THIS ARTICLE WILL PROVIDE AN OVERVIEW ON SOME OF THE MOST IMPORTANT BUS SYSTEMS CURRENTLY USED IN CARS: CAN, LIN, FLEXRAY™ AND MOST, AND FOCUSES ON EACH BUS SYSTEM'S APPLICATION AREA AND THE PROVIDED PROTOCOL.

Automotive Bus Systems

By: Markus Schmid, Atmel

The number of electronic components in vehicles has increased rapidly and continuously during recent years. On the one hand many new sensors and actuators and therefore new electronic control units have been developed to make passengers feel safer. On the other hand, entertainment and navigation systems have made their way into cars to make travel more comfortable.

To meet the design challenges due to the different requirements (capacitance, real-time operation and cost), several new bus systems have been developed or improved. This article will provide an overview on some of the most important bus systems currently used in cars: CAN, LIN, FlexRay™ and MOST. The article focuses on each bus system's application area and the provided protocol.

In a car, there are several operating fields with different requirements regarding the corresponding bus system. Each of the bus systems mentioned above is used to serve a certain communication requirement between the automotive electronic components. The designer selects the appropriate bus system depending on the required safety level, the data transmission rate and the costs. For example, the CAN bus - even though it is currently the most important automotive bus system - is not well-suited for very fast data transmission as needed for multimedia applications. Also, the CAN bus system is too sophisticated, and therefore too expensive for applications with low data rates, where only few parts of the system are involved in the transmission, for example sun roofs or heating systems. For these applications, new bus systems have been designed.

FlexRay, CAN, and LIN are mainly used for control systems, whereas MOST is used for telemetric applications.

	LIN	CAN	FlexRay	MOST
Application	Low-level communication systems	Soft real-time systems	Hard real-time systems (X-by-wire)	Multimedia, telemetrics
Control	Single-master	Multi-master	Multi-master	Timing-master
Bus Access	Polling	CSMA/CA	TDMA/FTDMA	TDM/CSMA
Bandwidth	19.6 kBit/s	500 kBit/s	10 Mbit/s	24.8 mbit/s
Data Bytes per Frame	0 to 8	0 to 8	0 to 254	0 to 60
Redundant Channel	Not supported	Not supported	Two channels	Not supported
Physical Layer	Electrical (single wire)	Electrical (twisted pair)	Optical, electrical	Mainly optical

Table 1: Bus System Overview

MOST - Media Oriented Systems Transport

The MOST bus was developed in 1998 under the leadership of BMW and DaimlerChrysler for all kinds of automotive multimedia applications such as audio, video, navigation and telecommunication systems. In August 2004 the new specification 2.3 was released.

The MOST bus features a very high data rate of up to 24.8 Mbit/s in synchronous and 14.4 Mbit/s in asynchronous transmission mode. It has an additional asynchronous control channel with a data rate of up to 700 kBit/s. These high data rates make the MOST bus the best fit for real-time audio and video transmission applications. To ensure a safe data transmission, an optical medium (Plastic Optic Fiber, POF), which is not susceptible to EMC, is used as physical layer.

Furthermore, MOST bus systems support plug&play of up to 64 nodes, which can be arranged in ring, star or chain topology. This enable to connect all parts of a MOST bus system, the so called MOST devices, in a very flexible way.

MOST Communication

In a MOST network, one device needs to be determined to be the master of the network. This device will be the so-called Timing Master, and all other connected devices are slaves.

The organization of a data transfer according to the MOST specification is shown in Figure 2.

For control data transport tasks and network management, the organization of data transfer in blocks of frames is required. 16 frames are combined in one block, each frame consists of 512 bits.

Table 2 (see next page) provides an overview on the content of these 512 bits.

Synchronous Data

The synchronous area is mainly used for real-time data transmission like audio/video or sensor values. Data access is realized by using Time Division Multiplexing (TDM). Physical channels can be allocated for a certain time while playing an audio source for example. It is possible to vary the band-width by allocating any number of bytes to one logical channel. To route the synchronous data to the appropriate sink, a routing engine is used.

The number of synchronous data bytes in one frame is limited to 60 bytes.

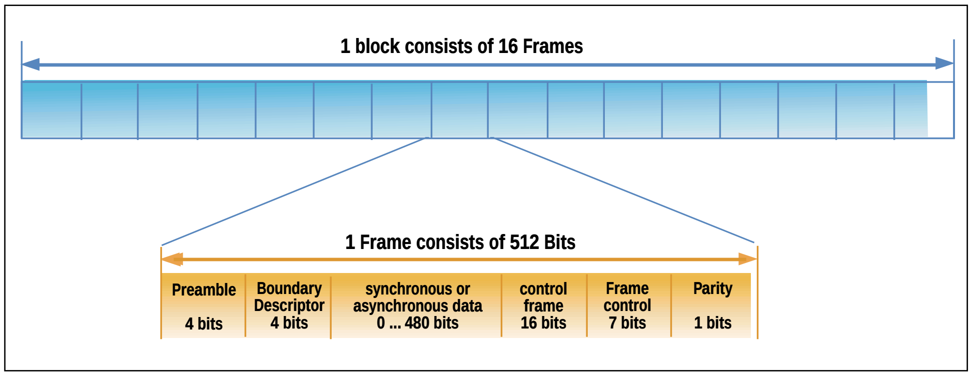
1 block consists of 16 Frames
1 Frame consists of 512 Bits
Preamble 4 bits
Boundary Descriptor 4 bits
synchronous or asynchronous data 0 ... 480 bits
control frame 16 bits
Frame control 7 bits
Parity 1 bits

Figure 2: MOST Communication

Asynchronous Data

If asynchronous data needs to be sent in addition to the synchronous, the boundary descriptor has to be set as described in table 2 to ensure that the beginning of the asynchronous data can be determined exactly. Asynchronous data transmission is mainly used for larger-sized blocks and if larger band-width is required. The number of asynchronous data bytes on an asynchronous channel is limited to 48 bytes when using the 48 byte data link layer. when using an alternative data link layer, the maximum packet length is 1014 bytes.

The structure of an asynchronous area in a frame is given in Table 3.

NAME	BITS	DESCRIPTION
Preamble	4	Synchronizes the MOST core and its internal functions to the bit stream
Boundary descriptor	4	If synchronous as well as asynchronous data is transmitted in one frame, the boundary descriptor marks the number of 4 byte blocks of data used for synchronous data in the data block. For example, if 40 bytes of synchronous data and 20 bytes of asynchronous data are transmitted, the boundary descriptor will be set to 10 by the timing master
Synchronous or Asynchronous data	0...480	See chapter “Synchronous Data” and chapter “Asynchronous Data”
Control frame	16	See chapter “Control Data”
Frame control	7	Frame control and status bits
Parity	1	Error detection

Table 2: MOST Frame Architecture

NAME	BITS	DESCRIPTION
Arbitration	8	Avoids collisions on the bus
Target address	16	When transmitting asynchronous data, the target address has to be transmitted, too.
Length	24	Length in four byte blocks
Source address	16	When transmitting asynchronous data, the source address has to be transmitted, too.
Data area	0...384	The actual data
CRC	32	Cyclic redundancy check

Table 3: Message Format in the Asynchronous Area

Control Data

The control data is mainly used for the communication between the separate nodes of the bus. Data access is realized by using Carrier Sense Multiple Access (CSMA), which offers fixed and predictable response times. Although a complete control data message is 32 bytes long, only two bytes can be transmitted in one frame. This means that one block (16 frames) is needed to transmit one control data message.

The structure of a control data message is shown in Table 4.

FlexRay

FlexRay was developed under the leadership of BMW and DaimlerChrysler in 1999 especially for the new X-by-wire systems, such as steer-by-wire systems or brake-by-wire, which require a very good error management along with high transmission data rates. Atmel is also a member of the FlexRay consortium, which released the latest specification 2.0 in June 2004.

FlexRay is based on the communication system “byteflight”, which was developed by BMW earlier. To meet the requirements of the new bus systems, the byteflight method has been improved in terms of chronological deterministic and fault tolerance.

FlexRay supports data transmission with a bandwidth of up to 10 Mbit/s and is thus well-suited for real-time operation. There is no need for a special physical layer, therefore, electrical and optical transmission mediums are supported by FlexRay. Furthermore, FlexRay is suited for several network topologies such as bus, star, cascaded star and hybrid network topologies.

FlexRay Communication

CAN supports CSMA (Carrier Sense Multiple Access), which means that every device starts a transmission as soon as no other device is sending. Since each device has different priorities, collisions on the bus will not occur. On the other hand, this prevents exact prediction of which time the sent data will be received (non-deterministic).

In contrast to that, FlexRay supports TDMA (Time Division Multiple Access). Each device has a fixed time window (time slot), during which the device has exclusive access to the bus. These time slots are repeated in a fixed pattern.

Using TDMA, it is possible to exactly predict the time when the data will be received by the bus (deterministic bus access). To properly handle that kind of communication, however, all nodes need to have the same global time.

Figure 3 shows an example of a typical data transmission using FlexRay with four components. Two out of the four (device A and device C) have a redundant second channel.

The second channel can be used for redundant transmission (C1 in figure 3) or for the transmission of two messages at the same time (A1 and A2 in Figure 3). The devices B and D are only connected to channel 1, so that the corresponding time slot on channel 2 elapses without being used.

If a device has exclusive access to the bus, but has no data to be sent, the designated time elapses without being used. In this case, the bandwidth is not used efficiently. If a device, however, has to send more data than fits into one time slot, the device needs to wait until it has exclusive access to the bus again to send the rest of its data. To avoid this, FlexRay splits the communication cycle into a static and dynamic part. The fixed time slots are designated in the static part, whereas the dynamic part has additional time slots, the so-called mini-slots, during which the exclusive bus access is limited for a short time. Only if a bus access occurs within this time, the mini-slots will be enlarged as necessary. This method helps to increase the efficiency of the bandwidth.

The message structure is shown in Figure 4 and a short description is given in Table 5.

LIN - Local Interconnect Network

In contrast to FlexRay, which serves more sophisticated application needs than CAN, LIN has been developed for less complex networks, where CAN would be

NAME	BITS	DESCRIPTION
Arbitration	24	Avoids collisions on the bus
Target address	16	When interchanging control data between separate nodes, it is important to specify the target node.
Source address	16	When interchanging control data between separate nodes, it is necessary to identify the source node.
Message type	8	There are two different message types. Normal messages include single cast, group cast and broadcast, system messages include resource allocate, resource de-allocate and remote getsource.
Data area	0...136	The actual data
CRC	16	Cyclic redundancy check
Transmission status	16	Indicates the current status of a transmission
Reserved	16	Reserved for further protocol use

Table 4: Message Format in the Control Frame

too expensive. The LIN specification was defined by a consortium with the initial members BMW, DaimlerChrysler, Audi®, Volvo, Motorola, VW® and Volcano. Atmel joined this consortium in 2001. After having gathered additional experience, the LIN consortium released the new LIN 2.0 specification in September 2003.

Typical LIN bus applications include the connection of intelligent actuators or sensors, such as small motors, temperature or rain sensors, sun roof or heating control. For these applications, high transmission data rates or complex fault management are not necessary.

This is why LIN supports only data transmission of up to 19.6 kBit/s. For this data rate, a cost-effective 12 V single wire is sufficient as transmission medium.

LIN is based on an SCI (UART) 8-bit interface and supports the Single-Master/Multiple Slave concept. UART interfaces are available in almost every micro-

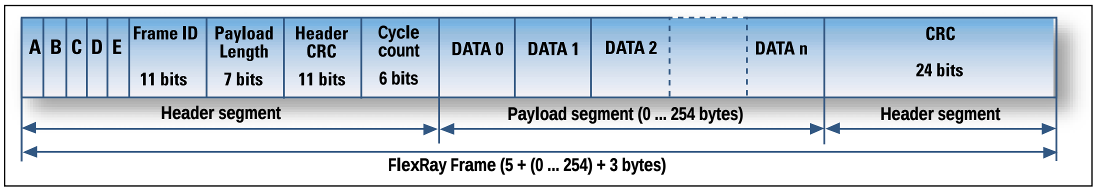

Figure 4: Message Architecture in FlexRay

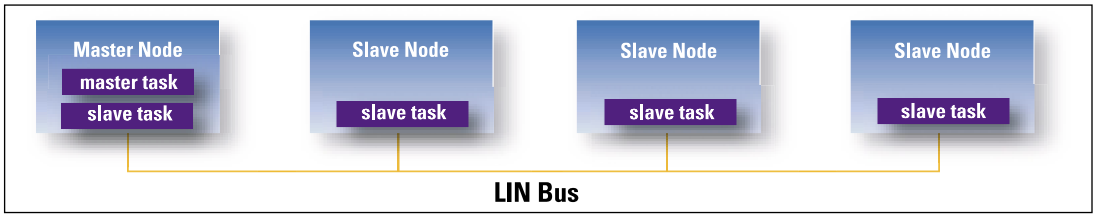

Figure 5: Typical LIN Bus Architecture

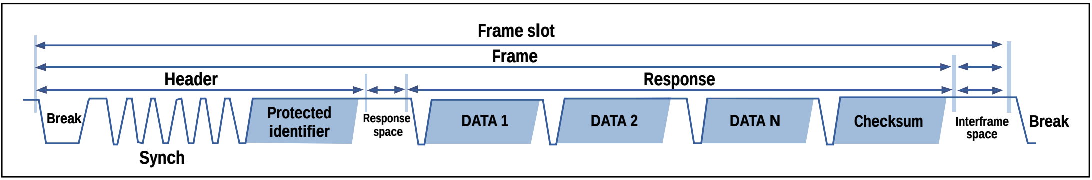

Figure 6: LIN Bus Message Frame

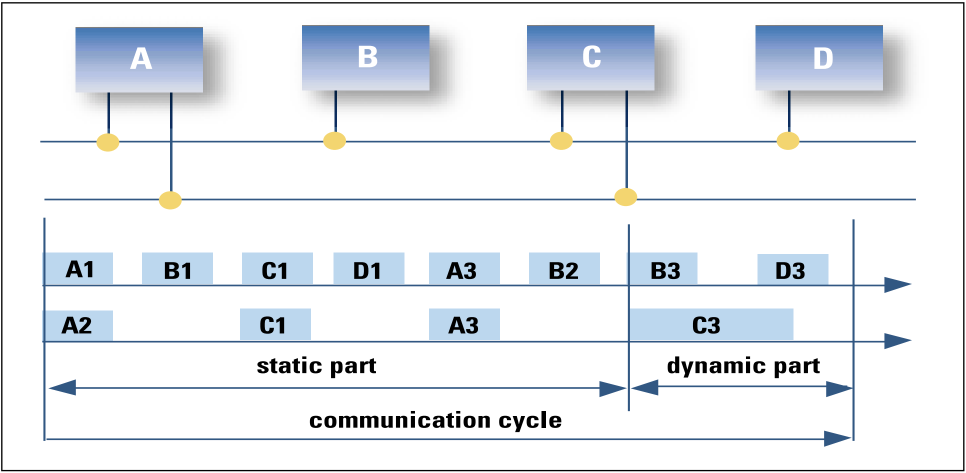

Figure 3: Example of a Typical Data Transmission with Four Bus Devices Using FlexRay

controller or ASIC, and can be implemented in almost any software or firmware. due to this, there is no need for the use of other expensive external components. A typical LIN cluster with one master node and three slave nodes is illustrated in Figure 5.

It is obvious that the master node performs both a master task as well as a slave task.

Due to the simple concept, no node within the LIN network, except the master, will be influenced by adding or removing another slave. In this case, the only necessary changes concern the master node.

A special feature of LIN is the synchronization mechanism which adjusts the clock rate of the slave nodes to the master without an external crystal or resonator.

Thanks to the simplicity of the UART communication, the single-wire transmission and the simplicity of the clock rate adjustment, a LIN bus system is very cost-effective.

LIN Communication

The structure of a LIN Bus Message Frame is illustrated in Figure 6.

Every LIN Bus Message Frame starts with the header sent by the master. This header consists of a Break byte field, the Synch byte field and the protected identifier. Before the slaves send the requested response, there is a short stop, the so-called response space. the interframe space at the end of each frame pulls the LIN bus to high level until the break byte of the next frame will force the bus line to low level again.

NAME	BITS	DESCRIPTION
A = Reserved bit	1	This bit is reserved for future protocol use and may not be used by the application.
B = Payload preamble indicator	1	This bit is used to indicate whether or not an optional vector is contained within the payload segment. This vector is a network management vector if the frame is transmitted in the static segment or a message ID if the frame transmitted in the dynamic segment.
C = Null frame indicator	1	If this bit is set, the message included contains no useable date in the payload segment.
D = Sync frame indicator	1	If this bit is set, the frame is used to synchronize all receiving nodes.
E = Startup frame indicator	1	This bit is used to indicate whether or not a frame is a start-up frame.
Frame ID	11	This ID defines a certain slot in which the frame should be transmitted.
Payload length	7	These bits determine the number of data bytes transmitted in the payload frame by setting the payload length bits to the number of the data bytes divided by two.
Header CRC	11 6	The Sync frame indicator, the start-up frame indicator, the frame ID and the payload length contribute to the Header CRC.
Cycle count		These bits contain the number of the current communication cycles.
Payload segment	2032	In that segment the data will be transmitted. It can contain up to 254 bytes (0...127 two-byte words).
CRC	24	This cyclic redundancy check uses the complete header segment as well as the complete payload segment.

Table 5: Description of Message Architecture in FlexRay

this response may contain up to eight data byte fields plus one checksum byte field.

With the release of the LIN2.0 specification, a new checksum calculation has been introduced. To enable compatibility to the still used LIN1.3 specification, the previous type of checksum calculation is also supported by LIN2.0. The new checksum is called enhanced checksum and is calculated by the inverted eight bit sum along with the carry bit over all data bytes plus the protected identifier. The LIN1.3 checksum is called classic checksum and is calculated as described above, but without the protected identifier.

Each slave is in alert state as soon as it detects the Break byte field. The following Synch byte field synchronizes the slaves with the master for the following transmission. The Synch sequence is always a byte field with the data value Ox55, so all slave noes within the network can easily synchronize to the clock of the master by detecting the edges of this signal.

Apart from the Break byte field, which is indicated by a low level on the LIN bus for at least 13-bit times followed by a high level for at least one bit time, all other byte fields n a LIN message are constructed as shown in Figure 7.

At the beginning, there is a low level start bit which is known by almost any UART-based communication, followed by eight data bits with the LSB first. The byte field is completed with a stop bit.

The protected identifier consists of six identifier bits and two parity bits, so there are 64 different identifiers within one LIN network.

The structure of the protected identifier byte field is shown in Figure 8. According to the protected identifier and after the response space, a slave begins to send the requested response, or it expects more data.

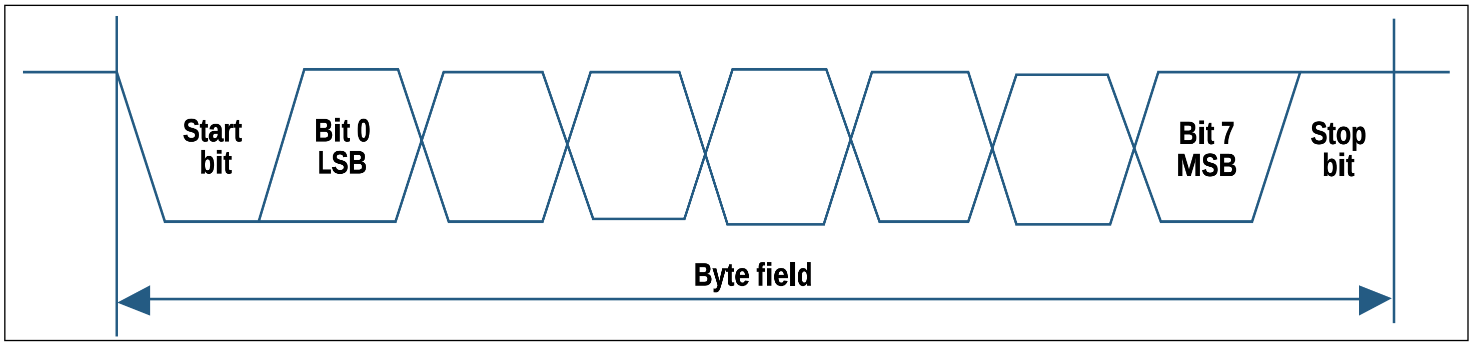

Figure 7: Structure of a Byte Field in a LIN Message Frame

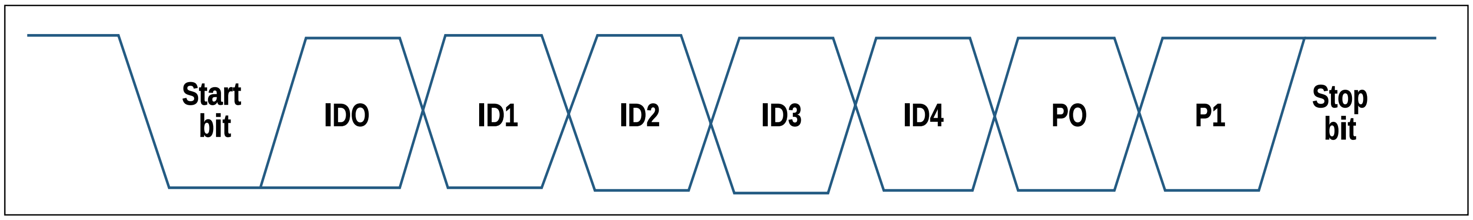

Figure 8: Structure of the Protected Identifier