ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA
SECONDA FACOLTÀ DI INGEGNERIA CON SEDE A CESENA
CORSO DI LAUREA IN INGEGNERIA
ELETTRONICA E TELECOMUNICAZIONI

Sistemi intelligenti di illuminazione pubblica: implementazione della comunicazione punto a punto su onde convogliate a banda stretta

ELABORATO IN:
Calcolatori elettronici

Relatore
Prof. Luca Roffia

Presentata da
Leonardo Ubaldi

Correlatore
Ing. Emanuele Montemurro

Terza Sessione
Anno Accademico 2011/2012

INDICE

1. SOMMARIO 5

2. INTRODUZIONE 6

2.1 PANORAMICA SULLE PLC 6
2.2 LE NORMATIVE DI RIFERIMENTO 6
2.3 PLC VS RF 7
2.4 PROGETTO "LA CITTÀ PULITA" 8
2.5 TOPOLOGIA DELLA RETE 9
2.5.1 Modulo COC 10
2.5.2 Modulo LIN 10

3. MODEM PLC 11

3.1 SCELTA DEL MODEM 13
31.1 Overview scheda 13
3.1.2 Caratteristiche ST7570 14
3.1.3 Caratteristiche EVALKITST7570 15
3.1.4 La GUI 16
3.1.5 Segnale S-FSK e Bit timing 17
3.2 PROTOCOL STACK 19
3.3 PHYSICAL LAYER 19
3.3.1 Pacchetto dati 19
3.3.2 Modi Operativi 20
3.3.3 Servizi disponibili 20
3.4 MAC LAYER 22
3.4.1 Servizi disponibili 22
3.4.2 Gestione degli indirizzi 23
3.4.3 Comportamento in trasmissione e in ricezione 24
3.5 INTERFACCIA CON L'HOST 24
3.6 PROTOCOLLO DI COMUNICAZIONE 25
3.6.1 Local frame 26
3.6.2 Status message 26
3.6.3 Acknowledgement message 27
3.6.4 Regole della comunicazione 27
3.7 MIB MANAGEMENT INFORMATION BASE 29
3.8 COMANDI 29
3.8.1 CMD_WriteDBRequest (41h) 30
3.8.2 CMD_SynchroIndication (10h) 31
3.8.3 CMD_DataRequest(51h) 31
3.8.5 CMD_DataConfirm (52h) 32
3.8.4 CMD_DataIndication (50h) 32

4. PROGETTO SOFTWARE 34

4.1 PROVE CON LA GUI 34
4.2 FUNZIONI DELLA PORTA SERIALE 34
4.3 FUNZIONI SECONDARIE 36
4.3.1 Invio dei dati sulla porta seriale 36
4.3.2 Ricezione del pacchetto dati dalla porta seriale 37
4.3.3 Ricezione del messaggio STATUS 37
4.4 MAIN 38
4.4.1 Descrizione del funzionamento del programma di esempio 41
4.4.2 Client 41
4.4.3 Server 45
4.5 PROBLEMATICHE RISCONTRATE A LIVELLO SOFTWARE 48

5. TEST DI FUNZIONAMENTO 49

5.1 TEST TRAFFICO DATI 49
5.2 TEST FREQUENZE E GUADAGNI 49
5.3 TEST DELLA COMUNICAZIONE CON DUE CLIENT 52

6. CONCLUSIONI 53

7. RIFERIMENTI 54

1. Sommario

L'obiettivo di questa tesi è stato quello di progettare una piccola rete di telecomunicazioni sfruttando la tecnica delle onde convogliate o *Powerline Comunication* per il controllo ed il monitoraggio punto-punto di un sistema di illuminazione pubblica. L'idea è che ogni lampione della rete di illuminazione sia munito di sensori per rilevare alcuni parametri ambientali (es. temperatura, umidità, CO2) e di attuatori per controllare l'intensità luminosa della lampada. Lo scenario prevede che tutti i lampioni di una strada condividano la stessa linea di alimentazione e che per ogni strada sia presente un unico centro di controllo e monitoraggio.

Lo studio parte da un lavoro compiuto in precedenza riguardo lo sviluppo delle *Powerline* a livello mondiale e su cosa è disponibile sul mercato, funzionale ad individuare la tecnologia da utilizzare in base allo scenario di riferimento. Sono state selezionate delle schede di sviluppo ottime per questo scopo e così si è dato inizio al lavoro vero e proprio.

In primo luogo si è studiato il funzionamento delle schede utilizzando un software dedicato fornito dalla casa costruttrice. Successivamente sono stati scritti due programmi in C, in ambiente Linux, uno per implementare la parte di comunicazione relativa al centro di controllo e monitoraggio e l'altro per implementare la parte di comunicazione relativa ad un lampione. Questo ha permesso infine di effettuare vari test per misurare la robustezza del collegamento, in una configurazione in cui sono presenti due lampioni ed un centro di controllo e monitoraggio.

Lo studio è stato svolto all'interno di una "convenzione di ricerca tra Ducati Energia Spa e ARCES".

L'elaborato è stato strutturato come segue.

Nel capitolo 2 viene riportato un quadro generale sulle PLC, cosa sono, dove si usano e perché sono così vantaggiose, esponendo le normative vigenti. Viene descritto poi più in dettaglio lo scenario di riferimento, riportando la tipologia della rete implementata e descrivendo i singoli componenti.

Nel capitolo 3 viene descritto il modem PLC utilizzato.

Nel capitolo 4 si riportano sezioni di codice, si illustrano nel dettaglio le singole funzioni e si mostrano i problemi affrontati.

Nel capitolo 5 si mostrano i test eseguiti. L'elaborato si conclude riportando alcune considerazioni finali e possibili sviluppi futuri.

2. Introduzione

2.1 Panoramica sulle PLC

La possibilità di trasmettere dati codificati su linee elettriche esistenti usando la tecnica delle onde convogliate, presenta numerosi vantaggi: quello principale è di poter utilizzare le linee già presenti sul territorio abbattendo tutti i costi di cablaggio; considerando che la rete elettrica è diffusa ovunque, si possono raggiungere anche i luoghi più isolati aumentando il numero di utenze.

Questa tecnica è stata usata per molti anni in applicazioni dell'"*Home automation"* ossia la telelettura del contatore, controllo di antifurti e condizionatori, utilizzando dei *bit-rate* non elevate in quanto la rete elettrica era stata progettata per la distribuzione di energia e non per la trasmissione di dati.

In ambito domestico c'è stato uno sviluppo più elevato nella realizzazione di LAN che andavano a sostituire le linee formate con i doppini di rame.

2.2 Le normative di riferimento

Quando non si ha la necessità di trasferire grosse quantità di dati e si devono coprire lunghe distanze in presenza di interferenze e attenuazioni, si usano le comunicazioni su *powerline* a "banda stretta".

L'ente che si occupa di stabilire gli intervalli di frequenze per i sistemi di comunicazione su linee elettriche in bassa tensione è il ***CENELEC*** (*Committee European de Normalisation Electrotechnique*).

Il *range* di frequenze parte da 3KHz e arriva a 148,5KHz suddiviso in 4 sottobande a seconda del tipo di impiego:

- CENELEC-A 3-95 kHz : riservata a fornitori di energia elettrica dotati di licenza;
- CENELEC-B 95-125 kHz : di libero uso, senza un protocollo d'accesso specifico;
- CENELEC-C 125-140 kHz : di libero uso con protocollo d'accesso CSMA/CA;
- CENELEC-D 140-148,5 kHz : di libero uso, senza protocollo d'accesso e di solito utilizzata per sistemi d'allarme e/o di sicurezza.

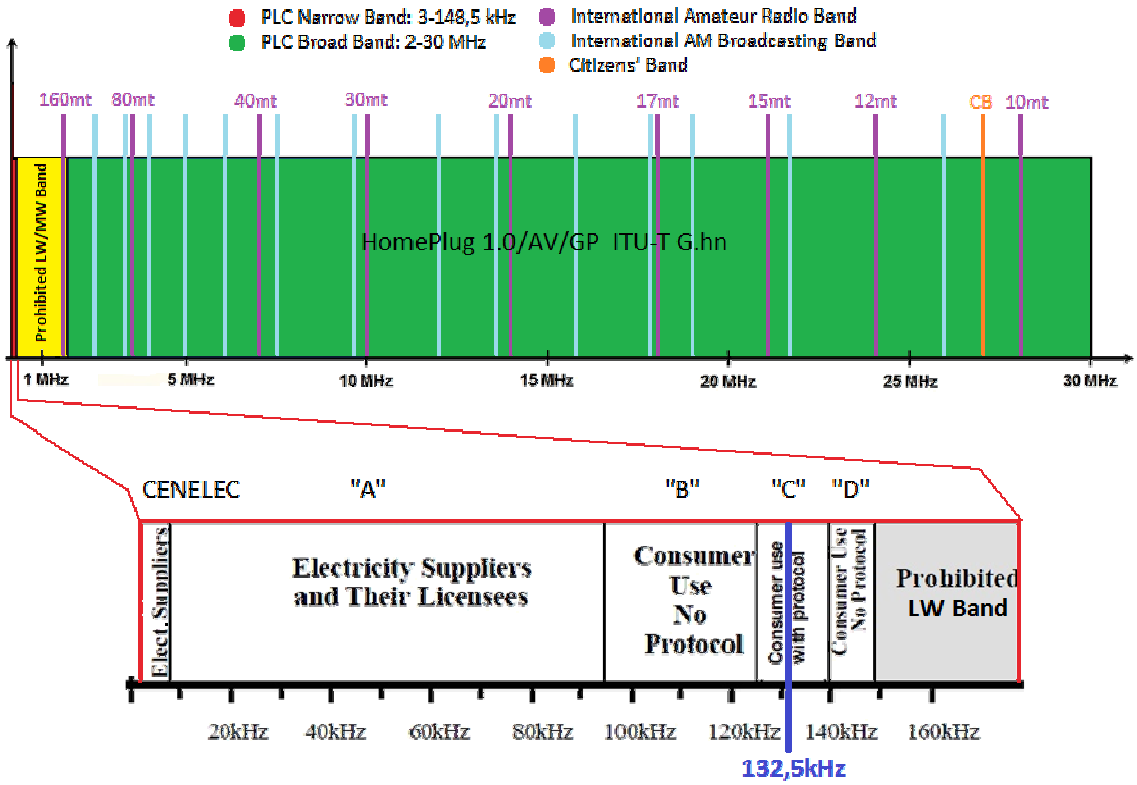

Figura 1: Distribuzione spettro

Per garantire la buona riuscita delle comunicazioni, ed evitare collisioni di dati, è opportuno utilizzare la banda CENELEC-C che prevede l'accesso al mezzo in modalità CSMA/CA.

In caso di maggiori interferenze o distanze più lunghe da coprire, si potrebbe diminuire la frequenza della portante utilizzando la banda CENELEC-B, anch'essa in modalità CSMA/CA, dato che in questa banda non è esplicitamente definito un protocollo specifico di accesso.

In caso di minori interferenze o distanze da coprire, si potrebbero invece utilizzare sistemi di trasmissione OFDM con più portanti distribuite in entrambe le bande in modo tale da aumentare il *bit-rate*[1].

2.3 PLC vs RF

Numerosi sono i difetti nell'utilizzare una rete *wireless* per impianti di illuminazione pubblica:

- i segnali nelle bande non licenziate (libere) sono maggiormente soggetti a interferenze e disturbi;
- i segnali trasmessi nelle bande licenziate sono meno soggetti ad interferenze ma hanno un costo molto più alto;

- la sicurezza in ambito *wireless* è sicuramente minore rispetto all'utilizzo del mezzo fisico in quanto la trasmissione non è confinata ed il segnale fisico è sempre in *broadcast*;
- il segnale radio perderebbe in affidabilità in presenza di strade non rettilinee, nelle zone altamente edificate o lungo strade d'altura ricche di tornanti e dislivelli.

Inoltre, la tecnologia *wireless* non permette di creare una topologia di rete a bus come la tecnologia PLC, ma rende necessario creare una rete di tipo lineare dotando ogni dispositivo di un ripetitore che però comporta notevoli svantaggi; primi tra tutti:

- il guasto di un dispositivo comporterebbe la perdita totale della comunicazione dal punto di guasto in poi;
- il sovraccarico dei dispositivi centrali soffocherebbe il traffico degli n/2 nodi presenti a destra e a sinistra della catena.

2.4 Progetto "La Città Pulita"

Il centro di ricerca ARCES dell'Università di Bologna e la Ducati Energia S.p.a. hanno stretto una convenzione di ricerca per studiare un sistema di illuminazione pubblica intelligente punto-a-punto trasformando i singoli lampioni in dispositivi in grado di fornire servizi maggiori rispetto a quelli di pura illuminazione, come il monitoraggio ambientale grazie all'uso di sensori. Inoltre sarà possibile controllare l'intensità luminosa di ogni singolo lampione grazie all'impiego di attuatori. Il controllo potrà essere manuale, in cui l'utente dispone di un'interfaccia per impostare l'intensità di ogni singola lampada, o basarsi sui dati rilevati da sensori (es. condizioni ambientali, presenza di traffico).

Grazie alla tecnologia *Powerline* si otterranno dei forti vantaggi poiché non sono richieste infrastrutture esterne per il trasferimento dei dati e si potranno rinnovare linee già esistenti con costi bassissimi.

Nella figura seguente è riportata l'architettura del sistema oggetto di studio, in cui sono presenti i vari lampioni (LIN) i quali sono collegati tra loro con i centri di controllo e comando (COC).

Questa architettura consente di suddividere la gestione della rete di illuminazione agendo, su ogni singola strada, che come mostrato il figura sarà gestita da un COC.

È previsto che i COC abbiano accesso ad internet (es. utilizzando un modem 3G) scambiando informazioni attraverso un'infrastruttura software chiamata *SMART-M3* [2][3] che consente di rappresentare i dati secondo gli standard del web semantico garantendo l'interoperabilità a livello di informazioni.

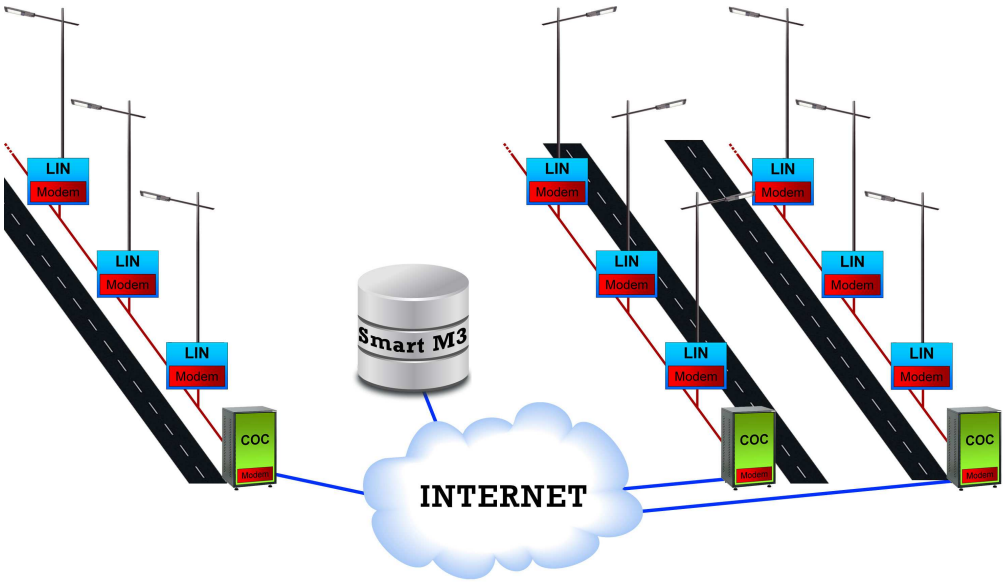

Figura 2: Schema collegamento lampioni

2.5 Topologia della rete

Il sistema di illuminazione pubblica intelligente prevede un collegamento di tipo punto-a-punto dove tutti i lampioni sono collegati alla rete elettrica in modo parallelo.

Sarà quindi indispensabile dotare ogni lampione di un modem PLC che gestirà il traffico dati presente sul bus.

Si svilupperanno quindi due tipologie di moduli presenti sulla rete: il modulo COC e il modulo LIN.

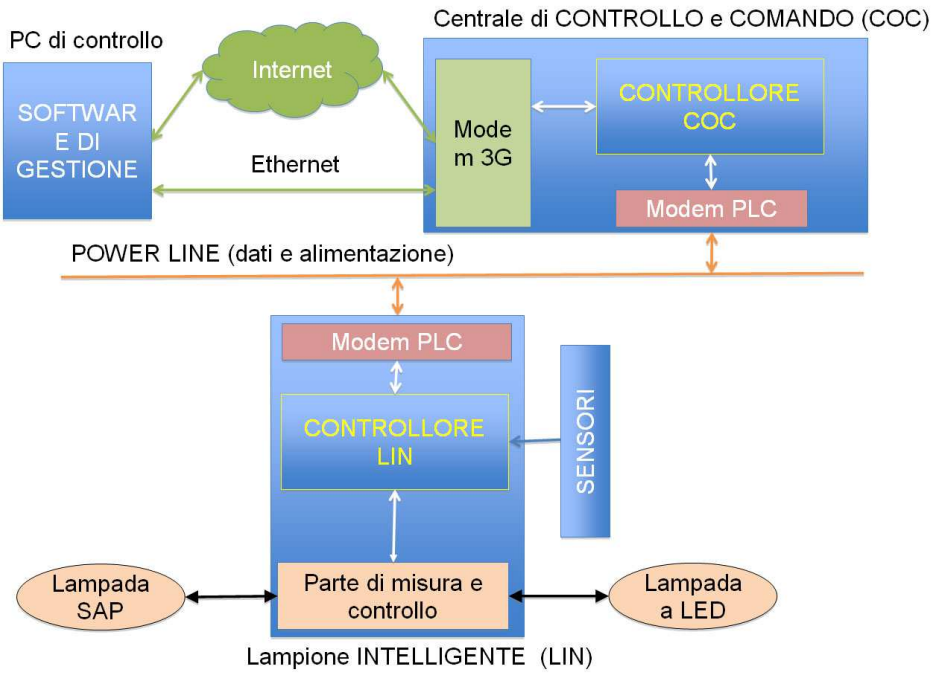

Figura 3: Componenti principali

2.5.1 Modulo COC

È la *centrale di controllo e comando* che ha la funzione di *router* verso i singoli lampioni e comunica con il software gestionale attraverso rete mobile (GPRS/UMTS/HSDPA).

Riceve comandi e invia dati al software gestionale per monitorare e pilotare ogni singolo lampione.

Sarà presente uno di questi moduli su ogni tratta di illuminazione.

2.5.2 Modulo LIN

È montato su ogni *lampione intelligente* e contiene tutti i sensori e gli attuatori per il monitoraggio ambientale e per il controllo degli elementi illuminanti come lampade a LED o quelle tradizionali.

Inoltre è presente anche il modem PLC senza il quale sarebbe impossibile la comunicazione tra COC e LIN.

3. Modem PLC

La scelta del modem è stata fatta basandosi su uno studio eseguito in precedenza riguardante i chip a semiconduttore che offre il mercato. Nella seguente tabella sono riportati i chip a disposizione [1]:

Marca	Prodotto	Standard	Velocità (kbps)	Banda	Interfacce	Crittografia/Codifica	Tipologia
ADD	ADD1021	PRIME	128	CENELEC A	2UART/SPI	128bit AES,FEC Viterbi	system on chip
ADD	ADD1010	EHS/KNX	4,8	CENELEC A/B/C	3UART, 2SPI	CRC,FEC,Viterbi	system on chip
Analog	ADE8165		0,8/2,4	CENELEC A/B	UART		modem
Ariane Controls	PLM-1		30	50-500 KHz	Parallel, SPI	FEC,CRC16	modem
Cypress	CY8CPLC10		2,4	CENELEC C	4UART, SPI	CRC	system on chip
Cypress	CY8CPLC20		2,4	CENELEC C	4UART, SPI	CRC	system on chip
Echelon	Smart Server 2.0	for all layers ISO/OSI		CENELEC C(plc)	ETH,RJ-45,RS-485,EIA-232,RJ-11		controller/gateway/server
Echelon	3120/3150		3,6	CENELEC A/C	UART, SPI	FEC,CRC	transceiver
Echelon	3170		5,4	CENELEC C	UART, SPI	FEC,CRC	transceiver
Infineon	UMF8110	G3-PLC PRIME,IEEE1901.2	576	9-490 KHz	UART, SPI,I²C, GPIO	128bit AES,CRC,Viterbi	modem
Maxim	MAX2992	G3-PLC	300	10-490 KHz	2UART, 2SPI	128bit AES,FEC,CRC16, Viterbi	system on chip
Maxim	MAX2990		100	CENELEC A/B/C	UART, SPI,I²C	56-DES FEC, CRC32	system on chip

ON Semicon.	AMIS 49587		2,4	CENELEC A	UART		modem
ON Semicon.	NCN 49597		4,8	CENELEC A-D	UART		modem
Semitech	SM2400	G3-PLC PRIME,IEEE1901.2	500	5-500 KHz	SPI	FEC, Viterbi	modem
Semitech	SM2200		175	5-500 KHz	SPI	CRC	modem
Semitech	SM6401	ANSI/EIA 709.1/2	5,4	CENELEC A/B/C	UART, SPI	3DES, FEC	system on chip
Semitech	SM8100		175	5-500 KHz	UART, RS232, RS485	CRC	module based on SM2200
ST	ST7540	protocol independent	4,8	CENELEC	UART/ SPI		transceiver
ST	ST7570	IEC61334-5-1 PHY +MAC	2,4	up to 148,5KHz	UART		system on chip
ST	ST7580	protocol independent	28,8	up to 250KHz	UART	128bit AES	system on chip
ST	ST7590	PRIME PHY+MAC+DLL	128	CENELEC A	UART/ SPI	128bit AES	system on chip
Texas Instruments	C2000 TMDSPLCKIT-V3	PRIME, G3-PLC		up to 500KHz	2UART, 2SPI	128bit AES, CRC	development kit
Ytran	IT700IC		2,5-7,5	CENELEC A/B	UART, SPI,I²C	128bit AES, CRC16	system on chip
Ytran	IT900		500	CENELEC A/B	UART, SPI,I²C	CRC16	system on chip

Tabella 1: Modem PLC sul mercato

Molti di questi chip sono venduti in kit di sviluppo, e dunque la scelta è ricaduta proprio su uno di questi kit.

3.1 Scelta del modem

Il modem scelto è l'*ST7570* montato sulla scheda di sviluppo *EVALKITST7570.*

Figura 4: EVALKITST7570

Questo kit è dotato di un alimentatore AC e una porta USB per la connessione al PC, inoltre è presente un connettore a vaschetta per il collegamento con altre schede in funzionamento *stand-alone.* Col kit viene fornita una utilissima GUI (*Graphical User Interface*) per PC per eseguire i primi test di funzionamento.

31.1 Overview scheda

La soluzione trovata ha molte caratteristiche interessanti:

- il chip ST7570 è basato su una modulazione *S-FSK narrow-band;*
- ha implementato a bordo lo standard *IEC61334-5-1* che comprende tre livelli di comunicazione: *PHY, MAC, MIB;*
- può essere configurato in diversi modi: *Server*, *Client*, *Sniffer* e *Test;*
- è equipaggiato con un alimentatore 110-240Vac basato sul chip *ALTAIR4-900;*
- la *GUI* è sviluppata in modo tale da poter programmare totalmente i parametri del modem [4].

3.1.2 Caratteristiche ST7570

È uno dei più potenti modem per PLC, è dotato di un performante processore che opera a livello fisico e un *analog front end (AFE)* integrato sulla linea.

Il processore fisico può essere programmato per operare con *bitrate* fino a 2.4Kbps sulla 50Hz, si possono regolare a *step* di 1Hz le portanti del segnale S-FSK fino a 148.5KHz e fornisce il rapporto segnale-rumore per il controllo della linea.

I dispositivi possono comunicare con il modem tramite la sua interfaccia UART.

All'interno dell'ST7570 vi è integrato un *front-end* analogico composto da un ADC, un DAC, un PGA con controllo automatico del guadagno per una elevata sensibilità in ricezione ed un generatore per il segnale della modulante.

Possiede un *Power Amolifier (PA)* a *single-ended* per il pilotaggio della linea con il controllo della corrente e di temperatura ed un filtro attivo configurabile [5].

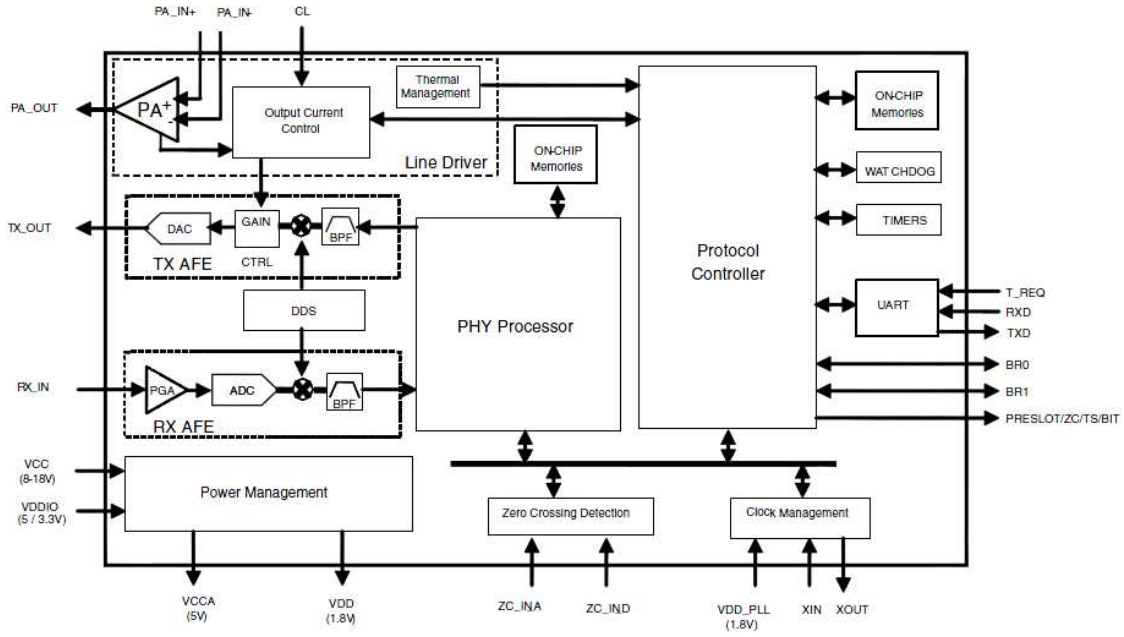

Figura 5: Diagramma a blocchi ST7570

3.1.3 Caratteristiche EVALKITST7570

La scheda su cui è montato il *chip* ST7570 è composta dai seguenti componenti [7]:

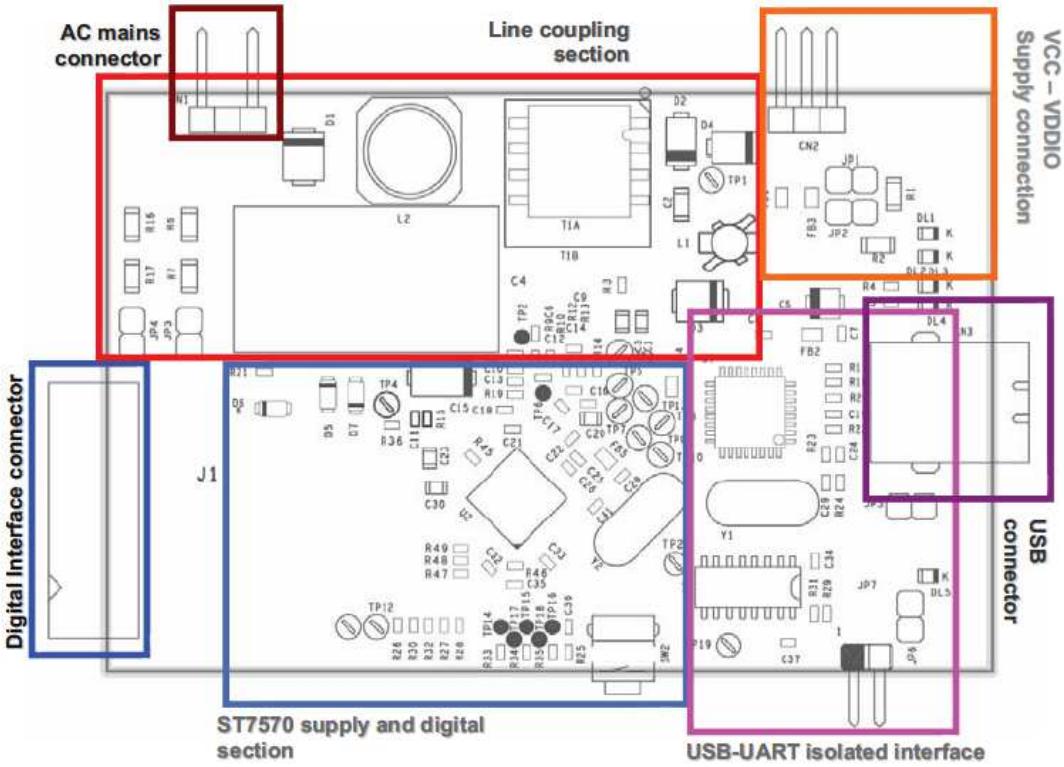

Figura 6: Disegno scheda diviso per sezioni

- interfaccia USB per il collegamento al PC dotata di un convertitore USB-RS232 di tipo FTDI;
- un connettore per interfacce digitali direttamente collegato al *chip* al quale è possibile collegare microcontrollori;
- tutti i componenti esterni al *chip* necessari per il suo funzionamento riguardanti la sezione di alimentazione e quella digitale;
- una sezione di accoppiamento alla linea composta di 4 differenti circuiti: filtro attivo in trasmissione, filtro passivo in ricezione, accoppiamento alla *power line* e lo *zero-crossing*.

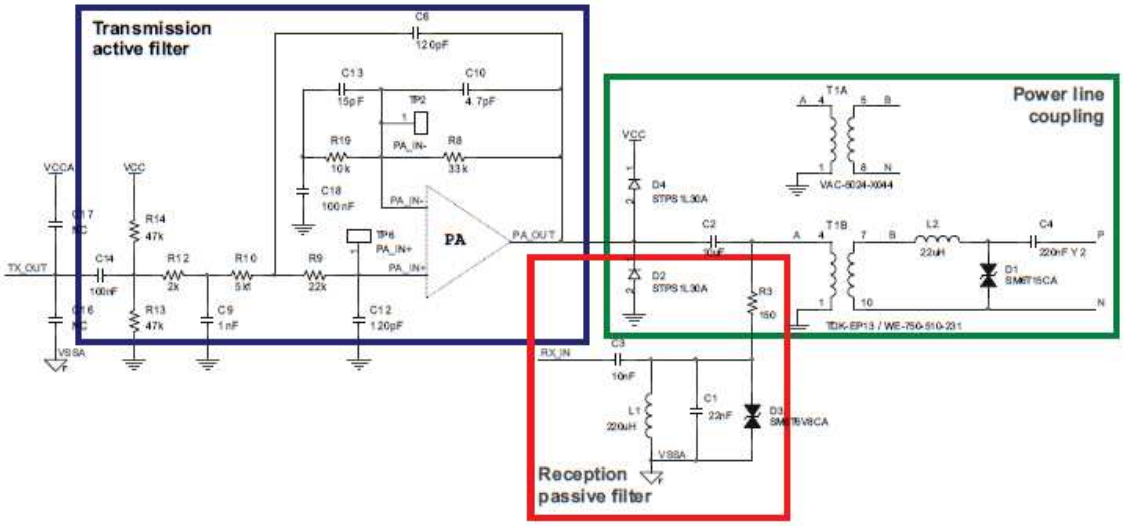

Figura 7: Schema filtro in ricezione e trasmissione

3.1.4 La GUI

La GUI è un *tool software* che permette di interfacciarsi con il modem ST7570 a bordo della scheda EVALKITST7570 avendo il completo controllo del dispositivo, potendo accedere quindi a tutti i registri e a tutte le sue funzioni.

Per usarla è sufficiente collegare la scheda al PC con un cavo USB installando inizialmente i driver per il convertitore FTDI e successivamente la GUI stessa [8].

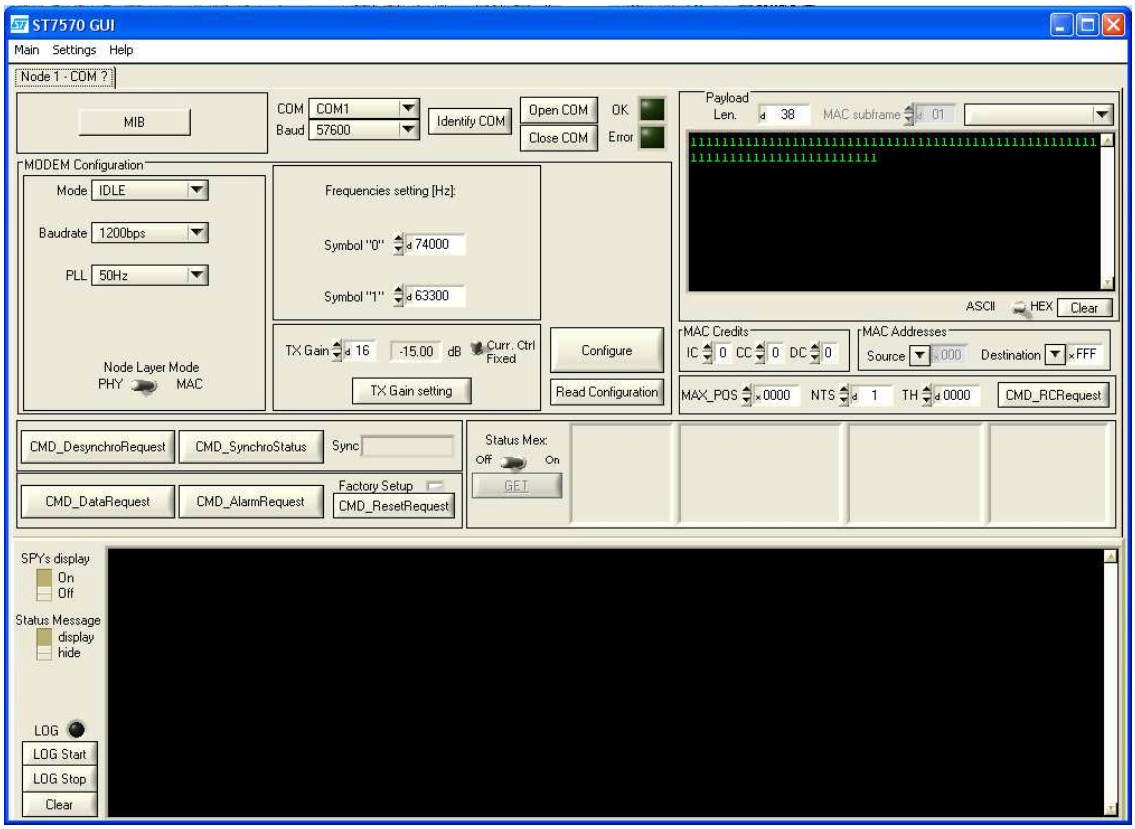

Figura 8: Schermata principale GUI

L’interfaccia grafica è molto *user friendly* quindi capirne il funzionamento non è complicato.

È formata principalmente da due sezioni: il *Node panel* sulla parte superiore dove si può controllare il funzionamento del modem e gestire contemporaneamente più modem su più porte, e nel pannello inferiore si nota la *console* che mostra tutti i messaggi scambiati tra i nodi.

Una volta stabilita la connessione al modem, si configurano tutti i suoi parametri come la modalità di lavoro, le frequenze e le *bitrate*, gli indirizzi MAC e i *timeout.*

In seguito si possono dare i comandi di trasmissione di pacchetti di dati e verificarne l’avvenuto inoltro.

Tutte queste funzioni verranno poi analizzate in dettaglio nei prossimi capitoli.

3.1.5 Segnale S-FSK e Bit timing

Il *IEC61334* (o 1334) è lo standard per una comunicazione su *Powerline* a bassa velocità ed è anche conosciuto come *S-FSK* ovvero *Spread Frequency Shift Keying*.

Questo standard definisce non solo tutto l’ambiente a livello fisico, ma il modo migliore per adattarlo alla *powerline* e l’interfaccia di gestione [5][6].

La modulazione S-FSK è un potenziamento della normale FSK aggiungendo maggiore robustezza in *narrow-band* sulle interferenze tipiche dell’approccio *spread-spectrum*.

Viene scelta una tecnica di *non-ritorno-a-zero (NRZ)* assegnando due differenti portanti sinusoidali per i bit 0 e 1.

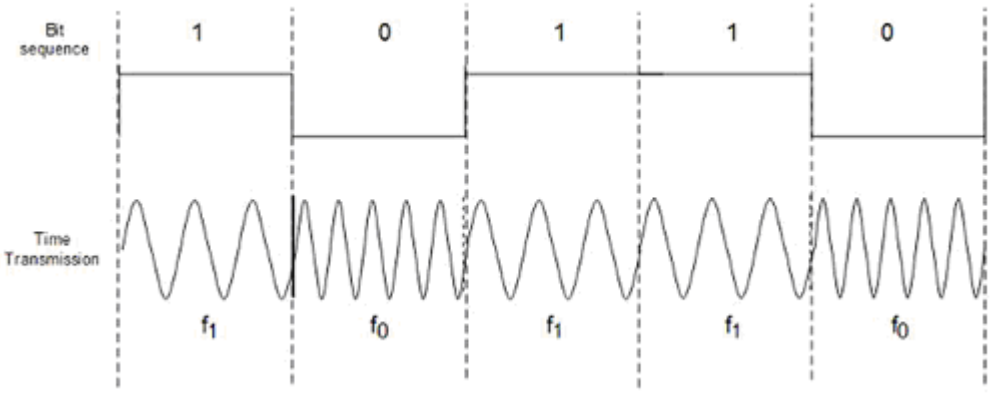

Figura 9: Forma d'onda segnale S-FSK (dominio del tempo)

È importante specificare che la differenza $|f_0 – f_1|$ sia almeno di 10kHz per diminuire la probabilità che i due toni interferiscano tra di loro evitando il fenomeno di *cross talk*.

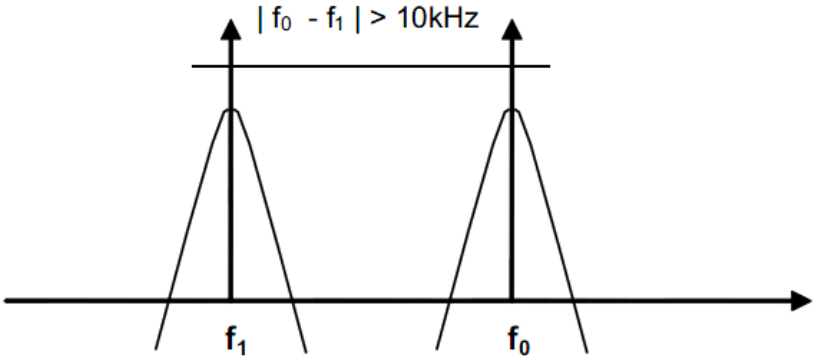

Figura 10: Forma d'onda segnale S-FSK (dominio delle frequenze)

Per la comunicazione i dati vengono sincronizzati con lo *zero-crossing* grazie al PLL interno, il tempo di bit è dinamicamente adattato per trasmettere sempre 24bit in ogni periodo e con la frequenza di 50Hz della rete, si trova come risultato una *bitrate* di 1200bit/sec.

Un'altra soluzione è di trasmettere 48bit in un periodo per arrivare ad una *bitrate* di 2400bit/sec.

Entrambe le configurazioni appena illustrate sono impostabili nel modem.

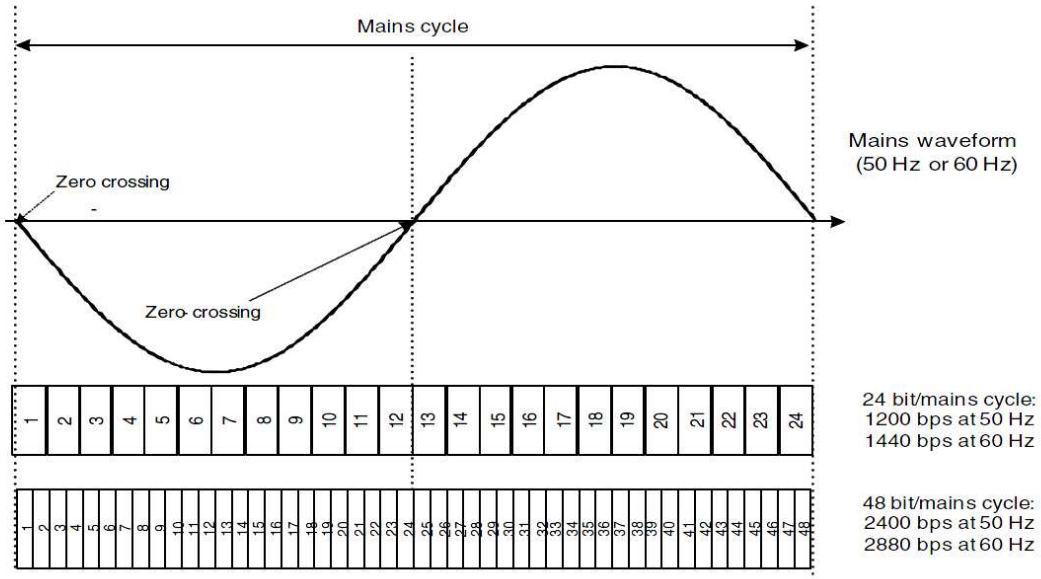

Figura 11: Bit timing

3.2 Protocol stack

L'ST7570 ha sviluppato al suo interno diversi livelli di accesso e funzioni [9]:

- Livello fisico: è implementato nel processore PHY e sviluppa tutte le funzioni primitive sviluppate dalla IEC 61334-5-1, in più ha funzioni aggiuntive per la configurazione, la gestione degli allarmi, la stima del rapporto segnale-rumore, il rilevamento dello sfasamento, e altre statistiche;
- Livello MAC: anche questo segue le funzioni della IEC 61334-5-1 aggiungendo servizi per la comunicazione;
- *Management Information Base* (MIB): è un database che contiene tutte le informazioni e le configurazioni per il funzionamento del sistema;
- *Host interface*: tutti i servizi PHY, MAC e MIB sono disponibili ad un *host* esterno attraverso una porta UART.

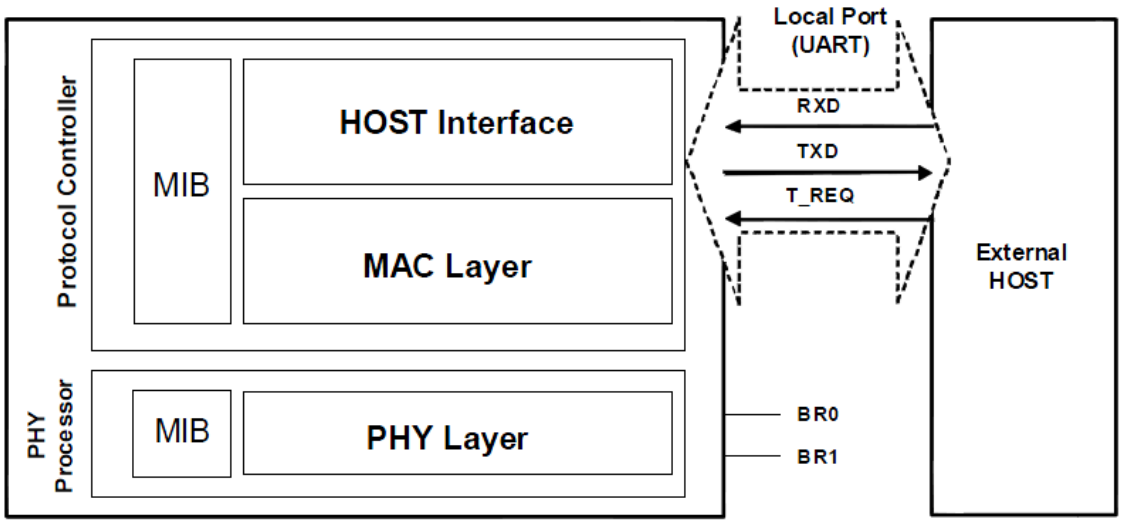

Figura 12: Panoramica delle funzioni

L'host che vuole comunicare con il modem può farlo in 2 modi: accedendo in maniera MAC o in maniera PHY.

3.3 Physical Layer

In questo paragrafo verrà spiegato come i modem interloquiscono tra loro, ossia come avviene la comunicazione tra il *layer* fisico e la *PowerLine.*

3.3.1 Pacchetto dati

Il *frame* formato a livello fisico si compone di 45 byte, dove i primi due sono di preambolo *(PRE)* e hanno valore *AAAAh*, i 2 successivi contengono lo *Start Subframe Delimiter (SSD)* con valore *54C7h*, poi comincia il pacchetto dati vero e proprio

composto di 38 byte contenente i dati forniti dal livello MAC; infine ci sono 3 byte di allarme o pausa. Da notare che i byte sono trasmessi da quello più significativo *(MSByte)* a quello meno significativo *(LSByte).*

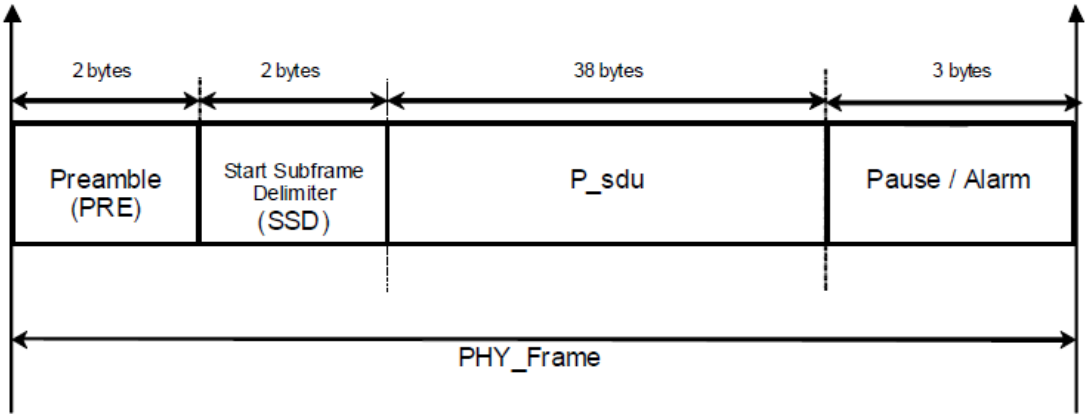

Figura 13: Formato del frame a livello fisico

Non appena viene ricevuta la sequenza di PRE+SSD da un nodo della rete, questo acquisisce la sincronizzazione.

È il *master* della comunicazione (*Client*) che spedisce la sincronizzazione sulla linea, solitamente allineandosi con un evento di *zero-crossing* sulla fase, mentre i nodi di tipo *slave* (*Server*) acquisiranno la sincronizzazione.

È fondamentale aver ricevuto correttamente la sincronizzazione altrimenti sarebbe impossibile avviare la comunicazione.

3.3.2 Modi Operativi

Come detto quindi ci sono due modi fondamentali di operare: come *Client*, ossia come *master* della comunicazione, o come *Server* che rappresenta lo *slave.*

Infatti, prima di poter cominciare una comunicazione, il modem configurato come *server* dovrà attendere di aver acquisito la sincronizzazione.

Ci sono inoltre altri modi di operare che non sono stati studiati come il *Monitor*, che è simile al server e può eseguire le funzioni di *debugging* sulla *Powerline* e la modalità *Test* per generare toni per testare la linea al fine di misurare le radiazioni elettromagnetiche prodotte dal segnale S-FSK.

3.3.3 Servizi disponibili

A livello PHY sono disponibili meno servizi rispetto a quello MAC ma possiamo distinguere:

- Servizio dati: per inviare e riceve dati sulla *Powerline*. Esistono tre tipologie di comandi che sono direttamente passati o dal *layer* MAC o da un *host* esterno: *Data.Request* per chiedere al livello fisico di inviare un pacchetto dati, *Data.Confirm* generato dal livello fisico in risposta alla *request* contenente il risultato della trasmissione (positivo o negativo), *Data.Indication* generata sempre dal livello fisico non appena viene ricevuto un dato sulla *Powerline.*

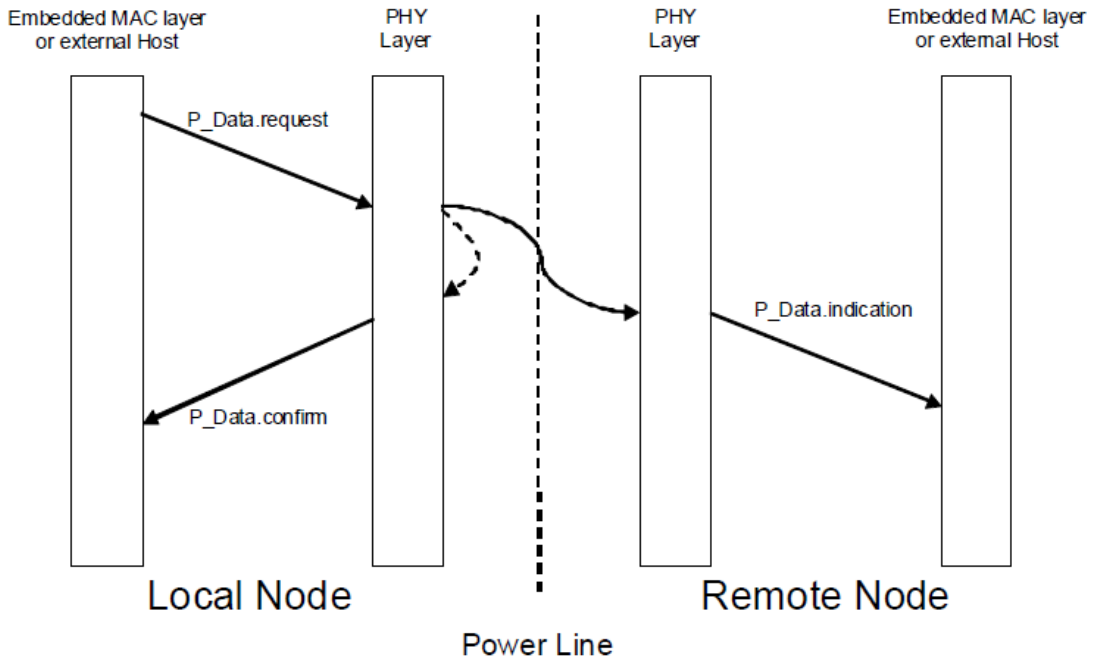

Figura 14: PHY servizio dati

Il servizio dati verrà usato per effettuare le prime prove di funzionamento del modem;

- Sevizio di allarme: può essere utilizzato per trasmettere messaggi di allarme sulla *Poweline*, ma nella presente fattispecie non se n'è mai reso necessario l'impiego;
- Servizio di sincronizzazione: gestisce l'acquisizione della sincronizzazione e si distinguono due funzioni: *Synch.Request* per rifiutare la sincronizzazione e *Synch.Indication* per archiviare la sincronizzazione.

Inoltre a livello fisico sono disponibili altri importanti servizi di tipo secondario:

- Stima del rapporto segnale-rumore: l'ST7570 calcola i livelli del segnale durante i byte di PRE e SSD e conserva tali valori in un oggetto MIB direttamente in dB. È possibile ricavare questi valori ogni volta che viene inviato un dato sulla *Powerline;*

- Determinazione dello sfasamento: anche qui durante i byte di PRE e SSD l'ST7570 misura la differenza di fase tra il riferimento in *zero-crossing* e la sequenza ricevuta. Il dato viene archiviato in un oggetto MIB ma anche reso subito disponibile dalla *Data.indication*.

La GUI è molto utile in questi due casi in quanto visualizza già sulla console i suddetti parametri.

È inoltre possibile impostare il guadagno in trasmissione grazie all'*Analog Front End* integrato del *chip.*

3.4 MAC Layer

Anche qui si distinguono due modi principali di operare: *Client* e *Server* e si individuano gli stessi servizi di dati, allarme e sincronizzazione, ma questi sono più complessi considerando che il *layer* MAC è ad un livello superiore rispetto al fisico e sono quindi disponibili molte più funzioni.

3.4.1 Servizi disponibili

- Servizio dati: viene usato per trasmette informazioni sulla linea. Le funzioni implementate sono le stesse di *Request*, *Confirm* e *Indication*, ma le informazioni al loro interno sono molte di più del semplice pacchetto dati del livello PHY. Il Servizio dati contiene ad esempio gli indirizzi di destinazione, la priorità e altri parametri.

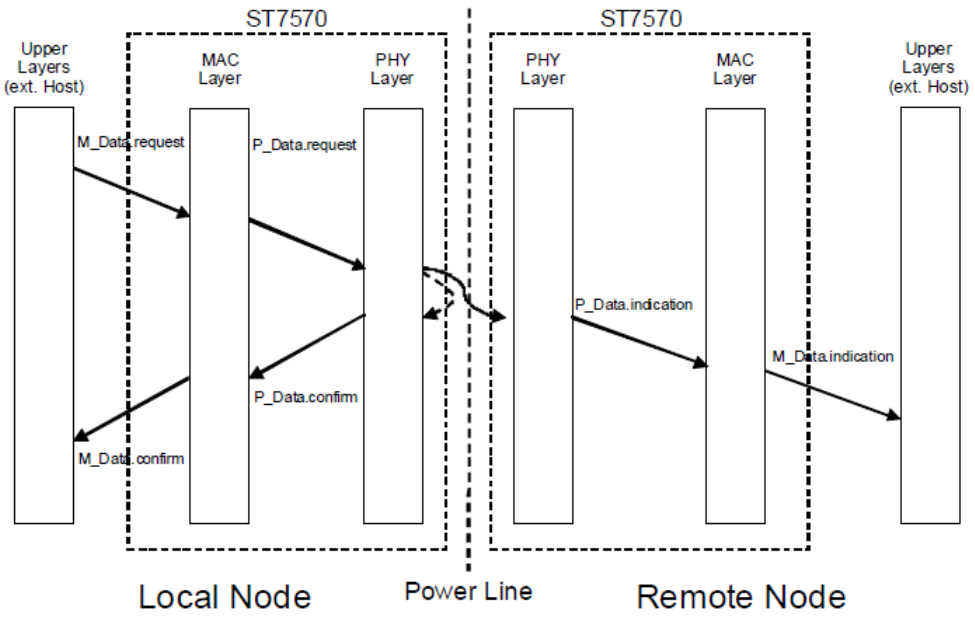

Figura 15: MAC servizio dati

- Servizio di allarme: del tutto simile al livello PHY.
- Servizio di sincronizzazione: del tutto simile al livello PHY.

3.4.2 Gestione degli indirizzi

Tra i campi del pacchetto dati MAC si distinguono due indirizzi: quello del **mittente** (SA) e quello del **destinatario** (DA) e sono entrambi a 12bit.

Infatti, ad ogni nodo della rete, viene destinato un indirizzo MAC configurabile tramite un oggetto MIB.

Il *range* degli indirizzi comincia con *000h* e termina con *FFFh* suddiviso così:

Value	Description
000h	Reserved
001h	Server
...	
FIMA-1	
FIMA	Client
...	
LIMA	
FFCh	Reserved
FFEh	New
FFFh	Reserved

Tabella 2: Indirizzi MAC

Ogni volta che viene inizializzato un modem, questo riceve automaticamente l'indirizzo *FFEh*, con la successiva configurazione si modifica l'indirizzo a seconda che il modo operativo sia *Server* o *Client.*

Tra gli oggetti MIB troviamo anche il *FIMA (Fist Initiator MAC Address)* che rappresenta il primo indirizzo disponibile per i *Client*, e il *LIMA (Last Initiator MAC Address)* ossia l'ultimo indirizzo disponibile per i *Client.*

Questi parametri devono essere configurati in ogni modem della rete nella stessa maniera affinché la comunicazione avvenga con successo.

Inoltre ogni modem imposta un "*Initiator MAC Address*" corrispondente al primo indirizzo dal quale poter riceve la sincronizzazione e quindi dati validi, che solitamente coincide con il primo indirizzo dei *Client* (FIMA).

3.4.3 Comportamento in trasmissione e in ricezione

Quando dall'*host* viene richiesta una trasmissione attraverso una *Data.Request*, il livello MAC copia l'indirizzo del mittente e quello del destinatario dentro il pacchetto dati.

In ricezione, tutti i server della rete confrontano l'indirizzo di destinazione presente nel pacchetto con il proprio indirizzo.

Si possono verificare i seguenti casi:

- l'indirizzo di destinazione coincide con il proprio o è il *broadcast* e l'indirizzo del mittente è uguale e/o superiore all'*Initiator MAC address* allora viene archiviata la sincroni zzazione e può avvenire lo scambio dati;
- in tutti gli altri casi la comunicazione non può avvenire.

Nel caso in cui si stata archiviata la sincronizzazione da parte di un server , scatta un timer per il *timeout* chiamato *"timeout-frame-not-OK"*. Questo *timeout* si aggiorna ogni qualvolta un *client* "chiama" un *server*. Durante il *countdown* il *server* può rispondere al *client* ma una volta scaduto questo, per poter trasmettere, dovrà aspettare di essere richiamato.

Il valore in secondi è configurabile in un oggetto MIB.

3.5 Interfaccia con l'HOST

Il modem ST7570 mette a disposizione una porta UART per una comunicazione *half duplex* asincrona usando un segnale per la ricezione *RXD* uno per la trasmissione *TXD* e uno per il controllo della comunicazione *T_REQ*.

Sono presenti altri 2 *input* BR0 e BR1 per configurare la *boudrate*, questi sono disponibili sul connettore a vaschetta presente sulla scheda e quindi non è configurabile via Software ma è necessario intervenire in modo Hardware.

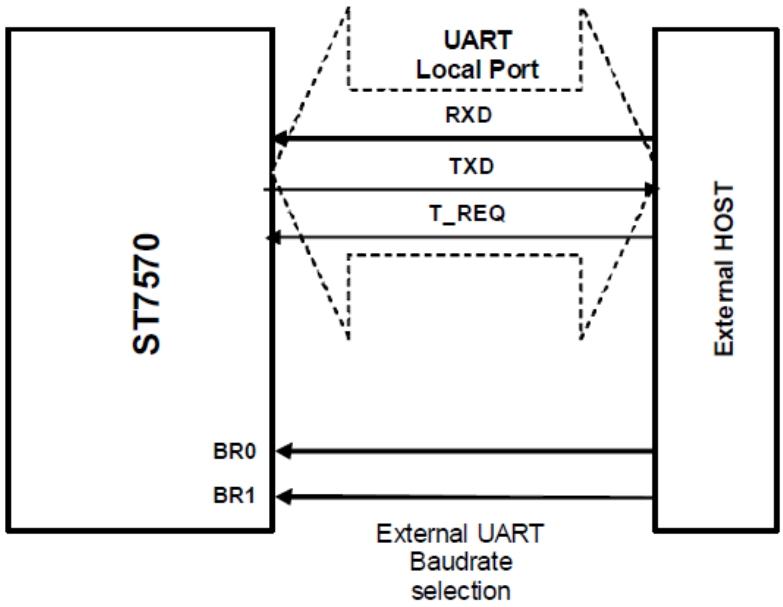

Figura 16: Comunicazione UART

Il formato del carattere trasmesso è il seguente: 8 bit di dati, uno di start e uno di stop, non è presente il bit di parità e usa la codifica NRZ.

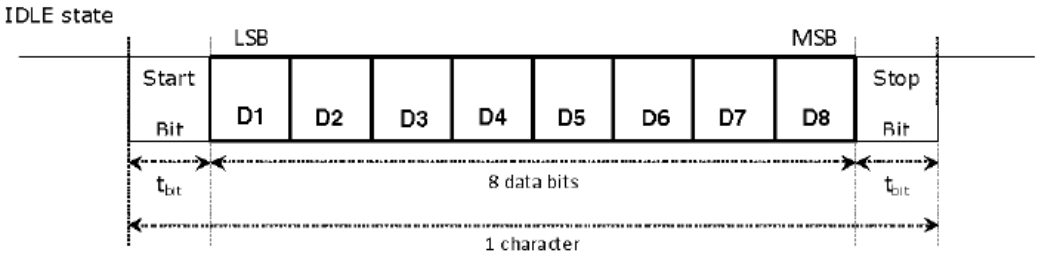

Figura 17: Formato carattere

3.6 Protocollo di comunicazione

Il protocollo di comunicazione implementa i seguenti compiti:

- definisce il formato del messaggio (o frame);
- implementa il meccanismo di ricezione secondo lo standard UART in *half duplex* con meccanismi di anticollisione;
- riconosce la ricezione dei *frame;*
- gestisce i *timeout;*
- controlla gli errori come la lunghezza del messaggio e l'errata sintassi .

I *frame* sono una sequenza di caratteri il cui contenuto rappresenta il dato scambiato tra *l'host* e l'ST7570.

Per ogni comando il protocollo definisce un formato per il *frame* e diversi passi da seguire per la comunicazione.

3.6.1 Local frame

Il formato dei *frame* scambiati tra *l'host* e ST7570 è composto da 5 campi principali:

Figura 18: Formato pacchetto dati locale

Campo	Lunghezza in Byte	Valore	Descrizione
STX	1	02h	Inizio del testo
Lenght	1	3..250	Lunghezza in byte di command + data + checksum
Command Code	1	0..FFh	Codice del comando
Data	0…247		Il dato
Checksum	2		Controllo degli errori

Tabella 3: formato pacchetto dati locale

Il campo dati è a sua volta suddiviso in sottocampi a seconda del comando; per i sottocampi che sono più lunghi di 1 byte verrà trasmesso prima il byte meno significativo.

Il *checksum* è calcolato come la somma byte a byte del *frame* escluso l'STX. Anche qui viene trasmesso prima l'LSByte.

3.6.2 Status message

Il messaggio di status è inviato dall'ST7570 ogni volta che *l'host* intende cominciare una comunicazione. È formato da 4 byte il cui contenuto dipende dalla configurazione del modem. In esso sono presenti molte informazioni utili sullo stato del modem come il modo operativo (*Client* o *Server*), a quale *layer* si ha accesso, (PHY o MAC), se il modem è sincronizzato o meno, lo sfasamento e altri parametri.

Tutte queste informazioni sono contenute negli ultimi 3 byte mentre il primo vale sempre il carattere "*?*".

3.6.3 Acknowledgement message

Dopo aver ricevuto il local frame, sia l'ST7570 che l'*host* devono rispondere all'altro dispositivo inviando sulla linea (TXD per il modem o RXD per l'host) un messaggio di risposta. Questo è lungo 1 byte e può valere:

Simbolo	Significato	Valore
ACK	Acknowledgement	06h
NAK	Not Acknowledgement	15h

Tabella 4: Codice ACK e NAK

Verrà inviato un ACK dall'ST7570 quando la lunghezza e il checksum sono entrambi corretti, in caso contrario verrà inviato un NAK. Lo stesso avviene per l'*host*: dovrà controllare se la lunghezza e il checksum del frame inviato dal modem e rispondergli con un ACK o un NAK.

3.6.4 Regole della comunicazione

Quando la comunicazione avviene dall'*host* all'ST7570, come primo passo deve essere impostato il pin T_REQ a livello basso, in risposta a questo il modem spedirà sulla linea il messaggio di *status* descritto in precedenza. Una volta ricevuto lo *Status message*, *l'host* può spedire il *frame*: è necessario rialzare la linea T_REQ subito dopo aver trasmesso il carattere STX del *frame* altrimenti l'ST7570 ignorerà il messaggio.

Dopo aver spedito il frame sulla linea RXD, l'ST7570 risponderà con l'ACK o con il NAK. Se la risposta è un NAK sarà necessario ripetere la sequenza.

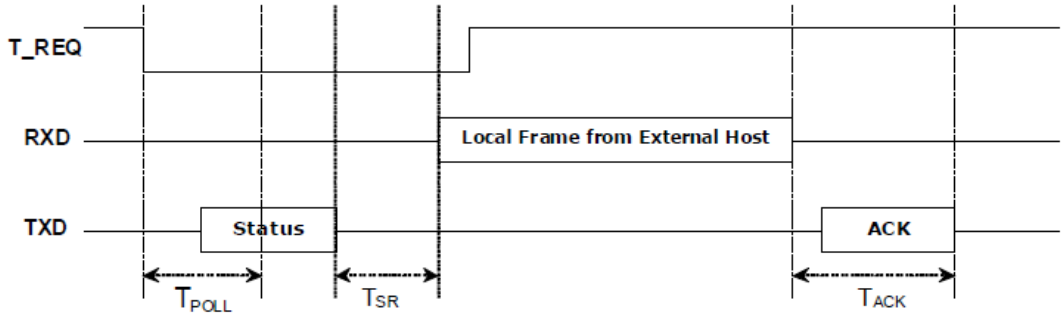

Figura 19: Flusso dati dall'Host all'ST7570

Se invece è l'ST7570 che ha bisogno di trasmettere un frame all'*host*, non abbasserà la linea T_REQ ma presenterà direttamente il frame sulla linea TXD. Sarà dunque l'*host* a rispondere con un ACK o NAK controllando la lunghezza e il

checksum: nel caso di risposta NAK il modem ripeterà il frame solo una volta. Se *l'host* non risponderà allora il modem lo interpreta come un ACK.

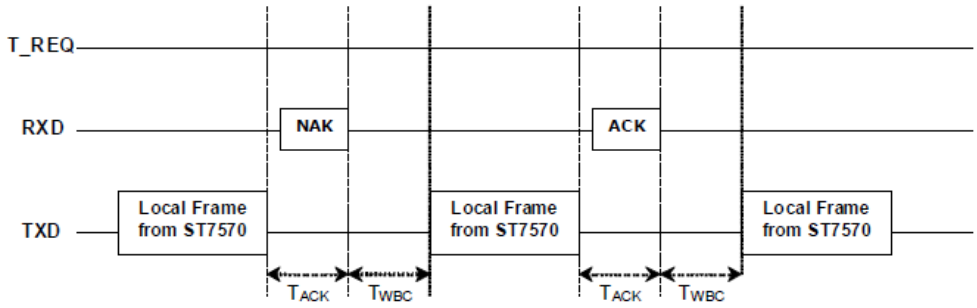

Figura 20: Flusso dati dall'ST7570 all'host

Si ricorda che i dati provenienti dall'UART sono di tipo asincrono quindi i protocolli dell'ST7570 devono formare un flusso di *frame* da un flusso di dati. Un parametro importante da tenere in considerazione è l'intervallo di tempo tra un carattere e l'altro che non deve superare un T_{IC} impostato di default 10ms, ma può essere cambiato.

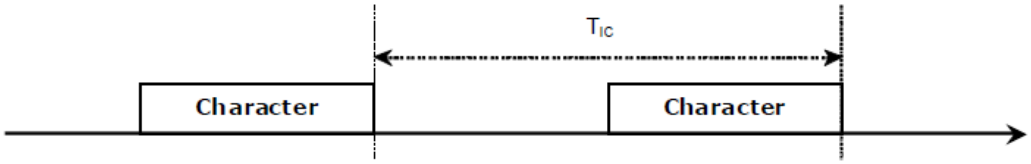

Figura 21: Timeout T_{ic}

Altri tempi importanti da rispettare sono:

- T_{POLL}: tempo entro il quale il modem risponde lo *status message*
- T_{SR}: tempo massimo in cui cominciare a trasmettere il frame dopo aver ricevuto lo *status message*
- T_{ACK}: tempo entro il quale il modem o *l'host* devono rispondere la conferma ACK
- T_{WBC}: tempo dopo il quale si può trasmettere un nuovo frame sia da parte dell'*host* che dal modem, dopo aver ricevuto un NAK o ACK

Timeout	Tempo in ms
T_{POLL}	20
T_{SR}	200
T_{ACK}	40
T_{WBC}	5
T_{IC}	10

Tabella 5: Valori dei timeout

3.7 MIB Management Information Base

Esiste una tabella all'interno dell'ST7570 in cui vengono raccolti tutti i parametri (o oggetti MIB) che permettono all'*host* di controllare il modem. In questa troviamo alcuni dei valori precedentemente analizzati come gli indirizzi FIMA e LIMA, i timeout, i rapporti segnale rumore e in generale le configurazioni della PLC come il modo operativo (PHY o MAC), la configurazione (*Client* o *Server*), la frequenza dei toni del segnale S-FSK e molti altri.

Indirizzo	Nome	Valore di default
0000h	FIMA / LIMA	0C00h / 0DFFh
0001h	LocalMacAdd/InitMacAdd	NEW (0FFEh) / NOBODY(0000h)
0002h	Timeout synchro confirmation	3 s
0003h	Timeout frame not ok	40 s
00A1h	PLC configuration	PHY mode, IDLE state, bit rate 1200 bits/s, ZC PLL 50 Hz, f0=74.0kHz, f1=63.3 kHz, current control disabled
…	…	…

Tabella 6: Tabella oggetti MIB

3.8 Comandi

Nel seguente capitolo verranno illustrati nel dettaglio i comandi usati per la comunicazione da PC a modem. I comandi in realtà sono molti di più, ma per il fine del progetto non sono stati utilizzati quindi non saranno spiegati.

Ogni comando è identificato da un codice detto *Command Code* di lunghezza un byte.

Si possono distinguere 3 differenti tipi di comando:

- Comandi in cui è l'*host* che richiede di usare l'ST7570;
- Comandi di conferma e di errore spediti dall'ST7570 in risposta a una richiesta precedentemente avvenuta;
- Comandi di indicazione inviati dall'ST7570 per informare l'*host* del che è avvenuto un cambiamento del suo servizio.

In tabella sono mostrati i comandi usati:

Gruppo di appartenenza	Comando	Codice
Sincronizzazione	CMD_SynchroIndication	10h
MIB	CMD_WriteDBRequest	41h
	CMD_WriteDBConfirm	42h
	CMD_WriteDBError	43h
Data	CMD_DataIndication	50h
	CMD_DataRequest	51h
	CMD_DataConfirm	52h

Tabella 7: Lista dei comandi

3.8.1 CMD_WriteDBRequest (41h)

Per configurare gli oggetti MIB all'interno della tabella si usa il comando ***CMD_WriteDBRequest***, è una delle operazioni da eseguire prima di iniziare la comunicazione sulla *Powerline*.

Questo è composto di due campi: il primo di due byte contiene l'indirizzo dell'oggetto che si vuole modificare, il campo successivo varia la sua lunghezza a seconda di cosa si vuole configurare. Anche qui viene trasmesso prima l'LSByte.

Se per esempio si volesse cambiare la "PLC configuration" il comando assumerebbe questa forma:

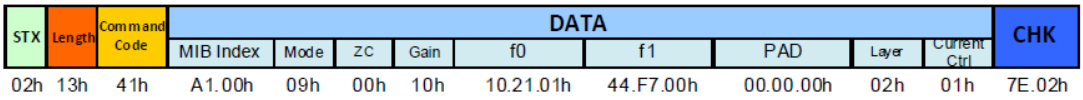

Figura 22: Esempio formato WriteDBRequest

Dalla forma del frame si possono notare tutti i campi descritti in precedenza: partendo dall'STX che ha sempre lo stesso valore, si aggiunge un byte indicante la lunghezza del messaggio, successivamente si inserisce il codice del comando. A questo punto si specificano nei primi due byte del dato l'indirizzo dell'oggetto MIB che si vuole modificare poi si specificano tutte le configurazioni del modem.

In risposta ad un comando di scrittura corretto, il modem invia una ***CMD_WriteDBConfirm*** (42h) che contiene lo stesso pacchetto dati della request con il *Comand Code* ovviamente diverso, perciò sarà diverso anche il *checksum*.

Se invece viene ricevuto un messaggio errato dal modem, esso rifiuterà la richiesta di modifica e risponderà con una ***CMD_WriteDBError*** (43h) che non contiene nessun tipo di dato.

3.8.2 CMD_SynchroIndication (10h)

È generato dall'ST7570 per indicare all'*host* che è avvenuto un cambio nel suo stato di sincronizzazione. Si differenziano due tipologie di questo comando a seconda se si sta lavorando in modo PHY o MAC:

- In *PHY mode* questo è generato ogni volta che viene ricevuta una sequenza PRE+SSD e il suo contenuto indica l'ampiezza del segnale, l'ampiezza del rumore, il guadagno del blocco PGA e lo sfasamento.
- In *MAC mode* viene generato quando si verifica un cambiamento nello stato della sincronizzazione quindi anche quando scadono i *timeout*.

Il contenuto dati in *MAC mode* è più articolato in quanto specifica oltre alle informazioni relative al *PHY mode* lo stato in cui si trova la sincronizzazione (trovata o persa), la causa che ha scaturito questo messaggio (timeout ecc..) e anche l'indirizzo del mittente.

3.8.3 CMD_DataRequest(51h)

Questo comando è inviato dall'*host* al modem ed è il cuore della comunicazione su *Powerline*: è infatti grazie a questa funzione che i dati possono essere inoltrati sulla linea da parte dell'*host*.

Anche qui si può differenziare a seconda del servizio attivo:

- In *PHY mode* nel pacchetto dati si inseriscono solamente i caratteri da voler inoltrare sulla linea che necessariamente devono essere 38; il frame assumerà questa forma:

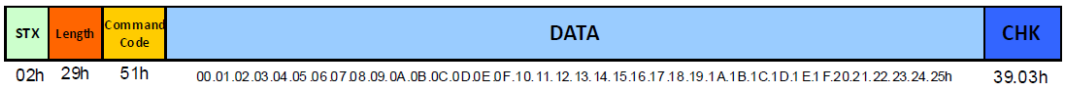

Figura 23: Esempio formato DataRequest a livello PHY

- In *MAC mode* si possono impostare l'indirizzo di destinazione e quello del mittente cosicché il dato venga ricevuto solo dall'utente designato semplificando molto il lavoro compiuto dal meccanismo di ricezione dati del software di tutti gli apparati LIN: infatti questi non dovranno controllare se ogni pacchetto che arriva se è indirizzato a loro, perché verrà loro inoltrato solo se sono i destinatari.

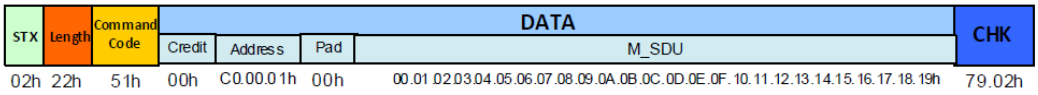

Figura 24: Esempio formato DataRequest a livello MAC

3.8.5 CMD_DataConfirm (52h)

In risposta al comando di *DataRequest*, l'ST7570 provvede a inviare una conferma dell'avvenuto inoltro del messaggio. In suo contenuto dati è di solo 1 byte e contiene il risultato della richiesta di invio dati:

Servizio	Descrizione
PHY	*LP_TU* (00h): PHY layer is busy *LM_SE* (03h): Length not correct on DataRequest *LP_NS* (04h): Modem is not mains-synchronized *LP_NOT_VALID* (06h): No data request pending or error at physical layer *LP_OK* (FFh) : No error
MAC	*LM_TU* (00h): MAC layer is busy *LM_NI* (01h): Not available type *LP_HF* (02h): Error at physical level *LM_SE* (03h): Length not correct on DataRequest *LM_NS* (04h): Modem is not synchronized *LM_IS* (05h): Modem is during Intelligent Synchronization Search *LM_NOT_VALID* (06h): No data request pending *LM_OK* (FFh): No error

Tabella 8: Lista risposte DataConfirm

Tra le risposte più frequenti si trova *LM_TU* che si verifica quando si cerca di trasmettere dei messaggi troppo repentinamente e non si da il tempo al MAC (o PHY) *layer* di completare l'inoltro del messaggio precedente; se si è configurati come Server e si cerca di inviare qualcosa senza aver ottenuto prima la sincronizzazione o semplicemente perché è scaduto un timeout, si troverà come risposta *LM_NS.*

Se invece la comunicazione va a buon fine, il modem risponderà con un *LM_OK.*

3.8.4 CMD_DataIndication (50h)

È inviato dall'ST7570 per indicare all'*host* la ricezione di un dato sulla linea. Anche qui il contenuto si differenzia a seconda del servizio attivo:

- *PHY*: nel pacchetto dati si trovano subito i caratteri trasmessi che sono necessariamente 38, poi sono presenti i rapporti segnale rumore.
- *MAC*: qui invece si trovano subito gli indirizzi del mittente e del destinatario e poi i dati veri e propri il cui numero di caratteri può variare fino a 242.

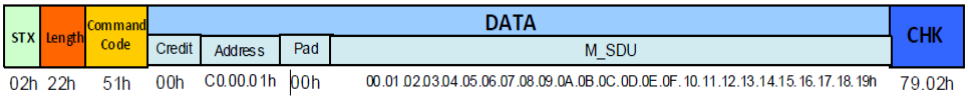

Figura 25: Esempio formato DataIndication a livello MAC

4. Progetto Software

4.1 Prove con la GUI

Come primo passo si è testato il funzionamento del modem tramite la **GUI.**

La GUI è molto utile per prendere confidenza con i protocolli di comunicazione del modem perciò da subito si configurano tutte le impostazioni di base.

Per provare una trasmissione vera e prioria è necessario utilizzare un'altra scheda e collegarla o ad un altro PC dotato di GUI o allo stesso PC operando sul medesimo programma: infatti questo permette di gestire più modem contemporaneamente.

Configurando anche l'altro modem si sono svolte diverse prove di funzionamento e trasmissione dati per verificare l'efficacia dell'ST7570.

Questi test hanno avuto una durata molto breve ma la GUI è stata ripresa alla fine del progetto per eseguire i test sulla robustezza della linea.

4.2 Funzioni della Porta Seriale

Tutte le funzioni per la comunicazione su porta seriale sono definite della libreria *termios.h* per cui è necessario includerla nel file di progetto.

Le funzioni utilizzate sono le seguenti [8]:

- ***open***: funzione utilizzata per accedere alla porta seriale. Prende in ingresso il nome della porta e i *flag* che indicano la modalità di accesso (lettura e/o scrittura) e ritorna un valore intero che indica se il processo di apertura è andato a buon fine;
- ***struct termios option***: viene creata una *struct* per impostare tutti i parametri della comunicazione seriale come la *bouderate*, il controllo di flusso, la parità, i bit di dato, i bit di start e stop.

Pper aprire e settare la porta le prime istruzioni da eseguire sono:

```
…
fd = open("/dev/ttyUSB2", O_RDWR | O_NOCTTY | O_NDELAY);
set_port(fd);
…
void set_port(int port){
      struct termios options;
      tcgetattr(port, &options);//si ottengono le info della porta
      cfsetispeed(&options, B57600);      // set BR input
      cfsetospeed(&options, B57600);      // set la BR output
      options.c_cflag |= (CLOCAL|CREAD);  //abilito la ricezione

      //No parity, 8bits, 1 stop bit (8N1)
      options.c_cflag &= ~CSIZE;
      options.c_cflag &= ~CSTOPB;
      options.c_cflag &= ~PARENB;
      options.c_cflag |= CS8;
      options.c_cflag &= ~CRTSCTS;

      //input non canonico
      options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
      //disabilito il ctrl flusso SW
      options.c_iflag &= ~(IXON | IXOFF | IXANY);
      options.c_oflag |= OPOST;//&= ~OPOST;
      //lunghezza minima byte che ci si aspetta dalla porta
      options.c_cc[VMIN] = 1;
      //tempo in cui rimane ad ascoltare la linea in Read in
      decimi di secondo
      options.c_cc[VTIME] = 4;
      tcsetattr (port, TCSANOW, &options); //salvataggio
      return;
}
```

- ***ioctl*:** ha molteplici funzioni ma in questo progetto è usata per alzare e abbassate la linea T_REQ corrispondente al pin RTS della porta seriale

```
ioctl(fd, TIOCMBIC, &sercmd);            //alzo RTS
ioctl(fd, TIOCMBIS, &sercmd);            //abbasso RTS
```

- ***write***: per scrivere dati sulla porta si usa questa funzione di semplice utilizzo. Questa ritorna il numero dei byte trasmessi o *-1* se la trasmissione ha fallito; in ingresso prende una stringa di caratteri e il numero di caratteri da inviare. In questo progetto alla funzione viene passato un byte alla volta perché è stato constatato che trasmettendo troppi caratteri la comunicazione fallisce.

```
str[0] = carattere;
write(fd, str, 1);     //scrivo il carattere sulla porta
```

  Si crea quindi un *array* formato da un solo carattere in cui si pone il valore da voler inoltrare, poi si passa questo *array* alla funzione specificando che si vuole trasmettere solo un carattere.

- ***read***: funzione per leggere i dati dalla porta. Prende in ingresso un *array* in cui vengono scritti i dati e la quantità di dati da ricevere. Ritorna il numero di dati

ricevuti e se il risultato equivale a *-1* la ricezione fallisce. Per lo stesso motivo della funzione *write* anche qui si riceve un byte per volta.

```
char rec[1];
n1 = read(fd, rec, 1);        //ricevo il carattere dalla porta
```

Il meccanismo di ricezione è più complesso di quello dell'inoltro. Nelle impostazioni della porta sono presenti due variabili: *options.c_cc[VMIN]* tramite la quale si può impostare il numero minimo di byte che la porta si aspetta di dover riceve e *options.c_cc[VTIME]* nella quale si imposta il tempo in decimi di secondo in cui la funzione *read* rimane ad aspettare sulla porta i caratteri dal momento in cui la si chiama. Il primo valore verrà impostato a 1, mentre per il secondo valore si veda al **capitolo 3.6.4** dove sono riportati tutti i tempi.

Si trova un valore ottimo di 40ms corrispondente al tempo massimo che può impiegare il messaggio di *ACK* ad essere inviato dal modem dopo un comando inviato dall'host.

- ***close***: utilizzato per chiudere la porta una volta conclusa la comunicazione.

```
close(fd);                    //chiudo la porta
```

4.3 Funzioni secondarie

Questa famiglia di funzioni ha lo scopo di "alleggerire" il codice: sfruttano a loro volta le funzioni della libreria relative alla comunicazione su porta seriale e ne creano altrettante che ricevono in ingresso meno variabili allo scopo di avere un notevole risparmio nel riuso del codice con ovvii vantaggi per il programmatore.

4.3.1 Invio dei dati sulla porta seriale

Questa semplicissima funzione riceve in ingresso un carattere che poi verrà posto in un *array* composto da un singolo elemento e quindi dato alla funzione *write* che lo inoltra sulla porta.

```
void send(char carattere){
        char str[1];
        str[0] = carattere;
        write(fd, str, 1);
}
```

4.3.2 Ricezione del pacchetto dati dalla porta seriale

Questa funzione è più complessa del semplice invio: essa restituisce un puntatore ad una stringa contenente il pacchetto dati ricevuto. Viene chiamata ogni qualvolta si intende ricevere dati dalla linea ossia quando si attende un messaggio dal modem ST7570.

La funzione si ferma subito in un ciclo *while* sfruttando la funzione *read*, aspettando che sulla porta si presenti il carattere STX. Una volta ricevuto il primo byte esegue un'altra *read* con la quale riceve il byte indicante la lunghezza del messaggio. In seguito è implementato un ciclo *for* che crea un *array* riempito che il ciclo riempie con i rimanenti byte del messaggio. Come ultimo carattere si pone il terminatore *\n*. Infine viene inviato come risposta il carattere *ACK* e si ritorna *l'array* appena creato.

```
unsigned char* receiv_data(){
        char rec[1]={0};
        while(rec[0]!=STX){
                n1 = read(fd, rec, 1);
        }
        n1 = read(fd, size_array, 1);
        size = size_array[0];
        for(i=0;i<size;i++){
                n1 = read(fd, rec, 1);
                risposta[i]=rec[0];
        }
        risposta[i+1] = '\n';
        send(ACK);
        return risposta;
}
```

4.3.3 Ricezione del messaggio STATUS

Come detto in precedenza, il primo *step* per iniziale la comunicazione con l'ST7570 è abbassare la linea T_REQ, successivamente ci si pone in ascolto aspettando il messaggio di STATUS, poi si spedisce il primo carattere del pacchetto dati che corrisponde a STX e infine si rialza la linea.

```
void init_comunicazione(){
    ioctl(fd, TIOCMBIS, &sercmd);       //abbasso RTS
    while(fist[0]!=0x3f){               //aspetto il primo byte di status
        n1 = read(fd, trans, 1);
    }
    n2 = read(fd, buff1, 3);            //ricevo gli altri byte
    usleep(1000);
    send(STX);                          //invio il primo byte del frame = STX
    ioctl(fd, TIOCMBIC, &sercmd);       //rialzo RTS
}
```

Viene creata una funzione ***void*** *init_comunicazione****( )*** che dopo aver abbassato la linea RTS aspetta con un ciclo *while* che gli venga presentato il primo carattere dello *status message* sulla porta di ingresso, poi riceve gli altri 3 byte, invia il primo carattere STX del pacchetto ed infine rialza la linea RTS come da protocollo.

4.4 Main

Il programma vero e proprio comincia aprendo la porta con la funzione *open*: viene subito controllato il risultato della funzione in quanto può scaturire un errore dovuto ad un errata impostazione della porta.

Successivamente si settano tutti i suoi parametri chiamando la struttura *set_port* descritta in precedenza.

Da questo momento comincia la comunicazione con il modem inviando subito le *PLC configurations* attraverso un'opportuna funzione chiamata in questo modo:

```
//configuro il modem a livello PLC
do{
        risultato = PLC_configuration(CLIENT,MAC);
}while(risultato == 0);
```

Viene eseguito un ciclo *do-while* perché la funzione ritorna il risultato dell'inoltro: in caso positivo (risutato = 1) si esce dal ciclo, in caso negativo (risultato = 0) viene ricevuto un NAK e la trasmissione delle configurazioni avviene nuovamente.

La sequenza di istruzioni da eseguire è illustrata nella figura 26:

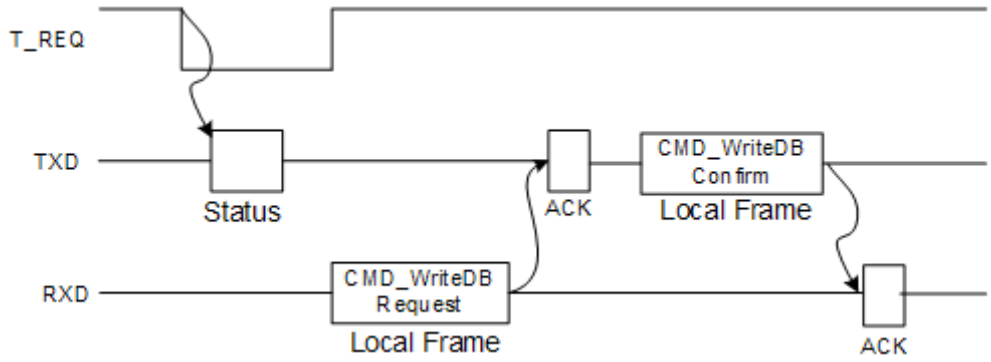

Figura 26: Sequenza istruzioni per la comunicazione dall'host all'ST7570

Come per ogni invio da parte *dell'host*, si riceve lo *status message*, poi si invia una *CMD_WriteDBRequest* e si aspetta *l'ACK*, quindi una *CMD_WriteDBConfirm* e infine si deve inoltrare *l'ACK*.

Si osservi nel dettaglio la funzione *PLC_configuration( )*: vengono dati in ingresso due parametri che indicano il servizio al quale si accede (*MAC* o *PHY*) il modo di configurazione (*Server* o *Client*); viene creato un *array* contenente il pacchetto dati dell'istruzione che ha il formato riportato nella figura 22.

Si inizializza la comunicazione usando la funzione *init_comunicazione( )* in cui si invia il carattere *STX*, quindi viene calcolata e spedita la lunghezza da inserire nel campo *lenght*, poi viene spedito il *Command Code*, si procede con il calcolo della *checksum* sommando tutti gli elementi del pacchetto e si spedisce trasmettendo prima l'*LSByte*. A questo punto si attende la risposta con un ciclo *while* e si controlla il suo valore sfruttando la funzione *swith-case:* si risponderà in maniera positiva solo se il risultato sarà un *ACK* dopo avere ricevuto anche la *CMD_WriteDBConfirm*.

```
int PLC_configuration(unsigned char mod, unsigned char how){
        unsigned char data[16]= {0xa1,0x0,mod,0x0,0x10,0x10,0x21,
        0x01,0x44,0xf7,0x0,0x0,0x0,0x0,how,0x0};
        init_comunicazione();
//invio la lunghezza:
        send(sizeof(data)+3);
//invio il comando più il dato
        send(CMD_WriteDBRequest);
        for(i=0;i<sizeof(data);i++){
                send(data[i]);
        }
//ora calcolo la checksum e la spedisco
        checksum = sizeof(data)+ 3 + CMD_WriteDBRequest;
        for(i=0;i<sizeof(data);i++){
                checksum = checksum + data[i];
        }
        check_array[0]=checksum;
        check_array[1]= (checksum>>8);
        send(check_array[0]);
        send(check_array[1]);
//ora aspetto la risposta
        w=0;
        while(w==0){
                n1 = read(fd, ack, 1);
                if(ack[0]!= (ACK | NAK)){
                        w = 0;
                }else{
                        w = 1;
                }
        }
        switch(ack[0]){
        case NAK:
                printf("ack: NAK\n");
                usleep(60000);
                return 0;
        case ACK:
//ho ricevuto l'ACK  aspetto la conferma con un CMD_writeDBConfirm
                printf("ack: ACK \n");
                receiv();
                return 1;
        default:
                usleep(60000);
                return 0;
        }
}
```

Dopo aver trasmesso le configurazioni PLC si devono configurare gli indirizzi sfruttando sempre il comando *CMD_WriteDBConfirm* ma puntando ad un altro oggetto MIB e quindi il comando avrà questa forma:

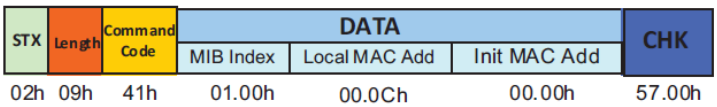

Figura 27: Esempio formato WriteDBConfirm

Anche qui la funzione è chiamata con un ciclo *do-while* per gli stessi motivi dell'altra configurazione passandogli in ingresso questa volta il proprio indirizzo e l'*Initiator MAC Address*:

```
//configuro il modem a livello MIB (indirizzi)
do{
      w = config_MAC_address(0x0C00,0x0000);
}while(w == 0);
```

La struttura della funzione è circa la stessa della precedente in quanto il comando coincide, cambia solo il contenuto del pacchetto dati.

Si crea perciò un *array* contenente gli indirizzi dividendo opportunamente in byte, poi si eseguono le stesse operazioni dell'altra funzione.

```
int config_MAC_address(unsigned int localMACaddress,unsigned int
initMACadress){
      unsigned char data[6] = {0x01,0x00,localMACaddress,
      localMACaddress>>8,initMACadress,initMACadress>>8};
      init_comunicazione();
      …
}
```

Quelle appena descritte sono le funzioni che devono eseguire sia i *Server* che i *Client* e coinciderà per entrambi i modem, tranne che per gli indirizzi che dovranno essere assegnati rispettando la tabella 42.

4.4.1 Descrizione del funzionamento del programma di esempio

Per simulare una comunicazione si configurano un *client* e un *server* e successivamente si aggiunge un terzo *server* per testare il collegamento.

Il programma si divide in due parti: la prima consiste nel far inviare dal *client* un comando al *server* chiedendo i dati dei sensori simulati o settando la luminosità di una lampada simulata; nella seconda il *server* risponderà al client i dati desiderati.

4.4.2 Client

Si analizzino ora le singole fasi della comunicazione descrivendo il codice in C implementato per il client:

```
while(1){
    int input[1];
    printf("selezionare modalitÃ : 1 cambiare lum., 2 chiedere
    stato -> ");
    fscanf(stdin, "%d", input);
    How_do=input[0];
    if(How_do == 1){
         printf("selezionare luminositÃ  %% -> ");
         fscanf(stdin, "%d", input);
         Lux_out=input[0];
    }
    do{
//invio al LIN il primo banco di dati dicendogli/chiedendogli cosa
fare
          risposta = call_LIN();
    }while(risposta == 0);
    printf("ok_call_LIN\n\n");

//aspetto la risposta dal LIN con i dati
    data_from_LIN();
}
```

Dopo aver inviato le configurazioni al modem ha inizio la *routine* principale in cui viene chiesto all'utente quale delle modalità sopra descritte vuole utilizzare. In seguito viene chiamata la funzione *call_LIN( )* con la quale si inviano i dati al server ed infine si attende che il server risponda con la funzione *data_from_LIN( ):*

- ***call_LIN( )*:** questa funzione segue lo schema riportato in figura 28:

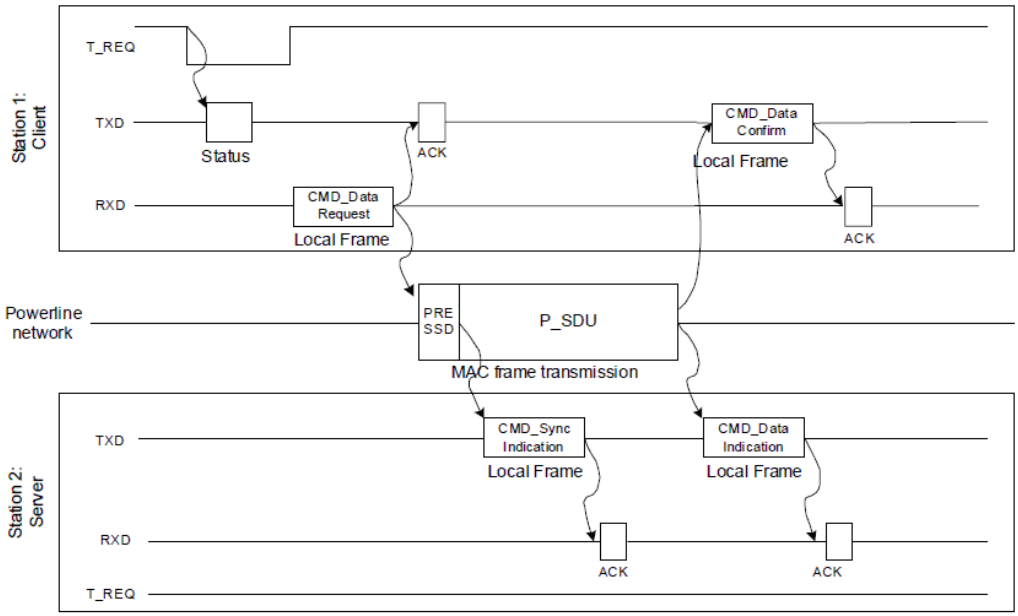

Figura 28: Diagramma dei tempi per la comunicazione quando è il client a iniziare la comunicazione

La funzione torna il risultato della trasmissione e non prende nessun dato in ingresso poiché questi sono assegnati in variabili globali; si procede alla prima fase chiamando la funzione *init_comunicazione( )* che acquisisce lo *status message* poi si invia il comando *DataRequest* specificando gli indirizzi di mittente e destinatario e il pacchetto

dati contenente la modalità di accesso al server (se cambiare la luminosità o acquisire i dati dai sensori); si aspetta la risposta e se equivale ad un ACK allora si attende la *DataConfirm* dal modem tramite la funzione *receiv_data( )* trasmettendo anche l'ACK di conferma.

```
int call_LIN(){
   unsigned char pre_data[5] = {0x00,0xC0,0x00,0x01,0x00};
   init_comunicazione();        //inizializzo la comunicazione
   send(1 + 5 + lenght_dato + 2
   send(CMD_DataRequest);
   for(i=0;i<5;i++){
        send(pre_data[i        }
        M_SDU[0]= How_do ; M_SDU[1]= Lux_out; M_SDU[2]= 0xAA;
        for(i=0;i<lenght_dato;i++){
              send(M_SDU[i]);
        }
        checksum = 1 + 5 + lenght_dato + 2 + CMD_DataRequest;
        for(i=0;i<5;i++){
              checksum = checksum + pre_data[i];
        }
        for(i=0;i<lenght_dato;i++){
              checksum = checksum + M_SDU[i        }
        check_array[0]=checksum;
        check_array[1]= (checksum>>8);
        send(check_array[0]);
        send(check_array[1
        ack[0] = 0; w=0;
        while(w==0){
                n1 = read(fd, ack, 1);
                if(ack[0]!= (ACK | NAK)){
                        w = 0;
                }else
                        w = 1;
                }
        }
        switch (ack[0]){
        case NAK:
                printf("ack: NAK\n");
                usleep(50000);
                return 0;
        case ACK:
//ho ricevuto l'ACK mi aspetto la conferma con WriteDBConfirm
                unsigned char* ric;
                ric = receiv_data();
                switch(ric[1]){
                case 0xff:
                        return 1;
                default:
                        usleep(50000);
                        return 0;
                }
        default:
                usleep(50000);
                return 0;
        }
}
```

- ***data_from_LIN( )***: grazie a questa funzione il client riceve i dati dal server seguendo lo schema di figura 29:

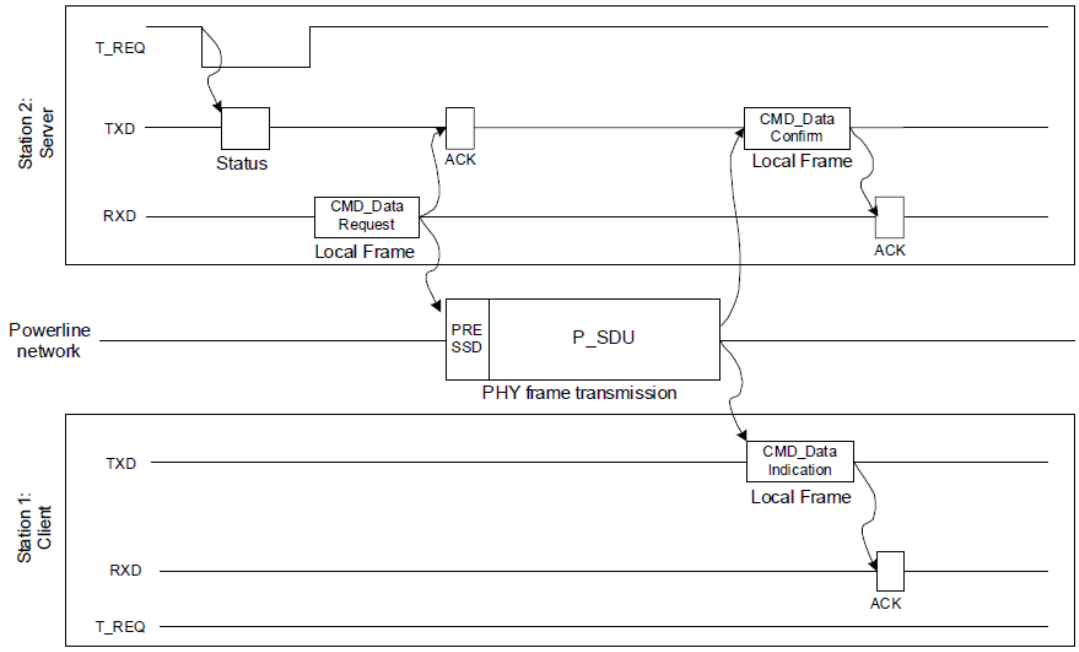

Figura 29: Diagramma dei tempi per la comunicazione quando è il server client a iniziare la comunicazione

La funzione chiama a sua volta *receiv_data( )* e controlla quale comando ha ricevuto. In questo semplice caso può riceve solo una *DataIndication* quindi aggiorna i parametri dei sensori e della luminosità nelle variabili globali e li stampa a video.

```
void data_from_LIN(){
        unsigned char* ric;
        ric = receiv_data();
        switch (ric[0]){
        case CMD_DataIndication:
                stato = ric[6];
                LED = ric[7];
                temperatura = ric[10];
                CO = ric[9];
                CO2 = ric[10];
                umidita = ric[11];
                printf("stato = %d, LED = %d%%, temp. = %dÂ°, CO =
                %dg/cmÂ², CO2 = %dg/cmÂ², umidita = %dg/kg
                \n",stato,LED,temperatura,CO,CO2,umidita);
                break;
        default:
                break;
        }
}
```

4.4.3 Server

La *ruotine* principale del server presenta aspetti diversi visto che non è questo che inizializza la comunicazione sulla *Powerline* perciò si metterà subito in attesa di dati chiamando la funzione *receiv_data(*): dallo schema in figura 28 si nota che il server può ricevere il comando di sincronizzazione tramite una *SynchroIndication* e una *DataConfirm* quindi sfruttando il *swith-case* si analizza il dato ricevuto. Nel caso di *SynchroIndication* si verifica quale evento ha scaturito il comando come descritto nel capitolo 3.8.2.

Quando si riceve invece una *DataConfirm* si controlla subito la richiesta del *Client* modificando la luminosità del lampione o acquisendo i dati dei sensori. Infine si utilizza la funzione *reply_to_COC( )* per rispondere al *Client*.

```
while(1){
       dato = receiv_data();
       switch (dato[0]){
       case CMD_SynchroIndication:
             printf("comando_Synchro:
             switch (dato[1]) {
                   case 1:
                         printf("Synchro_FOUND\n\n");
                         break;
                   case 2:
                         printf("Synchro_CONFIRMED\n\n");
                         break;
                   case 4:
                         printf("Synchro_LOST\n\n");
                          break;
                   case 5:
                         printf("Synchro_INTELLIGENT\n\n");
                         break;
                  }
                  break;
        case CMD_DataIndication:
              analizza_dato();
//controllo cosa chiede il COC
              if(How_do == 1){
//simulo la scrittura sul lampione
                        stato = ON;
                        LED = Lux_out;
              }else if(How_do == 2){
//simulo acquisizione dati lampione
                        temperatura = 21;
                        CO2 = 25;
                        CO = 20;
                        stato = ON;
                        LED = 75;
                        umidita = 0;
               }
//rispondo al COC
               do{
                         w = reply_to_COC();
                }while(w == 0);
                printf("ok_replay_to_coc\n\n");

                }

}
```

La funzione ***reply_to_COC( )*** segue lo schema di figura 29; da subito si chiama la funzione *init_comunicatzione( )*, si invia una *DataRequest* il cui pacchetto dati contiene tutte le informazioni dei sensori e lo stato della lampada, si aspetta l'ACK e successivamente la *DataIndication* tramite la funzione *receiv_data( )* ed infine si invia l'ACK di conferma.

```
int reply_to_COC(){
        unsigned char pre_data[5] = {0x00,0x00,0x1C,0x00,0x00};
        init_comunicazione();
        send(1 + 5 + lenght_dato + 2
        send(CMD_DataRequest);
        for(i=0;i<5;i++){
                send(pre_data[i]);
        }
        M_SDU[0]=stato ;M_SDU[1]=LED ;M_SDU[2]= temperatura;
        M_SDU[3]= CO; M_SDU[4]=CO2; M_SDU[5]=umidita;
        for(i=0;i<lenght_dato;i++){
                send(M_SDU[i]);
        }
        checksum = 1 + 5 + lenght_dato + 2 + CMD_DataRequest;
        for(i=0;i<5;i++){
                checksum = checksum + pre_data[i];
        }
        for(i=0;i<lenght_dato;i++){
                checksum = checksum + M_SDU[i        }
        check_array[0]=checksum; check_array[1]= (checksum>>8);
        send(check_array[0]); send(check_array[1]);
        ack[0] = 0; w=0;
        while(w==0){
                n1 = read(fd, ack, 1);
                if(ack[0]!= (ACK | NAK)){
                         w = 0;
                }else{
                        w = 1;
                }
        }
        switch (ack[0]){
        case NAK:
                printf("ack: NAK\n");
                return 0;
        case ACK:
                printf("ack: ACK \n");
                unsigned char* ric;
                ric = receiv_data();
                switch(ric[1]){
                case 0xff:
                        return 1;
                default:
                        return 0;
                }
        default:
                return 0;
        }
}
```

4.5 Problematiche riscontrate a livello software

Un problema inaspettato riguarda il comando del pin RTS della porta seriale: non è stato facile trovare una guida sulle librerie della porta seriale, perciò sono state compiute numerose prove che hanno impiegato diverso tempo per risolvere una questione che si pensava essere quasi scontata.

L'invio dei dati dalla porta seriale tramite la funzione *write* si è rivelato essere non molto efficiente dato che tale funzione non riusciva a inoltrare più di un certo numero di byte alla volta, quindi si è rimediato trasmettendo ogni singolo carattere indipendentemente, realizzando dei cicli *for*.

La stessa cosa accade in ricezione, la funzione *read* perde gran parte del contenuto del pacchetto dati e anche qui la soluzione è stata ricevere un byte alla volta utilizzando cicli *for*.

La ricezione dell'ACK non avviene quasi mai in maniera corretta: si ricevono grosse quantità di byte senza significato sulla linea tra ST7570 e PC al momento della ricezione dell'ACK, probabilmente dovuti alla presenza di variazioni spurie o residui di trasmissioni precedenti che "rimbalzano" tra i due dispositivi. Si è ovviato a questo problema controllando il carattere ricevuto e scartandolo quando il suo valore risultava diverso dalle uniche due configurazioni possibili ACK e NAK.

5. Test di funzionamento

Sin dall'arrivo delle schede sono stati eseguiti i test di funzionamento; partendo da quelli preliminari che utilizzano la GUI per capire il funzionamento dei modem fino ad arrivare a quelli sulla robustezza della linea aumentando il traffico dati e controllando i rapporti segnale/rumore su diverse tratte.

5.1 Test traffico dati

Questo test consiste nell'inserire sul bus di collegamento PLC tre modem configurati come un *client* e due *server*. Si è creato un programma dove il client interroga singolarmente i server e questi rispondono, con lo scopo di testare l'efficacia del livello MAC dei modem: questo scarta i pacchetti dati indirizzati a destinatari diversi da se stesso, controllando gli indirizzi.

Quando il *client* indirizza un server in particolare, solo quest'ultimo riceve i dati mentre l'altro acquisisce solamente la sincronizzazione dovuta alla presenza di un *frame* sulla *Powerline* e scarta il dato vero e proprio. Anche quando è uno dei *server* che trasmette i dati di risposta, solo il *client* li riceve poiché è lui il destinatario del messaggio.

Questo test è stato eseguito per diverso tempo (1 ora) con frequenze di richiesta dati molto alte (una chiamata ogni 3 secondi), allo scopo di verificare l'affidabilità in termini di robustezza nel tempo della comunicazione. Il test ha riportato un risultato positivo poiché la comunicazione è sempre rimasta attiva e non è mai stato perso un dato.

5.2 Test frequenze e guadagni

Grazie a questo test si è in grado di capire a quale frequenza è più opportuno trasmettere i bit: vengono trasmessi a livello MAC pacchetti dati di lunghezza fissa pari 242 byte dal client e dal server in diverse configurazioni di frequenza per i simboli "0" e "1" lasciando un gap di 10kHz come indicato dalle specifiche, inserendo una *baud rate* di 1200bit/sec e impostando diversi valori di guadagno in trasmissione, verificando i rapporti segnale-rumore forniti dal modem, anche su diverse lunghezze di cavo tra i due apparati. Nella seguente tabella sono illustrati le varie configurazioni e i risultati dei test:

Banda	frequenza 0 (kHz)	frequenza 1 (kHz)	guadagno (dB)	cavo (m)	S0/N0	S1/N1
A	80	90	-31	2	1,60	1,60
A	80	90	-16	2	1,70	1,59
A	80	90	0	2	1,67	1,67
B	100	110	-31	2	1,69	1,84
B	100	110	-16	2	1,90	1,78
B	100	110	0	2	1,79	1,66
B	115	125	-31	2	1,63	1,74
B	115	125	-16	2	1,88	1,93
B	115	125	0	2	1,84	1,79
C	128	138	-31	2	1,66	1,78
C	128	138	-16	2	1,85	1,93
C	128	138	0	2	2,00	2,00
D	140	148,5	-31	2	1,40	1,59
D	140	148,5	-16	2	1,74	1,95
D	140	148,5	0	2	1,67	1,70
A	80	90	-16	45	1,69	1,62
A	80	90	0	45	1,70	1,67
B	100	110	-31	45	1,67	1,78
B	100	110	-16	45	1,90	1,78
B	100	110	0	45	1,79	1,64
B	115	125	-31	45	1,61	1,71
B	115	125	-16	45	1,71	1,93
B	115	125	0	45	1,84	1,82
C	128	138	-31	45	1,52	1,67
C	128	138	-16	45	1,97	1,98
C	128	138	0	45	1,77	1,90
D	140	148,5	-31	45	1,43	1,57
D	140	148,5	-16	45	1,79	1,88
D	140	148,5	0	45	1,96	1,90
A	80	90	0	90 con giunte	1,70	1,70
B	100	110	-31	90 con giunte	1,54	1,69
B	100	110	-16	90 con giunte	1,90	1,75
B	100	110	0	90 con giunte	1,74	1,63
B	115	125	-31	90 con giunte	1,69	1,75
B	115	125	-16	90 con giunte	1,81	1,87
B	115	125	0	90 con giunte	1,77	1,81
C	128	138	-31	90 con giunte	1,51	1,61
C	128	138	-16	90 con giunte	1,71	1,95
C	128	138	0	90 con giunte	1,71	1,81
D	140	148,5	-31	90 con giunte	1,44	1,54
D	140	148,5	-16	90 con giunte	1,76	1,71
D	140	148,5	0	90 con giunte	1,76	1,85

Tabella 9: Dati e risultati test frequenze e guadagni

Si nota subito che aumentando la lunghezza del cavo, le frequenze alle quali la trasmissione ha successo, sono sempre più alte con un guadagno anch'esso più elevato.

Si possono costruire grafici per visualizzare meglio i valori in cui si riportano i rapporti segnale-rumore al variare del guadagno in *dB* impostato, tracciando una curva per ogni frequenza nel caso peggiore, ossia con un cavo lungo 90 metri:

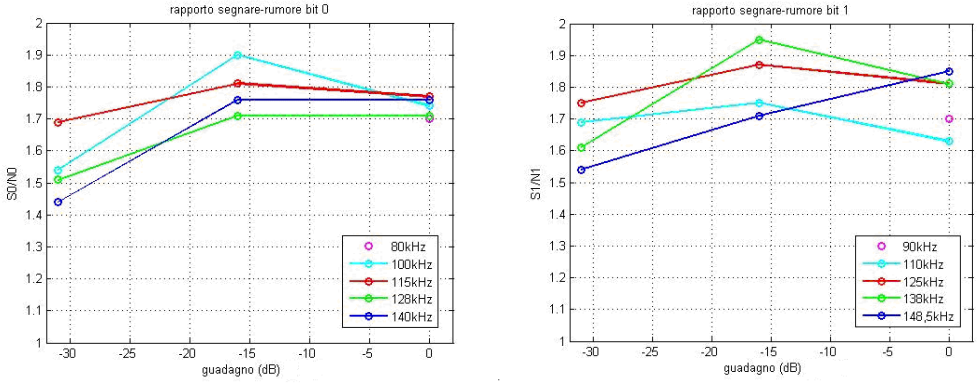

Figura 30: SNR al variare del guadagno per ogni frequenza

Da questi grafici si evince che trasmettere con un guadagno troppo basso o troppo alto, porta ad un risultato peggiore rispetto alla trasmissione con un guadagno intermedio.

Si può tracciare un altro grafico mettendo a confronto i rapporti segnale-rumore con le frequenze tracciando una curva per ogni guadagno:

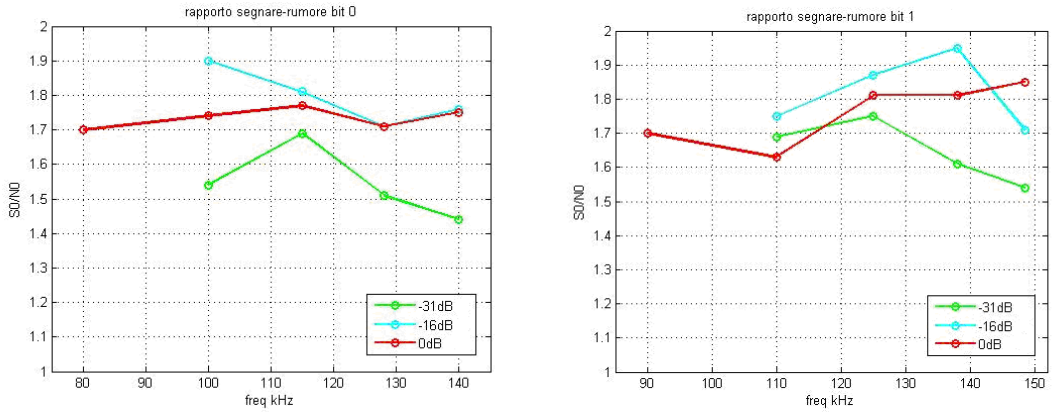

Figura 31: SNR al variare della frequenza per ogni guadagno

Anche da questo grafico si nota come trasmettendo con un guadagno intermedio il rapporto rimanga più elevato.

Per la scelta della banda in cui trasmettere con migliori risultati, si può notare che le bande centrali B e C hanno comportamenti migliori con valori di rapporto segnale-rumore più che apprezzabili. Sono d'altronde le bande in cui è più consigliabile trasmettere poiché sono quelle non licenziate e non adibite a nessun servizio.

5.3 Test della comunicazione con due client

Si inseriscono due *client* e un *server* nella stessa linea, si crea una coppia *client-server* che comunica sulle frequenze di 120-130kHz mentre il terzo modem, configurato come client, trasmette a frequenze limitrofe interferendo la comunicazione tra gli altri due apparati.

Se il secondo client trasmette a frequenze che superano i 10kHz di differenza, la comunicazione tra gli altri due apparati non viene interferita, quindi si possono sostenere più trasmissioni dati.

Se il *gap* si avvicina ai 2,5kHz allora gli altri modem riescono ad acquisire la sincronizzazione ma non ricevono il pacchetto dati.

Facendo trasmettere il secondo client alle frequenze si 121-131kHz, i dati sono ricevuti correttamente dagli altri due apparecchi.

6. Conclusioni

La tecnologia delle *Powerline* utilizzata nello scenario di una "*smart city*" risponde bene alle esigenze progettuali. È una valida alternativa alla tecnologia wireless, data la bassa mole di dati da trasferire e la presenza di cablaggi già istallati, abbatterebbe al massimo i costi.

I test effettuati in laboratorio hanno infatti dato prova che la comunicazione è affidabile anche fino a 100 metri. Quando si installeranno le apparecchiature in uno scenario reale, potrà essere necessario rieseguire tutti i test riguardanti l'affidabilità e la robustezza della comunicazione, correggendo le decisioni prese in laboratorio come la scelta delle frequenze e del guadagno in trasmissione, e nel caso in cui il segnale non riesca a raggiungere tutti i punti della linea, sarà necessario configurare dei modem come *repeater*.

Inoltre, all'atto dell'istallazione, sarà fondamentale valutare il livello di irraggiamento del segnale allo scopo di individuare possibili interferenze sulla trasmissioni wireless in bande limitrofe.

7. Riferimenti

[1] "*POWERLINE COMMUNICATIONS: Sistemi di comunicazione a onde convogliate"* Tesi di Laurea di Massimo Rocchi in Ingegneria Elettronica e Telecomunicazioni A.A. 2011/2012 seconda sessione.

[2] Smart-M3 *en.wikipedia.org/wiki/Smart-M3*

[3] Smart-M3 Source-Forge, sourceforge*.net/projects/smart-m3*

[4] "*EVALKITST7570-1" ST7570 S-FSK power line networking system-on-chip demonstration kit* ST datasheet.

[5] "*ST7570" S-FSK power line networking system-on-chip* ST datasheet.

[6] *http://en.wikipedia.org/wiki/IEC_61334* Modulazione S-FSK su Wikipedia.

[7] "*AN3213" ST7570 S-FSK power line networking system-on-chip design guide for AMR* ST Application Note.

[8] "*UM1008" Demonstration kit for the ST7570 power line modemwith graphical user interface* ST User Manual.

[9] "*UM0934" ST7570 S-FSK power line networking System-on-Chip* ST User Manual.

[10] *http://www.easysw.com/~mike/serial/serial.html* Serial Programming Guide.