Microcomputer Protocol Implementation at Local Interconnect Network

Georgi Krastev

Abstract: *The paper discusses the issues of microcomputer protocol implementation at local interconnect network LIN for automobile applications. This protocol is from class A according to the classification of SAE and it includes one master and several slave nodes. Single Chip Microcontrolers T89C51CC01 of the firm Atmel are used.*

Key words: *Computer Systems and Technologies, Single Chip Microcomputers, Local Interconnect Network.*

INTRODUCTION

The technical specifications of the protocol LIN (Local Interconnect Network) have been developed by a consortium of European automobile producers and other famous companies, including Audi AG, BMW AG, Daimler Chrysler AG, Motorola Inc., Volcano Communications Technologies AB, Volkswagen AG, Volvo Car Corporation [1, 2] and others. It is designated for creating cheap local networks for data exchange at short distances.

The main tasks, given to the LIN protocol from the consortium of European automobile producers – uniting of the automobile subsystems and nodes (as the door locks, window wipers, the serve-systems for window motion, the air-conditioner operation, the serve-system of the electrical hatchet, etc.) into a single electronic system. The LIN protocol has been affirmed by the European automobile consortium as a cheap additional part to the reliable Controller Area Network (CAN) protocol. These two protocols complement each other and allow the unification of all electronic automobile devices into a single multifunctional board network. The LIN standard includes the technical specifications both of the protocol and of the transfer medium. As a consecutive link protocol, LIN efficiently supports the operation of electronic nodes and devices in the automobile systems with networks from class A (two-way, half-duplex), with one master and several slave nodes.

BUS FEATURES

LIN Protocol supports bi-directional communication on a single wire, while using inexpensive microcontrollers driven by RC oscillators, to avoid the cost of crystals or ceramic resonators. Instead of paying the price for accurate hardware, it pays the price in time and software. The protocol includes an autobaud step on every message. Transfer rates of up to 20 Kbaud are supported, along with a low power SLEEP mode, where the bus is shut down to prevent draining the battery, but the bus can be powered up by any node on the bus. The bus itself is a cross between I^2C and RS232. The bus is pulled high via a resistor and each node pulls it low, via an open collector driver like I^2C. However, instead of having a clock line, each byte is marked via start and stop bits and the individual bits are asynchronously timed like RS232.

Figure 1 shows a typical LIN Protocol configuration. The bus uses a single wire pulled high through a resistor with open collector drivers. A Dominant state is signaled by a ground level on the bus and occurs when any node pulls the bus low. A Recessive state is when the bus is at VBAT (9 - 18V) and requires that all nodes let the bus float. In the idle state, the bus floats high, pulled up through the resistor. The bus operates between 9V and 18V, but parts must survive 40V on the bus. Typically, the microcontroller is isolated from the bus levels by a line driver/receiver. This allows the microcontrollers to operate at 5V levels, while the bus operates at higher levels. The bus is terminated to VBAT at each node. The Master is terminated through a 1KW resistor, while the Slaves are terminated through a 20-47KW resistor. Maximum bus length is designed to be 40 meters. At press time (early 2000), K-Line drivers are used until true LIN drivers are available.

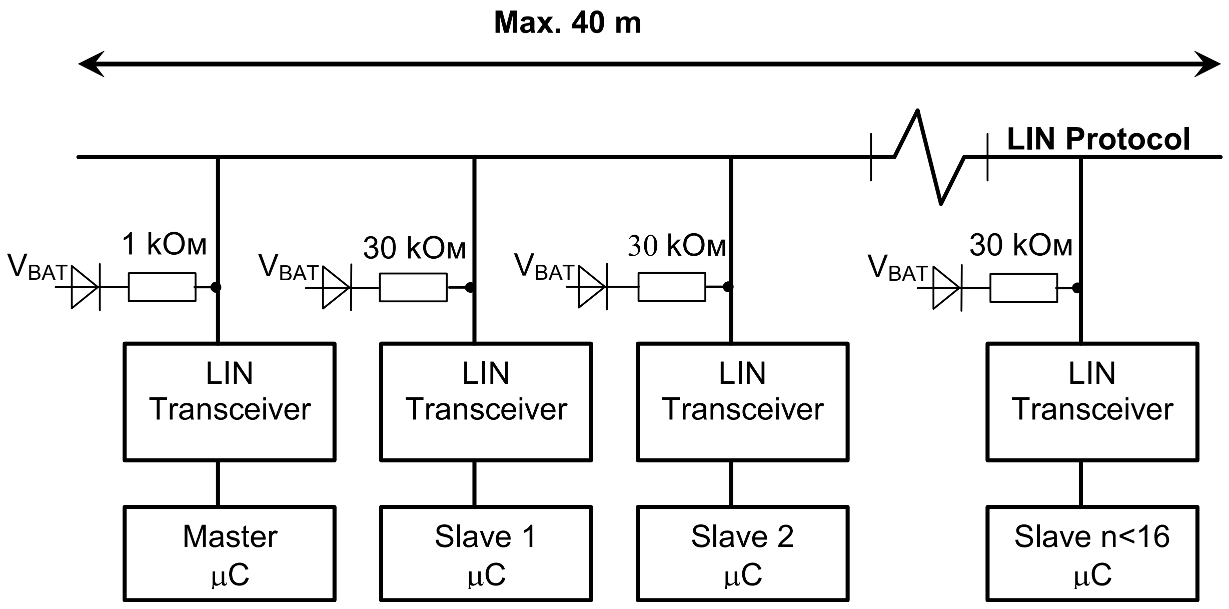

Fig.1. Bus configuration

Message Frames

All information transmitted on the LIN bus is formatted as Message Frames. As shown in Figure 2, a Message Frame consists of the following fields:

- Synch Break
- Synch Field
- Identifier Field
- Data Field
- Checksum Field

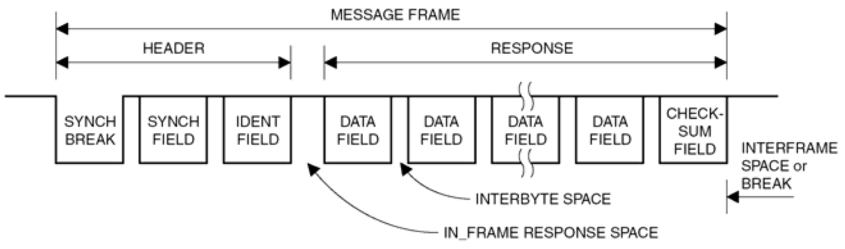

Fig. 2. LIN Message Frame

Byte Fields

The Byte Field format, shown in Figure 3, is identical to the commonly used UART serial data format with 8N1 coding. This means that each Byte Field contains eight data bits, no parity bits, and one stop-bit. Every Byte Field has the length of 10 bit-times (Tbit). As shown in the figure, the start bit marks the beginning of the Byte Field and is "dominant" while the stop-bit is "recessive". The eight data bits can be either "dominant" or "recessive".

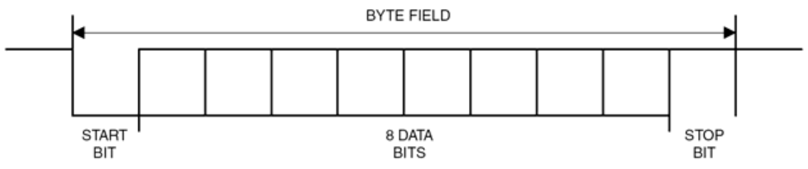

Fig. 3. LIN Byte Field

Synch Break

The Synchronization Break marks the beginning of a Message Frame. This field is always sent by the Master task, and provide a means for the Slave task to prepare for the synchronization field. The Synch Break consists of two different parts: a dominant (low) level that should be minimum 13 bit-times (Tbit) which is followed by a recessive (high) period that should be in the range one to four Tbit. The second field is necessary to detect the dominant (low) level of the start bit of the following Synch Field.

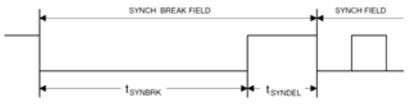

Fig. 4. Synch Break Field

The length of the first field is chosen to distinguish between the Synch Break and the maximum possible allowed sequence of dominant bits within a data frame. For example, a data field with all "0"s should not be mistaken for a Synch Break field.

Synch Field

The Synch Field contains the signalling required for the slave to synchronize with the master clock. The Synch Field is a Byte Field containing the data "0x55" giving the waveform shown in Figure 5.

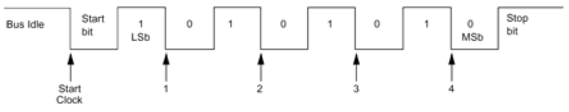

Fig. 5. Synch Field

As illustrated, the Synch Field is characterized by five falling edges (recessive to dominant edges) These edges are used during synchronization to tune the slave node transmission and reception speed to match the master node as explained later in this application note.

Identifier Field

The Identifier Field contains information about contents and length of a message. As shown in Figure 6, this field is divided into three sections: identifier bits (four bits), length control bits (two bits) and parity bits (two bits). This divides the set of 64 identifiers into four subsets of 16 identifiers each.

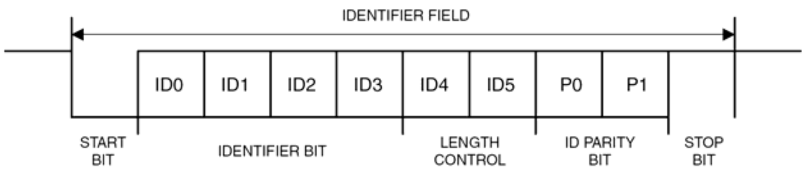

Fig. 6. Identifier Field

The LIN protocol defines this as follows:

Table 1. Number of Data Fields in a Data Frame

ID 5	ID 4	N_{data} (Number of Data Fields)
0	0	2
0	1	2
1	0	4
1	1	8

As shown in Table 1 there are two groups with two data fields, one group with four data fields, and one group with eight data fields. Note that the Identifier Field does not indicate the destination of the message but describes the contents of the message frame. It is up to all the Receiving Slave tasks to decide if they should act upon the received message or not. The last two bits of the Identifier Field contains parity information. LIN uses a mixed-parity algorithm that ensures that the Identifier Field never will consist of an all "recessive" or "dominant" pattern. Note that the parity check only detects errors, it does not correct them.

The parity check bits are calculated by the following mixed-parity algorithm:

$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$

$\overline{P1} = ID1 \oplus ID3 \oplus ID4 \oplus ID5$

Data Field

The data frame contains from two to eight Data Fields containing eight bits of data each. Transmission is done with LSB first. The data fields are written by the responding Slave task. Since there is no bus arbitration, only one slave task should be allowed to respond to each identifier. All other Slave tasks are limited to reading the response and act accordingly.

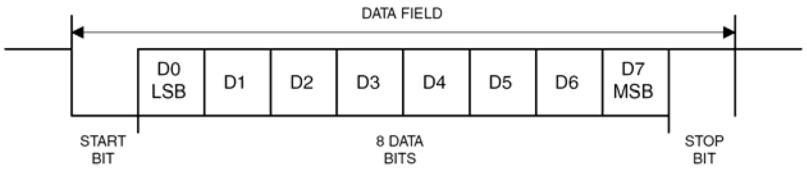

Fig. 7. Data Field

Checksum Field

The last field in the Message Frame is the Checksum Field. This byte contains the inverted modulo-256 sum of all data bytes (data frame not including the identifier). This sum is calculated by doing an "ADD with carry" on all data bytes and then inverting the answer. The properties of the inverted modulo-256 sum are such that if this number is added to the sum of all data bytes the result will be "0xFF".

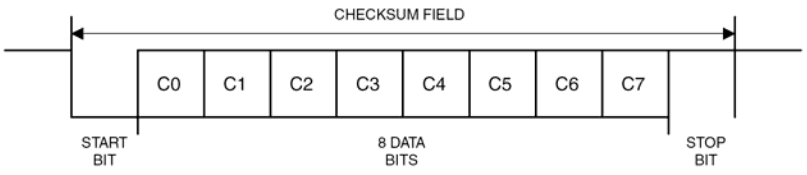

Fig. 8. Checksum Field

Hardware Implementation

The T89C51CC01 is an 8051 compatible microcontroller and is ideal for implementation of a LIN application. In a typical CAN and LIN application, the T89C51CC01/02 is an ideal candidate to be a CAN/LIN gateway unit and therefore the LIN Master.

Figure 9 shows an example implementation of two (Master and Slave) nodes of a network from type LIN. The used Single Chip Microcontrollers are T89C51CC01, made by the firm Atmel and LIN Transceivers are made by the firm Melexis.

The TH8080 is a physical layer device for a single wire data link capable of operating in applications where high data rate is not required and a lower data rate can achieve cost reductions in both the physical media components and in the microprocessor which use the network. The TH8080 is designed in accordance to the physical layer definition of the LIN Protocol Specification, Rev. 1.1 . The IC furthermore can be used in ISO9141 systems. Because of the very low current consumption of the TH8080 in the recessive state it's particularly suitable for ECU applications with hard standby current requirements, whereby no sleep/wake up control due to the microprocessor is necessary.

The TH8061 consist a low drop voltage regulator 5V/50mA and a LIN Bus transceiver, which is a bidirectional bus interface for data transfer between LIN bus and the LIN protocol controller. Additional integrated is a RESET output with a reset delay of 100ms and a fixed threshold of 4.65V.

LIN Slave Driver: Bit Manipulation

The LIN slave implementation uses the PCA function to synchronize with the LIN master. The bit manipulation method uses parallel port pins to be the Rx and Tx lines connecting

to the LIN transceiver. To implement the Rx line, port pins that can detect positive and negative transitions of the LIN signal are required for a bit manipulation implementation (see Figure 9). Any of the digital port pins can be used to be the Tx line to the LIN transceiver (e.g. P0.0 is used in the C-source code examples).
Pins P1.6 and P1.7 are used for detection of positive and negative transition. Since each pin can only be configured for either positive or negative transitions, two pins must be electrically connected together as shown in Figure 9.

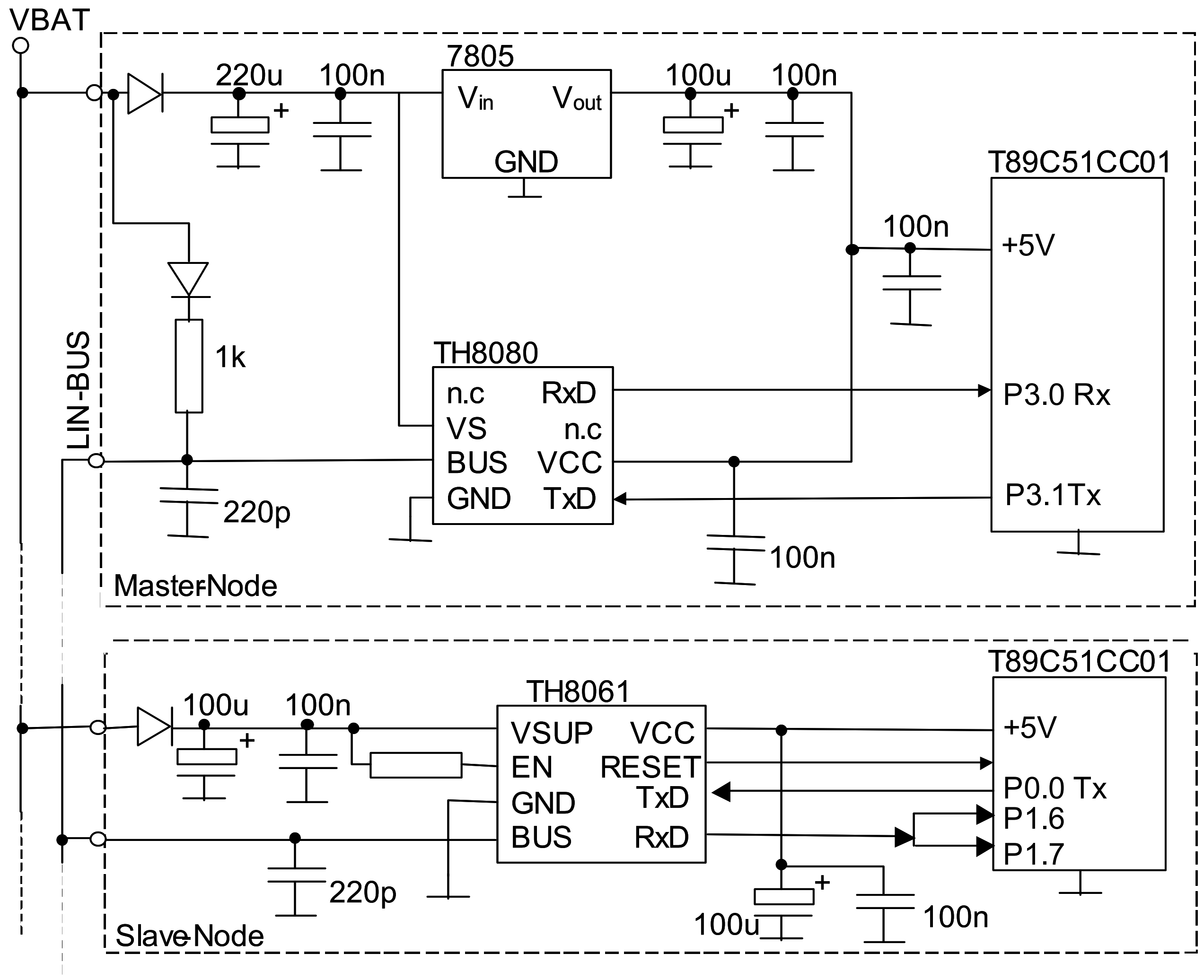

Fig. 9.

Bit Timing Configuration The bit timing is derived from the received Synch Field using the on chip PCA. Timer 2 is used to sample the bits received and to generate the bits for transmission on the LIN bus.

C-Source Files

The C-source code drivers contain the following source files:
• LINSLAVE.C – LIN Slave protocol driver routines
• LIN.H – Header file for LINSLAVE.C
• MAIN.C – Function main()
• TIMER0.C – Scheduler routines
• TIMER.H – Header file for TIMER.C
• 89C51CC0.H – T89C51CC01/02 register definitions

Code Size & Processor Loading

Using the Keil™ 8051 compiler, the code size is 1.5K bytes including main() and scheduler function in TIMER0.C. Processor loading will depend upon baud rate implemented and frequency of tasks for reading/writing to sensors and actuators. However under tests for baud rates 2400, 4800, 9600 and 19200 using a 20 MHz crystal (the maximum value for theT89C51CC01), processor loading was found to be in the region of 38 to 40%.

LIN Master Driver: USART Peripheral
The USART implementation of the LIN protocol is the most efficient way of implementing a Master Control Unit since the USART contains much of the hardware support for generation of LIN bytes, i.e. start bit, 8-data bits, stop bit format. To implement a LIN Master Control Unit with the T89C51CC01, at least two port pins are required as shown in Figure 9.

Bit Timing Configuration
Bit timing for the USART is achieved using Timer 1 via register TH1. Timer 1 is used in mode 1 which is auto-reload of TH1, therefore generating the baud rate.

Master Task Implementation
The first problem to solve for LIN Master Driver implementation using a USART is that of generation of the Synch Break. As previously mentioned, this must be at least 13 bit times with the bus at dominant. The maximum number of bits that can be driven to dominant with the T89C51CC01 USART is 10 bits; one start bit followed by nine data bits at dominant by writing 0x000 to the USART (note that the stop bit is recessive). Therefore, to generate the Synch Break of at least 13-bit times, a different strategy is used. The Synch Break is generated by changing the value in TH1, so that the baud rate is effectively decreased. Therefore writing 0x00 to the USART, will result in at least 13 bits times sent to dominant.
Once the Synch Break is generated, the baud rate is set to the full value again for transmission of the Synch and Identifier Fields. Bit Error detection is carried out upon each bit transmitted.

C-Source Files
The C-source code drivers contain the following source files:
• LIN.C – LIN protocol driver routines
• LIN.H – Header file for LIN.C
• MAIN.C – Function main()
• TIMER0.C – Scheduler routines
• TIMER.H – Header file for TIMER.C
• 89C51CC0.H – T89C51CC01/02 register definitions

CONCLUSION
The paper describes the LIN protocol for developing low speed networks for automobile applications. Single Chip Microcontrollers T89C51CC01, made by the firm Atmel and LIN Transceivers, made by the firm Melexis, are used for this development. The developed programming belongs to C.

REFERENCES

[1] Автомобильны стандарт LIN и микроконтроллеры для его реализации. Гамма Санкт Петербург. www.gamma.spb.ru.

[2] Dan Butler, Thomas Schmidt, Thorsten Waclawczyk. LIN Protocol Implementation Using PICmicro MCUs. Microchip Technology Inc.

ABOUT THE AUTHOR
Georgi Krastev, Department of Computer Systems, University of Rousse, Phone: +359 82 888 672, E-mail: gkrastev@ecs.ru.acad.bg.