Dispensa

sul

“Local Interconnect Network” (LIN)

(Ing. Stefano Maggi)
(Dottore di Ricerca – Politecnico di Milano)
(stefano.maggi@etec.polimi.it)

0 Introduzione al capitolo

In questo capitolo vengono illustrati i concetti base del protocollo LIN, limitando però i dettagli alle caratteristiche che si ritengono di una certa importanza per comprendere meglio il funzionamento del sistema.
Con l'ausilio di figure ed esempi verranno descritte e spiegate sia le principali caratteristiche del protocollo, sia il relativo funzionamento in un sistema bus ad architettura distribuita.

1 Introduzione al LIN

Il *LIN* (Local Interconnect Network) è un nuovo protocollo di comunicazione seriale progettato per funzionare nei sistemi distribuiti.
Viene normalmente utilizzato come sottorete locale alla rete bus principale, formando un sistema ad elettronica distribuita a basso costo interconnettendo vari componenti, principalmente sensori ed attuatori di tipo "smart", in applicazioni automotive.
Il protocollo LIN non intende sostituire i protocolli esistenti, per esempio TTP (Time Triggered Protocol), CAN (Controller Area Network), FlexRay, ecc…, ma soltanto completarli.

2 Approccio al sistema ad elettronica distribuita

Tutti i componenti (elettrici, elettronici, elettromeccanici, termici, ecc…) vengono interconnessi fra loro tramite una struttura distribuita a nodi (*Electronic Control Unit* "ECU").
In questo lavoro viene fatta una distinzione tra le ECU ed i componenti, dove le ECU devono essere considerate come elementi centrali del sistema distribuito.
I componenti in un sistema distribuito devono comprendere una logica che consenta loro di gestire i messaggi via bus (per esempio, avviare un motore passo-passo, accendere un LED, fornire una lettura di un sensore, ecc…).
Nei veicoli attuali vi sono differenti "*sistemi aperti*" che ricoprono varie funzioni.
Così come i requisiti relativi alla sicurezza, al comfort ed alla performance differiscono, così differisce anche il tipo di rete (vedi figura 1).
Oggi il sistema distribuito principale più diffuso è quello basato sullo standard CAN (non considerando i sistemi multimedia).

Il CAN (Controller Area Network), visto nel capitolo precedente, gestisce funzioni complesse in un moderno veicolo, con elevate caratteristiche in tempo reale.
Da questa rete principale, il LIN interviene quando le caratteristiche e la versatilità del protocollo CAN non sono richieste: si formano così localmente delle sottoreti per gestire il traffico di dati scambiato in ambito appunto locale.
Per esempio, una di queste sottoreti LIN può essere installata all'interno di una portiera, nei sedili, nel cofano posteriore, ecc… cioè dove si ha la necessità di gestire un traffico dati proveniente da un sistema del tipo sensore-attuatore e dove non si ha la necessità di avere elevate velocità di trasmissione (figura 4.1).
La seguente figura 1 mostra la delocalizzazione delle sole reti LIN (evidenziate in giallo) all'interno di un classico autoveicolo, insieme alle altre reti accennate.

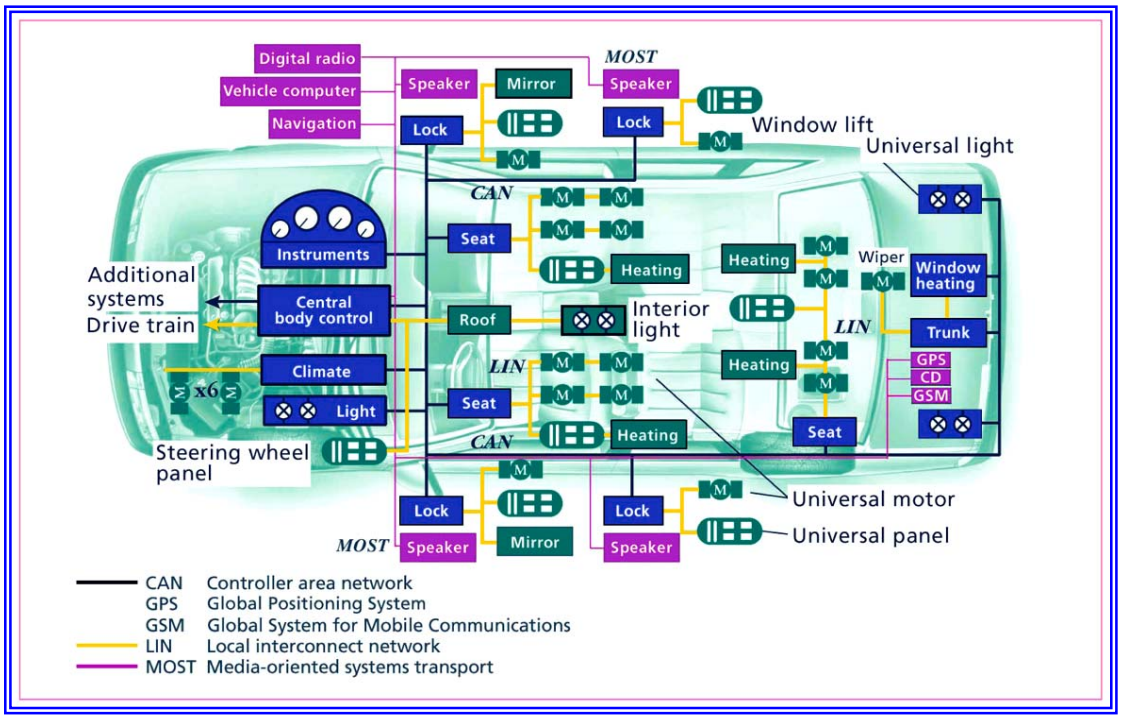


Fig. 1: Le diverse reti bus ad architettura distribuita all'interno di un veicolo

3 La nascita di LIN: l'Associazione

Il concetto di LIN è stato formulato dalla collaborazione fra diverse società leader di autoveicoli, da un'azienda di sviluppo software e da una di componentistica elettronica.
Questa collaborazione ha condotto più tardi alla costituzione dell'Associazione LIN nel 1998.
In figura 2 sono mostrate le società che fanno parte del comitato direttivo di questo consorzio.
Le prime specifiche sono state concluse nel 2000 e la presentazione ufficiale è avvenuta in SAE sempre nello stesso anno.
Le specifiche scritte dall'associazione (nel 2003 è uscita l'ultima revisione 2.0) coprono, oltre alla definizione del protocollo e del "Physical Layer", anche la definizione delle interfacce per gli strumenti di sviluppo e software di applicazione.
Il programma principale dell'associazione è di rendere il *LIN* uno *standard* "*de-facto*" per costruttori di veicoli, per i loro fornitori e per sviluppatori di tools applicativi.
L'obiettivo è quello della standardizzazione del protocollo, conforme a ISO (International Organization of Standardization) e a SAE (Society of Automotive Engineers).

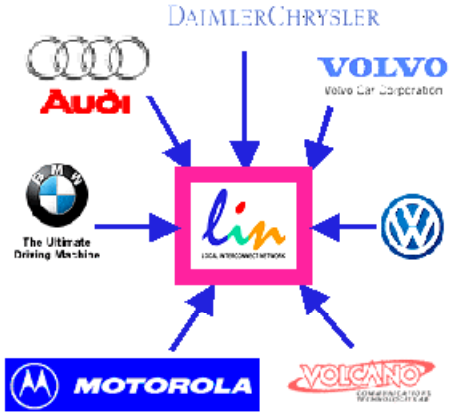


Fig. 2: Il consorzio LIN

3.1 Perché LIN?

Vi sono quattro campi distinti di applicazioni che utilizzano la comunicazione ed i protocolli di rete per gestire oggi diverse funzioni nei veicoli.
Ciascuno di questi campi necessita di un proprio tipo di protocollo di comunicazione.
Per ora non esiste nessun protocollo individuale "universale" che possa essere impiegato come soluzione standard per tutti.
Ogni protocollo richiede caratteristiche specifiche:

- Sicurezza, power train, controllo, ecc…, utilizzano protocolli con note caratteristiche di tempo reale e di velocità di trasmissione dati (CAN, TTP, FlexRay, ecc…)

- Applicazioni multi-media, richiedono protocolli che hanno il compito di fornire principalmente un'elevata larghezza di banda e di velocità di trasmissione (Bluetooth, MOST, D2B, ecc…, sono esempi tipici).

- Applicazioni critiche per la sicurezza, necessitano protocolli che abbiano tolleranza ai guasti (fault tolerant) e che siano ridondanti ed affidabili. X-by-wire è un mercato emergente che richiede protocolli come TTP/C, FlexRay, Byteflight e TTC.

- Applicazioni di tipo "Mechatronic", come l'interconnessione di sensori ed attuatori soprattutto di tipo "smart", o di complesse ECU con esigenze di comunicazione, sono indirizzate normalmente a protocolli come il LIN, TTP/A o ad alcuni protocolli proprietari.

Dopo questo ulteriore chiarimento, la scelta del LIN e del suo utilizzo in questo sistema integrato proposto, è stata dettata dall'esigenza di una comunicazione a basso costo, robusta ed affidabile fra sensori ed attuatori del sistema.
Per questo è necessario utilizzare un protocollo di tipo master/slave, dove la larghezza di banda e la versatilità del protocollo principale CAN, non sono necessari.
I principali vantaggi di questo protocollo, sono già stati illustrati nel precedente capitolo 2 di questo lavoro.
Evidenziare le principali differenze tra i protocolli CAN e LIN scelti, e l'adattabilità del LIN al CAN, sarà il tema del prossimo capitolo.

4 Prestazioni e risorse del LIN rispetto al CAN

Non ha alcun significato confrontare il protocollo di CAN con quello di LIN per quel che riguarda la concorrenza, perchè non sono orientati agli stessi risultati.
Tuttavia la seguente tabella 4.1 fornisce un generale confronto fra questi protocolli.
Le applicazioni LIN vengono realizzate dove il costo di comunicazione per nodo deve essere fino a due volte inferiore rispetto al costo di un nodo CAN, ma in cui non sia richiesta la prestazione, la larghezza di banda e la versatilità del CAN, come precedentemente accennato.
I principali fattori economici a favore del LIN sono la sua realizzazione con cavo singolo, il basso costo complessivo dei componenti, la comunicazione seriale si basa sulla comune interfaccia UART/SCI (Serial Communication Interface) e sull'assenza di oscillatori (al quarzo o in ceramica) nei nodi slave.
Si ha comunque una larghezza di banda minore ed un accesso al bus meno efficace, con una configurazione di tipo master-slave.
Le principali caratteristiche del protocollo LIN rispetto al CAN, sono quindi visibili nella seguente tabella 1.

	LIN	CAN
Medium access control	single master	multiple masters
Typical bus speed	2.4, 9.6 and 19.2kbps	62,5…1000kbps
Multicast message routing	6 bit identifier	11/29 bit identifier
Typical size of network	2…16 nodes	4…20 nodes
Data byte per frame	0… 8	2…8
Transmission time for 4 data bytes	6 ms at 20kbps	0.8 ms at 125kbps
Error detection (data field)	8-bit checksum	15-bit CRC
Physical layer	single-wire, 12 V	twisted-pair, 5 V
Quarts/ceramic resonator	master only	Yes
Relative cost per network connection	x 0.5[1]	x 1.0

Tabella 1: Confronto delle principali caratteristiche di LIN rispetto al CAN
([1] valutato per l'anno 2003)

5 Il protocollo LIN nel modello ISO/OSI

Secondo la specifica OSI (Open Systems Interconnection), con il termine "*sistema aperto*", si intende un sistema in cui lo scambio delle informazioni avviene secondo determinate norme di comunicazione sviluppate sulla base del modello di riferimento OSI.
Ai fini della descrizione del sistema, è necessario dividere in 2 parti la struttura interna.
Da una parte c'è il sistema di comunicazione che stabilisce il collegamento con il mezzo fisico e che quindi è responsabile esclusivamente della trasmissione dei dati, cioè del flusso non strutturato di bit.
Dell'elaborazione dei dati (pacchetti) ai fini di un determinato "task", è responsabile l'altra parte del sistema aperto (l'applicazione), che tuttavia esula dall'ambito delle attività di normazione ISO/OSI.
Nello specifico il protocollo LIN occupa i primi 2 livelli della "pila" ISO/OSI, cioè il *Physical Layer* ed il *Data Link Layer*.
Il *livello fisico* definisce come i segnali vengono trasmessi sul bus.
In particolare questo livello definisce le caratteristiche del "driver/receiver" (in pratica il transceiver), definisce il tempo di bit e la relativa sincronizzazione (come già accennato, gli slave di LIN non sono dotati di propri oscillatori al quarzo, per cui si richiede un meccanismo di autosincronizzazione col master).
Il *livello Data Link* è formato da 2 sottolivelli: il *Logical Link Control (LLC)* ed il *Medium Access Control (MAC)*.
Entrambi i livelli sono schematicamente rappresentati nella seguente figura 3.

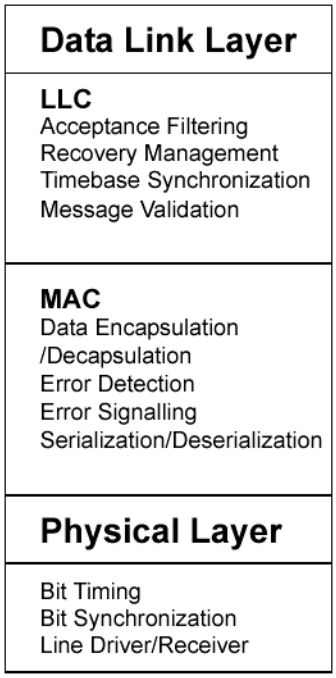


Fig. 3: Il protocollo LIN nel modello ISO/OSI

Il sottolivello LLC, gestisce il sistema di filtraggio dei messaggi, quindi la loro accettazione o rifiuto. Inoltre in questo sottolivello, si determina la sincronizzazione della base dei tempi del protocollo.
Il sottolivello MAC, rappresenta il "*kernel*" del protocollo.
Il MAC predispone i messaggi ricevuti dal sottolivello LLC per passarli al livello fisico, oppure accetta la sequenza di bit dal livello fisico, li "struttura" (cioè forma i pacchetti) e li passa al LLC.
Il MAC è anche il supervisore della sincronizzazione del sistema e del controllo degli errori.
Considerando la "pila" ISO/OSI, dal Data Link Layer in su aumenta l'astrazione dei livelli, infatti è da qui che interviene il modulo software di LIN.
Il cuore del sistema operativo (a livello software) è rappresentato dall'*Application Program Interface (API)*, che permette di programmare le singole ECU inserite nella rete LIN.
Questi aspetti software vengono trattati nel dettaglio in un successivo sottocapitolo dedicato.

6 Il protocollo LIN

La topologia della rete LIN consiste in un singolo master e uno o più slave (fino a 15 slave + un master, viene raccomandato dalle specifiche LIN).
Il master dà inizio a tutti i trasferimenti dati sul bus.
Di conseguenza non è necessario un sistema di arbitraggio (come esiste per il CAN) e quindi l'accesso al bus non è un problema.
Ciò conduce ad un comportamento di accesso al bus di *tipo deterministico* garantendo tempi di latenza calcolati e specifici per i frame che transitano sulla rete LIN.
La rete LIN è realizzata utilizzando un unico cavo, che generalmente significa avere valori più elevati in *EME* (ElectroMagnetic Emission), in confronto alle realizzazioni in cavi "twisted-pair" (doppino intrecciato) come per il CAN.
Per mantenere bassi i valori EME, viene controllata la "*slew-rate*" (velocità max. di risposta) dei segnali, nonché la larghezza di banda dei dati trasferiti.
La velocità dei dati non deve superare il valore specificato di 20kbits/s e la maggior parte dei dispositivi LIN dovrà sopportare velocità di bit standard di 2400, 9600 e 19200 bit/s.
Poiché vi è un bilanciamento tra "*slew rate/baud rate*" (velocità max. di risposta/velocità di baud) ed *EME*, è importante tenerlo in considerazione durante la progettazione o la valutazione del sistema nel quale il LIN dovrà funzionare.
Il protocollo LIN è costruito a partire dal protocollo *UART* (Universal Asynchronous Receiver Transmitter) e *SCI* (Serial Communication Interface).
Ciò significa che tutti i messaggi inviati al bus LIN sono codificati in byte in conformità a questo protocollo (UART).
L'unica eccezione è l'interruzione di sincronizzazione (SYNC) nel frame di messaggio LIN, che verrà trattato successivamente.
L'unica caratteristica dell'UART, che occorre brevemente trattare qui per comprendere il protocollo LIN, è che per ogni byte di dati (eccetto l'interruzione SYNC) viene inserito un bit di avvio (*Start bit*) e di arresto (*Stop bit*) prima e dopo il byte di dati.
Il bit di avvio è rappresentato da uno "zero" (attivo basso) ed il bit di arresto da un "uno" (attivo alto).

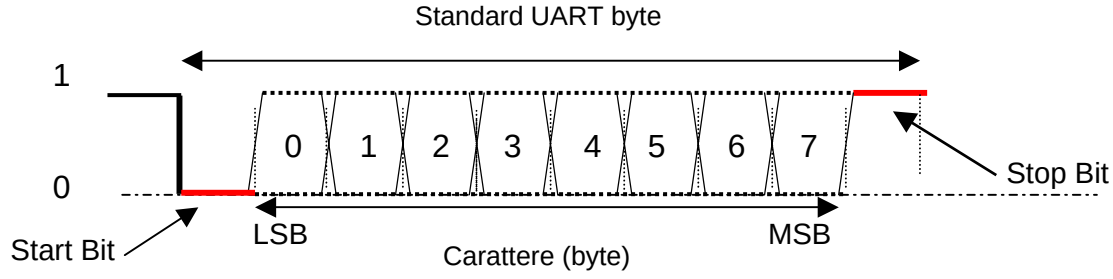


Fig. 4: Il formato UART di un byte

Per cui se un byte viene inviato all'interfaccia UART, questa lo codificherà in 10 bit (8+2 bit), inviandolo al transceiver LIN, cioè al bus.
Il formato del byte UART può essere visto graficamente in figura 4.

6.1 Il LIN Message Frame

Il frame di messaggio del LIN consiste in una parte d'intestazione (*message header*) ed in una parte di risposta (*message response*).
L'intestazione possiede una lunghezza fissa (a livello di bit), mentre la parte di risposta è variabile e più precisamente formata da 0 fino ad 8 byte di dati.
Il tempo di risposta tra un frame e l'altro è il tempo necessario allo slave per rispondere ad una richiesta (cioè ad un ID "Identifier") dal master e può variare tra i nodi della rete, poiché esso dipende dall'implementazione hardware e software in ciascun nodo.
Alla fine della parte di risposta è connesso un Checksum (somma di controllo) che è calcolato per la parte di dati.
L'intestazione, inviata dal master, è suddivisa in tre campi: *interruzione SYNC (synch break)*, *campo SYNC (synch field)* e *campo di identificazione ID (identifier field)*, che verranno successivamente trattati.
La risposta, normalmente da parte degli slave, è formata solo da due campi: *campo dati (data field)* e *somma di controllo (checksum)*.
Questa struttura completa del frame di messaggio è rappresentata in figura 5.

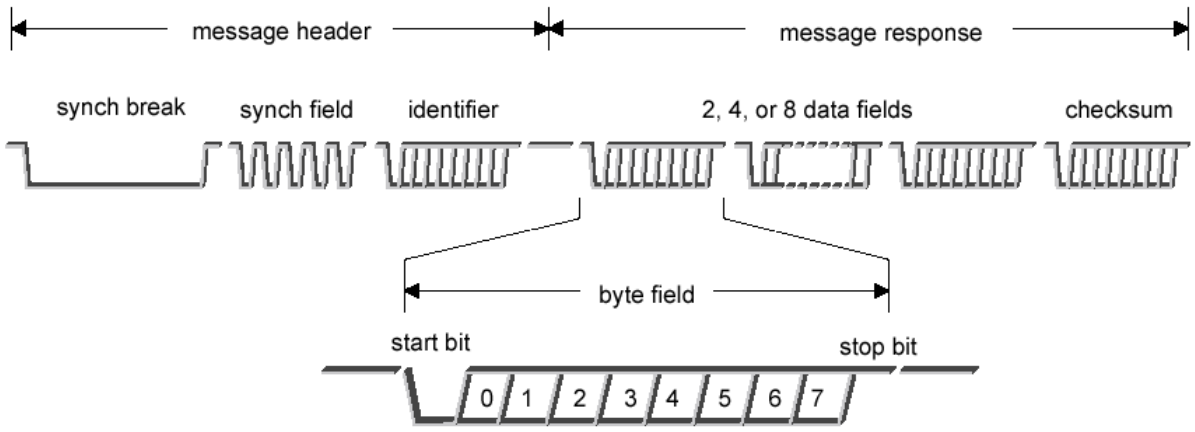


Fig. 5: Il Frame di messaggio del protocollo LIN

Il *message header*, rappresentato singolarmente, viene illustrato nella seguente figura 6.

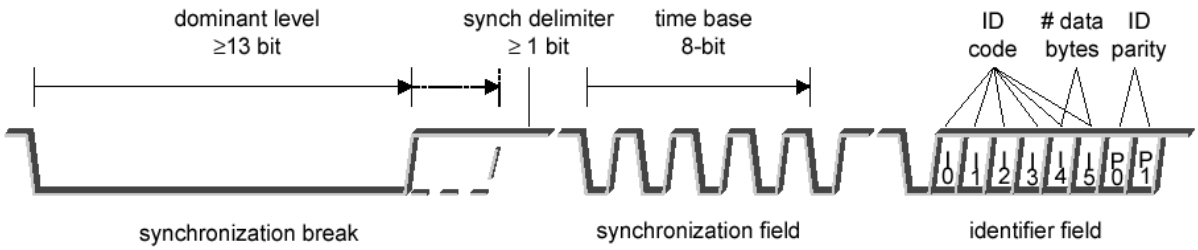


Fig. 6: Il message header

L'interruzione di sincronizzazione (Synchronization break):

La prima parte di un messaggio LIN è l'interruzione SYNC, che consiste di almeno 13 bit di "zeri".
L'interruzione è necessaria per consentire agli slave di rilevare che viene trasmesso un messaggio al bus.
Si ricorda che solo il master è dotato di un oscillatore vero e proprio (per esempio in questo lavoro si è utilizzato il quarzo), mentre gli slave sono dotati di semplici oscillatori RC a basso costo (dal punto di vista economico, questo rappresenta un vantaggio soprattutto in presenza di reti con un grande numero di slaves).

In conformità alle specifiche LIN, gli slave possono avere una frequenza di segnale che differisce del 15% rispetto ai master.
Quindi per rilevare l'interruzione un nodo slave, non sincronizzato con una frequenza di segnale che è del 15% inferiore al segnale del master, deve campionare almeno 10 bit come zeri, poiché possono sempre essere trovati 9 zeri in un byte UART ordinario.
Per il master significa che 12 bit (ottenuti da 10 per 1,15, cioè il 15% in più) saranno sufficienti agli slave affinché possono procedere al rilevamento dell'interruzione.
Tuttavia, le specifiche indicano che deve essere inviato un minimo di 13 bit.
Le routine del software LIN nello slave devono controllare che tutti i bit ricevuti nel byte dati siano zeri, al fine di assicurarsi che l'interruzione sia stata ricevuta.
Per il nodo master, realizzato da un microcontrollore, la procedura dell'invio dell'interruzione SYNC coinvolge anche qualche "manomissione" relativa al protocollo UART, perché è impossibile inviare 13 bit di zeri in fila in questo protocollo.
Un modo è quello di cambiare il baud rate in una velocità più lenta, in modo che il tempo impiegato per inviare un byte UART corrisponda a 13 bit con la giusta baud rate.
Oggi esistono già implementate UART evolute (o migliorate) al fine appunto di rilevare una stringa di almeno 13 "zeri" continuativi.

Il Campo di sincronizzazione (Synchronization field):

Poiché il master inizia sempre una trasmissione inviando un'intestazione, gli slave sono in grado di sincronizzare i loro segnali ogni volta che viene ricevuto un nuovo messaggio.
Ciò non rende necessario utilizzare degli oscillatori (più o meno costosi) sui nodi slave; in pratica è soltanto necessario un oscillatore preciso nel nodo master, come riferimento di tempo.
Lo slave sincronizza il proprio segnale misurando il tempo impiegato dal primo fronte di discesa del campo di sincronizzazione (cioè dal fronte di discesa del bit di start) al quinto fronte di discesa, cioè il bit 7 del byte di questo campo e lo divide per 8 al fine di ottenere il tempo di bit o il baud rate del master.
E' infatti questo il campo che da la base dei tempi al sistema LIN.
Il synch delimeter (di almeno 1 bit) che divide questo campo ed il precedente, è necessario alto per permettere di trovare il bit di start (basso) di questo campo.

Il Campo di identificazione (Identifier field):

Nell'ultima parte dell'intestazione del messaggio è collocato il campo ID.
Il campo ID, protetto da due bit di parità, evidenzia il contenuto e la lunghezza dati del messaggio.
Questo campo è formato da 6 bit identificatori e da 2 bit di parità, formando un insieme di 64 identifiers (2^6).

Nelle specifiche LIN, due dei bit nel campo ID vengono utilizzati per specificare la lunghezza della parte di dati del messaggio, ricordando che il numero di byte consentito è di 2, 4 o 8.
I nodi slave sulla rete sono quindi indirizzati dal campo ID.
I nodi non hanno indirizzi fisici (per esempio un qualsiasi indirizzo hardware), ma usano invece un elenco pre-programmato di ID valide nella memoria che utilizzano per scegliere a quali messaggi rispondere.
Come avviene nel CAN, anche nel LIN l'identificatore di un messaggio indica il contenuto di quel messaggio, ma non la destinazione!
La struttura di questo campo è rappresentata, come le altre, in figura 6.

Il *message response*, si veda in figura 5, è semplicemente formato dal campo dati (fino a 8 byte) e da un campo sempre di 1 byte che rappresenta la somma di controllo (checksum).
Il checksum è un controllo di parità che viene eseguito su tutti i data byte, secondo la regola del modulo-256.
Per scopi particolari, è anche possibile estendere (cioè oltre a 8 byte) la quantità di dati trasmessi nello stesso frame di messaggio (figura 7).
Si parla in questo caso di *LIN frame esteso*.
Questo lo si può ottenere con un particolare e predefinito identificatore nel message header, che avvisa gli slave di un arbitrario numero di data fields in arrivo.
Gli slave che ricevano tale identificatore, ad eccezione del destinatario programmato, ignorano tutti i data fields fino alla successiva sincronizzazione.

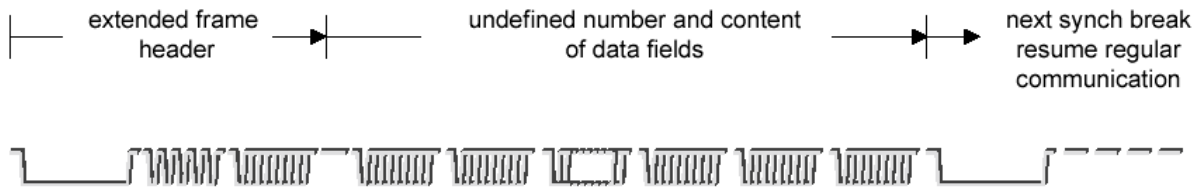


Fig. 7: LIN message frame esteso

In particolare due "*Extended Frame Identifiers*" sono riservati per permettere l'inserimento di formati di messaggi definiti dall'utente e per formati futuri del protocollo LIN, senza violare le attuali specifiche (ver. 2.0).
Questi "*Extended Frames*" vengono distinti da 2 "*Identifier Fields*" riservati:

- '0x3E': ID Field = 0xFE per *Extended Frame* definito dall'utente
- '0x3F': ID Field = 0xBF per future estensioni di LIN

L'identificatore '0x3E' un Frame esteso definito dall'utente libero per l'uso, dove ogni oggetto viene come prima, rappresentato da un byte. L'identificatore può essere seguito da un numero arbitrario di bytes (Data Field) e in questo caso la lunghezza dei Data Fields non viene più riferita dal campo di controllo "Length control" (ID4, ID5 bit) nell'Identifier Field.
Gli slave che ricevano un "*Extended Frame Identifier*" e non sono abilitati a valutarne il contenuto, ignorano tutti i bytes nel Data Field fino alla ricezione del successivo "*Sync Break*".

Osservazione:
Se il proprio sistema deve trasmettere continuativamente una quantità di dati superiore a 8 bytes per ogni *Message Frame*, è opportuno utilizzare un altro protocollo in quanto il traffico sul bus viene "sbilanciato" (vedi successiva figura 6) e si formerebbero troppi "tempi morti" in fase di trasmissione del master.
Cioè il traffico con più di 8 bytes, nel *Response Frame*, deve essere un'eccezione e non una regola di normale traffico.
Infatti una rete correttamente dimensionata, deve operare normalmente in condizioni di traffico leggero o medio (underload).

6.2 La comunicazione Master/Slave(s)

Vediamo con l'aiuto di figure, i 3 modi di comunicazione tra il master e i suoi slaves.

Comunicazione master – slave (figura 8.1).
La prima alternativa di trasmissione dati è quella di fare trasmettere dal master un intero frame di messaggio, cioè sia l'intestazione che la risposta, per uno o più slaves ("multicast" se più di uno o "broadcast" se tutti gli slave lo ascoltano) e viene normalmente riferito come un frame di comando (*CMD frame*).

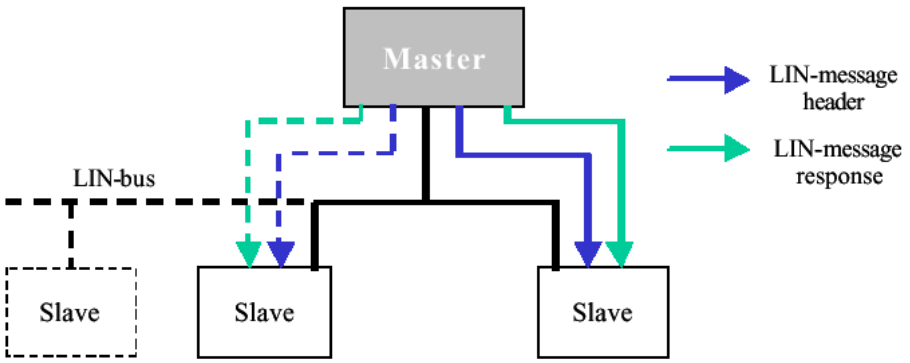


Fig. 8.1: Comunicazione dal master allo slave(s)

Comunicazione slave – master (figura 8.2).
Quest'alternativa si verifica quando il master richiede una risposta da uno slave specifico ("polling") e viene normalmente riferito come frame di richiesta (*REQ frame*).

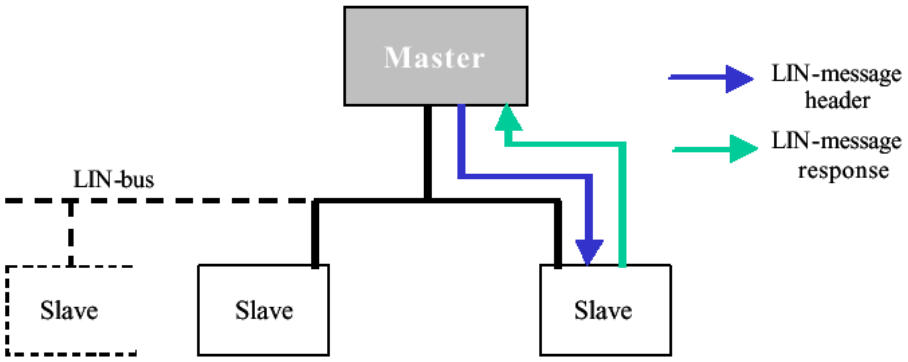


Fig. 8.2: Comunicazione dallo slave al master

Comunicazione slave – slave (figura 8.3).
L'ultima alternativa si ha quando uno slave invia la sua risposta ad uno o più slaves. Viene utilizzata questa comunicazione quando non vi è alcun bisogno di elaborazione dati mediante il master.
Un esempio può essere un semplice scambio di informazioni tra sensore ed attuatore.

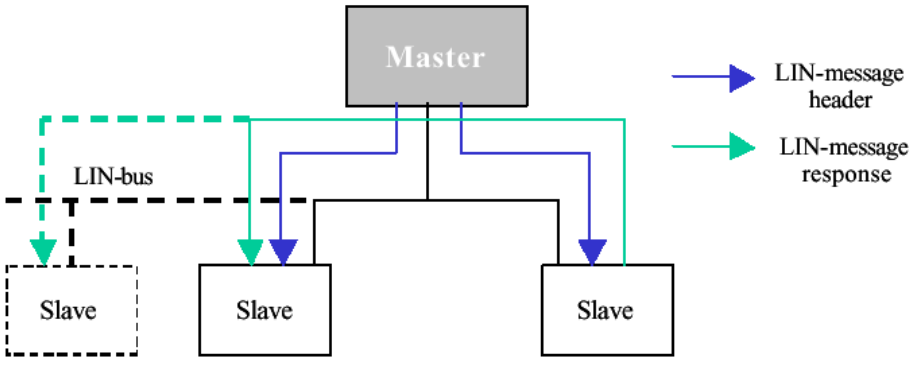
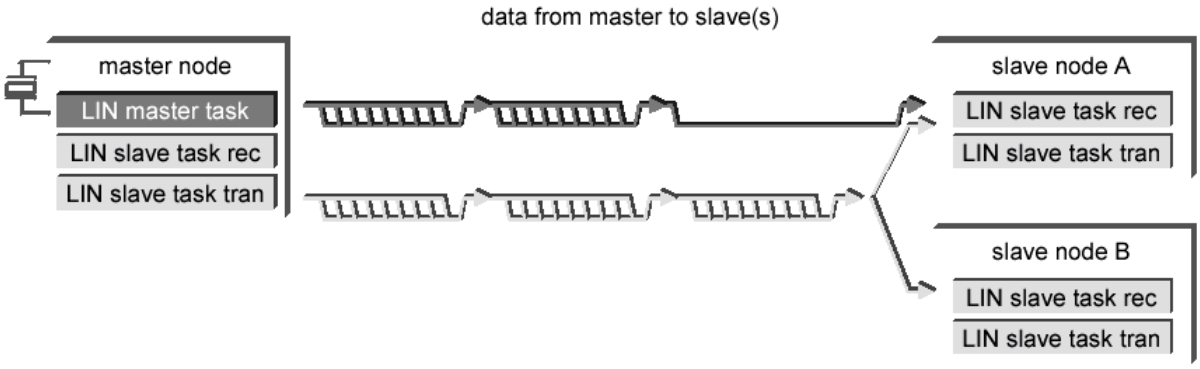


Fig. 8.3: Comunicazione da slave a slave

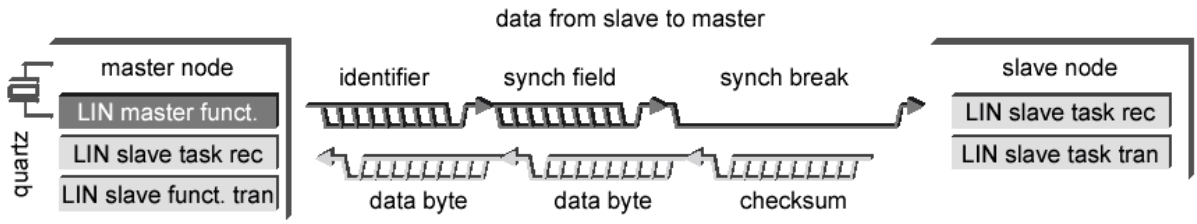
In forma più completa, questa comunicazione master/slave avviene a *livello Data Link* (modello ISO/OSI), mediante le funzioni di *master task*, di *slave task receiver* e di *slave task transmitter*.

La comunicazione può essere così rappresentata nella seguente forma più dettagliata.

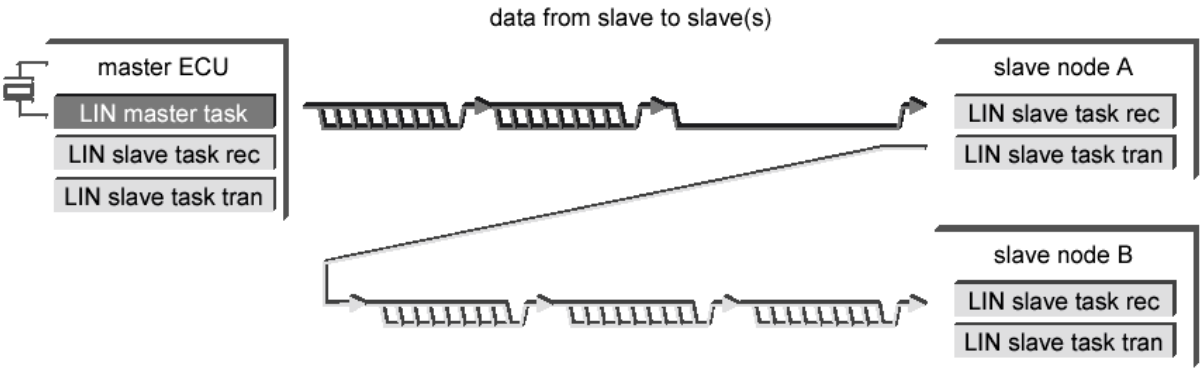
Comunicazione master – slave:



Comunicazione slave – master:



Comunicazione slave – slave:



6.2.1 Trasmissione (Tx) e ricezione (Rx) dei Message Frames

Dopo avere ricordato che il metodo di comunicazione utilizzato dal protocollo LIN è di tipo bidirezionale su conduttore unico, si può visualizzare nella seguente figura 8, l'occupazione del bus in funzione del traffico master-slave.
Il nodo master, prende il possesso del bus e trasmette un "*Header Frame*".
Ogni nodo slave riconosce un ID (identificatore) dal "Header Frame" appena ricevuto e quando l'ID è del nodo locale (slave locale), il nodo stesso trasmette una risposta (*Message Response*).

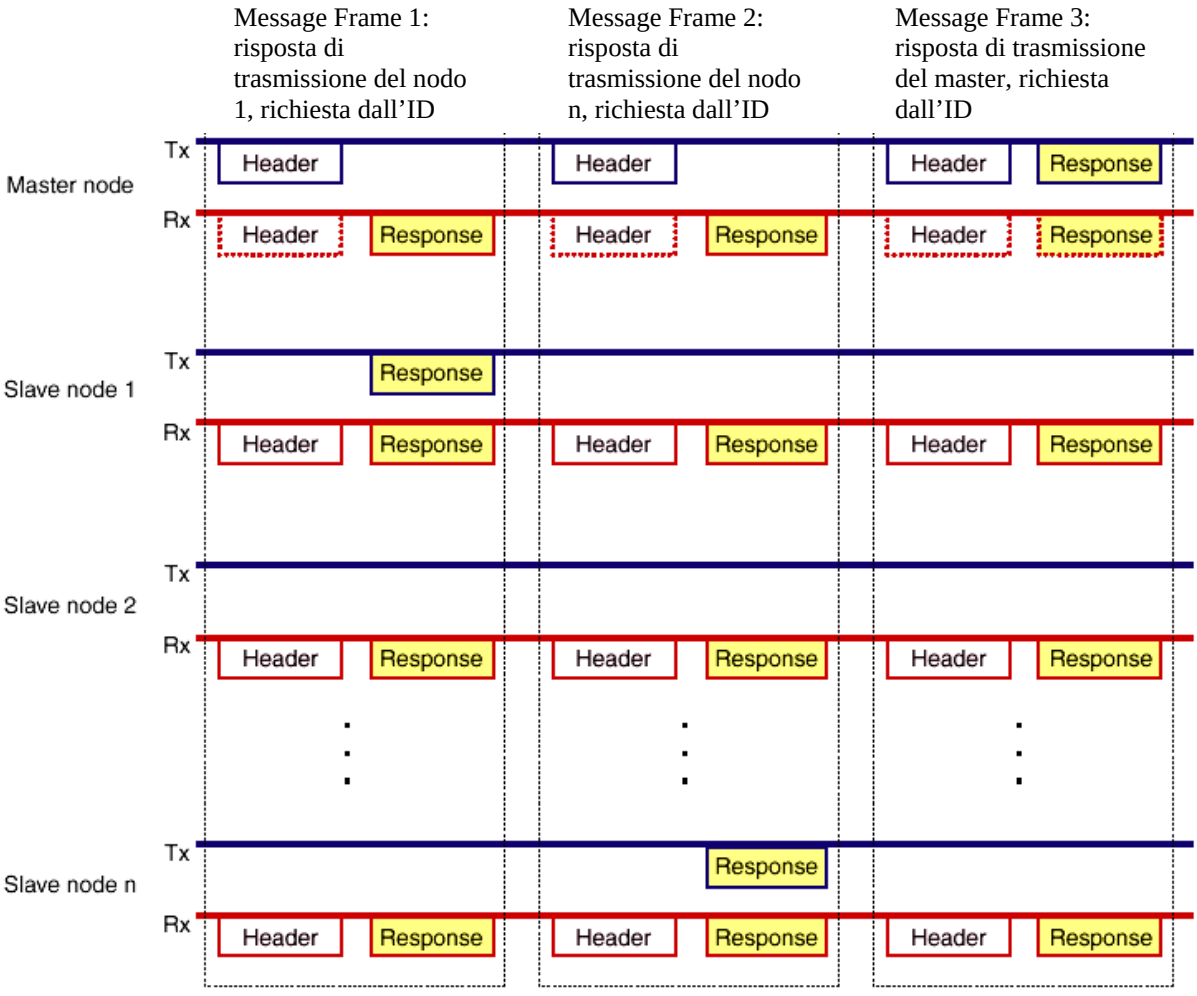


Fig. 8: Occupazione del bus nelle fasi di Tx e Rx

7.0 Rilevamento errori, confinamento guasti e protezione dati

Le azioni intraprese dai nodi della rete quando un messaggio viene interrotto, non vengono specificate dalle specifiche LIN ed è perciò compito del livello applicazione preoccuparsi delle procedure di confinamento guasti.
Il protocollo LIN specifica soltanto gli schemi base di rilevamento errori per bit, per la somma di controllo (checksum), per la parità ID (ID parity), e per errori di mancata risposta dello slave (del campo "synch") e di mancata attività del bus.
Del confinamento guasti si preoccupa il master, poiché esso è l'unico nodo che può emettere una ri-programmazione del messaggio.
I nodi slave non possono segnalare direttamente errori, per cui il master deve interrogare ciclicamente gli slave in merito ad errori.
Gli errori di bit al trasmettitore (sia master che slave) vengono rilevati confrontando il flusso di messaggi in uscita con il flusso di messaggi monitorati.
Gli errori di somma di controllo e parità ID sono facilmente rilevabili calcolando e confrontando localmente i dati.
L'errore del campo "*synch*" inconsistente, viene rilevato quando i fronti di questo campo di sincronizzazione si trovano al di fuori della tolleranza data.
Lo slave che non fornisce risposta e la mancata attività del bus vengono facilmente rilevati utilizzando dei temporizzatori implementati via software o hardware.
Se un nodo slave osserva un'incoerenza, lo slave interessato la può salvare come informazione diagnostica, che può ritrasmettere al master se richiesto dal software.

7.1 Un comando utile: il comando di riposo (Sleep mode)

Le specifiche LIN supportano un certa modalità di funzionamento chiamata "riposo".
È utile servirsi di questo comando quando si verificano solo eventi sporadici (per esempio quando viene premuto un pulsante), così il bus non viene "sovracaricato" inutilmente.
Il master è l'unico nodo valido che possa mettere a riposo la rete (cioè gli slave), mentre tutti i nodi sono in grado di ricevere questa modalità di riposo e successivamente, "attivare" nuovamente il bus.
Il master inizia lo "*sleep mode*" trasmettendo un frame di comandi predefinito (ID = 0x3C) con dati predefiniti (primo byte dati '0x00') agli slave della rete.
Gli slave rispondono a questo comando mettendo i loro transceivers nella modalità di basso consumo energetico ed aspettando dal bus un segnale di sveglia (*Wake-up signal*) o aspettando questo segnale da un nodo stesso (per esempio, premendo un pulsante collegato ad uno slave).

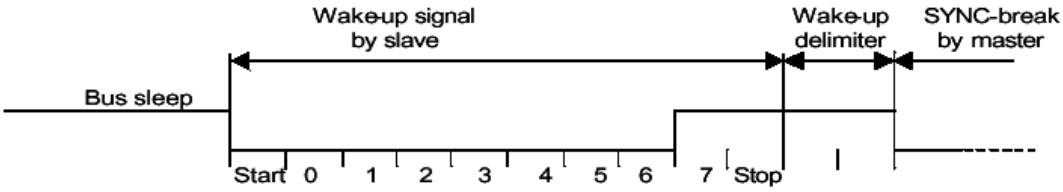


Fig. 9: Formato del segnale di “Wake-up”

Il segnale di sveglia consiste in un semplice byte dati con un carattere specifico (‘0x80’) come dato.
Dopo che il segnale di sveglia è stato ricevuto dal bus, tutti i nodi attraversano le loro procedure di avvio attendendo che il master invii un “*synch break*”.
Il bus deve essere a livello alto (uno) per un periodo di almeno 4 tempi di bit, dopo il periodo principale del segnale di sveglia, al fine di consentire ai nodi di poter eseguire le loro procedure di avvio.
Questa procedura viene anche visualizzata in figura 9.

8.0 Il LIN API (Application Program Interface)

L'*API* di LIN è un software di rete che "nasconde" all'utente i dettagli della configurazione di rete (per esempio, la mappatura dei segnali all'interno dei frame) mediante un programma interfaccia di applicazione per ogni arbitraria ECU (Electronic Control Unit) o nodo.
Questo rappresenta il cuore del sistema operativo della parte software di LIN, schematicamente rappresentata in figura 10.
Si utilizza quindi un tool (eseguibile da computer) chiamato *LCFG* (LIN Configuration tool) che aiuterà l'utente dalla configurazione della rete al codice del programma del nodo.
Questo permette di avere un'elevata flessibilità di configurazione, durante la fase di programmazione.

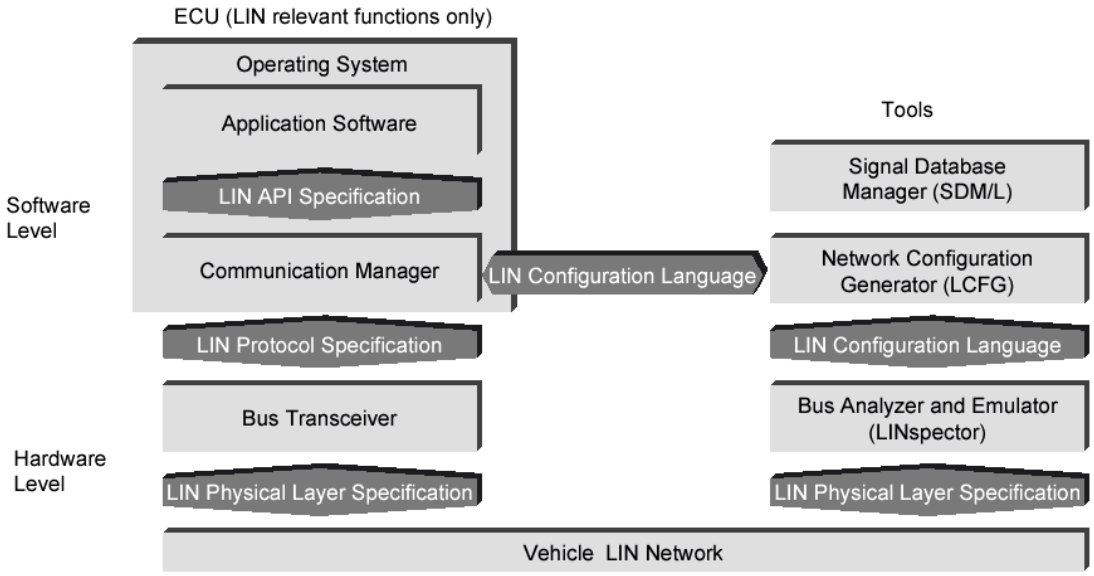


Fig. 10: Struttura del protocollo LIN secondo il modello ISO/OSI (a sinistra)
e strumenti (tools) di interfaccia (a destra)

I principali tools utilizzati a livello software sono:

- ✓ Il signal database manager per LIN (SDM/L)
- ✓ Il LIN configuration manager (LCFG)
- ✓ Il compilatore e il programmatore (linker)
- ✓ Vari tools di analisi del bus

Il "*signal database manager*" è uno strumento per definire, configurare e gestire una rete LIN. Gli input di questo programma sono le proprietà di progetto, come la definizione dei segnali, dei nodi, della velocità di comunicazione, le interfacce, i requisiti delle latenze, ecc…
Il *frame/schedule*, all'interno di questo programma, converte i segnali introdotti in frame e crea un message schedule (tabella di messaggi).
Questo "flusso" viene anche schematizzato nella seguente figura 11.

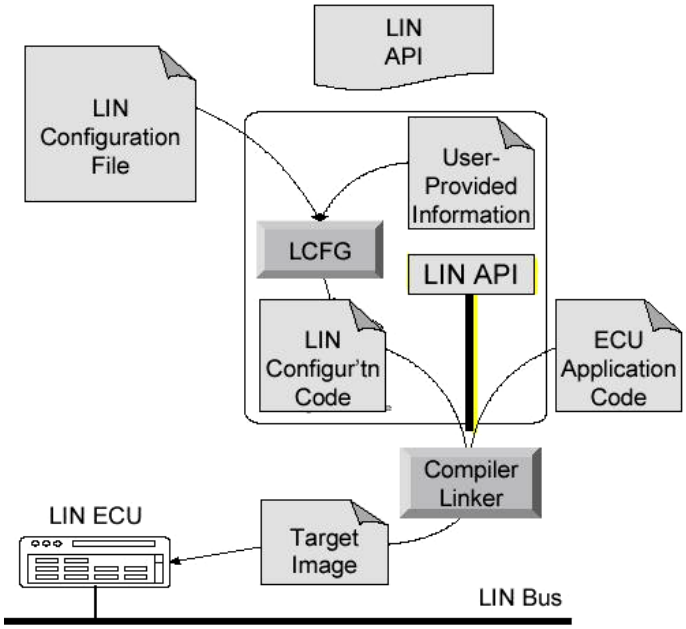


Fig. 11: Schema della configurazione software di LIN

Il *LIN configuration file* è un file che viene generato per la descrizione di parametri di un nodo specifico (per esempio descrive le funzionalità di quel nodo e le sue specifiche hardware).
Come si vede chiaramente in figura 11, gli input del *LCFG* sono il LIN configuration file e delle informazioni fornite dall'utente, riguardanti la rete complessiva e/o il singolo nodo (ECU).
Successivamente l'LCFG genera un file.c e un file.h che vengono compilati insieme con l'applicazione software utilizzata ("l_gen.c" e "l_gen.h").
Il codice così ottenuto è stato scaricato, tramite un programmatore, direttamente nel nodo (cioè nella memoria flash del microcontrollore).

Si può riassumere quanto detto evidenziando lo sviluppo del *source file* ed il *header file* nel seguente diagramma di flusso di figura 12.

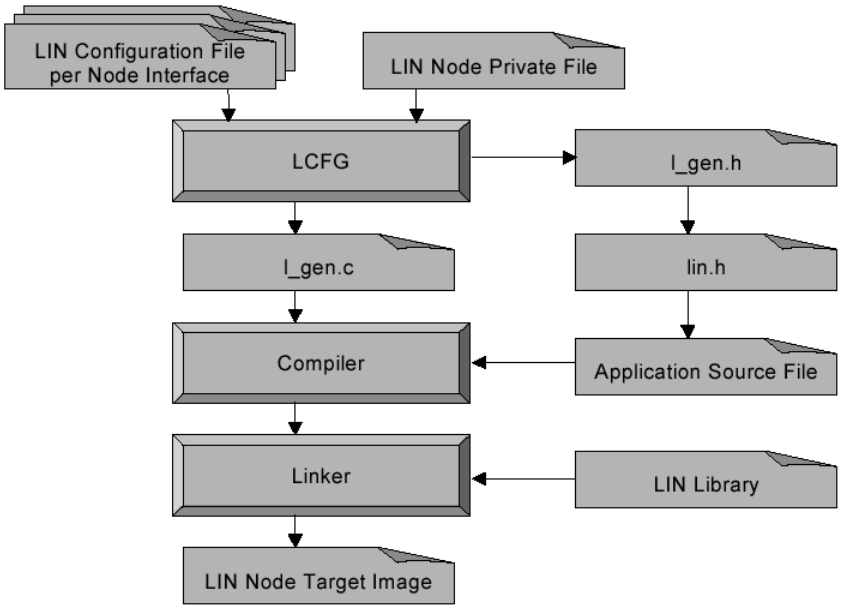


Fig. 12: Diagramma di flusso della generazione del codice con il Configuration Manager

Infine si può ricordare che le specifiche *LIN API* definiscono un set di funzioni utilizzate per inizializzare il sistema, per leggere o scrivere chiamate (per segnali o flags), per inizializzare e connettere nodi d'interfaccia, per gestire gli interrupt del microcontrollore, ecc…

9.0 Le latenze di segnale

La comunicazione tra i nodi della rete è amministrata e controllata dal master (unico).
Esistono comunque piccoli conflitti in termini temporali che si possono verificare sul bus.
Non vi è alcun sistema di arbitraggio sul bus essendo, come detto, questo protocollo di tipo non "event driven".
Quando qualcosa effettivamente non va, si rende necessario la ri-trasmissione del frame.
Nonostante il comportamento di tipo deterministico, vi sono alcuni fattori che sono importanti durante la progettazione e la programmazione dei frame e dei segnali della rete.
Alcuni di questi fattori riguardano le *latenze del segnale*, che vengono inflitte in diversi punti nella comunicazione e che realizzano le proprietà di "*real time*" del sistema.
In questo sottocapitolo si evidenziano alcune delle caratteristiche generali di tali latenze (in parte riscontrate durante le verifiche del sistema realizzato), di cui alcune sono specifiche del LIN, mentre altre sono presenti nella maggior parte dei sistemi di comunicazione.
Un altro fattore che interviene nella programmazione in tempo reale è il "tremolio" (*Jitter*) che verrà trattato successivamente.
Il modello di tempo rappresentato in figura 13, mostra gli intervalli temporali tra la generazione ed il consumo del segnale (in pratica dell'azione).
Vi possono essere diversi utenti di un singolo segnale nella rete ed i valori delle latenze descritte sono specifiche per ciascun nodo.
Questo avviene perché la rete può avere diversi tipi di soluzioni software e hardware per i nodi nel sistema.

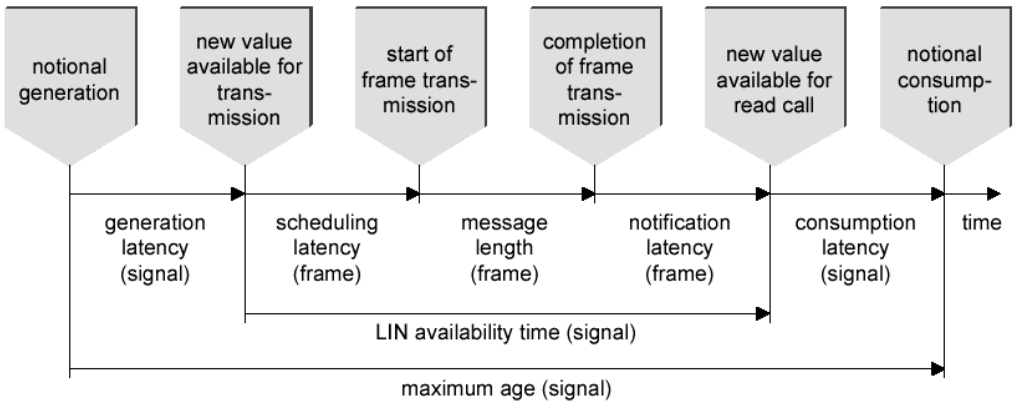


Fig. 13: Il "LIN timing model" e le latenze di segnale

Nello specifico, le varie latenze possono essere così descritte:

Latenza di generazione, T *GL*

Definisce il tempo impiegato dal momento in cui si verifica un evento (per esempio, la pressione su un pulsante) al momento in cui il corrispondente segnale viene aggiornato nel buffer e disponibile per i driver LIN per la trasmissione.
Questa è una latenza che dipende dalle proprietà dell'applicazione del software (per esempio, che tipo di hardware e software viene utilizzato) e perciò non influisce sulle proprietà di "real time" del protocollo LIN.

Latenza di programma, T *SL*

Definisce il tempo impiegato prima che venga effettuata la trasmissione corrente e dipende dal momento in cui il frame del segnale è pronto per la trasmissione che viene determinata dal programma del master.
La latenza è la stessa per tutti i segnali nel frame ed è anche comune per tutti gli utenti del segnale. Questa latenza dipende da come viene effettuata la programmazione dei frame, se il programma è conciso o se viene utilizzato un lasso di tempo lento tra i frame, per altre esecuzioni di operazioni sul microcontrollore.

Latenza di trasmissione, T *TL*

E' il tempo richiesto per la trasmissione di un frame sul bus e dipende dalla velocità di trasmissione e dalla lunghezza del bus.

Latenza di notificazione, T *NL*

Definisce il tempo impiegato tra la ricezione del segnale aggiornato e la sua memorizzazione nel buffer.
E' come *T GL*, dipende dalle implementazioni dell'hardware e del software e non influisce sulle proprietà di "real time" del protocollo LIN.

Latenza di consumo, T *CL*

Definisce il tempo intercorso dal momento della notifica del segnale aggiornato nel buffer, a quello corrispondente all'esecuzione dell'azione da parte dell'applicazione (per esempio l'avviamento di un motore, l'accensione di luci, ecc…).
Come la ***T*** *NL*, dipende dall'hardware e dal software e non è una latenza specifica per il protocollo LIN.

9.1 Base dei tempi e ritardo di frame

A ciascun frame trasmesso sul bus, viene data una certa quantità di tempo per il completamento della sua trasmissione, prima che venga fatto partire il successivo frame della *tabella di programma* (*message schedule*).
Questo viene chiamato *ritardo di frame*, il quale non è necessario per tutti i frame del programma.
Il ritardo dipende dalla dimensione dei frame ed anche dal tempo base del sistema.
In alcuni sistemi questo ritardo è addirittura necessario per avere un considerevole "gap" di tempo tra i frame, in modo che ad altri compiti sui nodi (master o slave) venga dato più tempo per l'esecuzione.
Su altri sistemi per esempio, il programma deve essere il più conciso possibile a causa delle richieste di "real time" molto esigenti.

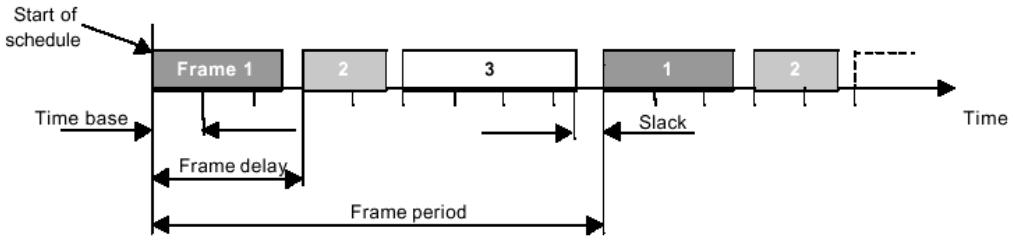


Fig. 14: Base dei tempi e ritardo di frame

La base dei tempi del sistema fornisce la risoluzione del ritardo di frame in millisecondi.
Mediante una base di tempo inferiore il ritardo di frame può essere organizzato in modo più conciso, cioè minimizzando la riduzione tra i frame.
La base di tempo non è un valore specificato dalle specifiche LIN, ma è un qualcosa che viene configurato dallo sviluppatore di sistema.
La figura 14 mostra il concetto del ritardo di frame e della base dei tempi.

Un esempio su questi concetti, può essere estrapolato da una progettazione di rete LIN:

Un frame *F* consiste di 2 byte di dati.
La lunghezza massima di bit del frame (*lung*) fornisce il tempo massimo entro cui deve essere completata la trasmissione del frame.
Esso viene calcolato in conformità alle specifiche LIN come segue:

$$lung_{MAX}(F) = lung_{MIN}(F) \cdot 1.4$$

$$lung_{MAX}(F) = \left[\left(10 \cdot n_{data} + lung_{MIN}(header) + checksumbyte\right) + 1\right] \cdot 1.4$$

$$lung_{MAX}(F) = \left[\left(10 \cdot 2 + 34 + 10\right) + 1\right] \cdot 1.4 = 91bits$$

Come si può vedere, la lunghezza minima di un frame si estende del 40%, per fornire la lunghezza massima consentita (dalle specifiche).
Questo viene fatto in modo che i frame avranno un certo tempo di risposta tra di loro, per i calcoli ed altri compiti come detto.
Se la velocità delle trasmissioni è tarata per questo progetto a 9.6kbits/s, il tempo di risposta massimo è calcolato come segue:

$$Time_{MAX} = \frac{91}{9600} \approx 9.5ms$$

Ora, se il tempo base del sistema è configurato in 5 ms, si può rilevare che il ritardo del frame *F* sarebbe almeno di $2 \cdot 5ms = 10ms$.
Gli 0,5 ms lasciati prima dell'ingresso successivo, rappresentano il tempo di "riempimento" (*slack time*) per il master.
La base dei tempi viene spesso realizzata come un timer, che fornisce un "interrupt" ogni volta che un periodo di tempo è scaduto.
L'uso di una base di tempo costituisce una raccomandazione fornita dalle specifiche LIN come un modo per l'implementazione della funzionalità del programma del master.

9.2 Il “Jitter”

Il *Jitter* (“tremolio”) è una variazione di tempo normalmente dovuta a clock più o meno imperfetti.
Con il LIN, il tremolio è causato più dai cambiamenti di tempo che dalla trasmissione dei frame.
Tali ritardi sono naturalmente importanti da osservare e misurare, in modo che il progetto e la programmazione dei frame e dei segnali, possano corrispondere alle esigenze di “*real time*”.
Inviando i frame ai nodi, si possono verificare dei tremolii sia nel master che nello slave.

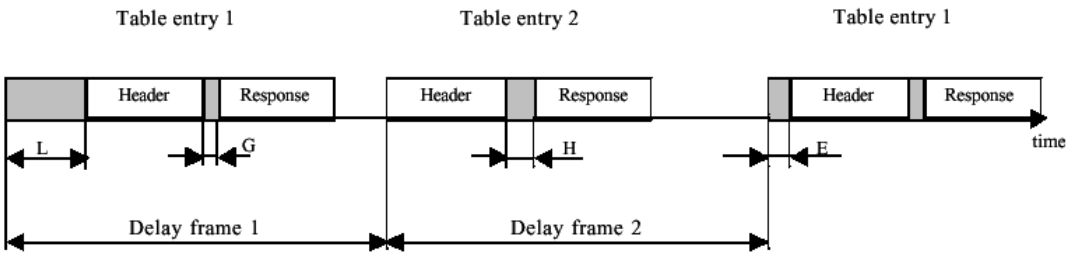


Fig. 15: Il Jitter dagli inter-frame di risposta e i ritardi di trasmissione

Il Jitter introdotto dal master può essere definito come la differenza tra il ritardo dal punto di avvio della base di tempo, all’intestazione del frame (fronte di discesa del “synch break”).
La figura 15 illustra questo concetto.
In questo esempio il tremolio introdotto dal master risulterebbe dalla sottrazione tra L ed E (L-E), ove L è il ritardo massimo ed E quello minimo (vedi figura sopra).
Nel caso del nodo slave, il tremolio può essere definito come differenza tra il tempo di risposta massimo e quello minimo tra i frame introdotti, quando lo slave risponde ad un messaggio di richiesta dal master.
Ciò può essere espresso come H-G nella stessa figura precedente.

10.0 Il LIN Physical Layer

Il bus, nel protocollo LIN, è costituito da un cavo singolo ed operante praticamente alla tensione di batteria V_{BAT} del veicolo (8-18V) (si ricorda che il LIN è nato per applicazioni Automotive).
Come illustra la seguente figura 16, ogni ECU (nodo LIN di rete) viene collegato, al resto della rete, solo da 3 cavi: l'alimentazione, il bus e il riferimento di massa.
Le linee driver/receiver (cioè il transceiver) sono implementate utilizzando lo *standard ISO 9141*, che rappresenta la base del Physical Layer per questo protocollo.

Osservazione:
Un grande vantaggio è l'utilizzo da parte del LIN e del TTP/A dello stesso "Physical Layer", definito dallo standard sopra citato.

Come si vede nella citata figura, il bus può avere 2 valori logici complementari: "zero", cioè fisicamente fino al 40% della tensione di batteria V_{BAT} e "uno", cioè dal 60% della tensione di batteria V_{BAT}.
All'ingresso di ogni ECU, il bus viene terminato mediante una *resistenza* di pull-up in serie ad un *diodo* di blocco.
Questa resistenza ha un valore variabile in funzione della natura del nodo: master o slave. Considerando che la rete LIN utilizza normalmente una topologia "a bus", dove tutti i nodi risultano in parallelo, è necessario elevare la resistenza di pull-up dei singoli slave (rispetto all'unico nodo master), per evitare una drastica riduzione dell'impedenza di rete.
Per il nodo master, essendo unico, risulta sufficiente una resistenza di $1k\Omega$.
Il diodo di blocco evita semplicemente che il bus alimenti la ECU, nel caso di abbassamento della tensione V_{BAT} di alimentazione.

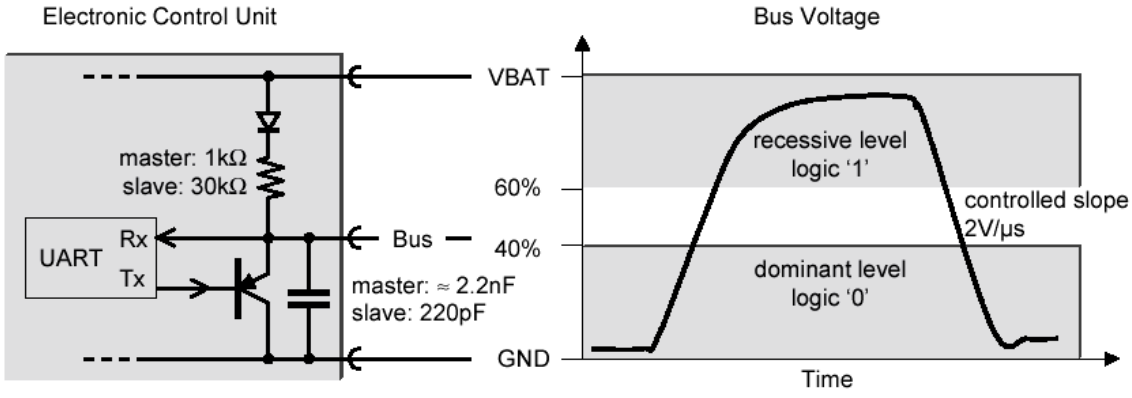


Fig. 16: Il livello fisico di LIN (a sinistra) e i livelli di tensione del bus (a destra)

Il condensatore (in parallelo tra il bus e la massa) fornisce una migliore stabilizzazione del segnale sul bus, ed in particolare nel master, avendo un valore di capacità maggiore che negli slave, può anche "servire" come "buffer" nel caso di reti con elevato numero di nodi slave.

Il valore tipico di capacità utilizzato per questo sistema (e consigliato dalle specifiche) negli slave è di $220\,pF$, mentre nel nodo master la capacità è più elevata, allo scopo di rendere meno dipendente la capacità complessiva della linea bus, al numero dei nodi slave.
Questo valore della capacità del master, è stato quindi calcolato come segue:

$$C_{master} = C_{bus} - n_{slave} \cdot C_{slave} - l_{linea} \cdot C_{linea}$$

dove: C_{bus} rappresenta la capacità propria del bus, C_{slave} quella dei nodi slave, C_{linea} quella della linea di rete, n_{slave} il numero complessivo dei nodi slave della rete e l_{linea} la lunghezza dell'intera rete LIN.

Per una rete di medie dimensioni, si possono scegliere infine questi valori:

$C_{bus} = 9nF$

$C_{linea} = 130\,pF / m$

$C_{slave} = 220\,pF$

Il comportamento del bus relativo alle interferenze elettromagnetiche (EMI), dipende principalmente dalla velocità di risposta (*slew-rate*) del segnale, unitamente ad altri fattori quali il di/dt e d^2V/dt^2.
Un valore accettabile di slew-rate potrebbe essere vicino ai $2V/\mu s$ per ridurre le emissioni da un lato e permettere velocità fino a 20kbit/s dall'altro (vedi figura 16).

Da verifiche pratiche si può affermare che questo valore di slew-rate risulta un buon compromesso fra due esigenze opposte: di sincronizzazione per alti slew-rate del segnale e di compatibilità elettromagnetica per valori di slew-rate più lenti.

11.0 Il nodo LIN

Un semplice nodo realizzato, collegato alla rete LIN è sostanzialmente costituito da un microcontrollore, un transceiver LIN e un regolatore di tensione (vedi figura 17).
Il nodo LIN, schematicamente illustrato, è da considerarsi un nodo slave in quanto, le funzioni del master vengono svolte dal microcontrollore nel nodo della rete principale d'interfaccia (se presente), oppure da un semplice altro nodo LIN.
Si ricorda che il LIN sopporta un'architettura di rete del tipo single-master/multiple-slave, quindi è assolutamente indispensabile avere un nodo master.

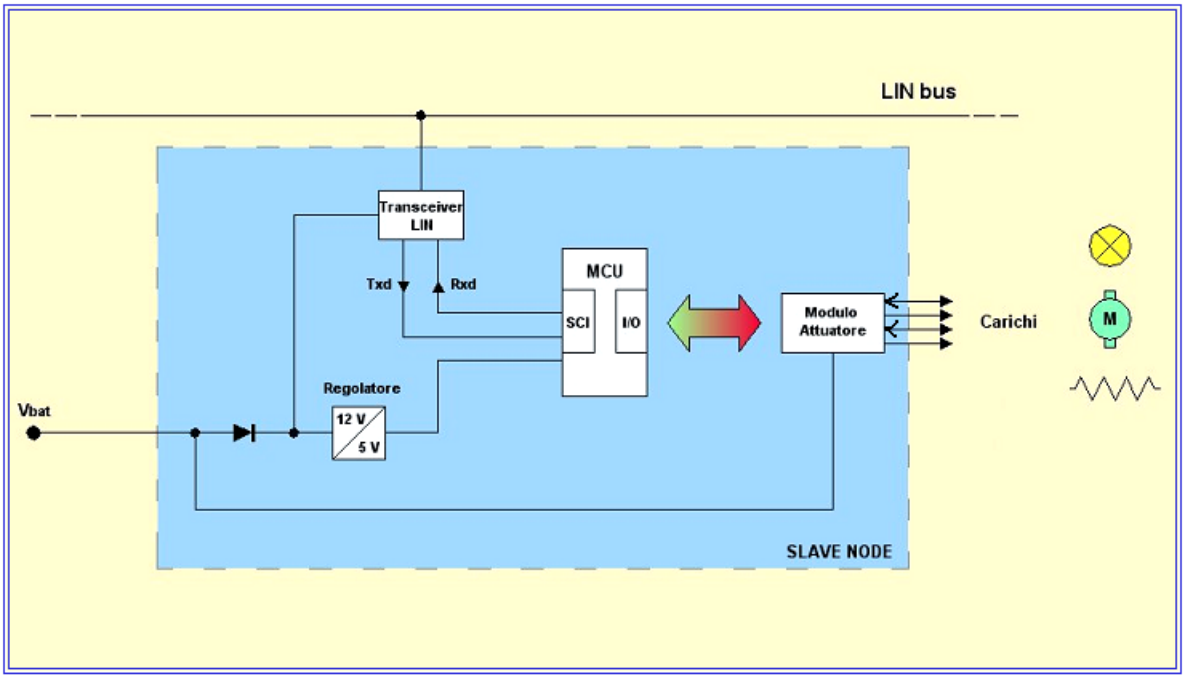


Fig. 17: Schema di principio del nodo LIN (inteso come slave)

Come si può notare, lo slave LIN non necessita di un controller specifico (come avviene per un nodo CAN), in quanto i dati (in trasmissione ed in ricezione) vengono gestiti dal microcontrollore per mezzo di una comune *interfaccia UART/SCI*, presente oggi in quasi tutti i "micro" in commercio.
Nei nodi LIN è presente un diodo che svolge la funzione di evitare in caso di guasto all'alimentazione della scheda, che il bus attraverso il transceiver, alimenti la stessa linea di alimentazione.

Si noti inoltre che l'alimentazione del transceiver LIN, viene fornita alla stessa tensione di alimentazione della scheda.
La struttura a blocchi di possibile utilizzo per una realizzazione e per lo sviluppo di nodi LIN, viene illustrata nella seguente figura 18.

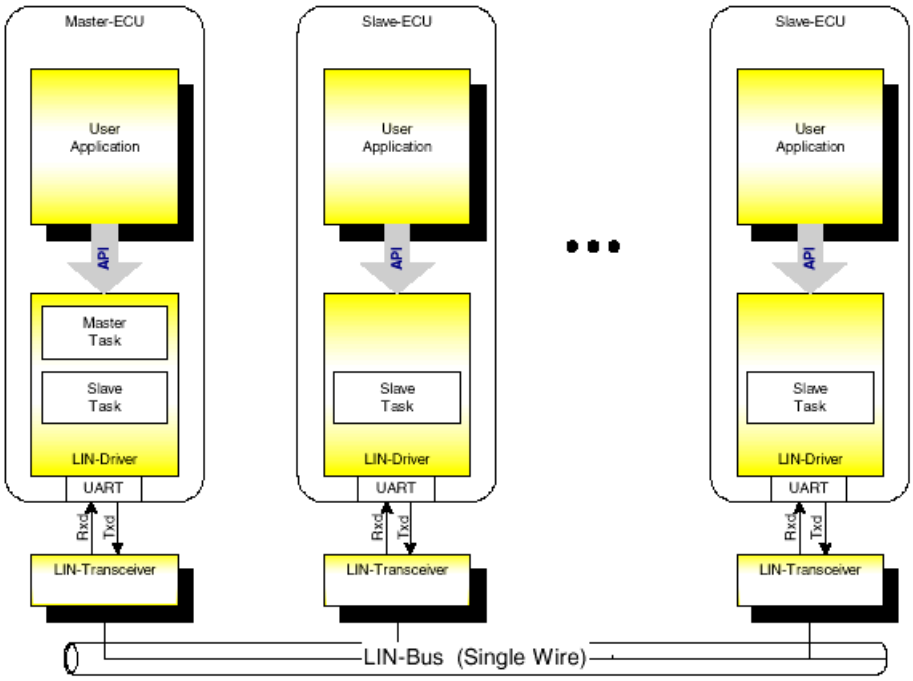


Fig. 18: Struttura a blocchi utilizzata per la realizzazione di nodi LIN

Il Physical Layer di ogni nodo LIN, in riferimento al modello ISO/OSI, viene realizzato dal transceiver e dall'interfaccia UART/SCI, mentre il livello Data Link utilizza i vari driver forniti dal protocollo.
L'applicazione è semplicemente rappresentata dal codice operativo creato e gestito dal microcontrollore.

11.1 Il Microcontrollore

L'implementazione di un nodo master o di un nodo slave di LIN, non è un'esclusiva delle specifiche LIN, ma soprattutto dipende dai requisiti del microcontrollore utilizzato.
Infatti, l'interfaccia sulla rete LIN richiede pochissimo al microcontrollore.
Ci sono diversi modi per realizzare l'interfaccia LIN in un microcontrollore ed essi sono in relazione al tipo di supporto hardware e software che il micro può offrire.
Vengono brevemente accennati alcuni microcontrollori con diversi tipi di soluzioni LIN.

- Con o senza UART:

Come già dettagliatamente illustrato, il LIN utilizza il *protocollo UART* come mezzo per trasmettere e ricevere i dati via bus.
La funzionalità UART può essere implementata nel software, ma molti microcontrollori disponibili oggi sul mercato hanno una UART implementata nell'hardware (questo è il caso più comune).
La soluzione software colloca tuttavia un carico supplementare al processore, poiché esso deve eseguire delle istruzioni aggiuntive generate per questa implementazione software dell'UART.
Così le procedure eseguite per ogni trasmissione sul bus, dovranno essere compensate da una potenza di elaborazione supplementare.
Questo naturalmente non avviene nel caso di utilizzo della soluzione hardware.

- Requisiti di tempo:

Un modulo hardware strettamente necessario per l'implementazione del LIN in un microcontrollore è il *modulo Timer*.
Esso viene utilizzato per avere traccia delle diverse proprietà di tempo del protocollo (per esempio, i settaggi della sincronizzazione, il massimo tempo di frame, il tempo di inattività tra le trasmissioni sul bus, ecc…).
La procedura di sincronizzazione condotta dai nodi slave, quando viene ricevuta un'intestazione di messaggio (message header), viene effettuata al fine di confrontare la frequenza di campionamento dello slave (cioè baud-rate), a quella del master.
Questo viene fatto per "testare" correttamente il flusso di bit e di conseguenza per ottenere una ricezione priva di errori del messaggio.
Le specifiche indicano che la differenza tra il baud-rate del master e quello dello slave, possono essere al massimo del 2% dopo una positiva sincronizzazione, altrimenti non viene garantita la corretta ricezione di un messaggio.
La procedura di sincronizzazione utilizzata (che verrà analizzata successivamente in questa bozza), è resa possibile mediante l'utilizzo come detto, di un timer.

- Caratteristiche speciali:

A completamento dei tipi di microcontrollori con le caratteristiche di base del LIN trattate in precedenza, possiamo citare diversi altri micro progettati in modo particolare, che emergono sul mercato dei nodi LIN.
Esempi di nuove caratteristiche "LIN" nei microcontrollori, sono le *interfacce UART migliorate* (per esempio, l'MC68HC908EY16, oppure l'MC68HC908QL4 di Freescale), dove la rilevazione di un'interruzione di sincronizzazione (*Synchronization Break*) dell'intestazione del messaggio (del master), viene effettuata automaticamente nell'hardware e viene notificata dopo la ricezione.
Pertanto non è assolutamente necessario, come avviene per l'UART comune, avere routine software per verificare se l'interruzione SYNC sia stata ricevuta.
Si ricorda che il "*Synchronization Break*", cioè un'interruzione formata da almeno 13 bit di "zeri" consecutivi, non sarebbe rilevata da un protocollo UART comune.
Un'altra caratteristica interessante relativa all'hardware, può essere trovata sui alcuni nuovi microcontrollori, nei quali il transceiver LIN viene incorporato nel micro stesso.
Questo rende il controllore molto compatto, piccolo e facile da usare.
Nella seguente figura 19, si mostra l'assegnazione dei pin del microcontrollore MC68HC908QL4, preso e proposto come esempio "tipico" per la realizzazione, a basso costo, di nodi slave.
Il motivo della sua scelta è rappresentato dall'ottimo rapporto prestazione/costo.
Appartiene alla famiglia dei microcontrollori a 8-bit di Freescale, è dotato di un oscillatore interno di 8MHz di clock, ha la possibilità di programmare "in circuit" una memoria flash di 4kbyte, è dotato di una UART migliorata (utilizzo della SCI) per LIN, dispone internamente di moduli Timer, di un convertitore ADC a 10-bit, ecc…(si rimanda al relativo Data Sheet per le altre caratteristiche).

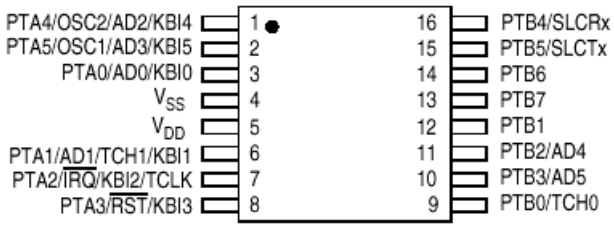


Fig. 19: Assegnazione dei pin del micro QL4

Si nota dalla figura, i pin 15 (TxD) e 16 (RxD) per il collegamento diretto al transceiver LIN.

11.2 Il Transceiver LIN

I transceivers LIN sono costruiti per funzionare come un'*interfaccia fisica*, tra il controllore di protocollo master/slave (implementato come detto sul microcontrollore) ed il bus LIN.
Normalmente i transceivers sono schermati, insieme al resto degli elementi elettronici incorporati in una ECU (nodo), per protezione contro influenze esterne (atmosferiche, meccaniche, ecc…).
Inoltre devono resistere a scariche di tensione, cortocircuiti ed altre variazioni elettriche generate da altri elementi dell'impianto elettrico o elettronico interno all'apparecchio.
Poiché il protocollo LIN utilizza un cavo singolo come bus fisico, il transceiver ha bisogno di controllare la quantità di emissione elettromagnetica (Electromagnetic Emission, EME) emessa dal ricetrasmettitore stesso e dal bus.
Ciò viene effettuato dallo "*slope control*" (spesso chiamato "*slew-rate control*") dei segnali trasmessi al bus.
Al fine di minimizzare il consumo di corrente, che è un fattore importante dal punto di vista del risparmio energetico, i transceivers sopportano una modalità di pausa "*standby*", in cui assorbono soltanto pochi micro-ampere di corrente.
Da tale modalità il transceiver può essere "risvegliato" in vari modi: in una normale modalità di funzionamento del componente di controllo (per esempio, dal microcontrollore), mediante un segnale locale (per esempio, un interruttore/pulsante esterno), oppure da un messaggio proveniente direttamente dal bus.
Una rappresentazione schematica di un classico transceiver LIN, può essere vista in figura 20.

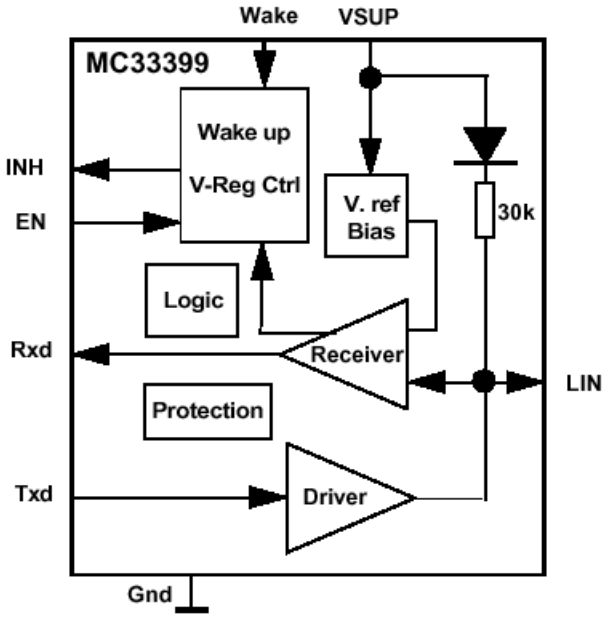


Fig. 20: Schema interno a blocchi del transceiver MC33399 (Freescale)

I transceivers vengono costruiti in conformità alle specifiche LIN.
La maggior parte dei transceivers possono gestire tensioni di funzionamento fino a 18V, che è il livello massimo specificato per il bus LIN.
Tuttavia, vi sono dei ricetrasmettitori che possono funzionare con tensioni di alimentazione fino a 40V (per esempio, Infineon TLE6259-2), oppure con semplici modifiche hardware, adattare un classico transceiver LIN a una tensione di alimentazione di 5V.
Tuttavia se alimentati a tensioni differenti da quelle consigliate, i transceivers non funzionerebbero entro i livelli richiesti dalle specifiche e come risultato sarebbe assai difficile stabilire come le proprietà (ad esempio i baud-rate e i valori EMI), potrebbero influenzare il sistema generale.
Un classico transceiver (illustrato in figura 18), è costituito principalmente da comparatori, regolatori e da buffers.
Si può brevemente notare che oltre all'interfaccia verso il bus LIN (pin "LIN") e quella verso il microcontrollore (pin "TxD" e "RxD"), è presenta anche un pin "EN", il quale, agendo direttamente sul regolatore, controlla la modalità di funzionamento operativa del transceiver (in particolare è "alto" quando il componente si trova in "normal mode" e "basso" quando è in "sleep mode").
Il pin "Wake" è utile per risvegliare il componente mediante uno switch esterno.
Si può comunque anche "risvegliare" il transceiver, direttamente da un messaggio via bus.
Per altre caratteristiche, si rimanda al relativo Data Sheet.

Nel caso si desideri costruirsi o modificare questa interfaccia fra il LIN bus e i pin del microcontrollore (chip), la seguente figura 21 mostra un possibile schema per realizzare quanto detto.

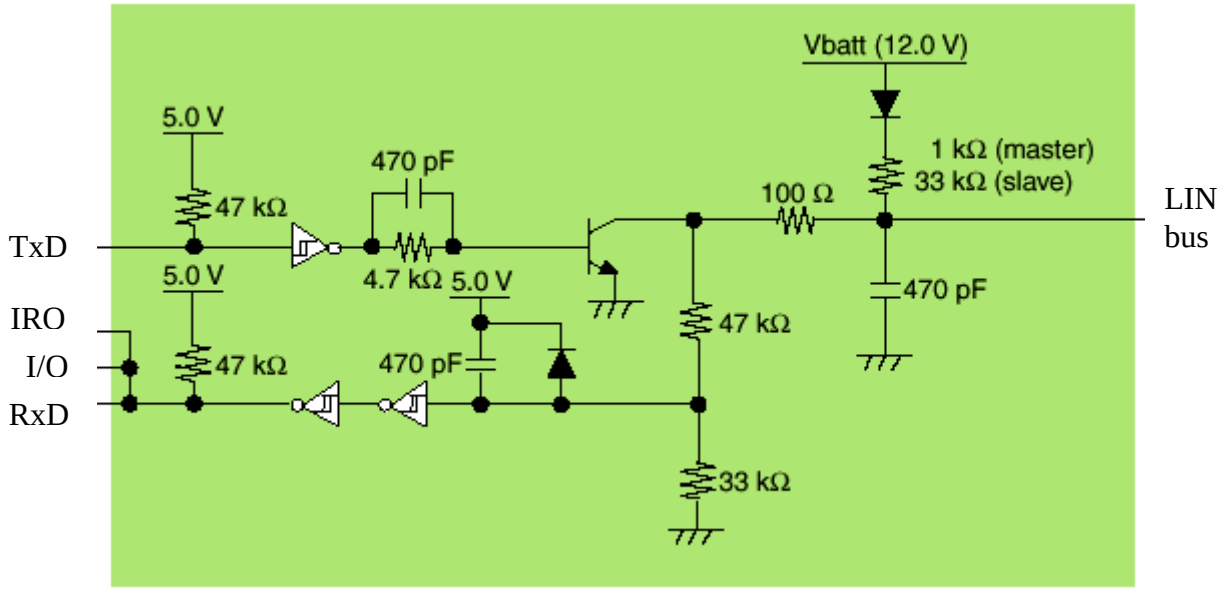


Fig. 21: Esempio d'interfaccia fra microcontrollore e bus

La parte di sinistra della figura 21, è il lato micro, mentre la parte di destra è il lato bus.

- TxD trasmette il: “sync field”, “ID field”, “response frame” e “wake-up signal”

- RxD riceve il: “response frame” e “communication error detection”

- IRQ per il: “wake-up signal detection”

- I/O Timer per la misura di: “sync break field”, “sync break delimiter” e “timeout detection”

12.0 Analisi sulla sincronizzazione

La fase più delicata del protocollo LIN, è quella di sincronizzazione degli slave sul clock imposto dal master.
Questa prima fase in un frame di messaggio LIN (inserita nel message header), è necessaria per stabilire l’inizio di un messaggio da parte del master e successivamente per calcolare il relativo tempo di bit (base dei tempi), permettendo quindi una esatta sincronizzazione con i vari slave (che come noto non sono dotati di propri oscillatori al quarzo).
E’ quindi il clock degli slave che deve adeguarsi a quello del master.
Si possono specificare 2 livelli di sincronizzazione per gli slave di LIN:

- “unsynchronized”: quando il clock dello slave $F_{unsynch}$ differisce meno del $\pm 15\%$ dal clock del master F_{master}

- “synchronized”: quando il clock dello slave F_{synch} differisce meno del $\pm 2\%$ dal clock del master F_{master}

Un nodo slave con un oscillatore stabilizzato, viene sempre considerato “synchronized”, mentre uno slave senza un clock stabilizzato, viene assunto come “unsynchronized”, per esempio, dopo la fine di un messaggio, dopo un reset o dopo l’uscita da una modalità “sleep”.
Gli oscillatori RC, integrati nel chip utilizzato, possono raggiungere un grado di tolleranza del clock migliore del $\pm 15\%$, mediante una semplice pre-calibrazione.
Dopo la fase di sincronizzazione vera e propria, l’oscillatore RC integrato nel microcontrollore, deve essere stabile (con $F_{unsynch}$) per il resto del messaggio, tenendo presente anche i vari disturbi, come le variazioni di temperatura o di tensione che possono presentarsi.

La sincronizzazione viene compiuta in 2 fasi:

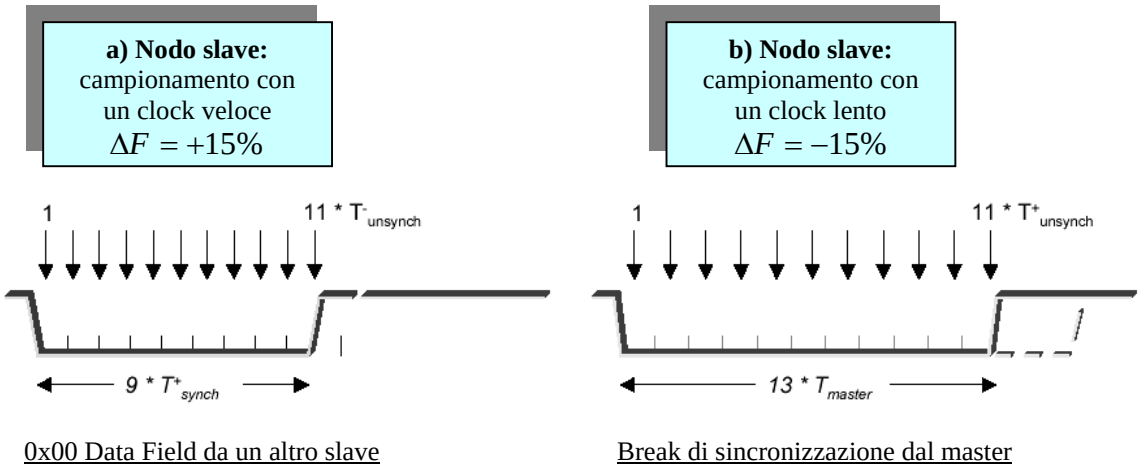
1) la fase di rilevazione del break di sincronizzazione (synch break)
2) la fase di sintonia per ottenere lo stesso tempo di bit (base dei tempi)

Nella prima fase vengono considerate le seguenti 2 ipotesi per la rilevazione del break:

a) Un "unsynchronized" slave che campiona velocemente il bus con $F^{+}_{unsynch} = F_{master} + 15\%$, non deve confondere un normale "0x00" dato in transito, come un break

b) Il segnale di break deve essere lungo abbastanza per essere identificato da un "unsynchronized" slave, che campiona lentamente il bus con $F^{-}_{unsynch} = F_{master} - 15\%$

Queste ipotesi considerate, vengono anche illustrate graficamente nella seguente figura 22.



0x00 Data Field da un altro slave

Break di sincronizzazione dal master

Fig. 22: Illustrazione dei requisiti per la rilevazione del break di sincronizzazione

L'ipotesi (a) porta ad avere una specifica soglia T_{soglia} per la rilevazione del break da parte degli slave.
Indicando con $T_{0x00}^{\max}$ la massima lunghezza di un normale 0x00 data byte, si ottiene:

$$T_{0x00}^{\max} = 9 \cdot T_{unsynch}^{+} = \frac{9}{F_{synch}^{-}} = \frac{9}{F_{master} \cdot (1 - 2\%)} = 9.18 \cdot T_{master} \qquad (1)$$

dove il termine $(1 - 2\%)$ evidenzia che il dato 0x00 può essere inviato da uno slave con il –2% di variazione.
Considerando le specifiche del protocollo LIN (dove T_{soglia} = 11 tempi di bit dello slave), uno slave veloce non può confondere un dato 0x00:

$$11 \cdot T_{unsynch}^{\min} > T_{0x00}^{\max} = 9.18 \cdot T_{master} \qquad (2)$$

Riscrivendo la (2) con:

$$T_{unsynch}^{-} = \frac{1}{F_{unsynch}^{+}} = \frac{1}{(1 + \Delta F^{+}) \cdot F_{master}} = \frac{T_{master}}{(1 + \Delta F^{+})} \qquad (3)$$

risulta:

$$11 \cdot \frac{T_{master}}{1 + \Delta F^{+}} > 9.18 \cdot T_{master} \quad \text{oppure} \quad \Delta F^{+} < \frac{11}{9} \cdot (1 - 2\%) - 1 \qquad (4)$$

rispettivamente.

Risolvendo l'equazione (4), risulta $\Delta F^{+} < 19.7\%$.
Questa è la deviazione di frequenza massima, sotto alla quale uno slave si sincronizzerà.

L'ipotesi (b) si riferisce alla lunghezza minima di un break di sincronizzazione, che viene specificato nelle norme di protocollo LIN, come $T_{syn\text{-}brk}$ = 13 tempi di bit del master.

Uno slave lento, rileverà il break se:

$$11 \cdot T^{+}_{unsynch} < 13 \cdot T_{master} \tag{5}$$

Riscrivendo la (5), con:

$$T^{+}_{unsynch} = \frac{1}{F^{-}_{unsynch}} = \frac{1}{(1-\Delta F^{-}) \cdot F_{master}} = \frac{T_{master}}{(1-\Delta F^{-})} \tag{6}$$

risulta:

$$11 \cdot \frac{T_{master}}{(1-\Delta F^{-})} < 13 \cdot T_{master} \qquad \text{oppure} \qquad \Delta F^{-} < 1 - \frac{11}{13} \tag{7}$$

rispettivamente.

Risolvendo l'equazione (7), risulta $\Delta F^{-} < 15.3\%$.

Questa è la deviazione di frequenza minima, sotto alla quale uno slave si sincronizzerà.

13.0 Una topologia consigliata per la rete LIN

Dalle considerazioni fatte in questa breve bozza, risulta ora semplice tracciare una struttura di rete LIN, come mostra la seguente figura 23.
In particolare, la figura illustra uno schema di principio classico, formato da un master e dai possibili 15 slaves, indicando inoltre la massima lunghezza della rete che risulta ampiamente conforme alle vostre specifiche.

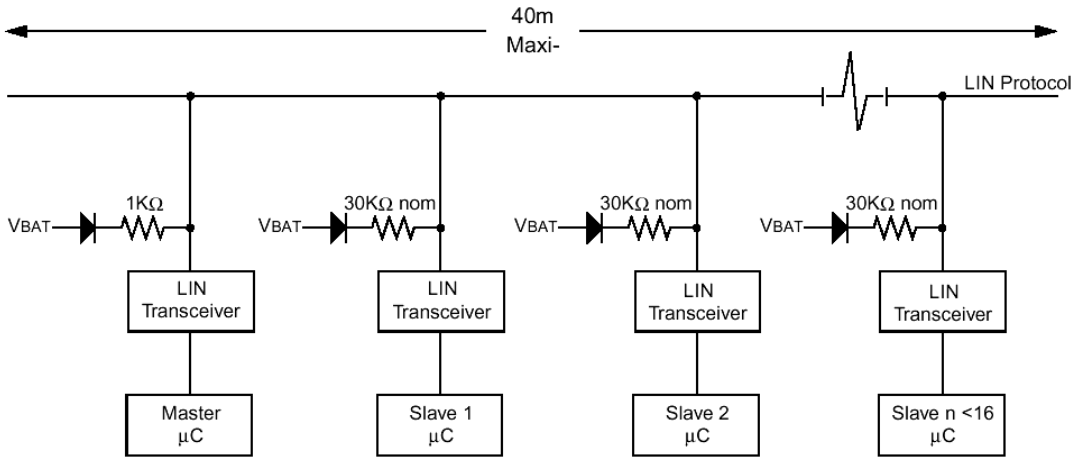


Fig. 23: Topologia classica della rete LIN

Come già ricordato, ogni nodo (compreso il master) è costituito da un microcontrollore per la gestione del nodo, da un transceiver per l'interfaccia sul bus e da qualche componente passivo necessario per il funzionamento pratico del nodo.
E' utile notare che anche se ogni nodo svolge funzioni complesse (per esempio, nella gestione di attuatori o di sensori relativi al nodo stesso), gli unici cavi di collegamento al resto della rete sono solo 3 (l'alimentazione, il bus e la massa).
Questa notevole riduzione del cablaggio e la possibilità di avere le ECU "embedded" in sensori o attuatori, rappresentano una delle scelte migliori ad architettura distribuita, di grande successo, sia in ambito automotive che in quello industriale del vostro settore.

14.0 Cenni sui tools software di possibile utilizzo specifici per il LIN

Un fattore importante durante lo sviluppo di una rete LIN, è l'utilizzo di strumenti per la progettazione, la gestione, l'analisi ed il controllo del protocollo stesso e delle sue funzioni.
Per il protocollo LIN vi sono svariati strumenti sul mercato, molti provenienti da aziende con un ampia esperienza nelle tecnologie di comunicazione in ambito automotive.
Il vantaggio di tale esperienza è nel possedere già strumenti sviluppati per altri protocolli, (per esempio per CAN), che possono essere adattati per "incorporare" il LIN.
Alcuni tools utilizzati per lo sviluppo di una rete LIN, sono fortemente indirizzati alla pura analisi e simulazione, per esempio il *Volcano Linspector* e il *Kvaser Navigator*.
Questi sono realizzati per inserirsi in una reale rete LIN ed interagire con essa o analizzarla.
Per fare questo è necessario un certo tipo d'interfaccia hardware tra computer ed il bus (normalmente disponibile in commercio).
Un'impostazione tipica può essere quella di figura 24.

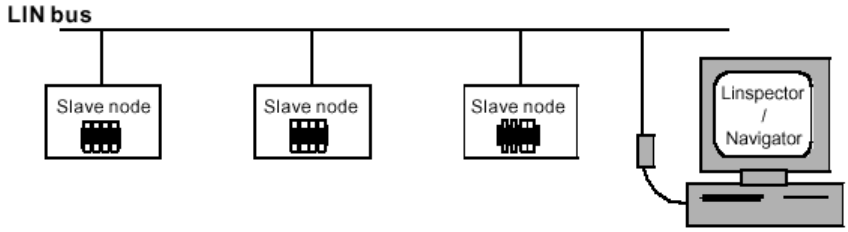


Fig. 24: Tipica impostazione per l'analisi e la simulazione con LINspector e Navigator

Altri strumenti incorporano le funzionalità per gestire i database del segnale LIN, generando le caratteristiche di comunicazione della rete ed impostando completamente la stessa.
Tra questi si può trovare l'*LNA* di Volcano.
Volcano Automotive Group e Vector Informatik GmbH sono aziende che forniscono completi processi per trattare l'intero flusso di lavoro (dalla progettazione di vari tipi di comunicazione e gestione di reti, fino al collaudo ed alla verifica finale).

14.1 Volcano Linspector

Il *LINspector* è sviluppato dalla Volcano Communications Technologies e rappresenta, come il Kvaser Navigator, uno strumento software progettato per lo studio e per l'analisi delle reti LIN.
Le somiglianze tra i due programmi sono numerose, così come le funzionalità.
Una differenza invece tra i due, è che il LINspector è esclusivamente orientato al protocollo LIN, mentre l'altro tool può anche essere utilizzato in reti CAN.
Per l'emulazione dei nodi (*LIN Emulation Control* "*LEC*"), vengono utilizzati degli scripts.
La funzione dello script è la stessa di quella dello script USR per il Navigator della Kvaser ed anche la sintassi relativa è molto simile.
Tuttavia gli script LEC non utilizzano una procedura guidata dell'evento, ma un loop infinito che viene eseguito durante l'intera simulazione.
Nel LINspector esiste la possibilità di utilizzare un programma periferico di output, per fornire un display grafico degli eventi durante una simulazione (*LINgo*).
Questo programma fornisce la possibilità all'utente di progettare i display grafici, per mostrare gli eventi di una simulazione.
Il display può quindi essere collegato alla simulazione ed interagire ai messaggi e ai dati.
Simulando per esempio una ECU, che gestisce una semplice funzionalità, l'utente può per esempio, fare accendere, lampeggiare o spegnere un led, dopo la ricezione dal bus di un certo segnale.
E' ovviamente anche possibile l'azione contraria, cioè tra il display LINgo ed il simulatore. Questo avviene modificando i valori di segnale, mediante l'interazione dell'utente con il display (per esempio premendo un pulsante).

14.2 Kvaser Navigator

Il *Kvaser Navigator*, della società svedese Kvaser, è un valido strumento non solo per l'analisi del traffico su reti CAN e LIN, ma anche per il collaudo sulla "correttezza" dei nodi.
Questo tool fornisce svariate finestre, mostrando vari dati e grafici interagenti fra loro.
Il programma Navigator richiede una configurazione iniziale, prima che venga avviata la fase di misurazione/simulazione.
Questo consiste nello sviluppo di un *LCF* (*LIN Configuration File*) con la configurazione della rete, programmando le finestre per mostrare i dati nella forma desiderata e scrivendo gli scripts necessari per emulare internamente i nodi della rete.
Avviata la simulazione, nelle finestre appaiono informazioni relative ai frame specifici ed alle statistiche di rete.
Possono essere rappresentati sia i dati numerici che quelli grafici.
Il linguaggio degli script di Navigator, chiamato *USR* (*Universal Scripting Real-time Language*), viene utilizzato per simulare il comportamento dei nodi emulati dal tool.
La sintassi di USR è simile al linguaggio C, quindi può essere facilmente recepita.
L'arrivo dei messaggi fa scattare le azioni corrispondenti nei nodi, cosa che avviene perché l'USR sta usando un tipo guidato di evento da eseguire.
Lo script è sostanzialmente configurato in modo che l'ingresso di certi messaggi origina delle azioni.
Il linguaggio consente inoltre all'utente, di impostare dei temporizzatori nel codice, che possono essere utilizzati per simulare le latenze nel nodo o per altri eventi periodici.
Una delle caratteristiche nel Navigator, durante la simulazione di un nodo master, (che non si trova nel LINspector) è la possibilità di modificare i messaggi, inviati nel tempo di esecuzione.
Questa caratteristica è utile quando non si desidera modificare un intero file *LCF*, ma si ha, per esempio, solo la necessità di verificare funzioni specifiche in un nodo slave.

15 Bibliografia

[1] S. Maggi, *"Sistemi elettronici distribuiti in Automotive"* Automazione Oggi - by VNU business publications, n°258 Maggio 2003, pag. 148-159

[2] S. Maggi, F. Castelli Dezza, *"Automotive electronics: sistemi elettronici distribuiti e comunicazione via Bus"*, Corso per la Formazione Permanente (Politecnico di Milano) n°42, 17-19 Maggio 2004, Milano

[3] S. Maggi, F. Castelli Dezza, *"Problemi e tecnologie in Automotive: verso la mobilità sostenibile"*, Giornata di Studio, (Politecnico di Milano e Regione Lombardia), 19 Novembre 2003, Milano

[4] http://www.lin-subbus.org sito ufficiale del Consorzio LIN (Local Interconnect Network)

[5] H. Kern, H. Geschloessl, *"Seamless solutions for LIN"*, (SAE 2001-01-0065), SAE 2001, pag.37-44

[6] H. Von der Wense, *"Introduction to Local Interconnect Network"*, SAE World Congress, Detroit, March 2000, (SAE 2000-01-0145), SAE Press

[7] M. Bender, *"Cost effective LIN bus Automotive Networking Microcontroller"*, (SAE 2004-01-1742), SAE 2004

[8] M. Ruff, *"Evolution of Local Interconnect Network (LIN) solutions"*, 2003 IEEE, pag.3382-89

[9] H. Von der Wense, A.J. Pohlmeyer, *"Building LIN application"*, (SAE 2001 World Congress), March 2001

[10] W. Elmenreich, M. Delvai, *"Time-Triggered Communication with UARTs"*, 4th IEEE International Workshop on Factory Communication Systems, Sweden, August 2002

[11] W. Specks, A. Rajnàk, *"LIN: Protocol, Development tools and Software Interfaces for Local Interconnect Network in Vehicles"*, 9th International Conference on Electronica Systems for Vehicles, Baden-Baden, October 2000

[12] *LINspector* tool della "Volcano Communications Technologies"

[13] *Kvaser Navigator* tool della "Kvaser"

Indice