A Comparison of Fieldbus Protocols: LIN 1.3, LIN 2.0, and TTP/A

Wilfried Elmenreich and Stefan V. Krywult
Vienna University of Technology
Institute of Computer Engineering
Treitlstraße 1-3/182-1, Vienna, Austria
{wil,stefan.krywult}@vmars.tuwien.ac.at

Research Report 3/2004

Abstract – *This paper compares the fieldbus protocols LIN 1.3, LIN 2.0, and TTP/A to the requirements of low-cost body electronic networks in the automotive domain. The examined protocols all aim at distributed smart sensor/actuator networks and provide support for low-cost implementation on standard microcontroller nodes.*

LIN comes with a rigid specification of the physical layer in order to support interoperability between LIN nodes of different vendors. The LIN specification defines also a complete application framework supporting developing and configuring LIN networks.

The TTP/A specification describes a pure communication system and leaves a lot of decisions regarding physical layer, fault detection, redundancy, etc. to the system designer. The time-triggered scheduling of messages leads to a very high data efficiency, even for short messages.

1 Introduction

Electronics has become the driving force of innovation in today's vehicle industry. Communication networks in automobiles are introduced with several purposes and requirements. As a consequence, future automobiles will concurrently host several networks, each optimized for a specific purpose. Examples for such specific busses will be control busses like Controller Area Network (CAN) [14], safety-critical busses like TTP/C [16], Flexray [5], or TTCAN [6], infotainment busses like MOST [10] and low-cost busses for body electronics.

Low-cost busses are introduced in automotive applications in order to interconnect smart sensors and actuators as an alternative to a full-featured electronic control unit (ECU). Smart sensors and actuators integrate a transducer element with a microcontroller and a network interface. Such a smart transducer encapsulates and hides the technical details from the physical transducers and provides a concise abstract interface of its features. With the advent of modern microcontrollers it became possible to built low-cost smart transducers by using commercial-off-the-shelf microcontrollers that provide a standard communication interface, such as a UART (Universal Asynchronous Receiver/Transmitter). As a further cost-decreasing factor, single-wire busses like ISO 9141 are used to interconnect the smart transducers. As a consequence, bandwidth and dependability of such a communication system is relatively low, thus, low-cost busses are mainly used for car body electronic functions.

Two communication protocols that address this field of application are LIN (Local Interconnect Network) and TTP/A (Time-Triggered Protocol for SAE class A applications). The LIN protocol was developed by a consortium of seven automotive partners (Audi, BMW, DaimlerChrysler, Volvo, Volkswagen, Motorola and VCT) as a complementary system to the widely used CAN bus [**?**]. In 2003, many updates to reflect the latest off-the-shelf microcontrollers as well as inputs from the SAE Task Force resulted in the definition of LIN 2.0. In fact, LIN 2.0 was a complete rework of the existing LIN 1.3 standard, but backward compatible.

The TTP/A protocol is the low-cost field-bus protocol that is harmonized with the fault-tolerant system bus TTP/C of the time-triggered architecture (TTA). It is intended for the connection of smart sensors and actuators in embedded real-time systems in

different application domains, e.g., industrial, automotive, etc. It is the objective of TTP/A to provide all services needed by a smart sensor, including timely communication, remote on-line diagnostics, and plug-and-play capability. While LIN is powered by industrial sponsors, TTP/A is an academic development started by the Technical University of Vienna, Austria and then broadened to include the Technical University of Munich, Germany and the University of Stuttgart, Germany. The first version of TTP/A was published at the SAE World Congress in 1995 [8]. Since then, TTP/A has been extended by a plug-and-participate function [4] and a unique interface scheme in form of an IFS [9].

It is the objective of this paper to compare the features of LIN 1.0, LIN 2.0, and TTP/A. First we will discuss the basic requirements for body electronics busses, then we will describe the common features and individual characteristics of the protocols. The paper is concluded by a discussion in Section 8.

2 Requirements on Automotive Body Electronic Busses

Cost factor: Obviously, such a system should be as low-cost as possible. The ways to achieve this goal are to reduce wiring costs (e. g., by using a single wire bus), a protocol based on UART communication (thus avoiding an extra communication chip), the support of COTS hardware, and a small footprint of the protocol implementation enabling the use of cheap microcontrollers. Moreover, complexity management by an adequate Plug and Play scheme may significantly reduce set-up costs.

Bandwidth: The provided bandwidth of the communication system should be as high as possible in order to serve a large set of possible fields of application, but increasing bandwidth above a particular level is opposed to the goal of decreasing costs.

As an example we will assume that a network of 32 transducers requires to be instrumented by a 2 byte message for each transducer at least every 100 ms. This will result in a minimum net bandwidth of 1920 Bytes/sec. Depending on the protocol efficiency, the required total bandwidth can be much higher.

Real-Time Capabilities: When the acquired sensor information is to be used in a real-time application, the determinism of measurements and its communication is of interest. Therefore we regard the protocols ability to globally synchronize measurements and to provide periodic measurements with minimal jitter, i. e., the difference between shortest and longest response time.

Error Diagnosis: Body electronic networks are not considered safety-critical, however, in order to keep maintenance costs low, error detection and diagnosis is required. To meet this diagnostic requirement, it must be possible to address some internal storage space in each smart transducer in order to collect extended data on its operation.

Extensibility: The number of connected nodes of a body electronics network are not considered to be constant during the lifetime of the system. For example, new devices might be attached to the car in order to enable some features. Thus, the communication system should support the connection and integration of new nodes with respect to new node detection and configuration.

3 Common features of LIN and TTP/A

Since both protocols have been developed with the same target application in mind, the systems are alike in several aspects:

Both protocols use UART frames encoding as basic communication units. Such a UART frame consists of a start bit (always low), 8 message bits in non-return-to-zero (NRZ) encoding, an optional parity bit and a stop bit (always high). LIN does not use the parity bit, thus encodes one data byte with 10 bits. TTP/A always uses the parity bit for error detection, thus encodes one data byte with 11 bits.

In order to support nodes with unstable clocks, a periodic synchronization pattern is provided in the LIN and TTP/A protocol. This allows the use of microcontrollers with internal RC oscillators for the implementation of the nodes. Such RC oscillators change their clock frequency with varying temperature and supply voltage so that they require frequent resynchronization.

Both protocols also agree on a time-triggered message scheme. This means that the messages are scheduled to be transmitted at a predefined point in time. This feature guarantees a collision-free media access scheme and a predictable message ordering.

The communication is initiated by a dedicated master node, the smart transducers are considered the slave nodes.

4 LIN 1.3 Specific Features

Each message in LIN is encapsulated in a single message cycle. The message cycle is initiated by the master and contains two parts, the frame header sent by the master and the frame response, which encompasses the actual message and a checksum field. The frame header contains a sync brake (allowing the slave to recognize the beginning of a new message), a sync field with a regular bit pattern for clock synchronization and an identifier field defining the content type and length of the frame response message. The identifier is encoded by 6 bit (allowing 64 different message types) and 2 bits for protection. Figure 1 depicts the frame layout of a LIN message cycle.

The frame response contains up to 8 data bytes and a checksum byte. Since an addressed slave does not know a priori to the reception of the respective frame header that it has to send a message, the response time of a slave is specified within a time window of 140% of the nominal length of the response frame. This gives the node some time to answer, for example to perform a measurement on demand, but introduces a noticeable message jitter for the frame response.

From the slave's view, the LIN protocol is a plain polling protocol, since the slaves only react on the frame header from the master. It is the master's task to issue the respective frame headers for each message according to a scheduling table. The configuration of the network must ensure, that each message has exactly one producer. Several slaves can subscribe to a particular message.

LIN clusters are configured during the design stage using the *LIN Configuration Language*. This language can be used to create a *LIN description file* (LDF). The LDF describes the complete LIN network. Its syntax is deliberately not specified to allow for vendor specific implementations.

In addition to the LIN configuration language and LDF, which are the most important tools to design a LIN cluster, the LIN specification defines a (mandatory) interface to software device drivers written in C. Also, many tools exist that can parse a LDF and generate driver modules by themselves. The LIN C API provides a signal based interaction between the application and the LIN core (core API).

LIN is supported by multitude of configuration and maintenance tools.

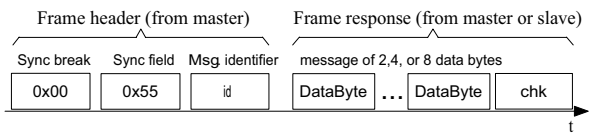

Figure 1: LIN frame format

5 LIN 2.0

In 2003, many updates to reflect the latest off-the-shelf microcontrollers as well as inputs from the SAE Task Force resulted in the definition of LIN 2.0. In fact, LIN 2.0 was a complete rework of the existing LIN 1.3 standard, but backward compatible, allowing for the integration of new LIN 2.0 master/slave nodes in existing LIN 1.2/1.3 clusters [1]. New features introduced in LIN 2.0 are an enhanced checksum, sporadic and event-triggered communication frames, improved network management (status, diagnostics) according to ISO 14230-3 / ISO 14229-1 standards, automatic baudrate detection, standardized LIN product ID for each node, and an updated configuration language to reflect the changes.

In addition to the unconditional frames (frames sent whenever scheduled according to the schedule table) provided by LIN 1.3, LIN 2.0 introduces *event-triggered frames* and *sporadic frames*.

Similar to unconditional frames, event-triggered frames begin with the master task transmitting a frame header. However, corresponding slave tasks only provide their frame response when the signals transmitted in the data fields have changed. Unlike unconditional frames, multiple slave tasks can provide the frame response to a single event-triggered frame, assuming that not all signals have actually changed. In the case of two or more slave tasks writing the same frame response, the master node has to detect the collision and resolve it by sequentially polling (i.e., send unconditional frames) the involved slave nodes. Event-triggered frames were introduced to improve the handling of rare-event data changes by reducing the bus traffic overhead involved with sequential polling.

Sporadic frames follow a similar approach. They use a reserved slot in the scheduling table, however, the master task only generates a frame header when

[1] however, a cluster with LIN 2.0 slaves requires a LIN 2.0 master, due to the slave nodes' improved configuration capabilities.

necessary, i.e. involved signals have changed their values. As this single slot is usually shared by multiple sporadic frames (assuming that not all of them are sent simultaneously), conflicts can occur. These conflicts are resolved using a priority-based approach: frames with higher priority overrule those with lower priority.

Each LIN 2.0 node possesses its *LIN Product Identification*. This unique number is stored in the microcontroller's ROM and encodes information about this node.

- supplier ID: assigned to each supplier by the LIN Consortium
- function ID: assigned to each node by supplier
- variant field: modified whenever the product is changed but its function is unaltered

In addition to signal-bearing messages, LIN 2.0 provides *diagnostic messages*. These messages use 2 reserved identifiers (0x3c, 0x3d). Diagnostic messages use a new format in their frame response called *PDU* (Packet Data Unit). There are two different PDU types: *requests* (issued by the client node) and *responses* (issued by the server node).

The LIN 2.0 configuration mode is used to set up LIN 2.0 slave nodes in a cluster. Configuration requests use SID values between 0xb0 and 0xb4. There is a set of mandatory requests that all LIN 2.0 nodes have to implement as well as a set of optional requests. Mandatory requests are:

- Assign Frame Identifier: This request can be used to set a valid (protected) identifier for the specified frame.
- Read By Identifier: This request can be used to obtain supplier identity and other property from the addressed slave node.

Optional requests are:

- Assign NAD: Assigns a new address to the specified node. Can be used to resolve address conflicts.
- Conditional Change NAD: Allows master node to detect unknown slave nodes.
- Data Dump: Supplier specific (should be used with care).

The LIN 2.0 API consists of two sections. In addition to the LIN Core API that came with LIN 1.x, there is a new *LIN Diagnostic API* that is used for configuration and the diagnostic transport layer. The LIN node Configuration API is available only to the master node and mainly implements functions to perform diagnostics and configuration [2]. LIN Diagnostic API functions:

- `check response`: gets the result of the last completed node configuration call
- `assign NAD`: assign a new address to slave nodes
- `conditional change NAD`: used to detect unknown nodes and assign new addresses to them

The LIN diagnostic transport layer is introduced to provide gatewaying functions between CAN and LIN slaves. It transports ISO diagnostic requests/responses and provides a simple *raw API*, as CAN ISO PDUs are very similar to LIN diagnostic frames.

6 TTP/A Specific Features

Communication is organized into rounds consisting of several messages. Each communication round is started by the master with a so-called fireworks byte. The fireworks byte defines the type of the round and is a reference signal for clock synchronization. The communication pattern for each round is predefined via the RODL, which is distributively stored in all nodes. The protocol supports eight different firework bytes encoded in a message of one byte using a redundant bit codesupporting error detection. One fireworks byte is a regular bit pattern, which is also used by slave nodes with an imprecise on-chip oscillator for startup synchronization. This bit pattern is identical to the sync pattern used in LIN.

Generally, there are two types of rounds:

Multipartner round: This round consists of a configuration dependent number of slots and an assigned sender node for each slot. The configuration of a round is defined in a datastructure called "RODL" (ROund Descriptor List). The RODL defines which node transmits in a certain slot, the operation in each individual slot, and the receiving nodes of a slot. RODLs must be configured in the slave nodes prior to the execution of the corresponding multipartner round. An example for a multipartner round is depicted in Figure 2.

Master/slave round: A master/slave round is a special round with a fixed layout that establishes

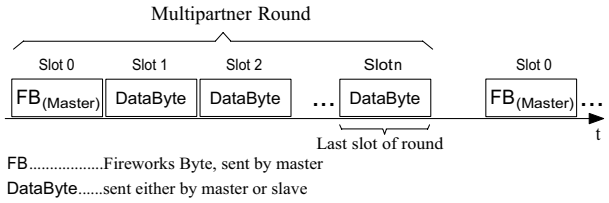

Figure 2: Example for a TTP/A multipartner round

a connection between the master and a particular slave for accessing data of the node's IFS, e. g. the RODL information. In a master/slave round the master addresses a data record using a hierarchical IFS address and specifies an action like reading of, writing on, or executing that record.

The master/slave rounds are used for diagnostics and configuration of the smart transducer nodes. The periodical multipartner rounds provide a predictable real-time communication among the nodes. Master/slave rounds are scheduled periodically between multipartner rounds as depicted in Figure 3 in order to enable maintenance and monitoring activities during system operation without a probe effect.

The master/slave rounds allow a point-to-point connection to a particular node for configuration, maintenance, and diagnosis purposes. Each node is assigned a logical node ID that is used in the master/slave rounds for addressing a node within a cluster. The logical IDs of TTP/A nodes can be assigned either at compile time or on-line when the node is integrated into the cluster. The online-assignment of logical node IDs is called baptizing. The baptizing algorithm is performed by the master and is based on binary search. It make use of the unique identification number of every TTP/A node, the conditional setting of node identifiers by executing a special record in the file system of the node and the ability to detect simultaneous bus access of multiple nodes. The baptizing enables true plug and play but is not mandatory in the TTP/A standard.

The syntactic specification of the TTP/A communication interface is part of the Smart Transducer standard of the Object Management Group (OMG) [11].

While the TTP/A standard does not define a particular configuration language for the network management, such an approach based on XML-descriptions has been proposed by Pitzek in [12, 13].

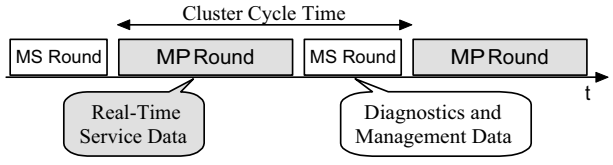

Figure 3: Recommended TTP/A Schedule

7 Comparison

Table 1 depicts the performance differences of LIN and TTP/A for a typical body electronics application. The bus speed of LIN networks is specified to be flexible, but the maximum specified bus speed is defined by 19,200 Bit/sec.

7.1 Efficiency

The number on efficiency for LIN varies, because of the possible delay of 40% in the slave responses. The first figure for efficiency of LIN 2.0 was calculated assuming unconditional frames.

When the new feature of event-triggered frames from LIN 2.0 is used, the best case efficiency is much better at the cost of worst case behavior. In the example in the table, we have assumed that always four sensors share one event-triggered slot.

The figures on efficiency have been derived as follows:

The specified 16 messages with 2 bytes each require at least $T_{\text{net}} = 16 \cdot 2 \cdot 8 = \mathbf{256}$ bit to be sent over the network.

In case of LIN 1.3 we would have to send 16 frame headers and 16 unconditional messages with 2 byte and a check byte. The LIN protocol specification 2.0 [2] defines the length of a frame header by a nominal value of $T_{\text{Header\_Nominal}} = 34t_{\text{bit}}$ and a maximum value of $T_{\text{Header\_Maximum}} = (T_{\text{Header\_Nominal}} \cdot 1.4$. The frame response length is given by $T_{\text{Response\_Nominal}} = 10 \cdot (N_{\text{DATA}} + 1) = 30t_{\text{bit}}$ and $T_{\text{Response\_Maximum}} = 1.4 \cdot T_{\text{Response\_Nominal}}$, respectively. Thus, for 16 unconditional frames we require between $16 \cdot (34 + 10 \cdot (2+1)) = \mathbf{1024}\ t_{\text{bit}}$ and $16 \cdot (34 \cdot 1.4 + (10 \cdot (2+1)) \cdot 1.4) = \mathbf{1433.6}\ t_{\text{bit}}$ to communicate the 16 messages, giving an efficiency $T_{\text{net}}/T_{\text{LIN1.3}}$ between **18%** and **25%**.

Using event-triggered messages changes the number of required messages and increases the number of data bytes per message by 1, since the first data

[2] In the opinion of the authors, the LIN protocol specification 1.3 gives an incorrect formula on the calculation of the message frame length [**?**, p. 25], therefore and for consistency reasons we will use the formula for unconditional frames from the LIN Protocol Specification 2.0 [2, p. 25]

Characteristic	LIN 1.3	LIN 2.0	TTP/A
Bus speed	up to 19.2 kBit/s	up to 19.2 kBit/s	arbitrary, typical up to 50 kBit/s
Message length	1...8 byte	1...8 byte	1...16 byte
Efficiency (for 16 messages of 2 byte, excluding diagnosis messages)	18...25%	18...25%(14...86%)	39 %
Typical implementation footprint (Slave)	362B Flash, 19B RAM [1]	2817B Flash, 49B RAM [3]	2672B Flash 63B RAM [15]
Message jitter	40% jitter	40% jitter	<1% jitter
Synchronized Measurements	only subsequent actions possible	only subsequent actions possible	full support of synchronized actions
Error detection	header 75% response 99.6%	header 99.9% after response response 99.6%	fireworks byte 98.4% messages >99.9%

Table 1: Comparison of Performance Characteristics

byte of an event-triggered frame has to carry the protected identifier of the corresponding unconditional message. In the best case, i. e., when each set of four sensors produces at most one update message per request, the number of messages drops down to 4, giving $4 \cdot (34 + 10 \cdot (3 + 1)) = \mathbf{296}\ t_{\text{bit}}$ as minimum time to communicate the information of the 16 messages, giving an efficiency of **86%**.

However, in the case that more than one sensor in an event-triggered set provides an updated value, extra unconditional frames are necessary to update the values and besides the 4 event-triggered messages another 16 unconditional messages could be necessary. Using the maximum frame length, the worst case scenario calculates to $4 \cdot (34 \cdot 1.4 + (10 \cdot (3 + 1)) \cdot 1.4) + 16 \cdot (34 \cdot 1.4 + (10 \cdot (2 + 1)) \cdot 1.4) = \mathbf{1848}\ t_{\text{bit}}$, giving an efficiency of **14%**.

For TTP/A, we assume a RODL with 16 messages where each message consists of 2 data bytes and one check byte. Furthermore, we have to take the fireworks byte and the inter round gap (1 byte of length) into account giving us a total number of $2 + 16 \cdot (2 + 1) = \mathbf{50}$ bytes to be sent. Each byte is encoded in a frame of 13 bit, thus the time to communicate all 16 messages evaluates to $50 \cdot 13 = \mathbf{650}\ t_{\text{bit}}$, giving a constant efficiency of **39%**.

7.2 Resource Requirements for Implementation

The minimum implementation footprint of LIN is much smaller due to the fact that a LIN node does not require to keep a local copy of the communication schedule. However, most of the available low-cost 8 bit microcontrollers provide at least the resources required for the TTP/A slave protocol.

Since a LIN slave is not aware of the global time, the protocol does not support synchronization of actions, for example synchronized measurements. This is inherently supported by the TTP/A protocol.

7.3 Error Detection

The error detection depict a weakness in LIN 1.3 on the protection with only two extra bit for the message identifier of the header message, giving an error detection probability of only $1 - 1/2^4 = 0.75$. The header message is critical for a LIN communication, since an undetected erroneous header message may cause a wrong slave to answer with a syntactically correct message.

The response frame of LIN is protected by a check byte, giving an error detection probability of $1 - 1/2^8 = 0.9961$.

LIN 2.0 solves the problem of the weak error detection of the header message by specifying an enhanced checksum for the frame response that includes the

protected message identifier in the checksum calculation. An erroneous header message may still cause a wrong slave to answer with a frame response, but the checksum of the frame response will be different from the expected one. Thus, there are 10 bits used for error detection of the header message, giving an error detection probability of $1-1/2^{10} = 0.9990$. However, note that there is still a chance of 75 % that an erroneous header message might cause the wrong slave to trigger a measurement at the wrong time.

The error protection of unconditional frames in LIN 2.0 is identical to LIN 1.3, thus rendering also an error detection probability of 0.9961.

The fireworks byte in TTP/A consists of a data word of 9 bits (including parity), whereof 3 bit define the round number and 6 bit are protection, giving an error detection probability of $1 - 1/2^6 = 0.9843$. The fireworks byte code has been especially designed to have a high resilience against frequent error patterns, such as burst errors of length less than 6 and all possible length of negative or positive pulses. The Hamming distance of the code is 4 [7].

The figure of 99.9% error detection for TTP/A messages has been given for the shortest message of 1 data byte (and one check byte), which constitutes the worst case scenario. For longer TTP/A messages, the error detection is higher because of the additional parity bit protection. Generally, the error detection of a TTP/A message of N bytes protected by one check byte is $1 - 1/2^{N+1+8}$, giving an error detection probability between 0.9990 and 0,99999997 (for a message of 16 bytes).

8 Discussion and Conclusion

Most of the new features in LIN 2.0 have greatly improved its applicability, so the diagnostic messages (which are very similar to the master/slave rounds in TTP/A) and the standardized LIN product identification, and the enhanced checksum which compensates for the comparatively weak protection of the frame header.

A major limitation of LIN in comparison to TTP/A might be the maximum communication speed. The LIN specification defines the maximum specified bus speed by 19,200 Bit/sec. This fact is explained by the use of the unshielded ISO k-line bus. The ISO k implementations, however, typically support up to 50,000 Bit/sec (according to the MC33290D datasheet). Taking the large overhead of LIN messages into account, the achievable net bandwidth is rather slow, even for simple body electronics applications such as windshield wipers or window lift motors.

LIN 2.0 specification tries to remedy the performance problem by the introduction of event-triggered messages in order to achieve a higher average network efficiency. The introduction of these event-triggered messages, however, is not well suited for real-time data, since it worsens the worst-case behavior. Moreover, event-triggered messages complicate network diagnosis, since a missing answer from a node could be interpreted as a crashed node or a correct node that has not perceived a change in the observed measurement property.

The LIN specification defines also a complete application framework supporting developing and configuring LIN networks. This standardization is an important step to support tool providers.

The TTP/A specification describes a pure communication system and leaves a lot of decisions regarding physical layer, fault detection, redundancy, etc. to the system designer. The time-triggered scheduling of messages leads to a high data efficiency, even for short messages.

TTP/A strongly supports a two-level design approach. The node developer does not deal with communication and message issues, while the system integrator does not have to deal with node-specific issues. This property is provided by the introduction of the IFS, which allows for an abstraction over the communication system. Therefore, it is possible to modify the communication system in a great deal supporting maximum flexibility.

Configuration tools for TTP/A are build on this IFS and the CORBA interface, which is specified in the OMG Smart Transducer Standard. Configuration data structures, such as electronic datasheets are implemented by XML descriptions and respective tools, but are currently not covered by the standard.

While the minimal footprint of TTP/A and LIN implementations both support implementations on low-cost microcontrollers, the protocols follow different philosophies. The TTP/A protocol as part of the OMG Smart Transducer Interface Standard supports an open and flexible system that can be applied for different fields of application. On the other hand, the properties of LIN like bandwidth, physical layer and message types/lengths are tailored to a subset of automotive applications and limit the application of the protocol as a general purpose fieldbus.

Acknowledgments

This work was supported in part by the European IST project DECOS under contract No. IST-511764. We would like to thank the anonymous reviewers and our colleagues at the Institute of Computer Engineering for their valuable comments on earlier versions of this paper.

References

[1] Atmel Corporation. *AVR 308: Software LIN Slave*, May 2002. Application note available at http://www.atmel.com.

[2] Audi AG, B. AG, D. AG, M. Inc. V. C. T. AB, V. AG, and V. C. Corporation. LIN specification v2.0, 2003.

[3] P. Cholasta. LIN 2.0 mirror unit slave based on the MC68HC908EY16 MCU and the LIN 2.0 communication protocol. Application Note AN2885, Rev. 0, 11/2004, Freescale Semiconductor, 2004.

[4] W. Elmenreich, W. Haidinger, P. Peti, and L. Schneider. New node integration for master-slave fieldbus networks. In *Proceedings of the 20th IASTED International Conference on Applied Informatics (AI 2002)*, pages 173–178, Feb. 2002.

[5] J. B. et al. FlexRay–The communication system for advanced automotive control systems. *SAE World Congress 2001, Detroit, Michigan, USA*, Mar. 2001.

[6] T. Fuhrer, B. Muller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on CAN. Technical report, Robert Bosch GmbH, http://www.can.bosch.com/, 2000.

[7] W. Haidinger and R. Huber. Generation and analysis of the codes for TTP/A fireworks bytes. Research Report 5/2000, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2000.

[8] H. Kopetz. TTP/A – A time-triggered protocol for body electronics using standard uarts. In *International Congress and Exposition*, Detroit, MI, USA, Feb.-Mar. 1995. The Engineering Society for Advancing Mobility Land Sea Air and Space, SAE International.

[9] H. Kopetz, M. Holzmann, and W. Elmenreich. A universal smart transducer interface: TTP/A. *International Journal of Computer System Science & Engineering*, 16(2):71–77, Mar. 2001.

[10] MOST cooperation. *MOST Specification Framework Rev 1.1*, Nov. 1999. http://www.mostnet.de.

[11] Object Management Group (OMG). *Smart Transducers Interface V1.0*, Jan. 2003. Specification available at http://doc.omg.org/formal/2003-01-01 as document ptc/2002-10-02.

[12] S. Pitzek. Description mechanisms supporting the configuration and management of TTP/A fieldbus systems. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[13] S. Pitzek and W. Elmenreich. Configuration and management of a real-time smart transducer network. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*, Lisbon, Portugal, Sept. 2003.

[14] SAE. Controller Area Network CAN, an in-vehicle serial communication protocol. In *SAE Handbook 1992*, pages 20.341–20.355. SAE Press, 1990.

[15] C. Trödhandl. Architectural requirements for TTP/A nodes. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[16] TTAGroup. *Specification of the TTP/C Protocol*. TTAGroup, 2003. Available at http://www.ttagroup.org.