Local Interconnect Network

LIN (**Local Interconnect Network**) is a serial network protocol used for communication between components in vehicles. The need for a cheap serial network arose as the technologies and the facilities implemented in the car grew, while the CAN bus was too expensive to implement for every component in the car. European car manufacturers started using different serial communication topologies, which led to compatibility problems.

In the late 1990s, the LIN Consortium was founded by five automakers (BMW, Volkswagen Group, Audi Group, Volvo Cars, Mercedes-Benz), with the technologies supplied (networking and hardware expertise) from Volcano Automotive Group and Motorola. The first fully implemented version of the new LIN specification (LIN version 1.3) was published in November 2002. In September 2003, version 2.0 was introduced to expand capabilities and make provisions for additional diagnostics features. LIN may be used also over the vehicle's battery power-line with a special LIN over DC powerline (DC-LIN) transceiver.

1 Network topology

LIN is a broadcast serial network comprising 16 nodes (one master and typically up to 15 slaves). All messages are initiated by the master with at most one slave replying to a given message identifier. The master node can also act as a slave by replying to its own messages. Because all communications are initiated by the master it is not necessary to implement a collision detection.[1]

The master and slaves are typically microcontrollers, but may be implemented in specialized hardware or ASICs in order to save cost, space, or power.

Current uses combine the low-cost efficiency of LIN and simple sensors to create small networks. These subsystems can be connected by back-bone-network (i.e. CAN in cars).[2]

2 Overview

The LIN bus is an inexpensive serial communications protocol, which effectively supports remote application within a car's network. It is particularly intended for mechatronic nodes in distributed automotive applications, but is equally suited to industrial applications. It is intended to complement the existing CAN network leading to hierarchial networks within cars.

In the late 1990s the Local Interconnect Network (LIN) Consortium was founded by five European automakers, Volcano Automotive Group and Freescale (Formerly Motorola). The first fully implemented version of the new LIN specification was published in November 2002 as LIN version 1.3. In September 2003 version 2.0 was introduced to expand configuration capabilities and make provisions for significant additional diagnostics features and tool interfaces.

The protocol's main features are listed below:

- Single master, up to 16 slaves (i.e. no bus arbitration). This is the value recommended by the LIN Consortium to achieve deterministic time response.[3]
    - Slave Node Position Detection (SNPD) allows node address assignment after power-up[4]
- Single wire communications up to 19.2 kbit/s @ 40 meter bus length.[3][5] In the LIN specification 2.2,[4] the speed up to 20 kbit/s.
- Guaranteed latency times.
- Variable length of data frame (2, 4 and 8 byte).
- Configuration flexibility.
- Multi-cast reception with time synchronization, without crystals or ceramic resonators.
- Data checksum and error detection.
- Detection of defective nodes.
- Low cost silicon implementation based on standard UART/SCI hardware.
- Enabler for hierarchical networks.
- Operating voltage of 12 V.[3]

Data is transferred across the bus in fixed form messages of selectable lengths. The master task transmits a header that consists of a break signal followed by synchronization and identifier fields. The slaves respond with a data frame that consists of between 2, 4 and 8 data bytes plus 3 bytes of control information.[4]

3 LIN message frame

A message contains the following fields:[4]

- Synchronization break
- Synchronization byte
- Identifier byte
- Data bytes
- Checksum byte

3.1 Frame types

1. **Unconditional frame.** These always carry signals and their identifiers are in the range 0 to 59 (0x00 to 0x3b). All subscribers of the unconditional frame shall receive the frame and make it available to the application (assuming no errors were detected).
2. **Event-triggered frame.** The purpose of this is to increase the responsiveness of the LIN cluster without assigning too much of the bus bandwidth to the polling of multiple slave nodes with seldom occurring events. The first data byte of the carried unconditional frame shall be equal to a protected identifier assigned to an event-triggered frame. A slave shall reply with an associated unconditional frame only if its data value has changed. If none of the slave tasks responds to the header the rest of the frame slot is silent and the header is ignored. If more than one slave task responds to the header in the same frame slot a collision will occur, and the master has to resolve the collision by requesting all associated unconditional frames before requesting the event-triggered frame again.
3. **Sporadic frame.** This frame is transmitted by the master as required, so a collision cannot occur. The header of a sporadic frame shall only be sent in its associated frame slot when the master task knows that a signal carried in the frame has been updated. The publisher of the sporadic frame shall always provide the response to the header.
4. **Diagnostic frame.** These always carry diagnostic or configuration data and they always contain eight data bytes. The identifier is either 60 (0x3C), called master request frame, or 61(0x3D), called slave response frame. Before generating the header of a diagnostic frame, the master task asks its diagnostic module if it shall be sent or if the bus shall be silent. The slave tasks publish and subscribe to the response according to their diagnostic module.
5. **User-defined frame.** These can carry any kind of information. Their identifier is 62 (0x3E). The header of a user-defined frame is always transmitted when a frame slot allocated to the frame is processed
6. **Reserved frame.** These shall not be used in a LIN 2.0 cluster. Their identifier is 63 (0x3F).

4 LIN hardware

The LIN specification was designed to allow very cheap hardware-nodes being used within a network. It is a low-cost, single-wire network based on ISO 9141.[6] In today's car networking topologies, microcontrollers with either UART capability or dedicated LIN hardware are used. The microcontroller generates all needed LIN data (protocol ...) (partly) by software and is connected to the LIN network via a LIN transceiver (simply speaking, a level shifter with some add-ons). Working as a LIN node is only part of the possible functionality. The LIN hardware may include this transceiver and works as a pure LIN node without added functionality.

As LIN Slave nodes should be as cheap as possible, they may generate their internal clocks by using RC oscillators instead of crystal oscillators (quartz or a ceramic). To ensure the baud rate-stability within one LIN frame, the SYNC field within the header is used.

5 LIN protocol

The LIN-Master uses one or more predefined scheduling tables to start the sending and receiving to the LIN bus. These scheduling tables contain at least the relative timing, where the message sending is initiated. One LIN Frame consists of the two parts **header** and **response**. The header is always sent by the LIN Master, while the response is sent by either one dedicated LIN-Slave or the LIN master itself.

Transmitted data within the LIN is transmitted serially as eight bit data bytes with one start bit, one stop-bit, and no parity. Bit rates vary within the range of 1 kbit/s to 20 kbit/s. Data on the bus is divided into recessive (logical HIGH) and dominant (logical LOW). The time normal is considered by the LIN Masters stable clock source, the smallest entity is one bit time (52 µs @ 19.2 kbit/s).

Two bus states — Sleep-mode and active — are used within the LIN protocol. While data is on the bus, all LIN-nodes are requested to be in active state. After a specified timeout, the nodes enter Sleep mode and will be released back to active state by a WAKEUP frame. This frame may be sent by any node requesting activity on the bus, either the LIN Master following its internal schedule, or one of the attached LIN Slaves being activated by its internal software application. After all nodes are awakened, the Master continues to schedule the next Identifier.

5.1 Header

The header consists of five parts:

BREAK: The BREAK field is used to activate all attached LIN slaves to listen to the following parts of the header. It consists of one start bit and several dominant bits. The length is at least 11-bit times; standard use as of today are 13-bit times, and therefore differs from the basic data format. This is used to ensure that listening LIN nodes with a main-clock differing from the set bus baud rate in specified ranges will detect the BREAK as the frame starting the communication and not as a standard data byte with all values zero (hexadecimal 0x00).

SYNC: The SYNC is a standard data format byte with a value of hexadecimal 0x55. LIN slaves running on RC oscillator will use the distance between a fixed amount of rising and falling edges to measure the current bit time on the bus (the master's time normal) and to recalculate the internal baud rate.

INTER BYTE SPACE: Inter Byte Space is used to adjust for bus jitter. It is an optional component within the LIN specification. If enabled, then all LIN nodes must be prepared to deal with it.

There is an Inter Byte Space between the BREAK and SYNC field, one between the SYNC and IDENTIFIER, and one between every Data byte in the payload.

IDENTIFIER: The IDENTIFIER defines one action to be fulfilled by one or several of the attached LIN slave nodes. The network designer has to ensure the fault-free functionality in the design phase (one slave is allowed to send data to the bus in one frame time).

If the identifier causes one *physical* LIN slave to send the response, the identifier may be called a Rx-identifier. If the *master's slave task* sends data to the bus, it may be called Tx-identifier.

RESPONSE SPACE: Response Space is the time between the IDENTIFIER field and the first Data byte which starts the LIN RESPONSE part of the LIN frame. When a particular LIN frame is transmitted completely, Header + Response, by the LIN MASTER, the LIN MASTER will use the full RESPONSE SPACE TIME to calculate when to send the response after sending the header. If the response part of the LIN frame is coming from a physically different SLAVE NODE, then each node (master & slave) will utilize 50% of the Response Space time in their timeout calculations.

5.2 Response

The response is sent by one of the attached LIN slave **tasks** and is divided into data and checksum.[4]

DATA: The responding slave may send zero to eight data bytes to the bus. The amount of data is fixed by the application designer and mirrors data relevant for the application which the LIN slave runs in.

CHECKSUM: There are two checksum-models available within LIN - The first is the checksum including the data bytes only (specification up to Version 1.3), the second one includes the identifier in addition (Version 2.0+). The used checksum model is pre-defined by the application designer.

6 LIN advantages

- Easy to use
- Components available
- Cheaper than CAN and other communications buses
- Harness reduction
- More reliable vehicles
- Extension easy to implement.
- No protocol license fee required

LIN is not a full replacement of the CAN bus. But the LIN bus is a good alternative wherever low costs are essential and speed/bandwidth is not important. Typically, it is used within sub-systems that are not critical to vehicle performance or safety - some examples are given below.

7 Applications

8 LIN API

The LIN application programming interface (API) provides a given set of function calls (base is the programming language C) which have to be implemented within each LIN software driver. Using this pre-defined set of driver routines, all LIN functions may be accessed.

The usage of API-compliant functions eases the implementation of standard software drivers and speeds up testing.

9 Development tools

When developing and/or troubleshooting the LIN bus, examination of hardware signals can be very important. Logic analyzers and bus analyzers are tools which collect, analyze, decode, store signals so people can view the high-speed waveforms at their leisure.

10 See also

- List of network buses

11 References

[1] "Lin Concept". *LIN Overview*. LIN Administration. Retrieved 28 October 2011.

[2] "Target Applications". *LIN Overview*. LIN Administration. Retrieved 28 October 2011.

[3] "Clemson Vehicular Electronics Laboratory: AUTOMOTIVE BUSES". 090114 cvel.clemson.edu

[4] LIN Specification Package Rev. 2.2a

[5] "LIN Bus Description, Automotive Bus, Local Interconnect Network". 090114 interfacebus.com

[6] LIN Technical Overview

12 External links

- LIN Consortium it is not longer available, because the latest LIN specification (2.2A) is being transcribed to the ISO (International Organization for Standardization) as part of the process to be accepted as ISO standard ISO 17987 Part 1-7.
- CAN/LIN Training
- Brief CAN/LIN Background Information (Chinese)
- Article about a free open hardware/software implementation of the LIN protocol
- An open source Arduino based LIN protocol analyzer
- Open source Arduino based platform with LIN connectivity
- A free online LIN checksum calculator

13 Text and image sources, contributors, and licenses

13.1 Text

- **Local Interconnect Network** *Source:* https://en.wikipedia.org/wiki/Local_Interconnect_Network?oldid=773986539 *Contributors:* Heron, Tedernst, Winelight, Blainster, Ukexpat, Thomas Willerich, Bender235, Laschewskir, Phansen, Btyner, Pingswept, Bhadani, Allen Moore, FlaBot, YurikBot, Crazytales, Arado, Gaius Cornelius, Ntropolis, Deville, Zzuuzz, X1cygnus, SmackBot, Aviageek, Frap, Idonra, Tionex, TastyPoutine, Cadaeib, Wleizero, Thijs!bot, Electron9, Magioladitis, Nyq, Cocytus, R'n'B, Cnilep, AlleborgoBot, SieBot, Dusti, Cavarooni, Fahidka, Lightmouse, Dodger67, Treekids, PipepBot, Rami823, Chickenfarmer73, Coccyx Bloccyx, SchreiberBike, Rafaelherrejon, DumZiBoT, Andre maier, Addbot, Meise, Mortense, JakeBinha, Szyx, Lightbot, Sergiej87, Yobot, AnomieBOT, Ciphers, LilHelpa, Nasa-verve, Jeevan44, Reply123, FrescoBot, Pdebonte, Betsytimmer, RCHenningsgard, Analog ic, Quondum, Sbmeirow, Rubinstu, Gandrewstone, BG19bot, ChrisGualtieri, Tony Mach, SFK2, Nvtj, Ginsuloft, JeanaS123, Light Engineer, Hansilicon and Anonymous: 96

13.2 Images

- **File:Folder_Hexagonal_Icon.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/48/Folder_Hexagonal_Icon.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?

13.3 Content license

- Creative Commons Attribution-Share Alike 3.0