Introduction to

LIN
(Local Interconnect Network)

Stéphane REY
Revision 1.0 - May 13th, 2003

Table of content

1 INTRODUCTION

1.1 Purpose of the document

This document is intended to give a general introduction to the LIN. It’s a compilation of informations from the lin specifications. It describes the features and highlights the main advantages of this communication bus.

1.2 Acronymous

CRC	Cyclic Redundant Code
CAN	Control Area Network
CPLD	Complex Programmable Logic Device
ECU	Electronic Control Unit
ISO	International Standard Organisation
LIN	Local Interconnect Network
LSB	Lower Significant Bit
MSB	Most Significant Bit
OSI	Open Systems Interconnections
SCI	Serial Communication Interface
UART	Universal Asynchronous Receiver Transmitter

1.3 Related documents

- LIN consortium steering committee : http://www.lin-subbus.org/

1.4 Reference documents

- LIN specification package revision 1.3 – *december 12th, 2002 – LIN consortium*

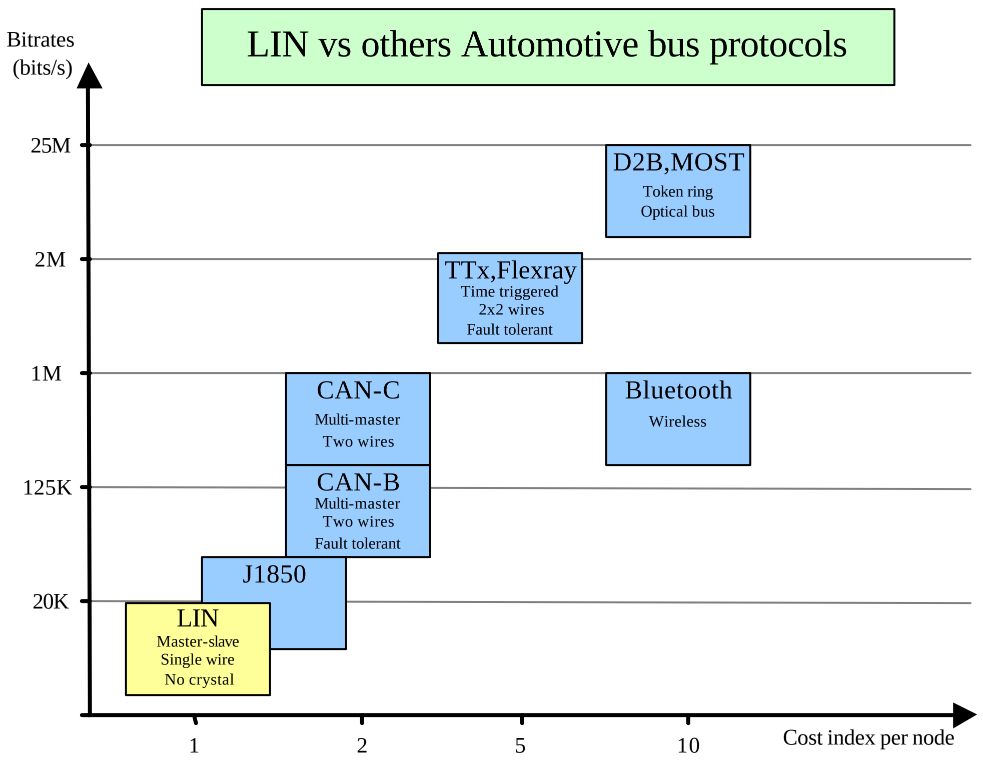
2 OVERVIEW

2.1 Introduction

LIN was designed by the LIN consortium and the first specification has been published in 1999. The consortium members are mainly European cars constructors : Audi AG, BMW AG, Daimler Chrysler AG, Volkswagen AG, Volvo Cars Corporation AB, Motorola and Volcano Communications Technologies.
Many car constructors are currently implementing LIN in their vehicles like PSA.

It was particularly designed for low-cost communication between smart sensors and actuators in automotive applications. It's intended to be used when high bitrates communications as CAN bus are not needed. It can be based on the UART/SCI hardware interface, software UART or state-machine.

Applications may be Instrument cluster, air conditioning, seats, mirrors, rain sensors, light sensors, door locking, windows, ...



2.2 Main features

- Mono-master, up to 15 slaves
- 1 wire bus
- Bitrates from 1 to 20 Kbits/s : 2.4, 9.6 and 19.2 Kbits are usually used in automotive applications
- Multicast (broadcast) messages

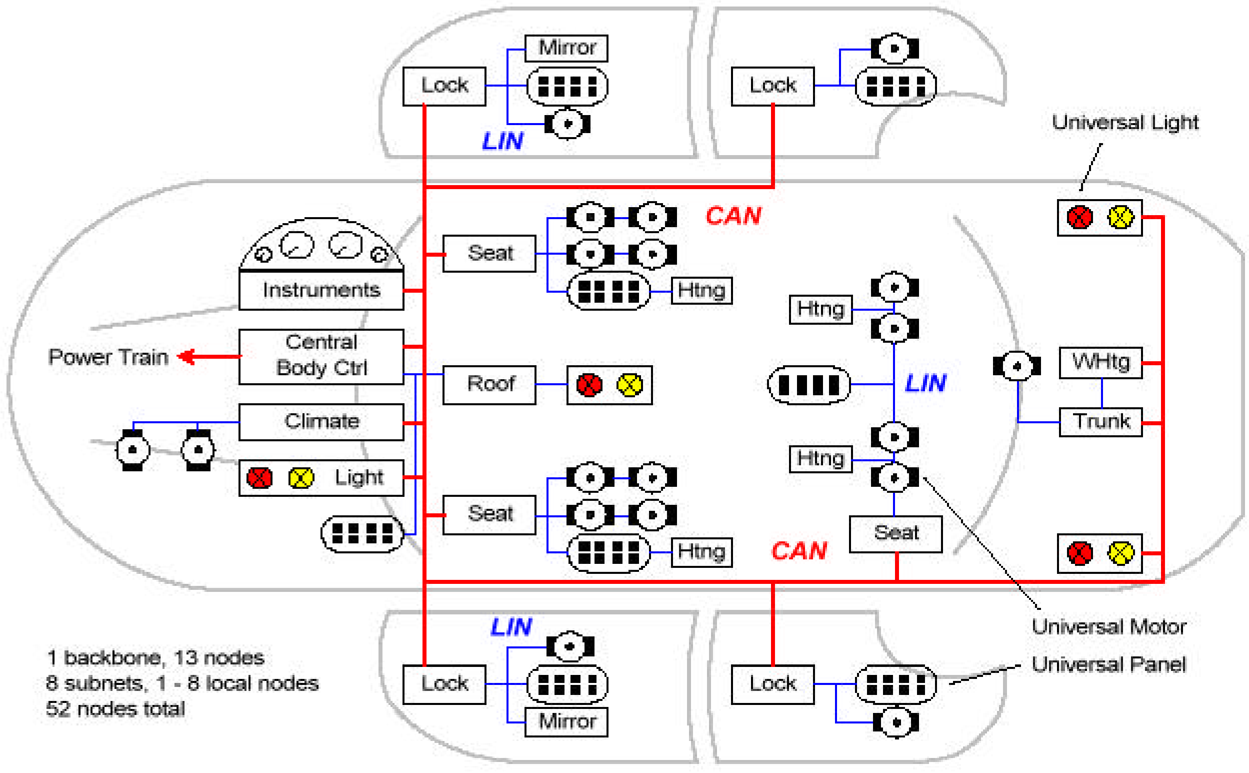
- Self-synchronization of the slave (only the master has an accurate clock as crystal)
- Messages with 2,4 or 8 data bytes, 3 control bytes
- Error detection by 8 bits checksum and 2 parity bits in identifier
- Physical layer : ISO9141
- Sleep / wake-up capability

The LIN allows to implement a serial communication in state-machine, small microcontrollers or CPLDs. A slave ECU doesn't need an accurate clock and crystals or resonators could be replaced by RC cell. This is a way to design some smart actuators or sensors, or smart connectors very cost effective.

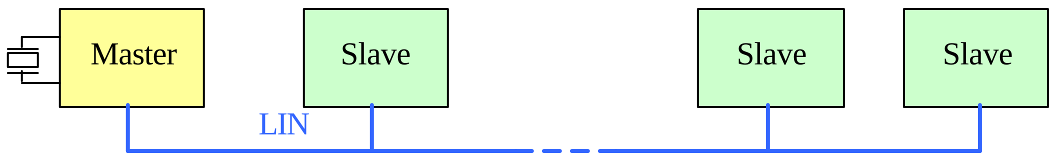
The specifications describes 3 of the 7 layers of the OSI model : physical layer, data link and application layers.

2.3 Topology

A LIN network consists of a LIN master and one or several LIN slaves. Usually in automotive application, the LIN bus is connected between smart sensor or actuators and an Electronic Control Unit (ECU) which is often a gateway with CAN bus.
You may find several LIN busses not interconnected between them as shown in the figure below. This is a major difference with other low-cost busses as K-line which was intended to link all the ECUs to an external analyse tool for diagnosis purpose.

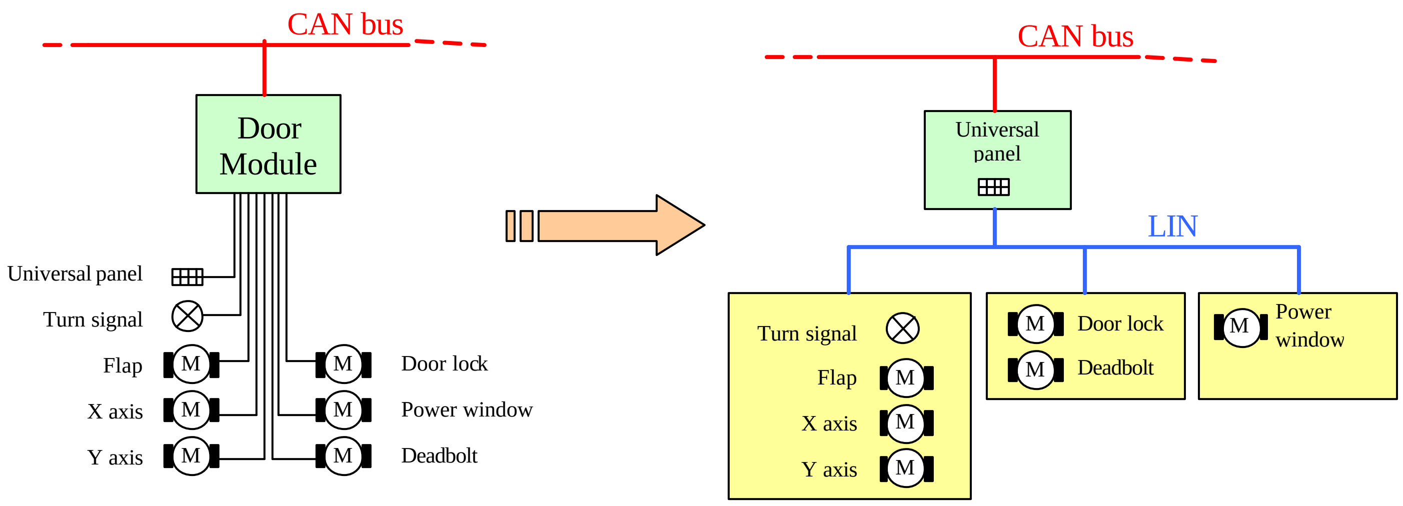


A LIN bus length is limited to 40 meters and up to 16 ECUs could be connected.



2.4 Example of LIN implementation

Source : Infineon Technologies



2.5 LIN advantages

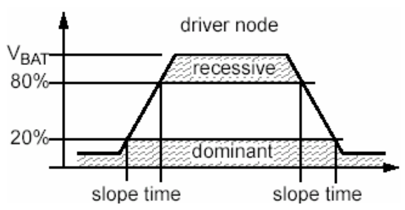
- Easy to use,
- Components available,
- Cheaper than CAN and other communications busses,
- Harness reduction,
- More reliable vehicles,
- Extension easy to implement.

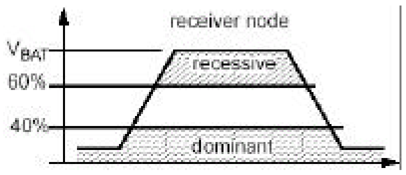
3 PROTOCOL

The LIN master knows the sequential order of all data to be transmitted and sends requests to slaves. These requests are achieved by sending a header.

3.1 Structure of a message

The recessive and dominant level are defined like this :



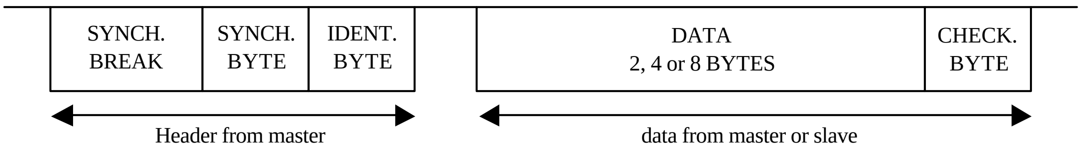


The bus is in recessive level when not busy.

A message contains the following fields :

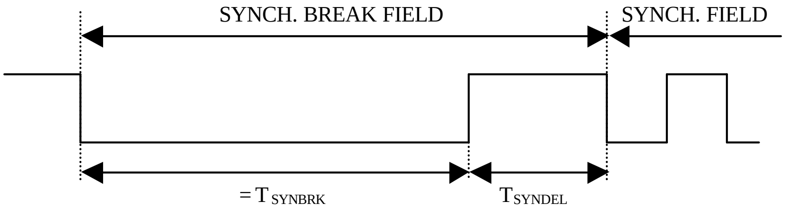
- Synchronization break,
- Synchronization byte,
- Identifier byte,
- Data bytes,
- Ckecksum byte.

The data are INTEL coded which means that LSB are sent first.



3.1.1 *Synchronization break*

A slave can detect a synchronization break by measuring the low phase duration which is higher than a databyte length.

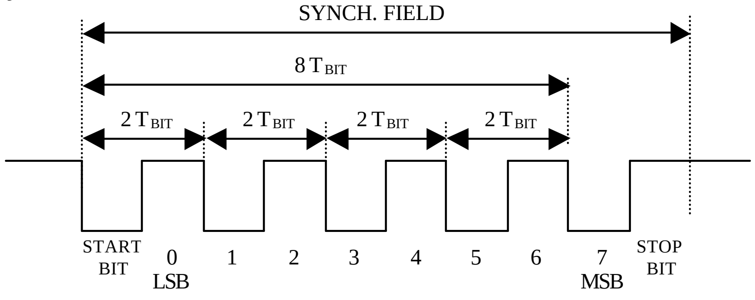


SYNCH. BREAK FIELD	LEVEL	NAME	MIN $[T_{BIT}]$	NOM $[T_{BIT}]$	MAX $[T_{BIT}]$
Synch. Break low phase	dominant	T_{SYNBRK}	13[a]		-
Synch. Break delimiter	recessive	T_{SYNDEL}	1[a]		-
Synch. Break threshold slave	Dominant	T_{SBRKTS}		10[b]	
				9[c]	

Clock timing from : (a) master, (b) slave with crystal or resonator, (c) slave without crystal or resonator

3.1.2 Synchronization byte

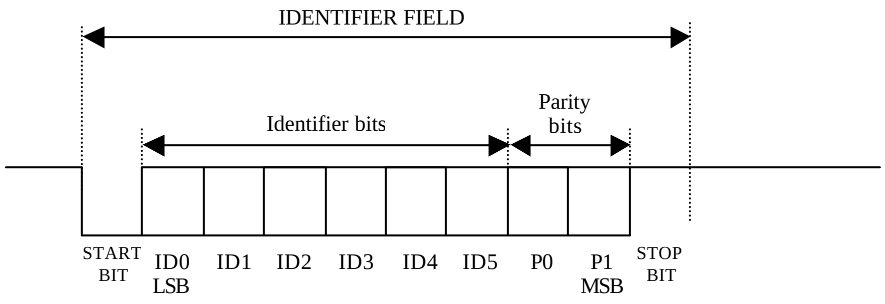
The synchronization byte is only sent by the master. It's used for slave self-synchronization.



A slave defines a bit duration by measuring the 8 bits length and dividing by 8. This defines the baudrate.

3.1.3 Identifier byte

The identifier byte defines the containt and length of the following data. There is 64 identifiers sorted in 4 groups of 16 messages.



- ID [0...3] = message number

This is the message identifier. There is identifiers from the specification which covers the LIN protocol. Some identifiers are reserved for future extension and custom messages.

- ID [4...5] = data length

This is the length of the following data bytes.

ID4	ID5	data length
0	0	2 bytes
0	1	2 bytes
1	0	4 bytes
1	1	8 bytes

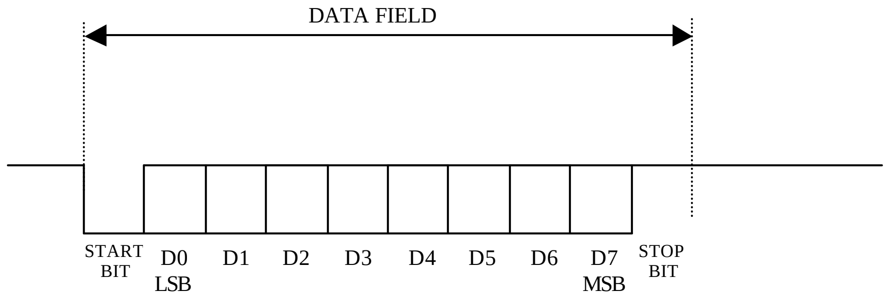
- P [0...1] = parity bits

The parity is computed on the identifier bits only.
P0 is the even parity bit, P1 is the odd parity bit.

$P0 = \text{not}(ID1 \oplus ID3 \oplus ID4 \oplus ID5)$
$P1 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$

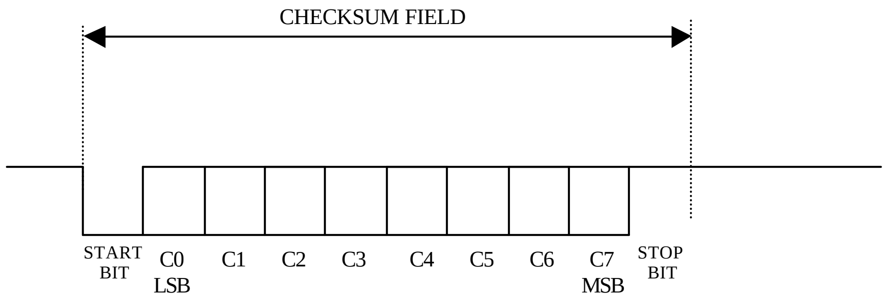
3.1.4 Data byte

The data length is defined by ID5 and ID6 from the identifier field. It can be 2, 4 or bytes. The data can be transmitted either by the master or the slave.



3.1.5 Cheksum byte

The checksum (CRC) is computed only on the data field. All other fields are not included.



The value of the checksum is equal to the inverted modulo 256 sum.
This means : Checksum = $(1 - \Sigma_{data})$ mod 256.
The carry of the previous addition is added to the LSB of the next addition.

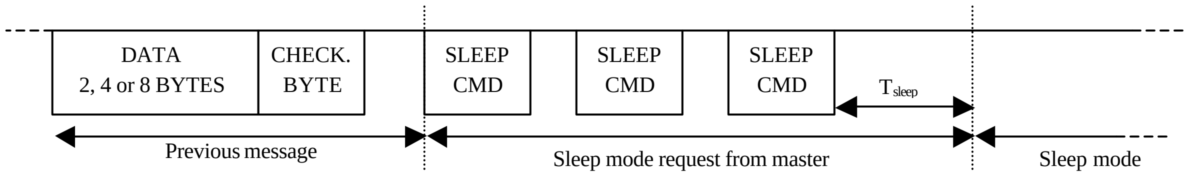
3.2 Wake-up / Sleep modes

The LIN protocol allows to be in sleep mode to be compliant with the standards and the environmental constraints. When the vehicle isn´t used, the consumption of the whole vehicle have to be less than a few milliamps in order to not discharge the battery. So, each ECU has to enter in sleep mode. Thus, strategies have been implemented to manage the sleep/wake-up modes on the communication busses.

3.2.1 Sleep mode conditions

3.2.1.1 Request from the master

The master has to send three times the sleep mode request : identifier 0x3C and data byte 0x00. The slaves have to enter in sleep mode in less than 25000 BitTime and then the bus remains to the recessive level.



The time between the first sleep command and the effective sleep mode is below 30ms+T_{sleep}.

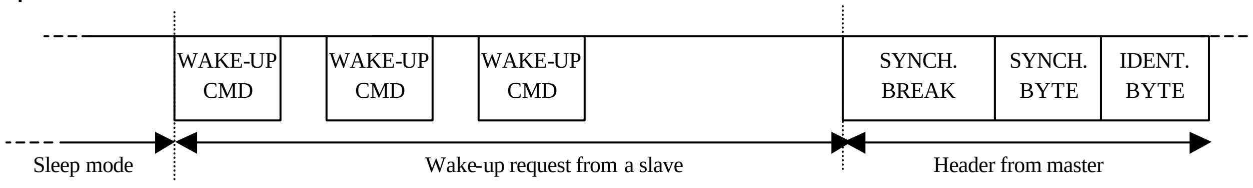
3.2.1.2 Bus inactivity

When the bus remains in the recessive level during a specified time, the ECUs have to enter in sleep mode.

3.2.2 Wake-up conditions

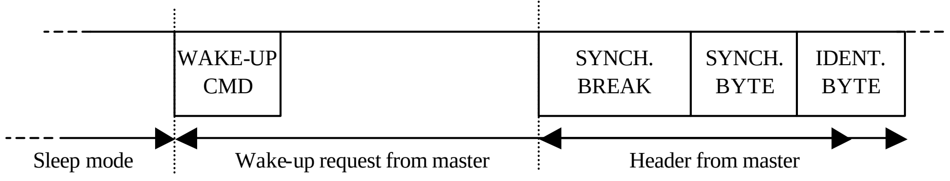
3.2.2.1 message from the slave after an event

If an event occurs, a slave can awake the master and the communication will start again. It will use the wake-up identifier 0x80. The master has to manage a bad wake-up signal. When the bus becomes dominant, it's a wake-up condition. However, the master has to check the wake-up message validity to start again the normal operation.



3.2.2.2 message from the master

The master can awake the slaves by sending the wake-up message 0x80. Once the request has been sent, the master starts again its normal communication.



3.3 Errors detection

There is no error diagnosis thru the LIN. This means that the detected errors are not sent on the LIN. However, the following errors have to be managed by the ECUs :

- bit error : the monitored output bit is different from the transmitted one,
- checksum error,
- identifier parity error,
- slave no answer,
- bad synchronization frame,
- inactive bus = sleep mode.

The master has to manage most of the errors recovering. It means that it has to apply a specified faliure mode when needed.

3.3.1 Master error detection

3.3.1.1 During transmission

During transmission the master has to detect the following errors :

- bit error,
- identifier parity error.

3.3.1.2 During reception

When receiving messages from slaves, the master has to detect :

- checksum errors,
- no slave answer.

3.3.2 Slave error detection

3.3.2.1 During transmission

When transmitting a message, the slave has to monitor the follwing error :

- bit error

3.3.2.2 During reception

When receiving, the slave has to detect :

- identifier parity errors,
- checksum errors,
- no slave answer,
- bad synchronisation frame.

- End Of Document -