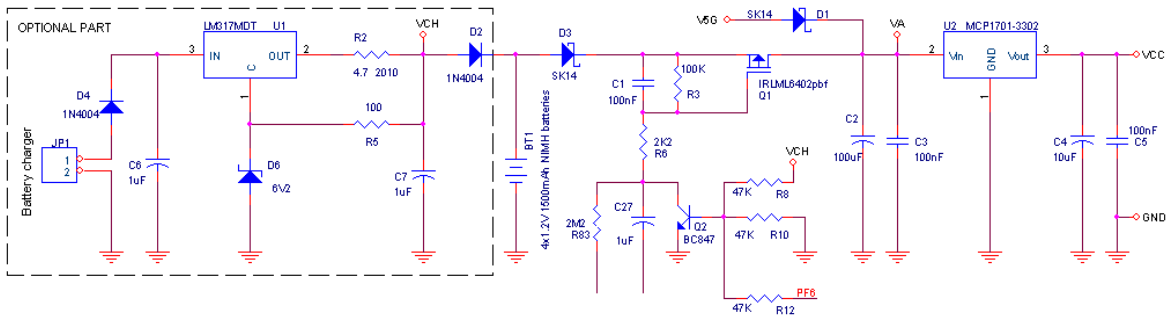
```
              change++;
              break;
      case 3: fdbk_kbrd_beep--;
              change=0;
              break;
      default:  change=0;
                tim26m.t.timTCi=0;
                buzz_stat = BUZZ_OFF;
                break;
    }
  } else  fdbk_kbrd_beep=0;
} else if (cnBeepErr){
  //---- Manage beep for error signalization
    if (cnBeepErr==1){
      // start sequence
      cnBeepErr++;
      if (buzz_stat != BUZZ_OFF){
        buzz_stat = BUZZ_OFF;
        TPM2C0SC = BUZZ_OFF;       //off
      }
    }else if (cnBeepErr<6){
      // buzzer on
      TPM2C0SC = BUZZ_ON;        //on
      cnBeepErr++;
    }else if (cnBeepErr==7){
      // buzzer off
      TPM2C0SC = BUZZ_OFF;       //off
      cnBeepErr++;
      buzz_stat = BUZZ_OFF;
    }else{
      cnBeepErr=0;
    }
} else {
  //---- Manage commanded buzzer status
  if(tim26m.t.timTCi==0){
    switch(buzzer.State.status){
      case NO_BEEP: TPM2C0SC = BUZZ_OFF;            //buzzer off
                    buzz_stat = BUZZ_OFF;
                    break;

      case BEEP_ON: TPM2C0SC = BUZZ_ON;             //buzzer on
                    buzz_stat = BUZZ_ON;
                    break;

      case BEEP_40m:    //40ms/40ms
                    stat_change=Beep();
                    break;

      case BEEP_1:  SeqBeep(15);
                    break;
      case BEEP_2:  SeqBeep(6);
                    break;
      case BEEP_3:  SeqBeep(3);
                    break;

    }
  }
  if(stat_change){
    if(buzz_stat==BUZZ_OFF){
      TPM2C0SC = BUZZ_ON;         //buzzer on
      buzz_stat = BUZZ_ON;
    } else {
        if(buzz_stat==BUZZ_ON){
```

```
            TPM2C0SC = BUZZ_OFF;       //buzzer off
            buzz_stat = BUZZ_OFF;
          }
        }
    }
  }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Beep
*
*    \detail  Required buzzer beeping
*/
////////////////////////////////////////////////////////////////////////////////
bool Beep(){
  bool stat=FALSE;
  if(buzzer.State.numbeep!=0){
    stat=TRUE;
    tmp_num++;
    if(tmp_num>=2){
      tmp_num=0;
      buzzer.State.numbeep--;
    }
  }
  return stat;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Sequence 1/2/3 Beep
*
*    \detail  Management of DLFB request of:
*               ..1 beep (120ms on / 600ms off) --> pausa = 15
*               ..2 beep (120ms on / 240ms off) --> pausa = 6
*               ..3 beep (120ms on / 120ms off) --> pausa = 3
*/
////////////////////////////////////////////////////////////////////////////////
void SeqBeep(byte pausa){
  switch(countBeep){
    case 0: TPM2C0SC = BUZZ_OFF;        //buzzer off
            buzz_stat = BUZZ_OFF;
            cnt_Buzz=3;
            countBeep++;
            break;
    case 1: TPM2C0SC = BUZZ_ON;         //buzzer on
            buzz_stat = BUZZ_ON;
            cnt_Buzz--;
            if(cnt_Buzz==0){
              countBeep++;
              cnt_Buzz=pausa;
            }
            break;
    case 2: TPM2C0SC = BUZZ_OFF;        //buzzer off
            buzz_stat = BUZZ_OFF;
            cnt_Buzz--;
            if(cnt_Buzz==0){
              cnt_Buzz=3;
              countBeep=1;
            }
            break;
  }
}
```

7.7 Analisi dello stato del sistema

Dato che il dispositivo può trovarsi in diverse modalità operative, si è resa necessaria una funzione che analizzasse il valore delle principali tensioni ed impostasse lo stato del TLC in uno dei seguenti:

- TLC alimentato da 5V (sensore di gas collegato) e carica batterie collegato;
- TLC alimentato da 5V (sensore di gas collegato) e carica batterie scollegato;
- TLC alimentato da carica batterie;
- TLC alimentato da batterie;

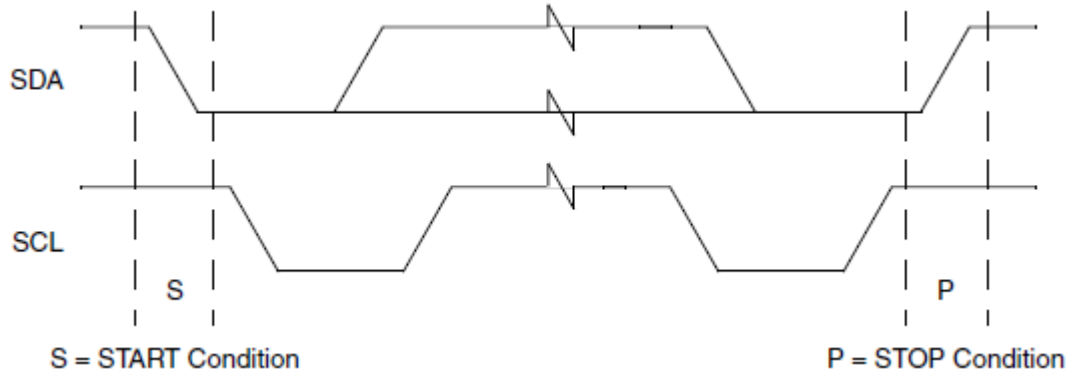



```
////////////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Power Mode analysis
*
*    \detail  This function reads analog input values and set device power mode
*             according to these values.
*/
////////////////////////////////////////////////////////////////////////////////////////
void PwMode(void){
  if(analogV.vAnalogMedia[V5G]>MIN_V5G){
    // TLC is powered from V5G
    if(fase!=ON_OFF){
      //reload power-on time if power is active
      timsec.t.timPwON=TEM_PWON;
    }
    if((analogV.vAnalogMedia[VCH]>MIN_VCH) &&
      (analogV.vAnalogMedia[VCH]>(analogV.vAnalogMedia[VA]+4))){
      pwFase=V5GVCH;
      inCarica=TRUE;     // charger is connected
    } else{
      pwFase=V5GFASE;
      inCarica=FALSE;    // charger is not connected
    }
  }else{
    // TLC is powered from battery or from charger
    if((analogV.vAnalogMedia[VCH]>MIN_VCH) &&
      (analogV.vAnalogMedia[VCH]>(analogV.vAnalogMedia[VA]+4))){
        // powered from charger
        pwFase=VCHFASE;
        inCarica=TRUE;
    }else{
        // powered from batteries
        pwFase=VBATFASE;
        inCarica=FALSE;
    }
  }
}
```

7.8 Parametri della EEPROM

Il TLC è dotato di una memoria esterna al microcontrollore, utilizzata per memorizzare alcuni parametri rivelati per l'uso del dispositivo (versione del firmware, lotto di produzione, etc). Per scrivere / leggere su questa memoria è stato utilizzato un bus interno di tipo I^2C, con il suo relativo protocollo di comunicazione.

L'I^2C è un bus bidirezionale basato su 2 fili in cui le linee di dato e di clock sono entrambe alte se la comunicazione non è attiva. Per generare la condizione si START della comunicazione, il segnale di dato deve compiere una transizione da alto a basso mentre il segnale di clock rimane alto. Al contrario, una transizione da basso a alto del segnale di dato mentre quello di clock è alto, è identificata come una condizione di STOP.



Durante la comunicazione, ogni byte trasmesso è seguito da un bit di conferma (ACK) e il segnale di clock genera un impulso positivo sia per gli 8 bit trasmessi che per il nono ricevuto.

Bit di ACK

Il bit di conferma (ACK) è un impulso a basso livello sulla linea di dato durante il nono colpo di clock alla fine di ogni trasmissione di byte sulla linea. Questo significa che nelle operazioni di scrittura lo slave genera un ACK al termine di ogni byte trasferito. Nelle operazioni di lettura lo slave genera l'ACK alla fine dei byte di comando e di indirizzo, e il master genera l'ACK quando riceve il byte letto dallo slave.

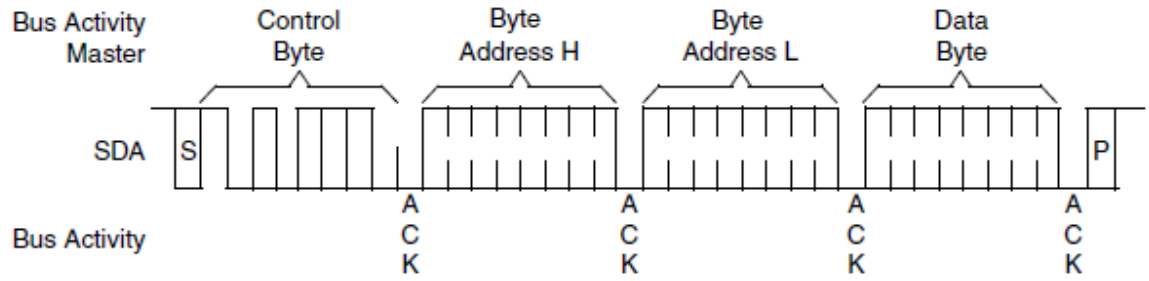
Byte di controllo

Il byte di controllo è sempre quello successivo alla condizione di START e serve per selezionare ed attivare il dispositivo specifico sul bus.

I primi 4 bit rappresentano il codice di controllo del dispositivo slave (ad esemprio, il codice di controllo di una EEPROM seriale è 0b1010). I successivi 3 bit rappresentano il chip select delle eventuali EEPROM connesse sul bus. L'ottavo bit è il bit R/W (Read/Write) che determina l'operazione richiesta per il dispositivo Slave: se R/W = 0 l'operazione è di scrittura, se R/W = 1 l'operazione è di lettura. L'ultimo è il bit di ACK ed è sempre generato dallo slave.

Byte di indirizzo

Il secondo byte porta le informazioni di indirizzo al dispositivo slave e determina l'indrizzo effettivo che verrà letto/scritto in EEPROM. Se l'indirizzo è composto da una word viene spezzato in 2 parti e quella più significativa viene inviata per prima.




```
//---------------------- IICSW variables-------------------------------------------------
typedef union {
  word EE_Address;

  struct{
    byte Address_H;
    byte Address_L;
  }Bytes;

}tAddr;
extern tAddr sADDR;


////////////////////////////////////////////////////////////////////////////////
/**
*    \brief    Read data from IICSW
*
*    \detail
*/
////////////////////////////////////////////////////////////////////////////////
byte IICSW_read_byte(word addr, byte *pnData){
  byte ctrl;
  byte data;
  byte maxTent=200;    // wait max 10ms (ending of last write)

  Address = addr;
  ctrl = IIC_SLAVE | ((Addr_H & 0x07) << 1);

  for (;;){
    IICSW_Start();
    if (IICSW_SnDt(ctrl)) break;
    else{
      maxTent--;
      if (!maxTent){IICSW_Err(); return ERR_FAULT;}
    }
```

```
  }
  if (!IICSW_SnDt(Addr_L)){IICSW_Err(); return ERR_FAULT;}
  ctrl=ctrl | 0x01;
  IICSW_Start();
  if (!IICSW_SnDt(ctrl)){IICSW_Err(); return ERR_FAULT;}
  data=IICSW_RdDt();
  IICSW_Stop();
  *pnData=data;
  return _OK;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief    Write data to IICSW
*
*    \detail
*/
////////////////////////////////////////////////////////////////////////////////
byte IICSW_write_byte(word addr, byte *pnData){
  byte temp;
  byte maxTent=200;    // wait max 10ms (ending of last write)

  Address = addr;
  temp = IIC_SLAVE | ((Addr_H & 0x07) << 1);

  for (;;){
    IICSW_Start();
    if (IICSW_SnDt(temp)) break;
    else{
      maxTent--;
      if (!maxTent){IICSW_Err(); return ERR_FAULT;}
    }
  }
  if (!IICSW_SnDt(Addr_L)){IICSW_Err(); return ERR_FAULT;}
  temp= *pnData;
  if (!IICSW_SnDt(temp)){IICSW_Err(); return ERR_FAULT;}
  IICSW_Stop();
  *pnData=temp;

  return _OK;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief    Send START sequence to IICSW
*
*    \detail
*/
////////////////////////////////////////////////////////////////////////////////
void IICSW_Start(void){
#pragma INLINE
  ONCK_IIC       // on ck
  ONDT_IIC       // on dt
  CK_OUT_IIC     // out
  DT_OUT_IIC     // out
  PAUSE_IIC
  OFFDT_IIC      // off dt (START)
  PAUSE_IIC
  OFFCK_IIC      // off ck
  PAUSE_IIC
  ONDT_IIC       // on dt
}
```

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief    Send STOP sequence to IICSW
*
*    \detail
*/
////////////////////////////////////////////////////////////////////////////////
void IICSW_Stop(void){
#pragma INLINE
  OFFDT_IIC      // off dt
  DT_OUT_IIC     // out
  ONCK_IIC       // on ck
  PAUSE_IIC
  ONDT_IIC       // on dt
  PAUSE_IIC
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief    IICSW error management
*
*    \detail
*/
////////////////////////////////////////////////////////////////////////////////
#pragma MESSAGE DISABLE C4301       // Inline expansion message removing
void IICSW_Err(void){
#pragma INLINE
  IICSW_Stop();
}
#pragma MESSAGE DEFAULT C4301       // Inline expansion message restore

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief    Send data to IICSW
*
*    \detail
*/
////////////////////////////////////////////////////////////////////////////////
bool IICSW_SnDt(byte data){
  byte counter;
  byte mask;
  byte temp;
  ONDT_IIC      // on dt
  DT_IN_IIC     // in
  counter=20;
  while(!DT_IIC){
    counter--;
    if (!counter)return FALSE;
  }
  ONDT_IIC      // on dt
  DT_OUT_IIC    // out
  mask=0x80;
  for (counter=0; counter<8; counter++){
    if (mask&data)  ONDT_IIC     // on dt
    else  OFFDT_IIC    // off dt
    PAUSE_IIC
    ONCK_IIC       // on ck
    PAUSE_IIC
    OFFCK_IIC      // off ck
    mask=(mask>>1);
  }
  ONDT_IIC         // on dt
  DT_IN_IIC        // in
```

```
  PAUSE_IIC
  PTCD_PTCD0=1;     // on ck
  PAUSE_IIC
  if (DT_IIC) temp=FALSE;
  else  temp=TRUE;
  OFFCK_IIC     // off ck
  asm(nop); asm(nop);
  DT_OUT_IIC    // out
  ONDT_IIC      // on dt
  return temp;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief    Read data from IICSW
*
*    \detail
*/
////////////////////////////////////////////////////////////////////////////////
byte IICSW_RdDt(void){
  byte counter;
  byte mask;
  byte data;
  ONDT_IIC     // on dt
  DT_IN_IIC   // in
  mask=0x80;
  data=0;
  for (counter=0; counter<8; counter++){
    PAUSE_IIC
    ONCK_IIC      // on ck
    PAUSE_IIC
    if (DT_IIC==1)  data=data | mask;
    OFFCK_IIC     // off ck
    mask=(mask>>1);
  }
  DT_OUT_IIC      // out
  return data;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Set Device Options
*
*    \detail  This function has to manage reading and setting byte values from and to
*             E2PROM memory according to I2CBUS interface protocol, on addresses from
*             0 to 1023.
*/
////////////////////////////////////////////////////////////////////////////////
void SetOption(void){
  byte temp;
  byte retCode;
  switch(countOpz){
    case 0: if(!pMyInput.initialized){
              MyPrint(STR_OPZ[1], 0, 16);
              MyPrint(STR_OPZ[2], 16, BUF_DISP);
              InitMyInput(0,99, 2, FALSE, 7);
              timsec.t.timoutOpz=TIMOPZ;
            }else{
              retCode=MyInput();
              if (retCode==INP_KEYPR){
                // numeric key
                timsec.t.timoutOpz=TIMOPZ; // reload timer
              }else if (retCode==2){
```

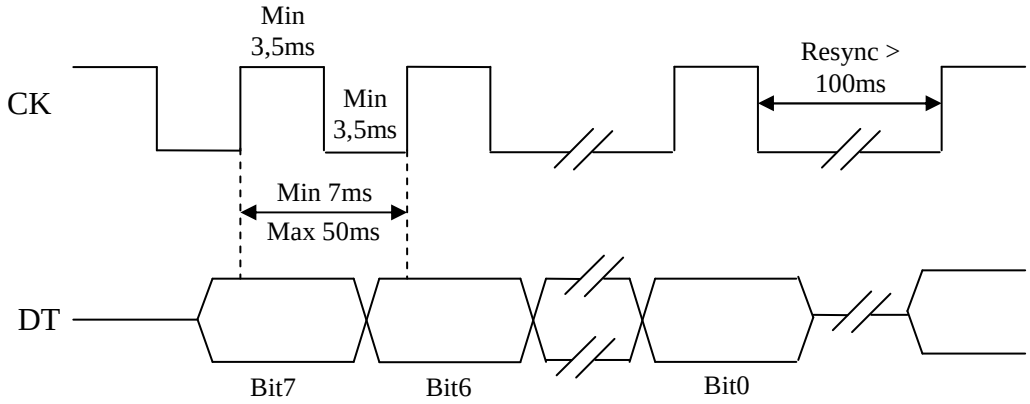
```
              //enter
              address=(byte)pMyInput.valInput;
              if(address==ADDSERVMOD) address=ADDSERVICE;
              else  address=address+OFFSETOPZ;
              countOpz++;

            }else if (retCode>2){
              // abort
              BackOpz();
              fase=INIT;
              ok_passw=FALSE;
            }
          }
          break;

    case 1:
          if(IICSW_read_byte(address, &opzData)==_OK){
            temp=opzData%100;
            disp_buf[23]=(opzData/100)+48;
            disp_buf[25]=(temp%10)+48;
            disp_buf[24]=(temp/10)+48;
            display.refresh=TRUE;
            timsec.t.timoutOpz=TIMOPZ;
            countOpz++;
          } else{
              MyPrint(CONF[lingua][3], 0, BUF_DISP);
              BeepErr();
              countOpz=0;
          }

    case 2: if(Get_last_key()){//wait a key
            if(commandK==K_MENU || commandK==K_ENTER){
              //menu --> modifica
              MyPrint(STR_OPZ[2], 16, BUF_DISP);
              timsec.t.timoutOpz=TIMOPZ;
              InitMyInput(0, 255, 3, FALSE, 23);
              countOpz++;
            }else if(commandK==K_BACK)  BackOpz();
          }
          break;

    case 3: retCode=MyInput();
          if (retCode==INP_KEYPR){
            // numeric key
            timsec.t.timoutOpz=TIMOPZ; // reload timer

          }else if (retCode==INP_DONE){
            //enter
            opzData=(byte)pMyInput.valInput;
            if (IICSW_write_byte(address, &opzData)==_OK){
              if(address-OFFSETOPZ==ADDSELDEV | address-OFFSETOPZ==ADDSELVIS)
                opzioni[address-OFFSETOPZ]=opzData;
              if(address==ADDSERVICE){
                opzioni[address]=opzData;
              }
              tim02.t.timOpz=10;
              Save_visual();   // save current visualization
              timsec.t.timoutOpz=TIMOPZ;
              countOpz++;
            } else{
                MyPrint(CONF[lingua][3], 0, BUF_DISP);
                BeepErr();
            }
```

```
            }else if (retCode>=INP_ABORTED){
              // abort
              BackOpz();
            }
            break;

    case 4: if(tim02.t.timOpz!=0){
              MyPrint(CONF[lingua][5], 0, 22);
              display.refresh=TRUE;
            }else{
              if(Restore_visual()){
                timsec.t.timoutOpz=TIMOPZ;
                countOpz=1;
              }
            }
            break;

    default:  BackOpz();
              countOpz=0;
              break;
  }
}
```

8 SVILUPPO GESTIONE GD1xx

8.1 Protocollo di comunicazione TTL

La comunicazione tra dispositivo TLC ed il rivelatore di gas è gestita in modalità "master-slave", in cui il master è rappresentato dal TLC e lo slave dal rivelatore di gas.
Il controllo dei segnali CK (clock) e DT (I/O data) è gestito direttamente da software tramite routine di interrupt collegate al timer interno del microcontrollore. Il clock è fornito dal TLC ed ha un periodo compreso tra min 7ms (3,5ms fase attiva, 3,5ms fase passiva) e max 50ms, il dato è letto dal sensore di gas sul fronte 5V-0V (fronte attivo). I comandi di protocollo sono eseguiti effettuando una pausa >100ms tra la fine di un comando e l'inizio del successivo per garantire la sincronizzazione tra master e slave.



I segnali resi disponibili dall'interfaccia sono i seguenti:

- **V+** Positivo +5V fornito dal rivelatore di gas per alimentare il terminale TLC;
- **C** Negativo. Comune segnali e alimentazione;
- **DT** Segnale di dato bidirezionale. Open collector normalmente aperto (livello logico 0) con pull-up a +5V, chiude verso C quando il segnale è attivo (livello logico 1). Il segnale può essere pilotato dal TLC o dal rivelatore**.**
- **CK** **S**egnale di clock fornito dal TLC. Open collector normalmente aperto (livello logico 0) con pull-up a +5V, chiude verso C quando il segnale di clock è attivo (livello logico 1). Il segnale è pilotato dal dispositivo TLC.

Nota: TLC tiene il segnale di clock chiuso a negativo (livello logico 1) nelle pause tra le fasi di trasmissione per segnalare la presenza del collegamento al sensore di gas.

Il protocollo di comunicazione è molto semplice, ed è costituito da 2 comandi.

Comando di lettura
Permette di leggere una cella di memoria ed ha il seguente formato:

Master:	Slave:
ID_Read	ID_Read
Indirizzo	Dato
LRC	LRC

LRC è calcolato partendo dal valore 0 ed effettuando lo XOR dei byte che costituiscono il messaggio escluso LRC. Dato è il valore letto dalla cella.

Comando di scrittura

Permette di scrivere una cella di memoria (solo indirizzi 000-099) ed ha il seguente formato:

Master:

ID_Write
Indirizzo
Dato
LRC

Slave:

ID_Write
Dato
LRC

LRC è calcolato partendo dal valore 0 ed effettuando lo XOR dei byte che costituiscono il messaggio escluso LRC.

Dato rappresenta l'esito dell'operazione di scrittura e può assumere i seguenti valori:

- 0xBB operazione di scrittura eseguita correttamente;
- 0xBD operazione di scrittura fallita;

```
//------------------- GD1xx management variables -------------------------------------
struct stc_kGas{
  byte count_vis;
  byte type_sens;
  byte unita_misura;
  byte decimali;
  byte fondoscala;
  byte mul_k;
  word valMaxUM;
  byte maxCifre;
};

extern volatile struct stc_kGas kGas;

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Transmittion to gas detector
*
*    \detail  This function manage the transmittion of a single byte to gas detector.
*/
////////////////////////////////////////////////////////////////////////////////
void Tx_Gd1xx(void){

  //if clock off then set next bit to send out
  if(PTFD_PTFD0==0){
    if(byteToSend & (MASK_BIT<<count_bit))  PTFD_PTFD1=1;
    else  PTFD_PTFD1=0;
    count_bit++;
    if(count_bit>7){
      count_bit=0;
      tx_bit=FALSE;
    }
  }

  clkAct=TRUE;
}
```

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Reception from gas detector
*
*    \detail  This function manage the reception of a single byte from gas detector.
*/
////////////////////////////////////////////////////////////////////////////////
void Rx_Gd1xx(void){
  //if clock on then read next bit received
  if(PTFD_PTFD0==1){
    var_tmp=PTFD_PTFD2;
    if((count_bit==0)&&(var_tmp==1))  byteReceived=0x0;
    if(count_bit>0)
      byteReceived=(byteReceived | (((var_tmp)&0x01) << (count_bit-1)));
    count_bit++;
    if(count_bit>8){
      count_bit=0;
      rx_bit=FALSE;
    }
  }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Gas detector communication management
*
*    \detail  This function sets data output to gas detector and check if data
*             receicived from device are correct.
*/
////////////////////////////////////////////////////////////////////////////////
void TRx_Manag(void){
  byte ii;

  switch(flag_trx_gd1xx){
    case NEED_TRX:  for(ii=0; ii<GD1_INP_BUF_SIZE; ii++)  inpBufGD1xx[ii]=0;
                    if(id_Gd1==READ){
                      //read operation

                      … codice omesso per riservatezza …

                    } else {
                        if(id_Gd1==WRITE){
                          //write operation

                             … codice omesso per riservatezza …

                        } else{
                            //wrong id selected
                            retCodeGd1=ERR_IDT;
                            flag_trx_gd1xx=0;
                          }
                      }
                    break;

    case TRX_GD1: //send and receive data
                  Gd1xx_TRx();
                  break;

    case TRX_END: //end of communication
                  if(inpBufGD1xx[0]!=inpBufGD1xx[1] | inpBufGD1xx[1]!=inpBufGD1xx[2]){
                    if(inpBufGD1xx[0]==id_Gd1){
                      if(lrc_rx==inpBufGD1xx[GD1_INP_BUF_SIZE-1]){
```

```
                            … codice omesso per riservatezza …

                      } else{
                          //wrong lrc received
                          retCodeGd1=ERR_LRCR;
                          flag_trx_gd1xx=0;
                          lrc_rx=0;
                          busyLed_status=BLINK;
                        }
                    } else{
                        //wrong id received
                        retCodeGd1=ERR_IDR;
                        flag_trx_gd1xx=0;
                        lrc_rx=0;
                        busyLed_status=BLINK;
                      }
                  } else{
                      on_gd1xx=FALSE;
                      bar_progr=16;
                      tim26m.t.timGas=75;
                      flag_trx_gd1xx=0;
                      lrc_rx=0;
                      busyLed_status=OFF;
                    }
                    break;
  }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Management of communication phases
*
*    \detail  This function manages phases of transmit and receive communication from
*             and to gas detector.
*/
////////////////////////////////////////////////////////////////////////////////
void Gd1xx_TRx(void){

  if(tx_bit){
    Tx_Gd1xx();
  } else{
      if(rx_bit){
        Rx_Gd1xx();
      } else{
          switch(count_Gd1){
            case 0: //send all data in output buffer
                    if(count<varTmp){
                      byteToSend=outBufGD1xx[count];
                      tx_bit=TRUE;
                      count++;
                    } else{
                        count_Gd1++;
                        tx_bit=FALSE;
                        stopClkFlg=TRUE;
                      }
                    break;

            case 1: //start to receive
                    count=0;
                    PTFD_PTFD1=0;
                    rx_bit=TRUE;
                    count_Gd1++;
                    break;
```

```
            case 2: //receive data from Gd1xx
                    inpBufGD1xx[count]=byteReceived;
                    if(count<GD1_INP_BUF_SIZE-1)  lrc_rx=lrc_rx^byteReceived;
                    count++;
                    rx_bit=TRUE;
                    if(count>GD1_INP_BUF_SIZE){
                      rx_bit=FALSE;
                      count_Gd1++;
                    }
                    break;
            case 3: //reset flag
                    count_Gd1=0;
                    count=0;
                    varTmp=0;
                    flag_trx_gd1xx=TRX_END;
                    clkAct=FALSE;
                    break;

            default:  count_Gd1=0;
                      count=0;
                      varTmp=0;
                      break;
          }
        }
    }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Management of gas detector
*
*    \detail  This function schedule all thing concern gas detector. It manages menù
*             selection and all different phases expected, selection = (
*                 1 --> Stato Sensore,
*                 2 --> Misura,
*                 3 --> Test,
*                 4 --> Preallarme,
*                 5 --> Allarme,
*                 6 --> Filtro Preallarme,
*                 7 --> Filtro Allarme,
*                 8 --> Ritardo Iniziale).
*/
////////////////////////////////////////////////////////////////////////////////
void Manag_Gd1xx(void){
  byte ii;
  char tmpVis[BUF_DISP];

  if(last_key)  timsec.t.timBL=TEMP_BKLIGHT;
  if(timsec.t.timLedGas==0){
    if(flag_trx_gd1xx==0){
      if(primo){
        tmpAdd=add_Gd1;
        SetTxGas(ADD_STATO);
        primo=FALSE;
      } else{
          if(flag_trx_gd1xx==0){
            timsec.t.timLedGas=10;
            primo=TRUE;
            if(retCodeGd1==0 && (inpBufGD1xx[1]<4)) LedGas(inpBufGD1xx[1]);
            SetTxGas(tmpAdd);
          }
        }
    }
```

```
} else{
    switch(status_menu){
      //management main menù
      case VERS_NAME: //show device's name and version
                      if (!tim02.t.timeErr){
                        if(flag_disp){
                          if (!CheckBusyDisp()){
                            VisVersName();
                            last_key=0;
                          }
                        } else{
                            if (Get_last_key()){//wait a key
                              if(commandK==K_MENU && on_gd1xx){
                                Back();
                                selectMenG=1;
                              }else if(commandK==K_BACK){
                                device=1;
                                kGas.count_vis=0;
                                flag_disp=TRUE;
                              }
                            }
                        }
                      }
                      break;
      case MENU_1:  if(flag_disp){
                       if (!CheckBusyDisp()) VisMenu();
                    } else if (Get_last_key()) KeyMenu(); // wait a key
                    break;
      case LIE_VAL: if(flag_disp){
                      if (!CheckBusyDisp()) VisLV();
                    } else if (Get_last_key()){// wait a key
                      if(commandK==K_ENTER && selectMenG!=8){
                        //menu = modifica
                        status_menu=MODIFICA;
                        pMyInput.initialized=FALSE;
                        fasModif=0;
                        flag_disp=TRUE;
                      }else if(commandK==K_BACK || commandK==K_MENU)  Back();
                    }
                    break;
      case STAT:  if(flag_disp){
                    if (!CheckBusyDisp()) VisStat();
                  }else if(timsec.t.timStat==0 && flag_disp==FALSE){
                    flag_disp=TRUE;
                    first=TRUE;
                  }
                  if (Get_last_key()){// wait a key
                    if(commandK==K_BACK || commandK==K_MENU) Back();
                  }
                  break;
      case MISBAR:  if(flag_disp){
                      if (!CheckBusyDisp()) VisBarra();
                    } else if(tim02.t.timRefMis==0 && flag_disp==FALSE){
                      first=TRUE;
                      second=TRUE;
                      third=TRUE;
                      flag_disp=TRUE;
                      bar_progr=16;
                    }
                    if (Get_last_key()){// wait a key
                      if(commandK==K_BACK || commandK==K_MENU)  Back();
                    }
                    break;
```

```
        case MISURA:  if(flag_disp){
                        if (!CheckBusyDisp()) VisMis();
                      }else if(tim02.t.timRefMis==0 && flag_disp==FALSE){
                        first=TRUE;
                        second=TRUE;
                        third=TRUE;
                        flag_disp=TRUE;
                      }
                      if (Get_last_key()){// wait a key
                        if(commandK==K_BACK || commandK==K_MENU)  Back();
                      }
                      break;
        case TEST:  //test_dev=0 --> Preallarme
                    //test_dev=1 --> Allarme
                    if(tim02.t.timRefMis==0){
                      for(ii=0; ii<16; ii++)  tmpVis[ii]=MAIN_MENU1[lingua][8][ii];
                      for(ii=0; ii<16; ii++)  tmpVis[ii+16]=MAIN_MENU1[lingua][9][ii];
                      MyPrint(tmpVis, 0, BUF_DISP);
                      disp_buf[test_dev+14+15*test_dev]='*';
                    }
                    if (Get_last_key()){ // Wait a key
                      //enter
                      if(commandK==K_ENTER){
                        //Preallarme
                        if(test_dev==0) TestG(92);
                        else{
                          //Allarme
                          if(test_dev==1) TestG(93);
                        }
                        status_menu=STAT;
                        selectMenG=1;
                        flag_disp=TRUE;
                        first=TRUE;
                        timsec.t.timStat=3;
                      }else if(commandK==K_UP){
                        //up
                        test_dev=0;
                      }else if(commandK==K_DWN){
                        //down
                        test_dev=1;
                      }else if(commandK==K_BACK || commandK==K_MENU)  Back();
                    }
                    break;
        case MODIFICA:  if (!tim02.t.timeErr){
                          if(flag_disp){
                            if (!CheckBusyDisp()) VisMod();
                          }else if (!Modifica()){
                            Back();
                            status_menu=LIE_VAL;
                          }
                        }
                        break;
        case CONFIG:  //configuration phase
                      Config();
                      break;
        default:  Back();
                  status_menu=0;
                  break;
      }
    }
}
```

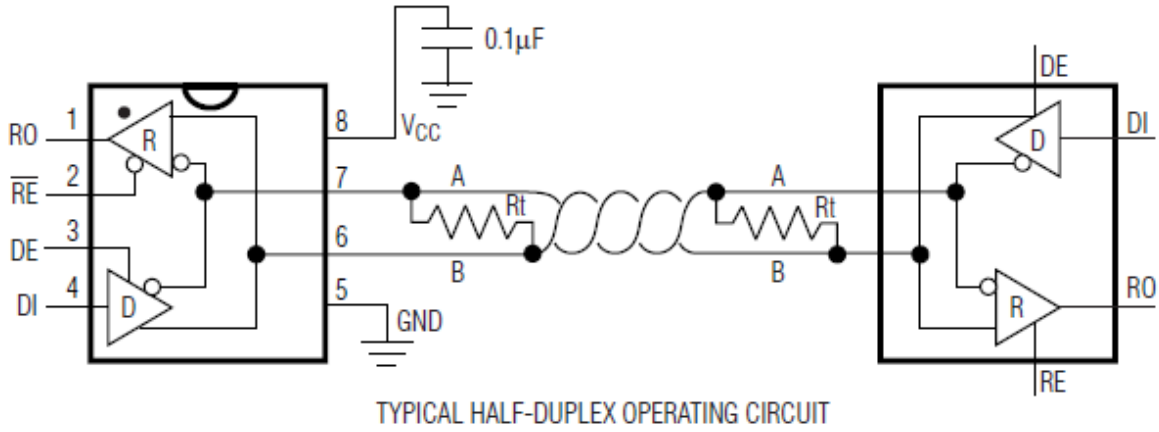
9 SVILUPPO GESTIONE DLFB

La comunicazione tra il rivelatore lineare di fumo e il dispositivo di controllo TLC è di tipo seriale asincrona, ed avviene tramite interfaccia SCI interna al microcontrollore che viene riportata su un'interfaccia fisica di tipo RS485 (half-duplex).

I parametri impostabili della comunicazione sono:

- Baud rate: impostato a 9600 baud;
- Num. Bit: impostato a 8 bit;
- Parity Bit: impostato a N, nessuna parità;
- Stop Bit: impostato a 1.

Il segnale in uscita viene trasformato in un segnale differenziale, come previsto dallo standar di comunicazione RS485. Questa particolare tipologia di interfaccia ha la caratteristica di essere molto resistente ai possibili disturbi che possono interferire con il segnale sulla linea. Infatti, se si verifica un disturbo, esso inciderà in modo uguale su entrambe le linee di dato, e quindi al ricevitore non verrà rilevato l'errore.



TYPICAL HALF-DUPLEX OPERATING CIRCUIT

9.1 Protocollo di comunicazione

La struttura dei dati prevista dal protocollo fornitoci dal produttore è la seguente:

	Significato
Byte 1	XON
Byte 2	Indirizzo
Byte 3	Dimensione
Byte 4	Comando
Byte 5 … Byte n	Dati
Byte n+1	LRC
Byte n+2	XOFF

XON: Indicatore di inizio messaggo (0x11)

Indirizzo: Indirizzo del dispositivo slave (DLFB=0)

Dimensione: Dimensione del messaggio da XON e XOFF compresi

Comando (ID): Identificativo del messaggio

Dati: Dati e parametri del messagio

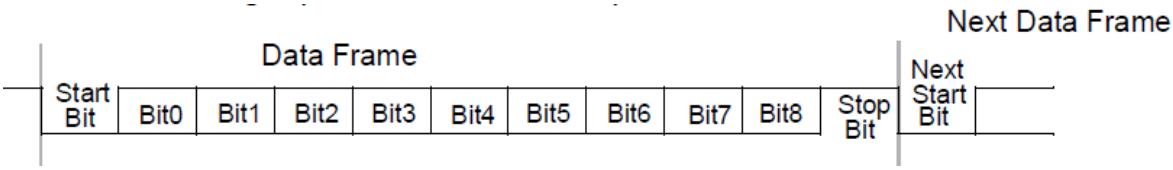
LRC: (Longitudinal Redundancy Check) Byte di controllo su Indirizzo, Dimensione, Comando e Dati. Si calcola nel seguente modo:

LRC=(Byte2) XOR (Byte3) XOR … (Byte n) XOR (0xFF)

XOR = or esclusivo

XOFF: Indicatore di fine messaggio (0x13)

I dati sono inviati con codifica Big Endian (bit più significativo per primo).
L'SCI è gestita dal microcontrollore attraverso routine di interrupt e prevede la seguente struttura di dati trasmessi e ricevuti:



Nel protocollo fornito dall'azienda produttrice dei sensori DLFB sono stati riservati 3 comandi (ID) per la comunicazione con il TLC:

ID (Hex)	Tipo	Interrogazione		Risposta	
		Dati	Dim	Dati	Dim
P (0x50)	Polling	Keys (2)	2	CmdSel (1) CmdVal (2) CmdParam (2) Text (32)	37
S (0x53)	Selezione lingua	Langue (1)	1	Langue (1)	1
E (0x45)	Risposta d'errore	-	-	E R ErrType	3

9.2 Sviluppo DLFB sul dispositivo TLC

Il master TLC interroga lo slave DLFB ogni 500ms per poter rilevare al meglio la pressione dei tasti (temporizzazione imposta dal protocollo). Il tempo di risposta del sensore dichiarato dal costruttore dipende dalla richiesta: varia da un minimo di 100µs ad un massimo di 2s.

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   AS1_InterruptRx (bean AsynchroSerial)
*
*    \detail  The method services the receive interrupt of the selected peripheral(s)
*             and eventually invokes the bean's event(s). This method is internal.
*/
////////////////////////////////////////////////////////////////////////////////
ISR(AS1_InterruptRx){
  byte tmp;

  if(flag_SCI1==RX_SCI){
    if(!active){
      (void)SCI1S1;
      tmp=SCI1D;
      if(tmp==XON){
        active=TRUE;
        inpBufDLFB[rx_count]=tmp;
        rx_count=1;
        tim02.t.timTTL=TIMERXOK;             //0.6 s
        giaConnesso=TRUE;
      }
      SCI1C2_RIE=TRUE;
    }else{
      // Reset interrupt request flag
      (void)SCI1S1;
      // Save received char to the receive buffer
      inpBufDLFB[rx_count]  = SCI1D;
      // Increse number of chars in receive buffer
      rx_count++;

      if(rx_count==3){  // check for message lenght reception
        maxLong=inpBufDLFB[2];            //size: 43 --> message input
                                          //size: 9  --> message error
                                          //size: 7  --> message language
        if (maxLong>AS1_INP_BUF_SIZE){
          SCI1C2_RIE=FALSE;
          flag_SCI1=NO_ACTIVE;
          retCode_SCI=ERROR_RX;
          rx_count=0;
          active=FALSE;
        }
      }else if(rx_count==maxLong){
        SCI1C2_RIE=FALSE;
        flag_SCI1=END_RX;
        retCode_SCI=OK_RX;
        rx_count=0;
        active=FALSE;
      }else  SCI1C2_RIE=TRUE;
    }

  } else {
    (void)SCI1S1;
    SCI1C2_RIE=FALSE;
    flag_SCI1=NO_ACTIVE;
    retCode_SCI=ERROR_RX;
    rx_count=0;
    active=FALSE;
    flag_DLFB=FALSE;
  }
}
```

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   AS1_InterruptTx (bean AsynchroSerial)
*
*    \detail  The method services the receive interrupt of the selected peripheral(s)
*             and eventually invokes the bean's event(s). This method is internal.
*/
////////////////////////////////////////////////////////////////////////////////
ISR(AS1_InterruptTx){
  byte tmp;

  if(flag_SCI1==TX_SCI){
    if(tx_count < sizeTx) {
      (void)SCI1S1;                      // Reset interrupt request flag
      SCI1D = outBufDLFB[tx_count];      // Store char to transmitter register
      tx_count++;
    } else {
      SCI1C2_TCIE=FALSE;
      (void)SCI1S1;
      tmp = SCI1D;           // Reset interrupt request flag
      SCI1C2_RIE=TRUE;
      //if(tim02.t.timOnlineDLFB!=0)  tim02.t.timTTL=TIMEMAX;   //2,2 sec
      //else  tim02.t.timTTL=TIMEMAX12S;   //12 sec
      flag_SCI1=RX_SCI;
      tx_count=0;
      rx_count=0;
      PTFD_PTFD3 = 0;
    }
  } else {
      (void)SCI1S1;
      SCI1C2_TCIE=FALSE;
      tx_count=0;
      flag_SCI1=NO_ACTIVE;
      retCode_SCI=ERROR_TX;
      flag_DLFB=FALSE;
    }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   AS1_InterruptError
*
*    \detail  Is invoked if an error occours on SC11. Increment the specific error
*             counter and reset the interrupt error flag
*/
////////////////////////////////////////////////////////////////////////////////
ISR(AS1_InterruptError){
  byte StatReg = getReg(SCI1S1);
  (void)SCI1D;                    // Dummy read of data register - clear error bits
  // Is an error detected?
  if(StatReg & (SCI1S1_OR_MASK|SCI1S1_NF_MASK|SCI1S1_FE_MASK|SCI1S1_PF_MASK))
    err_SCI1++;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   SCI1 transmission management
*
*    \detail  This function manages transmit and receive communication with SCI
*             interface.
*/
////////////////////////////////////////////////////////////////////////////////
```

```
void SCI1_TX(void){
  byte ii;

  switch(flag_SCI1){
    case RQS_TX:  //transmittion is needed
                  for(ii=0; ii<AS1_INP_BUF_SIZE; ii++)  inpBufDLFB[ii]=0;
                  PTFD_PTFD3 = 1;
                  tim02.t.timTTL=TIMEMAX;   //2,2s
                  tx_count=0;
                  sizeTx=outBufDLFB[2];
                  flag_SCI1=TX_SCI;
                  DisableInterrupts;
                  (void)SCI1S1;            // Reset interrupt request flag
                  SCI1C2_TCIE=TRUE;
                  EnableInterrupts;
                  break;
    case TX_SCI:  if(tim02.t.timTTL==0){
                    //Abort
                    SCI1C2_TCIE=FALSE;
                    flag_SCI1=NO_ACTIVE;
                    if(giaConnesso) retry++;
                    else  retry=NUMRETRY+1;
                    retCode_SCI=ERROR_TX;
                    flag_DLFB=FALSE;
                    busyLed_status=OFF;
                  }
                  break;
    case RX_SCI:  if(tim02.t.timTTL==0){
                    //Abort
                    SCI1C2_RIE=FALSE;
                    flag_SCI1=NO_ACTIVE;
                    retCode_SCI=ERROR_RX;
                    flag_DLFB=FALSE;
                    if(giaConnesso) retry++;
                    else  retry=NUMRETRY+1;
                    busyLed_status=OFF;
                  }
                  break;
  }
}
```

Il DLFB può quindi controllare:

- Suono dei tasti: può accendere/spegnere il suono quando i tasti vengono premuti;
- Buzzer: può attivare/silenziare il buzzer e, se è attivato, può scegliere se fargli eseguire 1 suono breve, 2 suoni brevi, 3 suoni lunghi o un suono continuo;
- Retroilluminazione: può attivare/disattivare la retroilluminazione e, se è attivata, può scegliere se mantenerla accesa sempre o solo per un breve periodo (20 sec);
- Led: può accendere/spegnere o far lampeggiare tutti e 3 i led (rosso, verde e giallo);
- Display: può mandare al terminale fino a 32 caratteri che saranno visualizzati sul display del TLC. Può inoltre far lampeggiare la prima riga, la seconda o entrambe.

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   DLFB communication management
*
*    \detail  This function sets data output to smoke detector and check if data
*             receicived from device are correct.
*/
////////////////////////////////////////////////////////////////////////////////*/
```

```
void Manag_DLFB(){
  byte tmp=0;
  byte tmp1=0;
  byte tmp2=0;
  byte ii;
  if(flag_SCI1==NO_ACTIVE){
    for(ii=0; ii<AS1_OUT_BUF_SIZE; ii++)  outBufDLFB[ii]=0;
      //required transmittion
      if(setLanguage){

       … codice omesso per riservatezza …

      } else{

       … codice omesso per riservatezza …

        }
      flag_SCI1=RQS_TX;
  } else {
      if(flag_SCI1==END_RX){
        flag_SCI1=NO_ACTIVE;
        if(retCode_SCI==OK_RX){
          if(inpBufDLFB[inpBufDLFB[2]-1]==XOFF){
            tmp=LrcIn();
            if(tmp==inpBufDLFB[inpBufDLFB[2]-2]){
              switch(inpBufDLFB[3]){
                case DATA_OK:

                      … codice omesso per riservatezza …

                      break;

                case DATA_ERR:

                      … codice omesso per riservatezza …

                      break;

                case DATA_LAN:

                      … codice omesso per riservatezza …

                      break;

                default:
                      BadMessage();
                      break;
              }
            } else  BadMessage();
          } else  BadMessage();
        } else  BadMessage();
      }
    }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Decode phase
*
*    \detail  This function done the management of input data from smoke detector and
*             arrange the telecomande for execute command received.
*/
////////////////////////////////////////////////////////////////////////////////
```

```
void Decode(void){
  byte ii;
  byte tmp;
  char temp[BUF_DISP];

  flag_DLFB=FALSE;
  if(offline==TRUE){
    PutComDisp(CLEAR,0);
    offline=FALSE;
    SpecialChar(lingua|DLFB_SET);
  }

  //--------------------------------update text--------------------------------------
  if(inpBufDLFB[4] & 0b00000001){
    if(!display.busy){
      display.dBlink=FALSE;
      display.cursPos=0;
      display.cursOn=FALSE;
      for(ii=0; ii<BUF_DISP; ii++)  temp[ii]=inpBufDLFB[ii+9];
      DecodCharSpec(temp);
      if(inpBufDLFB[8] & 0b00000010){
        //blink on first line
        for(ii=0; ii<16; ii++)
          disp_buf_blink[ii]=TRUE;   //blink display buffer first line
        display.dBlink=TRUE;
      } else{
          //blink on first line
          for(ii=0; ii<16; ii++)  disp_buf_blink[ii]=FALSE;
        }
      if(inpBufDLFB[8] & 0b00000001){
        //blink on second line
        for(ii=16; ii<32; ii++) disp_buf_blink[ii]=TRUE;   //blink display buffer
        display.dBlink=TRUE;
      } else{
          //blink on first line
          for(ii=16; ii<32; ii++)  disp_buf_blink[ii]=FALSE;
        }
    }
  }

  //-----------------------------update leds status-----------------------------------
  if(inpBufDLFB[4] & 0b00000010){
    if(inpBufDLFB[6] & 0b00000001){
      //led1 on
      led1_status=ON;
      if(inpBufDLFB[8] & 0b000000100) led1_status=BLINK;  //led1 blink
    } else  led1_status=OFF;  //led1 off
    if(inpBufDLFB[6] & 0b00000010){
      //led2 on
      led2_status=ON;
      if(inpBufDLFB[8] & 0b00001000)  led2_status=BLINK;  //led1 blink
    } else  led2_status=OFF;  //led2 off
  if(inpBufDLFB[6] & 0b00000100){
      //led3 on
      led3_status=ON;
      if(inpBufDLFB[8] & 0b00010000)  led3_status=BLINK;  //led1 blink
    } else  led3_status=OFF;  //led3 off
  }

  //-----------------------------update back light-----------------------------------
  if(inpBufDLFB[4] & 0b00000100){
    if(inpBufDLFB[6] & 0b00001000){
      //back light on
```

```
      if(inpBufDLFB[8] & 0b00100000)  flag_backLight=TRUE;  //back light fix
      else{
        //back light short
        timsec.t.timBL=TEMP_BKLIGHT;
        flag_backLight=FALSE;
      }
    } else{
        //back light off
        flag_backLight=FALSE;
        timsec.t.timBL=0;
      }
  }

  //-----------------------------update buzzer state---------------------------------
  if(inpBufDLFB[4] & 0b00001000){
    if(inpBufDLFB[6] & 0b00010000){
      //buzzer on
      tmp=inpBufDLFB[8] & 0b11000000;
      if(tmp==0b00000000){
        //1 beep
        if(buzzer.State.status!=BEEP_1){
          buzzer.State.status=BEEP_1;
          countBeep=0;
        }
      } else{
          if(tmp==0b01000000){
            //2 beep
            if(buzzer.State.status!=BEEP_2){
              buzzer.State.status=BEEP_2;
              countBeep=0;
            }
          } else{
              if(tmp==0b10000000){
                //3 long_beep
                if(buzzer.State.status!=BEEP_3){
                  buzzer.State.status=BEEP_3;
                  countBeep=0;
                }
              } else {
                  if(tmp & 0b11000000){
                    //beep_on
                    buzzer.State.status=BEEP_ON;
                    buzzer.State.numbeep=0;
                  }
                }
            }
        }

    } else{
        //no_beep
        buzzer.State.status=NO_BEEP;
        buzzer.State.numbeep=0;
      }
  }

  //---------------------------update keyboar beep status-----------------------------
  if(inpBufDLFB[4] & 0b00010000){
    if(inpBufDLFB[6] & 0b00100000)
      enable_kbrd_beep=TRUE;  //keyboard beep status = on
    else  enable_kbrd_beep=FALSE; //keyboard beep status = off
  }
}
```

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Status of first part of keyboard
*
*    \detail  This function calculate the status of the first part of keypad in the
*             form expected from the DLFB.
*/
////////////////////////////////////////////////////////////////////////////////
byte Keys1(void){
  byte tmp1 = 0;
  byte tmp2 = 0;
  byte ii;

  for(ii=0; ii<NUM_COL; ii++){
    tmp1=tmp1 | ((keyb_col[ii].status & 0b0001)<<ii);
    tmp1=tmp1 | ((keyb_col[ii].status & 0b0010)<<(3+ii));
  }

  if((last_key>0 && last_key<7) || (last_key>10 && last_key<13)){
    tmp2 = tabKeysMask[last_key-1];
    last_key = 0;
  }
  return tmp1 | tmp2;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Status of second part of keyboard
*
*    \detail  This function calculate the status of the second part of keypad in the
*             form expected from the DLFB.
*/
////////////////////////////////////////////////////////////////////////////////
byte Keys2(void){
  byte tmp1 = 0;
  byte tmp2 = 0;
  byte ii;

   for(ii=0; ii<NUM_COL; ii++){

    if(ii<2){
      tmp2=tmp2 | ((byte)(keyb_col[ii].status & 0b0100)>>(2-ii));
      tmp2=tmp2 | ((keyb_col[ii].status & 0b1000)<<(1+ii));
    } else{
        tmp2=tmp2 | ((keyb_col[ii].status & 0b0100)<<(ii-2));
        tmp2=tmp2 | ((keyb_col[ii].status & 0b1000)<<(1+ii));
      }
  }

  if((last_key>6 && last_key<11) || (last_key>12 && last_key<17)){
    tmp1 = tabKeysMask[last_key-1];
    last_key = 0;
  }
  return tmp2 | tmp1;
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Decode of special characters
*
*    \detail  Set special characters code in display buffer
*/
////////////////////////////////////////////////////////////////////////////////
```

```
void DecodCharSpec(byte buffer[BUF_DISP]){
  byte ii;
  byte jj;
  bool preced=FALSE;

  for(ii=0; ii<BUF_DISP; ii++){
    if(preced){
      preced=FALSE;
      for(jj=0; jj<16; jj++){
        if(buffer[ii]==CONV_CHAR2[jj][0]){
          disp_buf[ii]=CONV_CHAR2[jj][1];
          preced=TRUE;
          jj=16;
        }
      }
    }
    for(jj=0; jj<N_PATTERNS; jj++){
      if(buffer[ii]==CONV_CHAR[jj][0]){
        disp_buf[ii]=CONV_CHAR[jj][1];
        preced=TRUE;
        jj=N_PATTERNS;
      }
    }
    if(!preced) disp_buf[ii]=buffer[ii];
  }
  display.refresh=TRUE;
}
```

9.3 Sviluppo software di simulazione

Il simulatore realizzato doveva rispecchiare fedelmente il comportamento del sensore. Sono quindi stati presi in considerazione tutti gli aspetti del protocollo citati sopra (Cap. 9.1) ricreando tutte le possibili modalità operative del sensore, comprese quelle di errore.

```
Public Sub Main()
'-----------------------------------------------------
' DATA TLC_Test
'-----------------------------------------------------
    With Form1

inizio:
        On Error GoTo Gerr

       ' Carica default
        GetSet
        strs$ = nomeProgramma + "    Delta Erre Safe" ' MLS-> "  -   Delta Erre Safe "
        Form1.Caption = strs$
        strs$ = "TLC_Test" ' MLS-> " TLC_Test     "
        strs$ = strs$ + "     V" + versioneProgramma + " " + dataProgramma ' MLS-> " V "
        .Label8.Caption = strs$
        OpnCom

        tmp = False
        ii = 1
        vv = 0
        errXon = 0
        errAdd = 0
        errSize = 0
        errId = 0
        errLrc = 0
```

```
        errXoff = 0
        First = 0
        tot = &HFF
        flagErr = False

    ' Attende comando o dati
        Do
waicm:      fg0D = False
            Do
                If Form1.MSComm1.PortOpen = False Then
                    OpnCom
                End If
                tmh$ = ""
                tma$ = ""
                Form1.Text3.Text = errXon
                Form1.Text4.Text = errAdd
                Form1.Text5.Text = errSize
                Form1.Text6.Text = errId
                Form1.Text7.Text = errLrc
                Form1.Text8.Text = errXoff

                DoEvents
                If Form1.MSComm1.InBufferCount > 0 And ii = 1 Then
                    tmp = True
                    TrigTimer
                    DoEvents
                    ncar% = Form1.MSComm1.InBufferCount
                    RxBin() = Form1.MSComm1.Input
                    For jj = 1 To ncar%
                        vv = vv + 1
                        tmh$ = ""
                        tma$ = ""
                        aa$ = Chr$(RxBin(jj - 1))
                        valc = Asc(aa$)
                        tot = tot Xor Asc(aa$)
                        TrigTimer
                        Select Case vv
                            Case 1
                                tot = tot Xor Asc(aa$)
                                If valc <> XON Then
                                    jj = ncar%
                                    flagErr = True
                                    errXon = errXon + 1
                                End If
                            Case 2
                                If valc <> ADD Then
                                    jj = ncar%
                                    flagErr = True
                                    errAdd = errAdd + 1
                                End If
                            Case 3
                                If valc <> 8 Then
                                    If valc <> 7 Then
                                        jj = ncar%
                                        flagErr = True
                                        errSize = errSize + 1
                                    Else
                                        setLingua = True
                                    End If
                                End If
                            Case 4
                                If setLingua = False Then
                                    If valc <> ID Then
```

```
                        jj = ncar%
                        flagErr = True
                        errId = errId + 1
                    End If
                Else
                    If valc <> ID_LING Then
                        jj = ncar%
                        flagErr = True
                        errId = errId + 1
                    End If
                End If
            Case 5
                If setLingua = False Then
                    Tasti2 aa$
                Else
                    impLingua = valc
                    VisLingua (impLingua)
                End If
            Case 6
                If setLingua = False Then
                    Tasti1 aa$
                Else
                    tot = tot Xor Asc(aa$)
                    If valc <> tot Then
                        jj = ncar%
                        flagErr = True
                        errLrc = errLrc + 1
                    Else
                        Form1.Label6.Caption = "LRC : " + Hex$(tot)
                    End If
                End If
            Case 7
                If setLingua = False Then
                    tot = tot Xor Asc(aa$)
                    If valc <> tot Then
                        jj = ncar%
                        flagErr = True
                        errLrc = errLrc + 1
                    Else
                        Form1.Label6.Caption = "LRC : " + Hex$(tot)
                    End If
                Else
                    If valc <> XOFF Then
                        jj = ncar%
                        flagErr = True
                        errXoff = errXoff + 1
                    End If
                End If
            Case 8
                If setLingua = False Then
                    If valc <> XOFF Then
                        jj = ncar%
                        flagErr = True
                        errXoff = errXoff + 1
                    End If
                End If
        End Select
        tmh$ = Right$("00" + LTrim$(Hex$(Asc(aa$))), 2) + " "
        inmsH$ = inmsH$ + tmh$
        DoEvents
        Select Case valc
        Case &HD
            fg0D = True
```

```
                    Case &HA
                        If fg0D = True Then
                            fg0D = False
                            inmsH$ = inmsH$
                            OffTimer
                        End If
                    Case Else
                        fg0D = False
                    End Select
                Next jj
                If nTimeout_RTx > 0 Then
                    nTimeout_RTx = nTimeout_RTx - 1
                    inmsH$ = inmsH$
                End If
                lenj = Len(inmsH$)
                If lenj > 27 Then lenj = 27
                strtx$ = String(28 - lenj, " ")
                PrintS inmsH$, 1
                For tt = 0 To 6
                    If Form1.Check18(tt).Value = 1 Then
                        flagErr = True
                    End If
                Next tt
            End If
            DoEvents
            If nTimeout_RTx > 0 Then
                nTimeout_RTx = nTimeout_RTx - 1
                'inmsH$ = inmsH$ + vbCrLf
            End If
            If tmp = True Then
                If ii = 1 Then
                    If flagErr = False Then
                        If setLingua = False Then
                            Conv
                        Else
                            ConvLing
                        End If
                    Else
                        ConvErr
                    End If
                    Pausa (0.1)
                End If
                If flagErr = False Then
                    If setLingua = False Then
                        If ii < 10 Or ii > 41 Then
                            strtx$ = strtx$ + Right$("00" +
                                     LTrim$(Hex$(Asc(Chr(Tx(ii - 1))))), 2) + " "
                            Form1.MSComm1.Output = Chr(Tx(ii - 1)) 'invia messaggio
                        Else
                            str1 = Form1.Text1.Text
                            str2 = Form1.Text2.Text
                            If Len(str1) < 16 Then
                                For zz = Len(str1) To 15
                                    str1 = str1 + " "
                                Next zz
                            End If
                            If Len(str2) < 16 Then
                                For zz = Len(str2) To 15
                                    str2 = str2 + " "
                                Next zz
                            End If
                            For zz = 1 To Len(str1)
```

```
                                strtx$ = strtx$ + Right$("00" +
                                         LTrim$(Hex$(Asc(Mid(str1, zz, 1)))), 2) + " "
                                Next zz
                                For zz = 1 To Len(str2)
                                  strtx$ = strtx$ + Right$("00" +
                                         LTrim$(Hex$(Asc(Mid(str2, zz, 1)))), 2) + " "
                                Next zz
                                Form1.MSComm1.Output = str1
                                Form1.MSComm1.Output = str2
                                ii = 41
                            End If
                        Else
                            Form1.MSComm1.Output = Chr(Tx(ii - 1)) 'invia messaggio
                            strtx$ = strtx$ + Right$("00" + LTrim$(Hex$(Asc(Chr(Tx(ii -
                                     1))))), 2) + " "
                        End If
                    Else
                        Form1.MSComm1.Output = Chr(Tx(ii - 1)) 'invia messaggio
                        strtx$ = strtx$ + Right$("00" + LTrim$(Hex$(Asc(Chr(Tx(ii -
                                 1))))), 2) + " "
                    End If
                    ii = ii + 1
                    If setLingua = True And flagErr = False Then
                        If ii = 8 Then
                            tmp = False
                            flagErr = False
                            setLingua = False
                            ii = 1
                            vv = 0
                            tot = &HFF
                            strtx$ = strtx$ + vbCrLf
                            PrintS strtx$, 2
                        End If
                    Else
                        If ii = 44 Or (ii = 10 And flagErr = True) Then
                            tmp = False
                            flagErr = False
                            setLingua = False
                            ii = 1
                            vv = 0
                            tot = &HFF
                            strtx$ = strtx$ + vbCrLf
                            PrintS strtx$, 2
                        End If
                    End If
                End If
            Loop
        Loop

    ' subroutine gestione errori di sistema
Gerr:       GestErr
            Resume Next
     End With
End Sub

Sub Tasti1(tasto$)
    tas = Asc(tasto$)
    If tas And T1 Then
        Form1.Shape1.BackColor = &HFF00&
    Else
        Form1.Shape1.BackColor = &HFFFFFF
    End If
    If tas And T2 Then
```

```
        Form1.Shape2.BackColor = &HFF00&
    Else
        Form1.Shape2.BackColor = &HFFFFFF
    End If
    If tas And T3 Then
        Form1.Shape3.BackColor = &HFF00&
    Else
        Form1.Shape3.BackColor = &HFFFFFF
    End If
    If tas And T4 Then
        Form1.Shape11.BackColor = &HFF00&
    Else
        Form1.Shape11.BackColor = &HFFFFFF
    End If
    If tas And T5 Then
        Form1.Shape4.BackColor = &HFF00&
    Else
        Form1.Shape4.BackColor = &HFFFFFF
    End If
    If tas And T6 Then
        Form1.Shape5.BackColor = &HFF00&
    Else
        Form1.Shape5.BackColor = &HFFFFFF
    End If
    If tas And T7 Then
        Form1.Shape6.BackColor = &HFF00&
    Else
        Form1.Shape6.BackColor = &HFFFFFF
    End If
    If tas And T8 Then
        Form1.Shape12.BackColor = &HFF00&
    Else
        Form1.Shape12.BackColor = &HFFFFFF
    End If
End Sub

Sub Tasti2(tasto$)
    tas = Asc(tasto$)
    If tas And T1 Then
        Form1.Shape7.BackColor = &HFF00&
    Else
        Form1.Shape7.BackColor = &HFFFFFF
    End If
    If tas And T2 Then
        Form1.Shape8.BackColor = &HFF00&
    Else
        Form1.Shape8.BackColor = &HFFFFFF
    End If
    If tas And T3 Then
        Form1.Shape9.BackColor = &HFF00&
    Else
        Form1.Shape9.BackColor = &HFFFFFF
    End If
    If tas And T4 Then
        Form1.Shape13.BackColor = &HFF00&
    Else
        Form1.Shape13.BackColor = &HFFFFFF
    End If
    If tas And T5 Then
        Form1.Shape14.BackColor = &HFF00&
    Else
        Form1.Shape14.BackColor = &HFFFFFF
    End If
```

```
    If tas And T6 Then
        Form1.Shape10.BackColor = &HFF00&
    Else
        Form1.Shape10.BackColor = &HFFFFFF
    End If
    If tas And T7 Then
        Form1.Shape15.BackColor = &HFF00&
    Else
        Form1.Shape15.BackColor = &HFFFFFF
    End If
    If tas And T8 Then
        Form1.Shape16.BackColor = &HFF00&
    Else
        Form1.Shape16.BackColor = &HFFFFFF
    End If
End Sub

Sub Conv()
    kk = 0
    tt = &HFF

    Tx(kk) = XON
    kk = kk + 1

    Tx(kk) = ADD
    tt = tt Xor ADD
    kk = kk + 1

    Tx(kk) = SIZE
    tt = tt Xor SIZE
    kk = kk + 1

    Tx(kk) = ID
    tt = tt Xor ID
    kk = kk + 1

    'set comand selection
    tmp4 = 0
    If Form1.Check1.Value = Checked Then
        tmp4 = tmp4 + 1
        Form1.Check1.Value = Unchecked
    End If

    If Form1.Check2.Value = Checked Then
        tmp4 = tmp4 + 2
        Form1.Check2.Value = Unchecked
    End If

    If Form1.Check3.Value = Checked Then
        tmp4 = tmp4 + 4
        Form1.Check3.Value = Unchecked
    End If

    If Form1.Check4.Value = Checked Then
        tmp4 = tmp4 + 8
        Form1.Check4.Value = Unchecked
    End If

    If Form1.Check5.Value = Checked Then
        tmp4 = tmp4 + 16
        Form1.Check5.Value = Unchecked
    End If
```

```
    Tx(kk) = tmp4
    tt = tt Xor tmp4
    kk = kk + 1

    'set comand value
    Tx(kk) = 0
    kk = kk + 1

    tmp6 = 0
    If Form1.Check6.Value = Checked Then
        tmp6 = tmp6 + 1
    End If

    If Form1.Check7.Value = Checked Then
        tmp6 = tmp6 + 2
    End If

    If Form1.Check8.Value = Checked Then
        tmp6 = tmp6 + 4
    End If

    If Form1.Check9.Value = Checked Then
        tmp6 = tmp6 + 8
    End If

    If Form1.Check10.Value = Checked Then
        tmp6 = tmp6 + 16
    End If

    If Form1.Check11.Value = Checked Then
        tmp6 = tmp6 + 32
    End If

    Tx(kk) = tmp6
    tt = tt Xor tmp6
    kk = kk + 1

    'set comand parameter
    Tx(kk) = 0
    kk = kk + 1

    tmp8 = 0
    If Form1.Check12.Value = Checked Then
        tmp8 = tmp8 + 2
    End If

    If Form1.Check13.Value = Checked Then
        tmp8 = tmp8 + 1
    End If

    If Form1.Check14.Value = Checked Then
        tmp8 = tmp8 + 4
    End If

    If Form1.Check15.Value = Checked Then
        tmp8 = tmp8 + 8
    End If

    If Form1.Check16.Value = Checked Then
        tmp8 = tmp8 + 16
    End If

    If Form1.Check17.Value = Unchecked Then
```

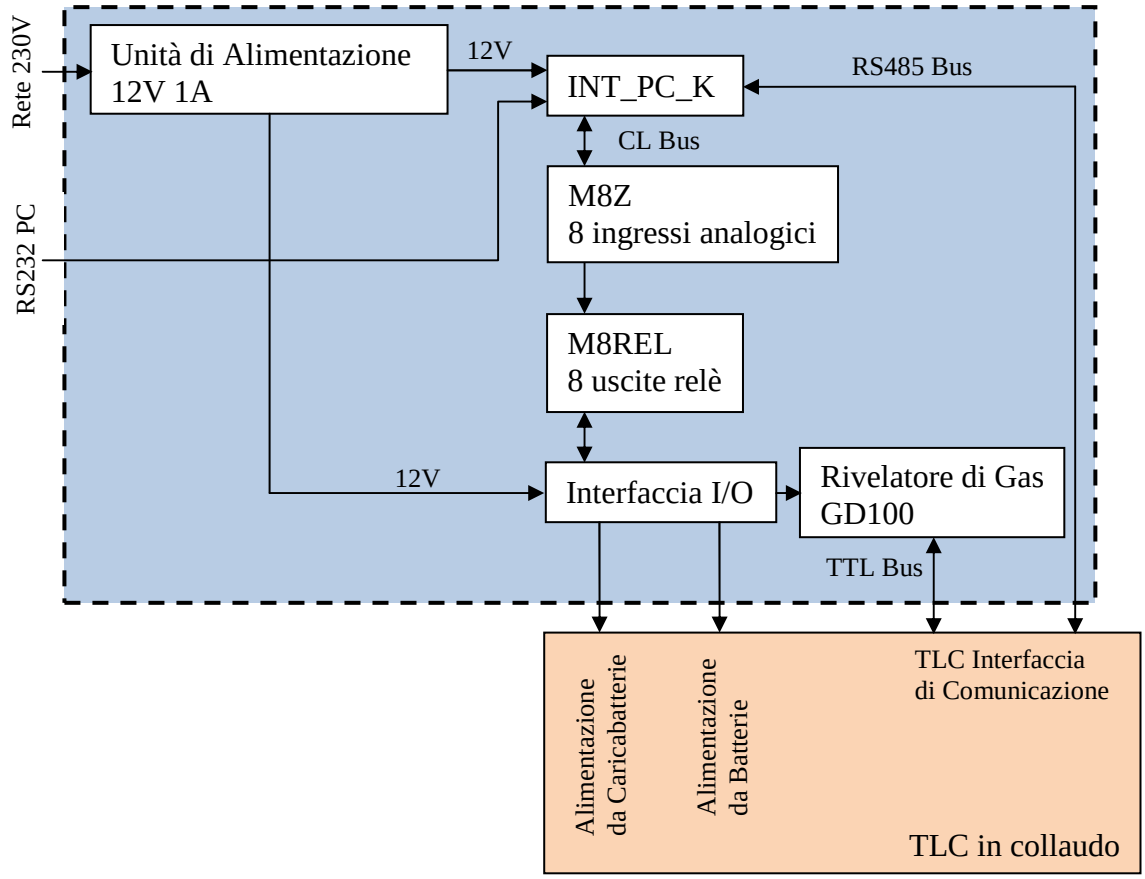
```
        tmp8 = tmp8 + 32
    End If

    If Form1.Option2.Value = True Then
        tmp8 = tmp8 + 64
    Else
        If Form1.Option3.Value = True Then
            tmp8 = tmp8 + 128
        Else
            If Form1.Option4.Value = True Then
                tmp8 = tmp8 + 192
            End If
        End If
    End If

    Tx(kk) = tmp8
    tt = tt Xor tmp8
    kk = kk + 1

    'set LRC
    kk = 41
    If Form1.Check19.Value = 1 Then
        varTmp = "01" + Chr(&HA0) + "23" + Chr(&HA1) + "45" + Chr(&HA2) + "67" +
                 Chr(&HA3) + "89" + Chr(&HA4)
        Form1.Text1.Text = varTmp
        varTmp = "AB" + Chr(&HA5) + "CD" + Chr(&HA6) + "EF" + Chr(&HA7)
        Form1.Text2.Text = varTmp
    End If

    varTmp = Form1.Text1.Text
    If Len(varTmp) < 16 Then
        For zz = Len(varTmp) To 15
            varTmp = varTmp + " "
        Next zz
    End If
    For ii = 1 To Len(varTmp)
        tt = tt Xor Asc(Mid(varTmp, ii, 1))
    Next ii

    varTmp = Form1.Text2.Text
    If Len(varTmp) < 16 Then
        For zz = Len(varTmp) To 15
            varTmp = varTmp + " "
        Next zz
    End If

    For ii = 1 To Len(varTmp)
        tt = tt Xor Asc(Mid(varTmp, ii, 1))
    Next ii

    Form1.Label9.Caption = "LRC: " + Hex$(tt)
    Tx(kk) = tt
    kk = kk + 1
    Tx(kk) = XOFF
End Sub

Sub ConvErr()
    erroreSend = 0
    kk = 0
    tt = &HFF

    Tx(kk) = XON
    kk = kk + 1
```

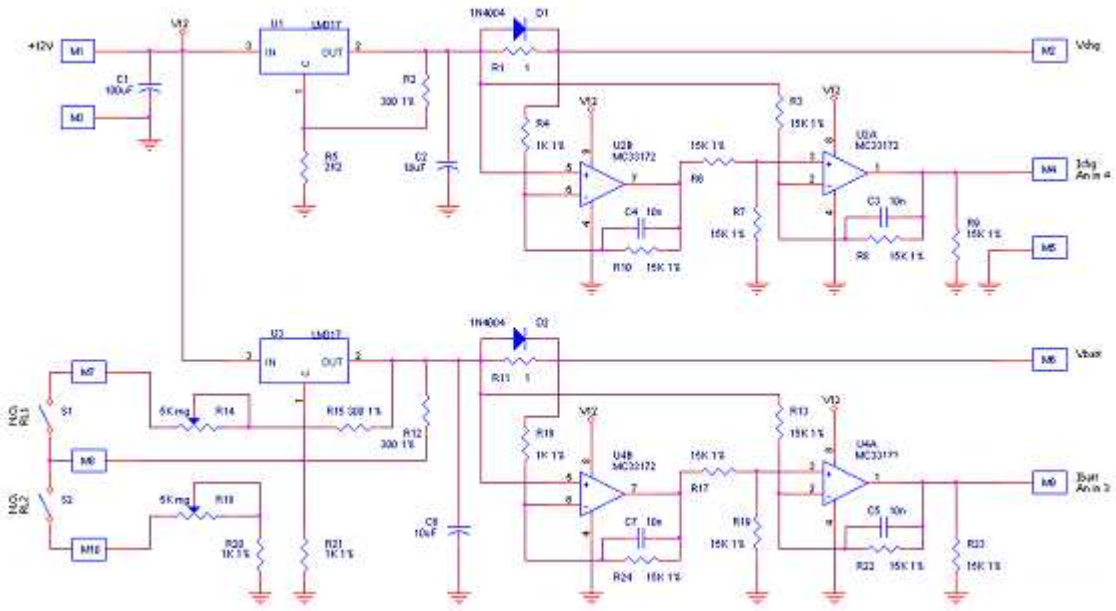
```
    Tx(kk) = ADD
    tt = tt Xor ADD
    kk = kk + 1
    Tx(kk) = SIZEERR
    tt = tt Xor SIZEERR
    kk = kk + 1
    Tx(kk) = ID_ERR
    tt = tt Xor ID_ERR
    kk = kk + 1
    Tx(kk) = E
    tt = tt Xor E
    kk = kk + 1
    Tx(kk) = R
    tt = tt Xor R
    kk = kk + 1

    If Form1.Check18(0).Value = 1 Then
        erroreSend = TYPEERR1
        'Form1.Check18(0).Value = 0
    Else
        If Form1.Check18(1).Value = 1 Then
            erroreSend = TYPEERR2
            Form1.Check18(1).Value = 0
        Else
            If Form1.Check18(2).Value = 1 Then
                erroreSend = TYPEERR3
                Form1.Check18(2).Value = 0
            Else
                If Form1.Check18(3).Value = 1 Then
                    erroreSend = TYPEERR4
                    Form1.Check18(3).Value = 0
                Else
                    If Form1.Check18(4).Value = 1 Then
                        erroreSend = TYPEERR5
                        Form1.Check18(4).Value = 0
                    Else
                        If Form1.Check18(5).Value = 1 Then
                            erroreSend = TYPEERR6
                            Form1.Check18(5).Value = 0
                        Else
                            If Form1.Check18(6).Value = 1 Then
                                erroreSend = TYPEERR7
                                Form1.Check18(6).Value = 0
                            Else
                                erroreSend = 0
                            End If
                        End If
                    End If
                End If
            End If
        End If
    End If

    Tx(kk) = erroreSend
    tt = tt Xor erroreSend
    kk = kk + 1

    Form1.Label9.Caption = "LRC: " + Hex$(tt)
    Tx(kk) = tt
    kk = kk + 1

    Tx(kk) = XOFF
End Sub
```

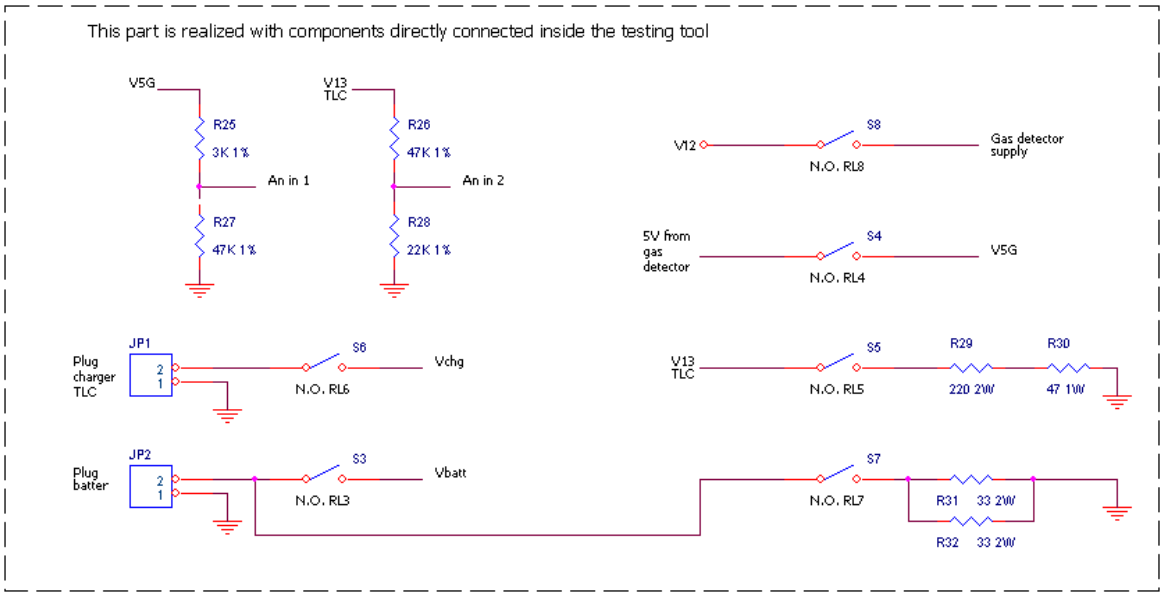
```
Sub ConvLing()
    kk = 0
    tt = &HFF

    Tx(kk) = XON
    kk = kk + 1
    Tx(kk) = ADD
    tt = tt Xor ADD
    kk = kk + 1
    Tx(kk) = SIZELING
    tt = tt Xor SIZELING
    kk = kk + 1
    Tx(kk) = ID_LING
    tt = tt Xor ID_LING
    kk = kk + 1
    Tx(kk) = impLingua
    tt = tt Xor impLingua
    kk = kk + 1

    Form1.Label9.Caption = "LRC: " + Hex$(tt)
    Tx(kk) = tt
    kk = kk + 1
    Tx(kk) = XOFF
End Sub

Sub VisLingua(lingua)
    Form1.Check20.Value = 2
    Select Case lingua
        Case 1
            Form1.Check20.Caption = Left(Form1.Check20.Caption, 16) + "  -->  Francese"
        Case 2
            Form1.Check20.Caption = Left(Form1.Check20.Caption, 16) + "  -->  Inglese"
        Case 3
            Form1.Check20.Caption = Left(Form1.Check20.Caption, 16) + "  -->  Italiano"
        Case 4
            Form1.Check20.Caption = Left(Form1.Check20.Caption, 16) + "  -->  Tedesco"
        Case 5
            Form1.Check20.Caption = Left(Form1.Check20.Caption, 16) + "  -->  Olandese"
        Case Else
            Form1.Check20.Caption = Left(Form1.Check20.Caption, 16) + "  -->  ERRORE!!"
            Form1.Check20.Value = 0
    End Select
End Sub
```

10 SVILUPPO COLLAUDO DEL TERMINALE

La comunicazione tra il TLC e il PC passa attraverso il banco di collaudo, che provvede ad inviare i comandi provenienti dall'interfaccia RS485 sulla SCI.







10.1 Sviluppo modalità Service su TLC

Sul terminale TLC è stata sviluppata una particolare modalità operativa di Service, che permette al dispositivo di eseguire operazioni specifiche di controllo degli ingressi e delle uscite. L'attivazione di questo particolare stato è vincolata e controllata, in modo che non sia possibile accedere per errore.

```
////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Management of Service mode
*
*    \detail  This function manage the main phases in service mode
*/
////////////////////////////////////////////////////////////////////////////////
void ServiceMode(void){
  byte addServ;
  byte dataServ;
  byte ii;

  switch(countService){
    case 0: for(ii=0; ii<NUM_COL; ii++) key_col_service[ii]=0;
            setLanguage=TRUE;
            tim02.t.timVisInit=3;
            countService++;
            break;

    case 1: if(tim02.t.timVisInit==0){
              MyPrint(MODSERVICE, 0, BUF_DISP);
              countService++;
            }
            if(last_key==15){
              last_key=0;
              fase=OPZIONI;
              display.cursOn=FALSE;
              display.cursPos=22;
              opzioni[ADDSERVICE]=ADDSERVICE;    // off service mode
            }
            break;
```

```
    case 2: Comm_Serv();
            break;

    case END_SERVICE: //write NOSERVICE e GIATEST bytes in E2PROM parameter
                      addServ=ADDSERVICE;
                      dataServ=ADDSERVICE;

                      if (IICSW_write_byte(addServ, &dataServ)==_OK){
                        opzioni[ADDSERVICE]=ADDSERVICE; // off service mode
                        dataServ=ADDGIATEST;

                        if (IICSW_write_byte(ADDGIATEST, &dataServ)==_OK){
                          opzioni[ADDGIATEST]=ADDGIATEST;
                          Reset_init();    // Reinitialise hardware and variables
                        }

                      }
                      countService=0;
                      break;
  }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Communication in service mode
*
*    \detail  This function sets data output to pc in service mode and check if data
*             receicived from device are correct.
*/
////////////////////////////////////////////////////////////////////////////////*/
void Comm_Serv(){
  byte tmp=0;
  byte tmp1=0;
  byte tmp2=0;
  byte datoOut;
  byte ii;

  if(flag_SCI1==NO_ACTIVE)  flag_SCI1=RX_SCI;

  else{
    if(flag_SCI1==END_RX){
      timsec.t.timPwON=TEM_PWON*2;   //reload power-on time
      if(retCode_SCI==OK_RX){
        if(inpBufDLFB[inpBufDLFB[2]-1]==XOFF){

          tmp=LrcIn();
          if(tmp==inpBufDLFB[inpBufDLFB[2]-2]){
            if(inpBufDLFB[2]==7){
              if(inpBufDLFB[3]==DATA_OK){
                //end good transittion
                countPausa++;

                if(inpBufDLFB[4]==GAS && !firstPass){
                  if(primaVolta && flag_trx_gd1xx==0){
                    flag_trx_gd1xx=NEED_TRX;
                    id_Gd1=READ;
                    add_Gd1=250;
                    primaVolta=FALSE;

                  } else{
                      if(secondaVolta && flag_trx_gd1xx==0){
                        tipoSens=inpBufGD1xx[1];
                        flag_trx_gd1xx=NEED_TRX;
```

```
                        id_Gd1=READ;
                        add_Gd1=19;
                        secondaVolta=FALSE;

                      } else{
                          if(flag_trx_gd1xx==0){
                            if(countPausa>CONST_PAUSA){
                              primaVolta=TRUE;
                              secondaVolta=TRUE;
                              countPausa=0;
                              datoOut=inpBufGD1xx[1];
                              for(ii=0; ii<AS1_OUT_BUF_SIZE; ii++)  outBufDLFB[ii]=0;

                              //required transmittion

                              … codice omesso per riservatezza …

                            }
                          }
                      }
                  } else{

                      if(countPausa>CONST_PAUSA){
                        countPausa=0;
                        if(!firstPass) datoOut=FaseService(inpBufDLFB[4]);
                        else datoOut=FaseService(PASS);
                        for(ii=0; ii<AS1_OUT_BUF_SIZE; ii++)  outBufDLFB[ii]=0;

                        //required transmittion

                        … codice omesso per riservatezza …

                    }
                } else  Bad_Comm();
              } else{
                  Bad_Comm();
                }
            } else  Bad_Comm();
          } else  Bad_Comm();
        } else  Bad_Comm();
    }
  }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   Decoding of input data
*
*    \detail  This function decide what data must be send out according to hardware
*             test required.
*/
////////////////////////////////////////////////////////////////////////////////
byte FaseService(byte phase){
  byte dataCodPro;
  byte ii;
  busyLed_status=OFF;
  led1_status=OFF;
  led2_status=OFF;
  led3_status=OFF;
  secondByte=0;

  if(phase!=TESTKEY)  first_key=TRUE;
```

```
switch(phase){

  case GIATEST: return opzioni[ADDGIATEST];
  case VERS:  secondByte=deviceVers[11]-0x30;
              return deviceVers[9]-0x30;
  case CKSUM: //numWords = ((addFine-addInizio)/numByteInWord) - 1
              //           (( 0xFFFF - 0x8000 ) /    2    ) - 1 = 16383
              return CheckSum((word *)(0x8000), 16383);
  case VOLT:  secondByte=analogV.vAnalogMedia[V13];
              return analogV.vAnalogMedia[V5G];
  case BATT:  secondByte=analogV.vAnalogMedia[VCH];
              return analogV.vAnalogMedia[VA];
  case CODPRO:  secondByte=opzioni[ADDCODLSB];
                return opzioni[ADDCODMSB];
  case CODLOT:  if(contaLotto<9){
                  secondByte=opzioni[ADDCODLOT+contaLotto];
                  contaLotto=contaLotto+2;
                  return opzioni[ADDCODLOT+contaLotto-1];
                } else{
                    contaLotto=0;
                    secondByte=0;
                    return 0;
                  }
  case LEDLINK: busyLed_status=BLINK;
                led3_status=BLINK;
                return LEDLINK;
  case LEDR:  led1_status=BLINK;
              return LEDR;
  case LEDY:  led2_status=BLINK;
              return LEDY;
  case LEDG:  led3_status=BLINK;
              busyLed_status=BLINK;
              return LEDG;
  case TESTBUZ: buzzer.State.status=BEEP_40m;
                buzzer.State.numbeep=3;
                return TESTBUZ;
  case TESTDIS: for(ii=16; ii<32; ii++) disp_buf_blink[ii]=TRUE;
                display.dBlink=TRUE;
                MyPrint(PROVADISP, 0, BUF_DISP);
                flag_backLight=TRUE;
                return TESTDIS;
  case TESTKEY: if(first_key){
                  first_key=FALSE;
                  secondByte=0;
                  return 0;
                } else{
                    secondByte=KeysS2();
                    return KeysS1();
                  }
  case BCKLGT_ON: flag_backLight=TRUE;
                  return BCKLGT_ON;
  case BCKLGT_OFF:  flag_backLight=FALSE;
                    timsec.t.timBL=0;
                    return BCKLGT_OFF;
  case ON_PG3:  spegni_V13=FALSE;
                return ON_PG3;
  case OFF_PG3: spegni_V13=TRUE;
                return OFF_PG3;
  case PASS:  if(!firstPass){
                firstPass=TRUE;
                secondPass=FALSE;
                thirdPass=FALSE;
                fourthPass=FALSE;
```

```
                } else{
                    if(!secondPass){
                      dataCodPro=inpBufDLFB[4];
                      if (IICSW_write_byte(ADDCODMSB, &dataCodPro)==_OK){
                        opzioni[ADDCODMSB]=inpBufDLFB[4];
                        secondPass=TRUE;
                      }
                    } else{
                        if(!thirdPass){
                          dataCodPro=inpBufDLFB[4];
                          if (IICSW_write_byte(ADDCODLSB, &dataCodPro)==_OK){
                            opzioni[ADDCODLSB]=inpBufDLFB[4];
                            thirdPass=TRUE;
                          }
                        } else{
                            if(!fourthPass){
                              dataCodPro=inpBufDLFB[4];
                              if (IICSW_write_byte(ADDCODLOT+countPass,
                                  &dataCodPro)==_OK)
                                opzioni[ADDCODLOT+countPass]=inpBufDLFB[4];
                              countPass++;
                              if(countPass>8) fourthPass=TRUE;
                            } else{
                                countService=END_SERVICE;
                                firstPass=FALSE;
                              }
                            }
                        }
                    }
                  return PASS;

    default:  return 0;
  }
}

////////////////////////////////////////////////////////////////////////////////
/**
*    \brief   CheckSum of flash
*
*    \detail  The function calculate a checksum of a range of byte. The checksum is
*             calculate over a long. The function return the least significant byte
*             of the result.
*/
////////////////////////////////////////////////////////////////////////////////
byte CheckSum(word *startAddr, int numWords){
       long tmpSum = 0;
       int i;
       for (i=0; i<numWords; i++){
              tmpSum += *startAddr++;
       }
       return (byte)tmpSum;
}
```

10.2 Sviluppo software di collaudo

Il software di collaudo sviluppato in Visual Basic ha il compito di comandare gli ingressi e controllare le uscite del banco, verificando così tutti i componenti e le funzionalità del terminale TLC.

```
Sub CollaudoManuale()
    statoCC = V5G Or V12
    txrxhw 1, statoCC, 1
    Pausa 0.5
    Do While manuale = True
        Select Case faseManu
            Case 0
                FormManuale.ShowMess ("Il led Rosso lampeggia?")
                'led red blink
                messaggio (RED)
                errore = 0
                vv = 0
                TxRxTLC
                FormManuale.Command1.Enabled = True
                FormManuale.Command2.Enabled = True
                datiOutput.OkLeds.LcResult = TESTOK
                Pausa 1

            Case 1
                If esitoManu = False Then
                    datiOutput.OkLeds.LcResult = TESTERR
                End If
                FormManuale.ShowMess ("Il led Giallo lampeggia?")
                'led yellow blink
                messaggio (YELLOW)
                errore = 0
                vv = 0
                TxRxTLC
                FormManuale.Command1.Enabled = True
                FormManuale.Command2.Enabled = True
                Pausa 1

            Case 2
                If esitoManu = False Then
                    datiOutput.OkLeds.LcResult = TESTERR
                End If
                FormManuale.ShowMess ("I 2 led Verdi lampeggiano?")
                'led green e link blink
                messaggio (GREEN)
                errore = 0
                vv = 0
                TxRxTLC
                FormManuale.Command1.Enabled = True
                FormManuale.Command2.Enabled = True
                Pausa 1

            Case 3
                If parziale = False Then
                    If esitoManu = True And datiOutput.OkLeds.LcResult = TESTOK Then
                        FormSeqColl.Check2(1).value = 1
                        FormSeqColl.Check2(1).Caption = "Led funzionanti"
                    Else
                        datiOutput.OkLeds.LcResult = TESTERR
                        FormSeqColl.Check2(1).value = 0
                        FormSeqColl.Check2(1).Caption = "Led guasti"
                        FormSeqColl.Check2(1).ForeColor = ROSSO
                    End If
                End If
                FormManuale.ShowMess ("Il buzzer suona correttamnte (3 beep)?")
                'buzzer 3 beep
                messaggio (BUZZER)
                errore = 0
                vv = 0
```

```
                TxRxTLC
                FormManuale.Command1.Enabled = True
                FormManuale.Command2.Enabled = True
                Pausa 1

            Case 4
                If parziale = False Then
                    If esitoManu = True Then
                        datiOutput.OkBuzzer.LcResult = TESTOK
                        FormSeqColl.Check2(2).value = 1
                        FormSeqColl.Check2(2).Caption = "Buzzer funzionante"
                    Else
                        datiOutput.OkBuzzer.LcResult = TESTERR
                        FormSeqColl.Check2(2).value = 0
                        FormSeqColl.Check2(2).Caption = "Buzzer Guasto"
                        FormSeqColl.Check2(2).ForeColor = ROSSO
                    End If
                End If
                FormManuale.ShowMess ("Il display visualizza il messaggio di prova?")
                'prova display
                messaggio (DISPLAY)
                errore = 0
                vv = 0
                TxRxTLC
                FormManuale.Command1.Enabled = True
                FormManuale.Command2.Enabled = True
                Pausa 1

            Case Else
                If parziale = False Then
                    If esitoManu = True Then
                        datiOutput.OkDisplay.LcResult = TESTOK
                        FormSeqColl.Check2(3).value = 1
                        FormSeqColl.Check2(3).Caption = "Display funzionante"
                    Else
                        datiOutput.OkDisplay.LcResult = TESTERR
                        FormSeqColl.Check2(3).value = 0
                        FormSeqColl.Check2(3).Caption = "Display Guasto"
                        FormSeqColl.Check2(3).ForeColor = ROSSO
                    End If
                    FormSeqColl.Command5.Enabled = True
                End If
                manuale = False
                Unload FormManuale
        End Select
    Loop
End Sub

Sub TestTastiera()
    avviaTest = True
    firstTasti = True
    FormSeqColl.Check2(4).Caption = ""
    FormSeqColl.Check2(4).value = 0
    FormSeqColl.Command2.Enabled = False
    FormSeqColl.Command3.Enabled = False
    FormSeqColl.Command4.Enabled = False
    FormSeqColl.Command6.Enabled = False
    FormSeqColl.Command8.Enabled = False
    Do While avviaTest = True
        messaggio (KEYPAD)
        vv = 0
        errore = 0
```

```
        TxRxTLC
        If errore = 0 Then
            numErr = 0
            FormSeqColl.Frame6.Enabled = True
            FormSeqColl.Command5.Enabled = True
            If firstTasti = True Then
                For ii = 0 To 15
                    FormSeqColl.ShapeT(ii).BackColor = BIANCO
                Next ii
                firstTasti = False
            End If
            DecTasti
        Else
            numErr = numErr + 1
            For ii = 0 To 15
                FormSeqColl.ShapeT(ii).BackColor = ROSSO
            Next ii
            FormSeqColl.Frame6.Enabled = False
            FormSeqColl.Command2.Enabled = True
            FormSeqColl.Command3.Enabled = True
            FormSeqColl.Command4.Enabled = True
            FormSeqColl.Command5.Enabled = False
            FormSeqColl.Command6.Enabled = True
            FormSeqColl.Command8.Enabled = True
            firstTasti = False
        End If
    Loop
End Sub

Sub DecTasti()
    Dim insieme(15)
    tasti1 = primoByte
    tasti2 = secondoByte
    premuti = 0
    For ii = 0 To 15
        insieme(ii) = False
    Next ii
    'decodifica tasti 1-8
    For ii = 0 To 7
        If tasti1 And CODTASTI(ii) Then
            insieme(ii) = True
            premuti = premuti + 1
            FormSeqColl.ShapeT(ii).BackColor = VERDE
        End If
    Next ii
    'decodifica tasti 9-16
    For ii = 0 To 7
        If tasti2 And CODTASTI(ii) Then
            insieme(ii + 8) = True
            premuti = premuti + 1
            FormSeqColl.ShapeT(ii + 8).BackColor = VERDE
        End If
    Next ii
    'controlla che sia stato premuto un solo tasto alla volta
    If premuti > 1 Then
        For ii = 0 To 15
            If insieme(ii) = True Then
                FormSeqColl.ShapeT(ii).BackColor = ROSSO
            End If
        Next ii
    End If
End Sub
```

```
Sub TestCentr()
    Do
        Select Case fasE

            Case FASEINIT
                '*****************************************************
                ' Accensione apparato collaudo
                '*****************************************************
                FormTestTLC.Frame1.Enabled = True
                FormTestTLC.Frame2.Enabled = True
                statoCC = V12 Or V5G
                txrxhw 1, statoCC, 1
                Pausa 0.5
                If retCod = OKRETCOD And fasE <> FASEESC Then
                    FormTestTLC.LabelEsito(0).ForeColor = NERO
                    FormTestTLC.LabelEsito(0).Caption = OK
                    numErr = 0
                Else
                    If fasE <> FASEESC Then
                        FormTestTLC.LabelEsito(0).Caption = "ERR TRX"
                        FormTestTLC.LabelEsito(0).ForeColor = ROSSO
                        FormErrMain.ShowMessMain (ErrHrdw)
                        Exit Sub
                    Else
                        Exit Sub
                    End If
                End If

                '*****************************************************
                ' Attivazione RS485
                '*****************************************************
                If fasE <> FASEESC Then
                    messaggio (GIATEST)
                    errore = 0
                    vv = 0
                    TxRxTLC
                    If errore = 0 And fasE <> FASEESC Then
                        numErr = 0
                        FormTestTLC.LabelEsito(1).ForeColor = NERO
                        FormTestTLC.LabelEsito(1).Caption = OK
                        If primoByte = 2 Then
                            messaggio (CODCENTR)
                            errore = 0
                            vv = 0
                            TxRxTLC
                            If fasE <> FASEESC Then
                                datiOutput.SerialNum = primoByte * 255 + secondoByte
                                FormTestTLC.Text2.Text = datiOutput.SerialNum
                                messaggio (VERSDISP)
                                errore = 0
                                vv = 0
                                TxRxTLC
                                If fasE <> FASEESC Then
                                    versioneTempor = "0." + CStr(primoByte) + "." +
                                                     CStr(secondoByte)
                                    FormTestTLC.Text1.Text = versioneTempor
                                    datiOutput.ProdVers = versioneTempor
                                    FormErrGiaTest.ShowMessGiaTest (ErrGiaTest)
                                    faseTestPre = fasE
                                    fasE = FASEWAIT
                                Else
                                    Exit Sub
                                End If
```

```
                        Else
                            Exit Sub
                        End If
                    Else
                        If parziale = False And fasE <> FASEESC Then
                            fasE = fasE + 1
                        Else
                            If parziale = True Then
                                fasE = FASEFINE
                            End If
                        End If
                    End If
                Else
                    If fasE <> FASEESC Then
                        numErr = numErr + 1
                        FormTestTLC.LabelEsito(1).Caption = "ERR TRX"
                        FormTestTLC.LabelEsito(1).ForeColor = ROSSO
                        FormErrMain.ShowMessMain (ErrTLC)
                        Exit Sub
                    Else
                        Exit Sub
                    End If
                End If
            Else
                Exit Sub
            End If

        Case FASEVERSCKS
            '*****************************************************
            ' Richiesta versione dispositivo
            '*****************************************************
            statoCC = V12 Or V5G
            txrxhw 1, statoCC, 1
            Pausa 0.5
            FormTestTLC.Frame3.Enabled = True
            messaggio (VERSDISP)
            errore = 0
            vv = 0
            TxRxTLC
            If errore = 0 And fasE <> FASEESC Then
                versioneTempor = "0." + CStr(primoByte) + "." + CStr(secondoByte)
                If (Dir(AddPath(CStr(versioneTempor))) <> CStr(versioneTempor)) Then
                    'il file non esiste ancora, versione nuova
                    Open AddPath(CStr(versioneTempor)) For Random As textVers
                    datiVersCks.Checksum = 0
                    Put #textVers, , datiVersCks
                    Close
                    appenaScritto = True
                Else
                    'file già esistente
                    Open AddPath(CStr(versioneTempor)) For Random As textVers
                    Get #textVers, , datiVersCks
                    Close
                    appenaScritto = False
                End If
                FormTestTLC.Text1.Text = versioneTempor
                datiOutput.ProdVers = versioneTempor
            Else
                If fasE <> FASEESC Then
                    numErr = numErr + 1
                    FormTestTLC.Text1.Text = "ERR TRX"
                    FormTestTLC.Text1.ForeColor = ROSSO
                    datiOutput.ProdVers = ""
```

```
                Else
                    Exit Sub
                End If
            End If

            '*****************************************************
            ' Richiesta checksum firmware
            '*****************************************************
            If fasE <> FASEESC Then
                messaggio (CKSUM)
                errore = 0
                vv = 0
                TxRxTLC
                If errore = 0 And fasE <> FASEESC Then
                    numErr = 0
                    If appenaScritto = True Then
                        datiVersCks.Checksum = primoByte
                        Open AddPath(datiOutput.ProdVers) For Random As textVers
                        Put #textVers, , datiVersCks
                        Close
                        FormTestTLC.LabelEsito(2).ForeColor = NERO
                        FormTestTLC.LabelEsito(2).Caption = OK
                    Else
                        If datiVersCks.Checksum = primoByte Then
                            FormTestTLC.LabelEsito(2).ForeColor = NERO
                            FormTestTLC.LabelEsito(2).Caption = OK
                        Else
                            FormTestTLC.LabelEsito(2).Caption = "ERR VAL" & " - " &
                                                                primoByte
                            FormTestTLC.LabelEsito(2).ForeColor = ROSSO
                        End If
                    End If
                Else
                    If fasE <> fasefient Then
                        numErr = numErr + 1
                        FormTestTLC.LabelEsito(2).Caption = "ERR TRX"
                        FormTestTLC.LabelEsito(2).ForeColor = ROSSO
                    End If
                End If
                If parziale = False And fasE <> FASEESC Then
                    fasE = fasE + 1
                Else
                    If parziale = True Then
                        fasE = FASEFINE
                    End If
                End If
            Else
                Exit Sub
            End If

        Case FASEVOLT
            '*****************************************************
            ' Controllo dei valori di 5V e 13V
            '*****************************************************
            FormTestTLC.Frame4.Enabled = True
            SequenzaVolt
            If parziale = False And fasE <> FASEESC Then
                fasE = fasE + 1
            Else
                If parziale = True Then
                    fasE = FASEFINE
                End If
            End If
```

```
Case FASECHRG
    '*****************************************************
    ' V e I recharger
    '*****************************************************
    FormTestTLC.Frame5.Enabled = True
    SequenzaBatt
    If parziale = False And fasE <> FASEESC Then
        fasE = fasE + 1
    Else
        If parziale = True Then
            fasE = FASEFINE
        End If
    End If

Case FASECONSUMI
    '*****************************************************
    ' verifica dei consumi
    '*****************************************************
    FormTestTLC.Frame6.Enabled = True
    SequenzaConsumi
    If parziale = False And fasE <> FASEESC Then
        fasE = fasE + 1
    Else
        If parziale = True Then
            fasE = FASEFINE
        End If
    End If

Case FASEGAS
    '*****************************************************
    ' verifica comunicazione rivelatore di gas
    '*****************************************************
    FormTestTLC.Frame7.Enabled = True
    'impostazione hardware
    statoCC = V5G Or V12
    txrxhw 1, statoCC, 1
    Pausa 0.5
    messaggio (OFF_PG3)
    errore = 0
    vv = 0
    TxRxTLC
    messaggio (GAS)
    errore = 0
    vv = 0
    TxRxTLC

    'controllo versione rivelatore gas
    If errore = 0 And fasE <> FASEESC Then
        numErr = 0
        If primoByte <> METANO Then
            If primoByte = VERSGAS Then
                FormTestTLC.LabelEsito(17).ForeColor = NERO
                FormTestTLC.LabelEsito(17) = OK
            Else
                FormTestTLC.LabelEsito(17) = "ERR VAL"
                FormTestTLC.LabelEsito(17).ForeColor = ROSSO
            End If
        Else
            FormTestTLC.LabelEsito(17) = "ERR VAL"
            FormTestTLC.LabelEsito(17).ForeColor = ROSSO
        End If
    Else
```

```
                    If fasE <> FASEESC Then
                        numErr = numErr + 1
                        FormTestTLC.LabelEsito(17) = "ERR TRX"
                        FormTestTLC.LabelEsito(17).ForeColor = ROSSO
                    Else
                        Exit Sub
                    End If
                End If
                fasE = FASEFINE
                messaggio (ON_PG3)
                errore = 0
                vv = 0
                TxRxTLC

            Case FASEWAIT
                Pausa 0.3

            Case FASEFINE
                '*****************************************************
                ' Fine Test
                '*****************************************************
                FormTestTLC.Frame8.Enabled = True
                If riscrivi = False Then
                    'ricava codice prodotto da ultimo record salvato+1
                    GetNewCodeSer
                    FormTestTLC.Text2.Text = Trim(datiOutput.SerialNum)
                End If
                faseTest = faseTest + 1
                Exit Sub

            Case FASEESC
                faseTest = faseTest + 1
                Exit Sub

            Case Else
                faseTest = faseTest + 1
                Exit Sub

        End Select
        If numErr > 3 Then
            fasE = FASEESC
        End If
    Loop

    Stop
    faseTest = faseTest + 1
    Exit Sub
End Sub

'-----------------------------------------------------
' Subroutine di colloquio con hardware di test. Si aspetta i seguenti parametri:
'    nmodulo%  = modulo indirizzato
'    comouts%= switch di comando uscite
'    retcod% = return code caricato in uscita.
'
'    Retcod% puo' assumere i seguenti valori:
'         &HFF -> ok transazione.
'         &H00 -> uscita con escape
'         &H01 -> messaggio da inviare errato
'         &H02 -> header da inviare errato
'         &H81 -> err. la centrale non risponde
'         &H82 -> err. timeout risposta
'         &H83 -> messaggio di risposta errato
```

```
'         &H84 -> header di risposta errato
'         &H85 -> err. checksum messaggio da centrale
'         &H86 -> indirizzo di risposta errato
'
'-------------------------------------------------
Sub txrxhw(nmodulo%, comouts%, viser%, Optional onlyRead As Boolean = False)
    ' Preliminari
        retCod = 0
        retryh = 2
        tmouh = 0.2
        errX = 0
        lockErr = 1
        OpnComHw
        If errX Then GoTo enhtxrx
        lockErr = 0
        If errX Then
           If errX <> 24 Then
                  rig1$ = "** ERRORE" + Str$(errX) + " **"
                  rig2$ = ""
           Else
                  rig1$ = "COLLEGAMENTO " + com$
                  rig2$ = "ERRATO O MANCANTE"
           End If
           MsgBox ""
           Call ErrSpec(rig1$, rig2$)
           GoTo enhtxrx
        End If

        retr% = retryh
        msgsnd$ = Chr$(&HFF) + Chr$(nmodulo%) + "S"
        If onlyRead = False Then
           dd% = comouts%
           dl% = dd% And &HF
           dh% = dd% And &HF0
           msgsnd$ = msgsnd$ + Chr$(2) + Chr$(dl%) + Chr$(dh%)
        Else
           msgsnd$ = msgsnd$ + Chr$(0)
        End If
        lrc% = 0
        For jj% = 2 To Len(msgsnd$)
           lrc% = ((lrc% Xor (Asc(Mid$(msgsnd$, jj%, 1)))) And &HFF)
        Next jj%
        msgsnd$ = msgsnd$ + Chr$(lrc%)

' Invia messaggio a modulo
sendh:  FormMain.MSComm1.InBufferCount = 0 'vuota buffer di ricezione
        FormMain.MSComm1.Output = msgsnd$   'invia messaggio
        DoSleep

' Attende risposta
lodth1: toutcm = Timer
        Do Until FormMain.MSComm1.InBufferCount > 0
            DoSleep
            If Timer - toutcm > tmouh Then
                  retr% = retr% - 1
                  If retr% = 0 Then retCod = &H81: GoTo enhtxrx
                  Pausa (0.2)
                  GoTo sendh
            End If
        Loop

' Ricezione risposta
        FormMain.MSComm1.InputLen = 1
```

```
        inmsg$ = ""
        aa$ = ""
lodth2: toutcm = Timer
        Do While FormMain.MSComm1.InBufferCount = 0
            DoEvents
            If Timer - toutcm > tmouh Then
                   retr% = retr% - 1
                   If retr% = 0 Then retCod = &H82: GoTo enhtxrx
                   Pausa (0.5)
                   GoTo sendh
            End If
        Loop
        rxBin() = FormMain.MSComm1.Input
        aa$ = Chr$(rxBin(0))
        inmsg$ = inmsg$ + aa$
        If Len(inmsg$) < 13 Then GoTo lodth2 'skip se non conclusa ricezione

' Controlla messaggio ricevuto
        GoSub contrlh
        If retCod <> OKRETCOD Then
           retr% = retr% - 1
           If retr% = 0 Then GoTo enhtxrx
           Pausa (0.3)
           GoTo sendh
        End If
        GoTo enhtxrx

' Controlla messaggio ricevuto e carica risposta
contrlh:
        If Asc(Mid$(inmsg$, 1, 1)) <> 255 Then retCod = &H83: GoTo endcth
        If Asc(Mid$(inmsg$, 2, 1)) <> nmodulo% Then retCod = &H83: GoTo endcth
        inmsg$ = Right$(inmsg$, (Len(inmsg$) - 1)) 'toglie codice di inizio
        lndat% = Len(inmsg$) - 1    'lndat% = lunghezza messaggio
        lrc% = 0
        For jj% = 1 To lndat%
           lrc% = ((lrc% Xor (Asc(Mid$(inmsg$, jj%, 1)))) And &HFF)
        Next jj%
        If lrc% <> Asc(Mid$(inmsg$, lndat% + 1, 1)) Then retCod = &H85: GoTo endcth
        inmsg$ = Left$(inmsg$, lndat%)   'toglie CHECK

        For ii% = 1 To 8
           tmp% = Asc(Mid$(inmsg$, ii% + 3, 1))
           inpts(nmodulo%, ii%) = tmp% 'carica dati risposta
        Next ii%
            retCod = OKRETCOD
endcth: Return
enhtxrx:
        If retCod <> OKRETCOD Then
           If viser% > 0 Then
                  If retCod > 0 Then
                         numErr = numErr + 1
                         Call erroh
                  End If
           End If
        End If
        Exit Sub
errrhw: Call GestErr
        Resume Next
End Sub

'----------------------------------------
' Apre porta di comunicazione MSComm1 per comunicazione con circuito di collaudo
'----------------------------------------
```

```
Public Sub OpnComHw()
    On Local Error GoTo Err_OpCHW

' Attivazione normale
    If FormMain.MSComm1.PortOpen = True Then
       FormMain.MSComm1.PortOpen = False
       Pausa 0.2
    End If

    retCod = 0
    errX = 0
    lockErr = 0
    tmpa$ = Left$(paritY$, 1)
    opens$ = "4800" + "," + tmpa$ + "," + nBit$ + "," + stopB$
    FormMain.MSComm1.CommPort = Val(Right$(serial$, 1))
    FormMain.MSComm1.Settings = opens$
    FormMain.MSComm1.InputMode = comInputModeBinary
    FormMain.MSComm1.RTSEnable = True
    FormMain.MSComm1.InputLen = 0
    FormMain.MSComm1.PortOpen = True
    retCod = OKRETCOD
    Exit Sub

Err_OpCHW:
    GestErr
    Exit Sub

End Sub

Sub TxRxTLC()
    tot = &HFF
    If nTimeout_RTx > 0 Then
        nTimeout_RTx = nTimeout_RTx - 1
    End If
    DoEvents
    For ii = 0 To 6
        OpnCom
        FormMain.MSComm1.Output = Chr(Tx(ii)) 'invia messaggio
        CloseCom
    Next ii
    timeTras = Timer

    Do While esci = False
        If Timer - timeTras <= TIMEOUT Then
            If FormMain.MSComm1.PortOpen = False Then
                OpnCom
            End If
            DoEvents
            If FormMain.MSComm1.InBufferCount > 0 Then
                TrigTimer
                DoEvents
                ncar% = FormMain.MSComm1.InBufferCount
                rxBin() = FormMain.MSComm1.Input
                For jj = 1 To ncar%
                    vv = vv + 1
                    aa$ = Chr$(rxBin(jj - 1))
                    valc = Asc(aa$)
                    tot = tot Xor Asc(aa$)
                    TrigTimer
                    Select Case vv
                        Case 1
                            tot = tot Xor Asc(aa$)
                            If valc <> XON Then
```

```
                                jj = ncar%
                                errore = errore + 1
                            End If
                        Case 2
                            If valc <> ADD Then
                                jj = ncar%
                                errore = errore + 1
                            End If
                        Case 3
                            If valc <> 8 Then
                                jj = ncar%
                                errore = errore + 1
                            End If
                        Case 4
                            If valc <> ID Then
                                jj = ncar%
                                errore = errore + 1
                            End If
                        Case 5
                            primoByte = valc
                        Case 6
                            secondoByte = valc
                        Case 7
                            tot = tot Xor Asc(aa$)
                            If valc <> tot Then
                                jj = ncar%
                                errore = errore + 1
                            End If
                        Case 8
                            If valc <> XOFF Then
                                jj = ncar%
                                errore = errore + 1
                            End If
                            esci = True
                            OffTimer
                    End Select
                Next jj
                If nTimeout_RTx > 0 Then
                    nTimeout_RTx = nTimeout_RTx - 1
                End If
            End If
            If nTimeout_RTx > 0 Then
                nTimeout_RTx = nTimeout_RTx - 1
            End If
        Else
            esci = True
            errore = errore + 1
        End If
    Loop
End Sub

Sub SequenzaVolt()
    '*****************************************************
    ' set gas supply on
    '*****************************************************
    If parziale=False Or (parziale=True And numFaseTest=0) And fasE <> FASEESC Then
        statoCC = V12 Or V5G
        txrxhw 1, statoCC, 1
        Pausa 0.5
        messaggio (VOLT)
        errore = 0
        vv = 0
```

```
        TxRxTLC
        txrxhw 1, statoCC, 1
        If errore = 0 And fasE <> FASEESC Then
            'controllo V5G
            numErr = 0
            verificaV5 = primoByte
            If primoByte >= datiMinMax.Min5V And primoByte <= datiMinMax.Max5V Then
                If inpts(1, 1) >= MINV5RIF And inpts(1, 1) <= MAXV5RIF Then
                    FormTestTLC.LabelEsito(3).ForeColor = NERO
                    FormTestTLC.LabelEsito(3) = OK
                Else
                    FormTestTLC.LabelEsito(3) = "ERR RIFERIMENTO"
                    FormTestTLC.LabelEsito(3).ForeColor = ROSSO
                End If
            Else
                FormTestTLC.LabelEsito(3) = "ERR VAL TLC"
                FormTestTLC.LabelEsito(3).ForeColor = ROSSO
            End If
            FormTestTLC.LabelEsito(3) = FormTestTLC.LabelEsito(3) & ":   V " &
                                        Left(primoByte * KVAV5G, 4)
        Else
            If fasE <> FASEESC Then
                numErr = numErr + 1
                FormTestTLC.LabelEsito(3) = "ERR TRX"
                FormTestTLC.LabelEsito(3).ForeColor = ROSSO
            Else
                Exit Sub
            End If
        End If
        statoCC = V12 Or V5G
        txrxhw 1, statoCC, 1
        Pausa 0.5
        messaggio (BATT)
        errore = 0
        vv = 0
        TxRxTLC
        If errore = 0 And fasE <> FASEESC Then
            'controllo VA
            numErr = 0
            'V5G - VA > 0.2V & < 0.4V
            If (verificaV5 - primoByte) >= 7 And (verificaV5 - primoByte) <= 15 Then
                FormTestTLC.LabelEsito(4).ForeColor = NERO
                FormTestTLC.LabelEsito(4) = OK
            Else
                FormTestTLC.LabelEsito(4) = "ERR VAL"
                FormTestTLC.LabelEsito(4).ForeColor = ROSSO
            End If
            FormTestTLC.LabelEsito(4) = FormTestTLC.LabelEsito(4) & ":   V " &
                                        Left(primoByte * KVAV5G, 4)
        Else
            If fasE <> FASEESC Then
                numErr = numErr + 1
                FormTestTLC.LabelEsito(4) = "ERR TRX"
                FormTestTLC.LabelEsito(4).ForeColor = ROSSO
            Else
                Exit Sub
            End If
        End If
        If parziale = True Then
            Exit Sub
        End If
    End If
```

```
'*****************************************************
' set battery supply on
'*****************************************************
If parziale=False Or (parziale=True And numFaseTest=1) And fasE <> FASEESC Then
    statoCC = VMED Or ONBATT
    txrxhw 1, statoCC, 1
    Pausa 0.5
    If parziale = True Then
        FormFineTest.ShowMess msgAccendi
        Do While passaProssimo = False
            Pausa 0.2
        Loop
    End If
    messaggio (OFF_PG3)
    errore = 0
    vv = 0
    TxRxTLC

    If errore = 0 And fasE <> FASEESC Then
        numErr = 0
        Pausa 2
        messaggio (VOLT)
        errore = 0
        vv = 0
        TxRxTLC
        txrxhw 1, statoCC, 1
        If errore = 0 And fasE <> FASEESC Then
            numErr = 0
            '13V spento
            If secondoByte > (datiMinMax.Max13V + 22) Then
                If inpts(1, 2) < (MIN13VRIF / 2) Then
                    FormTestTLC.LabelEsito(5).ForeColor = NERO
                    FormTestTLC.LabelEsito(5) = OK
                Else
                    FormTestTLC.LabelEsito(5) = "ERR RIFERIMENTO"
                    FormTestTLC.LabelEsito(5).ForeColor = ROSSO
                End If
            Else
                FormTestTLC.LabelEsito(5) = "ERR VAL TLC"
                FormTestTLC.LabelEsito(5).ForeColor = ROSSO
            End If
            FormTestTLC.LabelEsito(5) = FormTestTLC.LabelEsito(5) & ":   V " &
                                        Left(inpts(1, 2) * K13VRIF, 4)
        Else
            If fasE <> FASEESC Then
                numErr = numErr + 1
                FormTestTLC.LabelEsito(5) = "ERR TRX"
                FormTestTLC.LabelEsito(5).ForeColor = ROSSO
            Else
                Exit Sub
            End If
        End If
    Else
        If fasE <> FASEESC Then
            numErr = numErr + 1
            FormTestTLC.LabelEsito(5) = "ERR TRX"
            FormTestTLC.LabelEsito(5).ForeColor = ROSSO
        Else
            Exit Sub
        End If
    End If
    messaggio (ON_PG3)
    errore = 0
```

```
        vv = 0
        TxRxTLC
        If errore = 0 And fasE <> FASEESC Then
            Pausa 1
            numErr = 0
            messaggio (VOLT)
            errore = 0
            vv = 0

            TxRxTLC
            txrxhw 1, statoCC, 1
            If errore = 0 And fasE <> FASEESC Then
                numErr = 0
                '13V acceso senza carico
               If secondoByte>=datiMinMax.Min13V And secondoByte<=datiMinMax.Max13V Then
                    If inpts(1, 2) >= MIN13VRIF And inpts(1, 2) <= MAX13VRIF Then
                        FormTestTLC.LabelEsito(6).ForeColor = NERO
                        FormTestTLC.LabelEsito(6) = OK
                    Else
                        FormTestTLC.LabelEsito(6) = "ERR RIFERIMENTO"
                        FormTestTLC.LabelEsito(6).ForeColor = ROSSO
                    End If
                Else
                    FormTestTLC.LabelEsito(6) = "ERR VAL TLC"
                    FormTestTLC.LabelEsito(6).ForeColor = ROSSO
                End If
                FormTestTLC.LabelEsito(6) = FormTestTLC.LabelEsito(6) & ":   V " &
                                            Left(secondoByte * KV13, 5)
            Else
                If fasE <> FASEESC Then
                    numErr = numErr + 1
                    FormTestTLC.LabelEsito(6) = "ERR TRX"
                    FormTestTLC.LabelEsito(6).ForeColor = ROSSO
                Else
                    Exit Sub
                End If
            End If
        Else
            If fasE <> FASEESC Then
                numErr = numErr + 1
                FormTestTLC.LabelEsito(6) = "ERR TRX"
                FormTestTLC.LabelEsito(6).ForeColor = ROSSO
            Else
                Exit Sub
            End If
        End If
        statoCC = statoCC Or LOAD13V
        txrxhw 1, statoCC, 1
        Pausa 1
        messaggio (VOLT)
        errore = 0
        vv = 0

        TxRxTLC
        txrxhw 1, statoCC, 1
        If errore = 0 And fasE <> FASEESC Then
            numErr = 0
            '13V acceso con carico
            If secondoByte>=datiMinMax.Min13V And secondoByte<=datiMinMax.Max13V Then
                If inpts(1, 2) >= MIN13VRIF And inpts(1, 2) <= MAX13VRIF Then
                    FormTestTLC.LabelEsito(7).ForeColor = NERO
                    FormTestTLC.LabelEsito(7) = OK
                Else
```

```
                    FormTestTLC.LabelEsito(7) = "ERR RIFERIMENTO"
                    FormTestTLC.LabelEsito(7).ForeColor = ROSSO
                End If
            Else
                FormTestTLC.LabelEsito(7) = "ERR VAL"
                FormTestTLC.LabelEsito(7).ForeColor = ROSSO
            End If
            FormTestTLC.LabelEsito(7) = FormTestTLC.LabelEsito(7) & ":   V " &
                                        Left(secondoByte * KV13, 5)
        Else
            If fasE <> FASEESC Then
                numErr = numErr + 1
                FormTestTLC.LabelEsito(7) = "ERR TRX"
                FormTestTLC.LabelEsito(7).ForeColor = ROSSO
            Else
                Exit Sub
            End If
        End If
    End If
End Sub

Sub SequenzaBatt()
    '*******************************************
    ' Verifica Batterie
    '*******************************************
    If parziale=False Or (parziale=True And numFaseTest=0) And fasE <> FASEESC Then
        statoCC = VMED Or ONBATT
        txrxhw 1, statoCC, 1
        Pausa 0.5

        If parziale = True Then
            FormFineTest.ShowMess msgAccendi
            Do While passaProssimo = False
                Pausa 0.2
            Loop
        End If
        messaggio (OFF_PG3)
        errore = 0
        vv = 0
        TxRxTLC

        If errore = 0 And fasE <> FASEESC Then
            messaggio (BATT)
            errore = 0
            vv = 0
            TxRxTLC
            If errore = 0 And fasE <> FASEESC Then
                'Vbatt on, no load -> VCH=0
                If secondoByte < (datiMinMax.MinVchr / 2) Then
                    FormTestTLC.LabelEsito(8).ForeColor = NERO
                    FormTestTLC.LabelEsito(8) = OK
                Else
                    FormTestTLC.LabelEsito(8) = "ERR VAL"
                    FormTestTLC.LabelEsito(8).ForeColor = ROSSO
                End If
                FormTestTLC.LabelEsito(8) = FormTestTLC.LabelEsito(8) & ":   V " &
                                            Left(secondoByte * KVCHG, 4)
            Else
                If fasE <> FASEESC Then
                    numErr = numErr + 1
                    FormTestTLC.LabelEsito(8) = "ERR TRX"
                    FormTestTLC.LabelEsito(8).ForeColor = ROSSO
                Else
```

```
                Exit Sub
            End If
        End If
    Else
        If fasE <> FASEESC Then
            numErr = numErr + 1
            FormTestTLC.LabelEsito(8) = "ERR TRX"
            FormTestTLC.LabelEsito(8).ForeColor = ROSSO
        Else
            Exit Sub
        End If
    End If
    If parziale = True Then
        Exit Sub
    End If
End If

'*****************************************
' verifica Carica Batterie
'*****************************************
If parziale=False Or (parziale=True And numFaseTest=1) And fasE <> FASEESC Then
    statoCC = ONCHRG
    txrxhw 1, statoCC, 1
    Pausa 0.5
    messaggio (BATT)
    errore = 0
    vv = 0

    TxRxTLC
    If errore = 0 And fasE <> FASEESC Then
        If secondoByte>=datiMinMax.MinVchr And secondoByte<=datiMinMax.MaxVchr Then
            FormTestTLC.LabelEsito(9).ForeColor = NERO
            FormTestTLC.LabelEsito(9) = OK
        Else
            FormTestTLC.LabelEsito(9) = "ERR VAL"
            FormTestTLC.LabelEsito(9).ForeColor = ROSSO
        End If
        If primoByte >= datiMinMax.MinVA And primoByte <= datiMinMax.MaxVA Then
            FormTestTLC.LabelEsito(10).ForeColor = NERO
            FormTestTLC.LabelEsito(10) = OK
        Else
            FormTestTLC.LabelEsito(10) = "ERR VAL"
            FormTestTLC.LabelEsito(10).ForeColor = ROSSO
        End If
        FormTestTLC.LabelEsito(9) = FormTestTLC.LabelEsito(9) & ":   V " &
                                    Left(secondoByte * KVCHG, 4)
        FormTestTLC.LabelEsito(10) = FormTestTLC.LabelEsito(10) & ":   V " &
                                     Left(primoByte * KVAV5G, 4)
    Else
        If fasE <> FASEESC Then
            numErr = numErr + 1
            FormTestTLC.LabelEsito(9) = "ERR TRX"
            FormTestTLC.LabelEsito(10) = "ERR TRX"
            FormTestTLC.LabelEsito(9).ForeColor = ROSSO
            FormTestTLC.LabelEsito(10).ForeColor = ROSSO
        Else
            Exit Sub
        End If
    End If
    messaggio (VOLT)
    errore = 0
    vv = 0
    TxRxTLC
```

```
        txrxhw 1, statoCC, 1
        If errore = 0 And fasE <> FASEESC Then
            If primoByte <= (datiMinMax.Min5V / 2) Then
                FormTestTLC.LabelEsito(11).ForeColor = NERO
                FormTestTLC.LabelEsito(11) = OK
            Else
                FormTestTLC.LabelEsito(11) = "ERR VAL"
                FormTestTLC.LabelEsito(11).ForeColor = ROSSO
            End If
            FormTestTLC.LabelEsito(11) = FormTestTLC.LabelEsito(11) & ":   V " &
                                         Left(primoByte * KVAV5G, 4)
        Else
            If fasE <> FASEESC Then
                numErr = numErr + 1
                FormTestTLC.LabelEsito(11) = "ERR TRX"
                FormTestTLC.LabelEsito(11).ForeColor = ROSSO
            Else
                Exit Sub
            End If
        End If
        statoCC = statoCC Or BATTLOAD
        txrxhw 1, statoCC, 1
        Pausa 1
        messaggio (BATT)
        errore = 0
        vv = 0

        TxRxTLC
        txrxhw 1, statoCC, 1
        If errore = 0 And fasE <> FASEESC Then
            If secondoByte>=datiMinMax.MinVCgL And secondoByte<=datiMinMax.MaxVCgL Then
                FormTestTLC.LabelEsito(12).ForeColor = NERO
                FormTestTLC.LabelEsito(12) = OK
            Else
                FormTestTLC.LabelEsito(12) = "ERR VAL"
                FormTestTLC.LabelEsito(12).ForeColor = ROSSO
            End If
            If inpts(1, 4)>=datiMinMax.MinIchr And inpts(1, 4)<=datiMinMax.MaxIchr Then
                FormTestTLC.LabelEsito(13).ForeColor = NERO
                FormTestTLC.LabelEsito(13) = OK
            Else
                FormTestTLC.LabelEsito(13) = "ERR VAL"
                FormTestTLC.LabelEsito(13).ForeColor = ROSSO
            End If
            FormTestTLC.LabelEsito(12) = FormTestTLC.LabelEsito(12) & ":   V " &
                                         Left(secondoByte * KVCHG, 4)
            FormTestTLC.LabelEsito(13) = FormTestTLC.LabelEsito(13) & ":   mA " &
                                         Left(inpts(1, 4) * KICHRG, 5)
        Else
            If fasE <> FASEESC Then
                numErr = numErr + 1
                FormTestTLC.LabelEsito(12) = "ERR TRX"
                FormTestTLC.LabelEsito(13) = "ERR TRX"
                FormTestTLC.LabelEsito(12).ForeColor = ROSSO
                FormTestTLC.LabelEsito(13).ForeColor = ROSSO
            Else
                Exit Sub
            End If
        End If
    End If
End Sub

Sub SequenzaConsumi()
```

```
'*****************************************************
' set DLFB off and backlight off
'*****************************************************
If fasE <> FASEESC Then
    statoCC = VMED Or ONBATT
    txrxhw 1, statoCC, 1
    Pausa 0.5
    If parziale = True Then
        FormFineTest.ShowMess msgAccendi
        Do While passaProssimo = False
            Pausa 0.2
        Loop
    End If
    messaggio (BACKLIGHT_OFF)
    errore = 0
    vv = 0
    TxRxTLC
    txrxhw 1, statoCC, 1
    'verifica ingressi analogici inpts
    If inpts(1, 3)>=datiMinMax.MinIbatt1 And inpts(1, 3)<=datiMinMax.MaxIbatt1 Then
        FormTestTLC.LabelEsito(14).ForeColor = NERO
        FormTestTLC.LabelEsito(14) = OK
    Else
        FormTestTLC.LabelEsito(14) = "ERR VAL"
        FormTestTLC.LabelEsito(14).ForeColor = ROSSO
    End If
    FormTestTLC.LabelEsito(14) = FormTestTLC.LabelEsito(14) & ":   mA " &
                                 Left(inpts(1, 3) * KIBATT, 4)
Else
    Exit Sub
End If

'*****************************************************
' set DLFB off and backlight on
'*****************************************************
If fasE <> FASEESC Then
    messaggio (BACKLIGHT_ON)
    errore = 0
    vv = 0
    TxRxTLC
    Pausa 0.5
    txrxhw 1, statoCC, 1
    'verifica ingressi analogici inpts
    If inpts(1, 3)>=datiMinMax.MinIbatt2 And inpts(1, 3)<=datiMinMax.Maxibatt2 Then
        FormTestTLC.LabelEsito(15).ForeColor = NERO
        FormTestTLC.LabelEsito(15) = OK
    Else
        FormTestTLC.LabelEsito(15) = "ERR VAL"
        FormTestTLC.LabelEsito(15).ForeColor = ROSSO
    End If
    FormTestTLC.LabelEsito(15) = FormTestTLC.LabelEsito(15) & ":   mA " &
                                 Left(inpts(1, 3) * KIBATT, 5)
Else
    Exit Sub
End If

'*****************************************************
' set DLFB on and backlight off
'*****************************************************
If fasE <> FASEESC Then
    statoCC = statoCC Or LOAD13V
    txrxhw 1, statoCC, 1
    messaggio (ON_PG3)
```

```
        errore = 0
        vv = 0
        TxRxTLC
        messaggio (BACKLIGHT_OFF)
        errore = 0
        vv = 0
        TxRxTLC
        Pausa 0.5
        txrxhw 1, statoCC, 1
        'verifica ingressi analogici inpts
        If inpts(1, 3)>=datiMinMax.MinIbatt3 And inpts(1, 3)<=datiMinMax.Maxibatt3 Then
            FormTestTLC.LabelEsito(16).ForeColor = NERO
            FormTestTLC.LabelEsito(16) = OK
        Else
            FormTestTLC.LabelEsito(16) = "ERR VAL"
            FormTestTLC.LabelEsito(16).ForeColor = ROSSO
        End If
        FormTestTLC.LabelEsito(16) = FormTestTLC.LabelEsito(16) & ":   mA " &
Left(inpts(1, 3) * KIBATT, 5)
    End If
End Sub
```

10.3 Organizzazione DataBase

La registrazione dei prodotti e dei collaudi sul DataBase è suddivisa in 2 parti: una principale su ZKP_ProdReg, dove vengono salvati i dati generali riguardanti il singolo prodotto, e una secondaria su ZKP_ProdRegDetail, in cui si registrano i dettagli riguardanti collaudi / tarature / etc.

10.3.1 Struttura di ZKP_ProdReg

- DocID: identificativo univoco del collaudo. Numero progressivo inserito automaticamente dal DataBase;
- DocType: tipo di "documento", se collaudo normale o verifica Banco di collaudo;
- ProdType: tipo di prodotto collaudato (vedi registrazione prodotti);
- ProdCode: codice prodotto magazzino del prodotto;
- ProdVers: versione Firmware o Hardware del prodotto (vedi registrazione prodotti);
- LotNum: numero del lotto del collaudo nel formato mmaa/Oxxx (→ mm = mese, aa = anno, O = Sigla Operatore, xxx = Identificativo Lotto);
- SerialNum: numero seriale del collaudo;
- ProdOrd: codice ordine di produzione;
- UserName: cognome dell'operatore che ha effettuato il collaudo;
- StardTD: ora di inizio del collaudo;
- StopTD: ora di fine del collaudo;
- GlResult: esito generale del collaudo, se positivo o negativo;
- Note: campo per eventuali commenti o annotazioni;
- Value1 - 3: campi per l'inserimento di valori globali che interessano tutto il collaudo;
- Spare 1 – 5: campi liberi.

10.3.2 Struttura ZKP_ProdRegDetail

- RowId: identificativo univoco della riga relativa ad una singola fase del collaudo. Numero progressivo inserito automaticamente dal DataBase;
- RowType: tipo di riga. Se contiene un valore booleano (significativo solo LcResult)
  → RowType = 1, se invece la fase comprendeva il salvataggio di dati numerici
  → RowType = 2 e anche i valori Value 1 – 3 diventano significativi;
- DocID: chiave esterna che fa riferimento ad un collaudo riportato nella tabella principale ZKP_ProdReg. Specifica a quale collaudo appartiene la fase salvata;
- LcResult: risultato locale della fase del collaudo, se positiva o negativa;
- Descr: stringa che descrive la fase di collaudo salvata;
- Value 1 – 3: campi per l'inserimento di valori relativi alla fase del collaudo;
- Spare 1 – 3: campi liberi;

10.3.3 Dati Collaudo TLC

Dati Globali
DocType: 333 → documento di tipo collaudo prodotto;
ProdType: 852 → Collaudo del prodotto TLC
ProdVers: versione Firmware su microprocessore nel formato x.x.x;
I dati Note, Value1 – 3 e Spare 1 – 5 non sono utilizzati.

Fase CheckSum
RowType: 1 → valore Booleano;
Descr: "Verifica Checksum";
I dati Value1 – 3 e Spare 1 – 5 non sono utilizzati.

Fase 5V e Va
RowType: 2 → valori significativi;
Descr: "Verifica 5V e VA";
Value1: valore 5 V;
Value2: valore VA;
I dati Value2 – 3 e Spare 1 – 5 non sono utilizzati.

Fase 13V
RowType: 2 → valori significativi;
Descr: "Verifica 13V";
Value1: valore 13 V Off;
Value2: valore 13 V On con carico spento;
Value3: valore 13 V On con carico attivo;
I dati Spare 1 – 5 non sono utilizzati.

Fase Batteria
RowType: 2 → valori significativi;
Descr: "Verifica Batteria ";
Value1: verifica tensione batterie;
I dati Value2 – 3 e Spare 1 – 5 non sono utilizzati.

Fase Carica Batteria
RowType: 2 → valori significativi;
Descr: "Verifica Carica Batterie ";
Value1: tensione carica batterie senza carico;
Value2: tensione carica batteria con carico attivo;
Value3: corrente di ricarica batterie con carico attivo;
I dati Spare 1 – 5 non sono utilizzati.

Fase Consumi
RowType: 2 → valori significativi;
Descr: "Verifica Consumi ";
Value1: consumi a riposo (con DLFB Off e BackLight Off);
Value2: consumi con BackLight accesa (DLFB Off);
Value3: consumi con comunicazione DLFB attiva (BackLight Off);
I dati Spare 1 – 5 non sono utilizzati.

Fase Comunicazione Gas
RowType: 1 → valore Booleano;
Descr: "Verifica Comunicazione Gas ";
I dati Value1 – 3 e Spare 1 – 5 non sono utilizzati.

Fase Led
RowType: 1 → valore Booleano;
Descr: "Verifica Led ";
I dati Value1 – 3 e Spare 1 – 5 non sono utilizzati.

Fase Buzzer
RowType: 1 → valore Booleano;
Descr: "Verifica Buzzer ";
I dati Value1 – 3 e Spare 1 – 5 non sono utilizzati.

Fase Display
RowType: 1 → valore Booleano;
Descr: "Verifica Display ";
I dati Value1 – 3 e Spare 1 – 5 non sono utilizzati.

Fase KeyPad
RowType: 1 → valore Booleano;
Descr: "Verifica Keypad ";
I dati Value1 – 3 e Spare 1 – 5 non sono utilizzati.

```
Type RigaDB
      RowID As Integer               'identificativo riga
      RowType As Integer             'tipo di riga (1=bool, 2=value)
      DocID As Integer               'chiave esterna collaudo
      LcResult As String             'ok o errore
      Descr As String * 64           'descrizione tipo test
      Value1 As String * 32          'se tipo=2 -> valore del test
      Value2 As String * 32          'se tipo=2 -> valore del test
      Value3 As String * 32          'se tipo=2 -> valore del test
  End Type
```

```
Type RecordDati
    DocID As Integer                'identificativo collaudo
    DocType As Integer              'tipo documento
    ProdType As Integer             'tipo prodotto -> TLC=??
    ProdVers As String * 10         'versione prodotto ?.?.?
    ProdCode As String * 10         'codice prodotto magazzino
    SerialNum As String * 10        'codice seriale prodotto
    LotNum As String * 10           'codice lotto prodotto
    ProdOrd As String * 16          'codice ordine produzione
    UserName As String * 32         'nome + cognome
    StartTD As Date                 'ora inizio
    StopTD As Date                  'ora fine
    GlResult As String             'risultato generale

    OkChksum As RigaDB
    Ok5VeVA As RigaDB
    Ok13V As RigaDB
    OkBatt As RigaDB
    OkChrg As RigaDB
    OkConsumi As RigaDB
    OkGas As RigaDB
    OkLeds As RigaDB
    OkBuzzer As RigaDB
    OkDisplay As RigaDB
    OkKeys As RigaDB
End Type
```

CONCLUSIONI

Gli obbiettivi prefissati sono stati tutti conseguiti durante lo svolgimento del tirocinio. Sono stati sviluppati:

- Il dispositivo principale (TLC) in grado di interfacciarsi con i due sensori intelligenti indicati nelle specifiche del progetto;
- Un software di simulazione della barriera rivelatrice di fumo DLFB;
- Un banco di collaudo per il test finale del terminale durante la fabbricazione;
- Un software di collaudo che guida l'operatore durante tutte le fasi di controllo;
- Un database in cui salvare i dati relativi al collaudo dei dispositivi.

Non è stato possibile, al contrario del funzionamento con il rivelatore di gas, verificare l'utilizzo del TLC con il sensore di fumo in quanto l'azienda produttrice del DLFB, non ne aveva ancora terminato lo sviluppo e la realizzazione.
Il banco di test automatico ha permesso di ridurre notevolmente i tempi medi di collaudo dei terminali, abbassando i costi di produzione del dispositivo e riducendo la possibilità di errore durante le procedure.
Il database realizzato non solo permette di reperire tutti i dati relativi al collaudo in modo semplice e veloce, eliminando molti dei documenti cartacei previsti in precedenza, ma la sua struttura flessibile ne ha permesso l'impiego in tutti i collaudi supportati da PC realizzati in seguito.

La grande versatilità del TLC è quella che, quando viene collegato ad un sensore che comunica con il protocollo DLFB, cede completamente il controllo di tutti i suoi componenti di interfaccia (led, buzzer, display e testiera). Questo rende il dispositivo estremamente versatile e ha permesso il suo utilizzo, senza dover apportare modifiche al firmware del teminale, con numerosi dispositivi sviluppati in seguito (sensori, alimentatori, schede, etc) che altrimenti non avrebbero avuto nessun modo per interagire direttamente con l'utente.