Research Report 4/2000 Institut für Technische Informatik, TU Wien, Austria

A Comparison of LIN and TTP/A

H. Kopetz
W. Elmenreich
C. Mack
hk@vmars.tuwien.ac.at

Institut für Technische Informatik
Technische Universität Wien
Austria

April 10, 2000

Abstract: This paper compares two novel field-bus protocols for low-cost single-chip smart sensor and actuator nodes, LIN and TTP/A. Both protocols are central-master UART protocols, where the master with its precise oscillator establishes the stable time-base required by the slaves to synchronize their imprecise on-chip oscillators. While LIN provides the basic services needed for real-time communication, the TTP/A standard additionally specifies an interface-file system to perform on-line configuration, diagnostics and maintenance of smart sensor nodes. With TTP/A it is thus possible to produce preprogrammed simple transducer nodes or generic smart transducer nodes that can be configured dynamically to the given application requirements.

Keywords: LIN, TTP/A, real-time systems, field bus, smart sensors, interface

INTRODUCTION

With the development of a new generation of compact low-cost microcontrollers that contain the frequency oscillator on chip, the need for a new class of field-bus communication protocols arises, because these new microcontrollers are not in the position to support any serial communication without a start-up calibration of the on-chip oscillator. The potential for cost saving that can be expected from these new microcontrollers is significant. Since all functional blocks that are required to realize a smart sensor/actuator node, including network access and often a MEMS sensing element, can be implemented on a single silicon die, the economics of Moore's law are leading to very cost-effective smart sensor nodes that will challenge any other solution in the emerging networked embedded market.

Two new field-bus protocols that address this important future market are LIN (Local Interconnect Network) and TTP/A (Time-Triggered Protocol for SAE class A applications). The LIN protocol was presented to the public at a press conference (LIN 2000) at the SAE World Congress in Detroit on March 6, 2000 by a consortium of seven automotive partners (Audi, BMW, DaimlerChrysler, Volvo, Volkswagen, Motorola and VCT). The LIN activities were initiated by a workgroup of these companies in October 1998 and a first specification draft was released by this workgroup in July 1999. The objectives of LIN are to provide a standard low-cost sensor network below the CAN functionality that results in communication costs per node that are two to three times lower when compared to CAN. According to the authors (LIN 2000) "LIN shall complement CAN and not replace CAN".

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. In contrast to LIN, which is driven 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 (Kopetz 1995). In the mean time, a start-up synchronization and an interface-file system (IFS) has been added to TTP/A (Kopetz, Holzmann et al. 2000).

It is the objective of this paper to compare the two protocols LIN and TTP/A from a number of different perspectives. The basis of this comparison is the LIN protocol as described in (Wense 2000) and the TTP/A Protocol Specification (TTP/A 2000). The paper starts with an elaboration of the three interfaces, the real-time service (RS) interface, the diagnostic and maintenance (DM) interface, and the configuration planning (CP) interface that a smart sensor field bus should support. In Chapter three the criteria for comparison are established. Chapter four outlines the principles of operation of LIN and TTP/A. Chapter five compares the two protocols from the points of view of timeliness, dependability, on-line diagnostics, "plug-and-play" capability, physical layer, start-up-synchronization, and cost. The paper finishes with a conclusion in Chapter six.

2. THREE INTERFACES OF A SMART SENSOR NODE

An interface is a common boundary between two subsystems. An information exchange across an interface is only possible if the engaged subsystems share a common background of concepts and a common coding system. In the context of a distributed control system, the area of concern is a cluster, consisting of a set of sensor, actuator, and processing nodes connected by a communication medium (see, e.g., Fig. 4). The set of all nodes of the cluster must thus share the same concepts and must agree on common code spaces.

In the industrial control industry, the classic 4-20 mA analog current loop interface has been highly successful for many years because of its simplicity and its understandability: The name space of the relevant state variables is formed by the set of interfacing wires, each wire denoting one particular state variable. Being an analog system with minimal delay, the time of delivery (reading a value at a receiver) is about the same as the time of generating the value and is implicit to the reading operation, resulting in a protocol with negligible delay and jitter. Any value between 0 and 100% of the chosen range is mapped into the standardized 4 to 20 mA current interval, that denotes the code space of good values. This codespace is extended to provide an in-band error code, the value 0 mA that denotes an error (no signal). The 4-20 mA interface technology was primarily developed to interface an analog sensing element with a small amount of local analog processing logic to an (remote) analog controller. The noise pickup on the analog transmission line is one of the limiting factors for the accuracy of 4-20 mA signals.

A smart sensor is the combination of an analog or digital sensor or actuator element and a local microcontroller that contains the interface circuitry, a processor, memory and a network controller in a single unit. The smart sensor transforms the raw sensor signal to a standardized digital representation, checks and calibrates the signal, and transmits this digital signal via a secure communication protocol to its users. More and more sensor elements are themselves microelectronics mechanical systems (MEMS) that can be integrated on the same silicon die as the associated microcontroller. The smart sensor technology offers a number of advantages from the points of view of technology, cost and complexity management:

(i) Electrically weak non-linear sensor signals that originate from an MEMS sensor can be conditioned, transformed into digital form, and calibrated on a single silicon die without any noise pickup from long external signal transmission lines (Deirauer and Woolever 1998).

(ii) It is possible to locally monitor the operation of the sensing element and thus simplify the diagnostics at the system level. In some cases it is possible to build smart sensors that have a single simple external failure mode--fail-silent, i.e., the sensor operates correctly or does not operate at all.

(iii) The interface of the smart sensor to its environment is a well-specified digital communication interface to a sensor bus, offering "plug-and-play" capability if the sensor contains a reference to its documentation on the INTERNET.

(iv) The internal complexity of the smart-sensor hardware and software and the internal sensor failure modes can be hidden from the user by well-designed fully specified smart sensor interfaces that provide just those services that the user is interested in. Thus a proper interface design enabled by the smart sensor technology can contribute to a reduction of the complexity at the system level.

A smart sensor needs a much larger name space than a simple analog sensor. In addition to the actual measured values, the parameters for range selection, alarm limits, signal conditioning, and calibration must be set by the user (see the IEEE standard 1452 on "A smart transducer interface for sensor and actuators", IEEE 1997). Information about sensor performance and diagnostic information must be stored in the sensor and accessed during maintenance. Furthermore it must be possible to configure a "generic" smart sensor into a new application context. Many instances of the same sensor type, e.g., a temperature sensor, may be used in differing roles within an application. In the emerging market of massively distributed embedded systems, these configurations should be performed on-line during system operation.

From the point of view of complexity management and composability, it is useful to distinguish between three different types of interfaces of a smart sensor: the real-time-service (RS) interface, the diagnostic and maintenance (DM) interface, and the configuration planning (CP) interface. In the next section we discuss the properties of these three different types of interfaces.

2.1 The Real-Time-Service (RS) Interface

In the abstract, the purpose of the RS interface is the timely exchange of "observations" of real-time (RT) entities between the engaged subsystems. An RT entity is a state variable of interest that has a name and a value. An observation (Kopetz 1997 p.99) is thus an atomic triple

<RT entity name, instant, value>

where *RT entity name* is an element of the common namespace of RT entities, *instant* is a point on the time-line and *value* is an element of the chosen domain of values. An observation thus states that the referenced RT entity possessed the stated value at the indicated instant. In control applications, the temporal access pattern of information at the RS interface is periodic. A small delay and minimal jitter are important for the quality of control. These temporal parameters must be stable in order to support the composability at the RS interface. The user of the observations at the RS interface must only know about the meaning of these observations, but does not need any knowledge about the internal structure or operation of the smart sensor. If an observation is produced by a set of replicated sensors, the user is only interested in the availability of a timely observation, but not which one of the set of replicated sensor produced this observation. The unit of addressing is thus the name of the observation, but not the node that produced this observation.

2.2 The Diagnostic and Maintenance (DM) Interface

The DM interface opens a communication channel to the internals of a smart sensor for the purpose of diagnostic and maintenance. It is used for setting sensor parameters and for retrieving information about the internals of the sensor. The maintenance engineer that accesses the sensor internals via the DM interface must have detailed knowledge about the internal structure and behavior of the sensor. The end-points of communication are the internals of a sensor on one side and some maintenance expert sitting on a remote terminal on the INTERNET on the other side. The communication pattern is thus point-to-point and the messages between the sensor and the maintenance engineer must be routed transparently through a set of networks. Since maintenance is performed on a physical sensor, the addressing system must reach each physical sensor instance. If, for example, ten identical temperature sensors are used in differing roles in an embedded application, it must be possible to address each one of these physical sensors individually. Ideally, the DM interface should be independent from the RS interface, since these two interfaces are directed towards two different user groups.

There is a need to support on-line maintenance while a system is operational. To achieve this objective, the sporadic maintenance traffic must coexist with the time-critical real-time traffic without disturbing the latter. The traffic pattern across the DM interface is normally sporadic and not time-critical, although precise knowledge about the point in time when a particular value was observed is important.

2.3 The Configuration Planning (CP) Interface

The CP interface is used to connect a component to other components of a system. It is used during the integration or reconfiguration phase to generate the "glue" between a generic sensor and the embedded application it is serving. The use of the CP interface does not require detailed knowledge about the internal operation of a sensor, but requires knowledge about the structure of the middleware. The CP interface is point-to-point and not time-critical.

The following Table 1 summarizes the characteristics of these three interfaces

	RS-Interface	DM-Interface	CP- Interface
Purpose	provide timely real-time images	perform diagnostics and maintenance	configure a generic sensor into an application
Required Knowledge	Knowledge about the role of the sensor output in the real-time application	Knowledge about the internal operation of the sensor	Knowledge about the configuration process
End--points of the communication	Sensor Output and communication network interface of the master	Internals of the sensor and service technician on the INTERNET	Middleware and Configuration Process
Topology	multicast	point-to-point	point-to-point
Timeliness	low latency, small jitter	time-of observation of sensor value	none
Temporal access pattern	periodic	sporadic/periodic	sporadic
Unit of addressing	real-time image	physical node	physical node
Relevant for temporal composability	yes	no	no

Table 1: Characteristics of the three interfaces of a smart sensor

3. REQUIREMENTS OF A FIELD BUS

In this section, the requirements for a field bus to smart transducer nodes are outlined. In the following sections, the two protocols, LIN and TTP/A will be compared with respect to these requirements.

3.1 Timeliness

The most important requirements for a sensor bus that is used for the periodic RS service in embedded control applications are temporal predictability, low latency and minimal latency jitter. The latency can be improved if the protocol operates with high data efficiency. Since in many scenarios a smart sensor will only deliver a few bits of information (e.g., the status of a photo-cell or the position of a switch) it is important that these very short periodic data fields can be transported with a small protocol overhead.

In a number of control applications precise knowledge about the instant when a RT-entity has been observed is essential. To meet this requirement, the sensor node must contain a synchronized global time that can be used to time-stamp the occurrence of a significant event within the smart sensor node. The time-stamp value is then transported to the master just as any other data value (time as data, see Kopetz 1997 p.194).

3.2 Dependability

Within the constraints given, a field bus must support a trustworthy level of dependability and high error-detection coverage.

3.3 On-line Diagnostics

Smart sensors can vary from simple devices (e.g., a photocell that observes a single RT entity) to large subsystems that encapsulate complicated measurement methods of an RT entity that can only be measured indirectly by observing a number of state variables with different sensing elements. These large sensor subsystems must be parameterized, calibrated, and diagnosed in case of malfunction. Ideally, it should be possible to monitor and investigate the internal operation of a sensor subsystem by a domain expert from a remote terminal (that maybe connected to the sensor via the INTERNET) while the system is in operation (see also Figure 2).

To meet this diagnostic requirement, it must be possible to address an internal storage space of sufficient size within each *physical* sensor via the network. This storage space contains the parameters, diagnostic information and intermediate results that are important for the analysis of the correct operation of a sensor by a domain expert. Ideally, it should be possible to open a dynamic real-time communication channel with known bandwidth, guaranteed latency and minimal jitter to the internal storage space of every sensor in order to provide a real-time monitoring capability.

3.4 "Hot Plug and Play" Capability

Massively distributed embedded systems will contain a very large number of smart sensor nodes that join and leave a system during the operation of the system (Nesett 1999). This requires the capability to identify new generic sensors, to parameterize the generic sensor for the given application, to reconfigure sensors and to remove sensors while the system is operational.

3.5 Low Complexity and Low Cost

The field-bus must support compact single-chip smart sensor nodes and low wiring and installation costs. The protocol complexity should be such that the protocol can be implemented on low-cost microcontrollers.

4. PRINCIPLES OF OPERATION OF LIN AND TTP/A

In this section the principles of operation of LIN and TTP/A are elaborated. For a more detailed description of the protocols, the interested reader is advised to consult the respective specifications on the INTERNET, (LIN 2000) for LIN and (TTP/A 2000) for TTP/A.

4.1 Common Characteristics

Protocols, LIN and TTP/A are UART protocols that require a central master that communicates with a set of slave nodes. This central master with its stable timebase provides the signals for the start-up and continuous synchronization of the slave-nodes and coordinates the traffic on the bus. Both protocols support a "master-slave" dialogue that is initiated by the master. This master slave dialogue consists of a "command frame" sent by the master and a "data frame" that is sent either by the master (if the data is an output message from the master to the sensor) or by a slave node (if the data is input message from the sensor to the master). In both protocols, the master slave dialogue has a structure as shown in Fig.1. There are significant differences between LIN and TTP/A in the way data is named in the command frame.

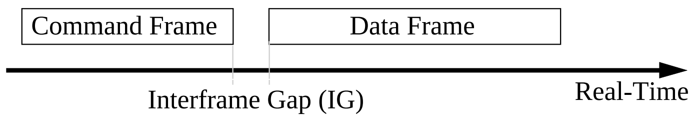

Figure 1: Master -Slave Dialogue

In LIN, the interframe gap is variable, in TTP/A it is constant. This seemingly minor variance between LIN and TTP/A is caused by a fundamentally different philosophy of protocol design. In LIN it is assumed that the temporal predictability of protocol execution in the slave node is subordinated to the local application task processing within such a slave. In LIN, local temporal properties of the application tasks are thus considered more important than global temporal properties of the communication protocol, resulting in a significant permitted jitter concerning the point in time when a message must be sent (see the LIN standard LIN 2000). In TTP/A, the global timing properties, e.g., the timing specification of the communication protocol, have priority over the local timing properties within a slave. In TTP/A the protocol tasks within a slave node must thus have a higher priority than the node-local applications tasks. In the architecture based TTP/A protocol design a good end-to-end timing and minimal jitter of a distributed transaction has been considered most important (see also Table 2).

The software development for both protocols, LIN and TTP/A, is supported by software tools. The software tools specification for LIN includes a standard for the Configuration File description and a corresponding application program interface (API). The TTP/A development tools will be integrated in the TTP tool chain (see TTTECH 2000).

Neither the LIN standard, nor the TTP/A standards require that all implementations support the full protocol functionality. An implementation is thus free to decide which functionality to support. However, if a given functionality is included in an implementation it must adhere to the standard.

4.2 LIN

LIN supports open master-slave dialogues. In an open master-slave a named data frame is put on the LIN bus (either by a slave or a slave task within the master) after the command frame from the master, according to the *a priori* associated attributes of the selected identifier. A data frame on the bus can be accessed by any node interested in this data frame. In LIN a one-byte identifier names a data frame. This identifier consists of two fields, a six-bit address field and a two bit-check field. The six-bit address field provides a name space of 64 distinct identifiers within a cluster. The following attributes are statically (*a priori*) associated with each identifier:

- Meaning of the data
- Direction of the data (input or output)
- Length of the data field (determined by the two least significant bits of the address field), allowing for a total of 32 two byte messages, 16 four byte messages and 16 eight byte messages within a cluster

There is no mechanism in LIN to directly name the node that produces or consumes a data element. An open master slave dialogue in LIN consists of a three byte command frame (one byte synchronization break, one byte synchronization field, one byte identifier) and a variable length data frame of between three and nine bytes (two to eight data bytes and one check byte).

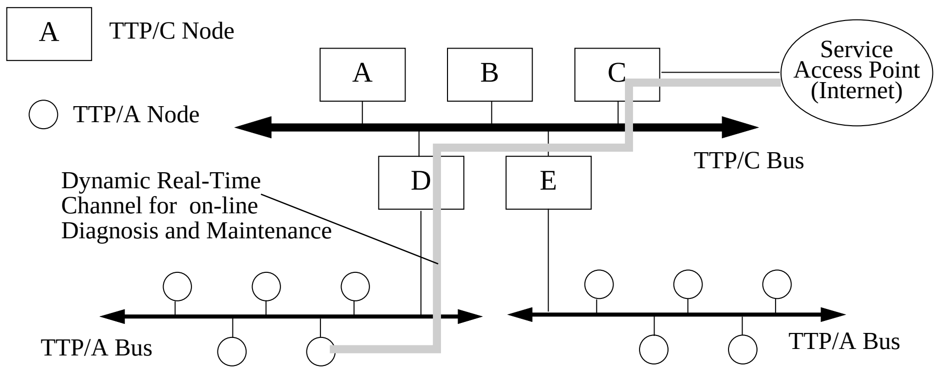
A
TTP/C Node
TTP/A Node
A
B
C
D
E
Service Access Point (Internet)
TTP/C Bus
Dynamic Real-Time Channel for on-line Diagnosis and Maintenance
TTP/A Bus
TTP/A Bus

Figure 2: On-line diagnostic transaction into a TTP/A sensor node

4.3 TTP/A

During the design of TTP/A it has been an objective to provide on the one side a “function-rich” specification that supports dynamic sensor configuration and diagnosis, but on the other side the possibility of implementing very simple low-cost pre-configured sensor nodes that can be implemented in a state machine.

TTP/A supports open master-slave dialogues and multipartner rounds. Both are initiated by a command frame (a "fireworks byte") from the master. The end-points of the communication in TTP/A are the records (or part of a record) of a local interface file system (IFS) that resides in each node. The union of all local interface file systems of all nodes in a cluster forms the distributed IFS of the cluster.

The IFS is an index-sequential file system with a constant record length of 5 bytes (4 data bytes and one check byte). Each node can contain of up to 60 files with 255 records each. The last record of each file is a check record. TTP/A supports three operations on files: read a record, write a record, or execute a file. A command frame to access a file in TTP/A consists of five bytes as follows

<master-slave ("fireworks"), file op and identifier, record number, logical node name, check byte>

A master slave dialogue in TTP/A consists of two frames, a command frame of five bytes (4 command bytes and one check byte) followed by a constant length five-byte data frame (4 data bytes and one check byte). Master slave dialogues are used in TTP/A to implement the DM interface and the CP interface. The constant length of a master-slave frame makes it relatively easy to route this record via a dynamic real-time channel through a set of interconnected TTP/C clusters to a local INTERNET WEB site (Fig.2). Thus it is possible to monitor the internals of any TTP/A sensor via the master-slave rounds while the system is in operation.

TTP/A provides multipartner rounds to implement the periodic traffic across the time-critical RS interface. A multipartner round can be up to 64 bytes long. The master starts a multipartner round in TTP/A by sending a one-byte *execute-RODL-file* command. The file-execute operation points to a distributed IFS file in each node that must contain a round descriptor list (RODL) that specifies the structure of the round. The RODL determines for each message the message length in bytes, which node has to send the message and which nodes are listening to the message (and some additional parameters). Furthermore, the relevant IFS addresses contained in the RODL specify the sources and the sinks of the data. Since the specification of the multipartner round, the RODL, is itself contained in a distributed file of the IFS, it is possible to reconfigure rounds via the CP interface while the system is in operation. The one-byte *execute-RODL-file* command supports a name space of 8 different RODL files. These RODL names are coded into the "fireworks" byte with a Hamming distance of 4.

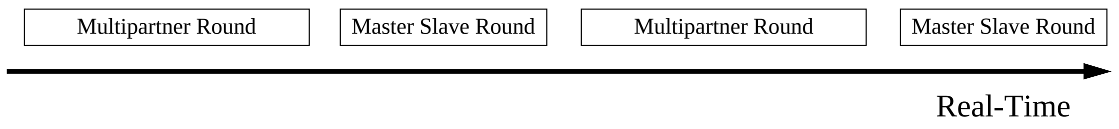

Figure 3: Traffic on the TTP/A Bus

During normal TTP/A operations there is a regular sequence of multipartner rounds and master-slave rounds (Fig. 3). The periodic multipartner rounds exchange current real-time observations for the RS service, while the sporadic master-slave rounds access a TTP/A file for a DM or CP service, if required. In case there is a need for an optimized RS service and no need for the DM or CP service, then the number of occurrences of master-slave rounds can be reduced (e.g., only one master-slave round after every tenth multipartner round) or they can be totally omitted during the time-critical phases of a process.

TTP/A provides an address space for up to four special commands in a master-slave round. In addition to the standard "read", "write" and "execute" command on a file, one special command is used to send all nodes of a cluster to the "sleep mode. Another special command is used to "baptize" nodes, i.e., to associate a logical name (a role name in the cluster) to the physical name of the node. In TTP/A every node has a physical name and a logical name (see Section 5.4). In low-cost implementations the logical name can be assigned *a priori*. Such a node does not need to support the baptize command.

The *a priori* assignment of the logical name, a RODL execute command, and the position of the output information (the "output" byte) within the specified multipartner round may lead to a minimal TTP/A node (see Fig. 5c). Such a node only waits for the appearance of "its" firework byte on the bus and then transmits its output byte in the *a priori* specified slot. Such a minimal node will not support file operations for diagnostics and or dynamic configuration. If such a minimal node needs to be synchronized after start-up, then it must also understand the start-up synchronization command (see Fig. 5b).

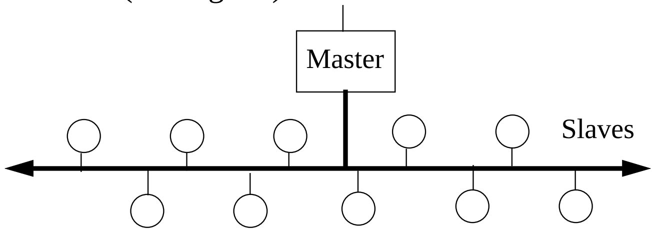

Figure 4: Sample configuration for response time comparison.

5. COMPARISON

5.1 Timeliness

Both protocols, LIN and TTP/A, require a master to periodically poll the slaves to learn about an event occurrence in one of the slaves. The minimum period of such a polling round is thus the best response time that can be achieved in such architecture. In order to compare LIN and TTP/A, the configuration of Fig. 4 has been chosen. A master is connected to 10 slave nodes. For the time-critical RS service the slave nodes need to send (or receive) periodically data from the master.

10 nodes, response time in milliseconds on a 20 kbit bus	Minimum LIN	Maximum LIN	Minimum TTP/A	Maximum TTP/A
Every nodes sends four bytes of data	46.75 msec	65.4 msec	35.4 msec	35.6 msec
Every nodes sends two bytes of data	35.75 msec	50.05 msec	22.2 msec	22.3 msec
Every node sends one byte of data	35.75 msec	50.05 msec	15.6 msec	15.7 msec
Every node sends four bits of data	35.75 msec	50.05 msec	9 msec	9.1 msec
Every node sends four bits of data, additional master-slave round for DM service between any two multipartner rounds in TTP/A	not supported	not supported	16.8 msec	16.9 msec

Table 2: Achievable response times of LIN and TTP/A

Response Times: We assume that both protocols operate on a 20kbit/second bus. If the bus speed is increased the response times will be reduced about proportionally.

Five scenarios have been evaluated. Rows two to five denote the response time of the RS service if the slaves (or the master) send the specified number of data bytes in each round. The last row of Table 2 denote the response times of a TTP/A system when a master/slave dialogue for on-line diagnostics (the DM or the CP service) is inserted between any two multipartner rounds. In the chosen configuration, such a master slave dialogue takes 13 bytes, i.e., 7.8 msec (including start-up synchronization, fireworks, and the necessary intervals of silence). This scenario of the last row is of particular significance for a TTP/A application. A multipartner round with 4 bits of data from each slave allows the master to recognize one (or more) of four important events that has occurred at the slave. The master can then communicate with the service-requesting slave by executing a master-slave dialogue directed to this slave. Both rounds together take less than half of the time of a comparable LIN round. Even if a master slave round of 7.8 msec is added after every multipartner RS round, the response time of TTP/A is lower than that of LIN.

Jitter: A low jitter is important in real-time applications, where control loops are closed by a field bus, since any jitter results in a degradation of the control quality. Both protocols, LIN and TTP/A are free of conflict and therefore have small jitter from the point of view of the communication system. In the LIN standard, the interframe gap between the frames is variable in the millisecond range, causing a jitter as shown in Table 2. The LIN standard allows this jitter to be 40% of the duration of a transmission in order to provide processing capacity to high-priority application tasks in a LIN node (see also Section 4.1). In TTP/A the interframe gap, which can be parameterized, must be constant during any protocol execution in order to avoid any additional jitter. As a consequence in TTP/A the difference between the minimum and maximum response time is bounded by the maximum clock offset between synchronization events. Assuming that the clock drift of an imprecise on-chip oscillator is less than 10%/second, the jitter in TTP/A, even in the longest multipartner round of 64 bytes, is less than 1%. If a resynchronization event is scheduled after every 8 bytes of a multipartner round, then the jitter in TTP/A is less than 0.1%.

Temporal Awareness: In order to record the occurrence of an event that happens within a polling period, TTP/A provides a global time-base within each sensor node. The epoch of this time-base is the start of the command frame of each periodic multi-partner round. The precision of this time base is about one third of the bit-cell length. TTP/A provides a standard time format for the representation of the slave-node local time. In order to distinguish between overlapping rounds, this time format contains an alternating bit.

The LIN standard does not contain any information about a local time in a LIN node, although, according to our view, such a service could be implemented in LIN.

Characteristic	LIN	TTP/A
Parity on every byte	no	yes
Protection of command frame ("fireworks")	2 bit check field	5 bit check field
Protection of normal data frame	check byte	check byte
Protection of short (4 bit) data frame	not applicable	check nibble
Protection of IFS record	not applicable	check byte

Protection of IFS file	not applicable	check record

Table 3: Error detection in LIN and TTP/A

5.2 Dependability

Both protocols, LIN and TTP/A contain a number of comparable mechanisms for error detection, as shown in Table 3. TTP/A includes a parity bit in every byte, whereas LIN does not require such a parity bit. While LIN protects each data frame with a check byte, TTP/A offers the flexibility of byte-protected, nibble-protected, or unprotected data fields. It is assumed that the application will provide an application-specific end-to-end protection for data fields that are not protected by the protocol. There is also a significant difference in the protection of the command frame: LIN protects the command frame by a two-bit check field, while TTP/A provides a five-bit check field.

The IFS design of TTP/A contains enough redundancy to correct single bit errors in an IFS file by a periodic background file check. Such a period background check that corrects single bit errors is very effective for the increase of reliability with respect to SEUs (single-event upsets).

The TTP/A standard provides recommendations for in-line error codes. Optional confidence markers are proposed in order that a sensor can express its confidence in an observed value. No such mechanisms are mentioned in the LIN standard.

5.3 On-line Diagnostics

The most marked differences between LIN and TTP/A are in the areas of on-line diagnostics and plug-and-play capability.

The topic of smart-sensor diagnostics and maintenance is not addressed in the LIN standard. The limited identifier name space of LIN (64 different identifier within a cluster) and the inability to directly address a physical slave node make it difficult for LIN to access data structures within a node for the purpose of diagnostics and maintenance.

The design of TTP/A, particularly the interface file system IFS and the addressing schema of TTP/A, has been driven by the objective to provide an effective framework for on-line diagnostics of smart sensors while the system is in operation. In combination with the dynamic real-time channel service of TTP/C it is possible to monitor from a service access point the internals of any TTP/A sensor connected to a TTP/C system in real-time, while the system is fully operational (Figure 2). For example, with TTP/A it is possible to look into every smart sensor of a car (e.g., via the telephone from a maintenance shop) while the car is operating on the road.

5.4 "Hot Plug-and-Play" Capability

The "hot plug-and-play" capability, i.e., the dynamic insertion of new nodes and the on-line reconfiguration of nodes are not addressed in the LIN standard.

TTP/A contains a number of mechanisms that support the "plug-and-play" capability.

Naming schema design: A TTP/A node has a universally unique physical name that refers to the physical identity of the node. The physical name is formed by the concatenation of a four-byte series number and a four-byte serial number. The series number refers to the node-type, while the serial number uniquely identifies each node within its type. The role of a node within a TTP/A cluster is identified by a cluster-unique one-byte logical name. From the point of view of the namespace, a cluster can thus comprise of up to 256 nodes. During normal operation, a node is addressed by its one-byte logical name. The assignment of the logical name to the physical name is called the "baptism" of the node. This assignment can either be done *a priori* (then no "baptize command" has to be implemented in the slave node), after power up, or dynamically during the operation of the system (by executing the "baptize" command across the CP interface).

Dynamic configuration: If a set of (unknown) nodes is connected to a TTP/A bus, then a TTP/A configuration agent can detect the physical names of the newly connected nodes by executing a binary search algorithm (provided this optional functionality of the baptize command has been implemented in the node). Since the four byte series number characterizes the node type, the configuration agent can then access a documentation data-base (which can be anywhere in the INTERNET) to fetch the detailed technical documentation for this node type. In the next phase the configuration agent can determine the roles of the nodes and assign a logical name to each one of the newly attached nodes. After a logical name has been assigned, the appropriate RODL files can be downloaded to each node from the configuration agent and the multi-partner rounds of the real-time service can be started. It is important to note that all these configuration activities are performed via the DM and CP service using the master-slave dialogue. Since the master-slave dialogues coexist with the real-time service, no unintended side effect to the ongoing real-time service of the cluster will occur.

Recommendations for data encoding: to further enhance the "hot plug-and-play" capability, the TTP/A standard contains recommendations for encoding the measured values. These recommendations also contain a proposal for in-band error codes (that replace a sensor value in case of an error) and out-of-band confidence markers that inform the receiver about the confidence the sender has in its delivered value. Confidence markers are important if replicated sensor or different sensing methods are used in a fault-tolerant application.

5.5 Physical Layer

The following Table 4 denotes the physical layers of LIN and TTP/A:

Transmission speed up to	LIN	TTP/A
20 kbits/second	ISO 9141 (ISO-K)	ISO 9141 (ISO-K)
1 Mbit/second	not specified	RS 485 or CAN
above 1 Mbit/second	not specified	fiber optics

Table 4: Transmission speed of LIN and TTP/A

Both protocols support the sleep and wakeup function on the ISO 9141 physical layer.

5.6 Start-up Synchronization and Resynchronization

The start-up synchronization in LIN for *non-time-aware* slave nodes occurs before each master-slave round by sending a "break" byte and a synchronization byte. Since start-up synchronization occurs before every master-slave round, no mechanism for the resynchronization of already integrated nodes is required.

The start-up synchronization in TTP/A for *non-time-aware* slave nodes is performed during a special two-byte master-slave synchronization round that contains a defined synchronization pattern (see Fig. 5). Such a two-byte synchronization round can be either inserted before every multipartner round, before every master-slave round, or instead of some master-slave round, depending on the quality of the available resonators. For already integrated slave nodes resynchronisation is performed at the instance when a byte from a node with a precise clock arrives, e.g., the fireworks byte.. In a multi-partner round those bytes that originate from a node with a precise clock are marked in the RODL. The slave can thus consider the arrival instants of these bytes as precise time-references. The first byte of every multipartner round, the "fireworks" byte is always marked (since it originates at the master with a precise clock). In master-slave rounds, the arrival instants of the first five bytes (sent by the master) are used as precise synchronization events.

5.7 Complexity and Cost

Figure 5 depicts the simplest messages in LIN and TTP/A. The simplest message in LIN consists of a break, a synchronization byte, an identifier byte, a variable interframe gap, two data bytes and a check byte (Figure 5a). In TTP/A a comparable minimal message consists of one-byte silence, a synchronization byte, one-byte silence, an execute-RODL command (fireworks byte), and at least one data byte (Figure 5b). If a node does not require start-up synchronization, then the three-byte sequence of Figure 5c applies. Of course, in a TTP/A system, the data byte can be in any timeslot of the round, if so specified.

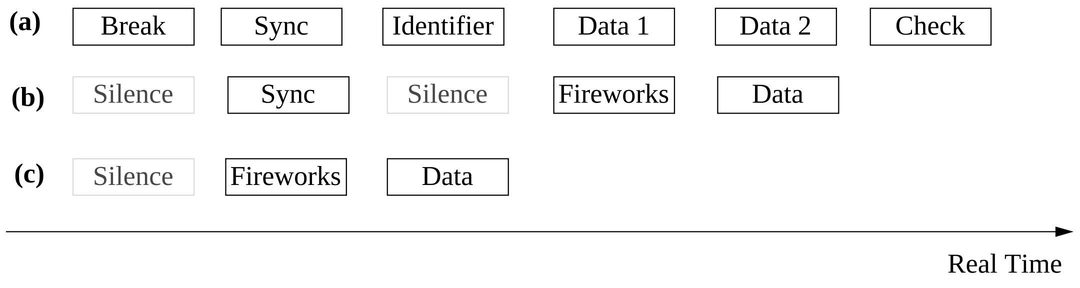

Figure 5: Byte Sequence of the simplest message in LIN (a), in TTP/A with start-up synchronization (b) and in TTP/A without start-up synchronization (c).

In the simplest TTP/A implementation all parameters of the round (e.g., the fireworks name, the position of the byte in the round) and the meaning of the data byte must be set *a priori,* i.e., outside the protocol. Such an TTP/A implementation does not support the interface file system. Similarly in LIN, the contents of the identifier and the meaning of the data must be set outside the protocol. The minimal implementation of LIN and TTP/A are simple enough for implementation by a state machine.

We do not have any experimental data on the size of LIN implementations.

The TTP/A protocol, including the interface file system, has been implemented on a number of low-cost single-chip microcontrollers, e.g., the ATMEL AT90S2313 (with 2 kbytes of flash memory and 64 bytes of RAM), the MICROCHIP PIC16F877 and the MOTOROLA 68HC711P2. These implementations, including the logic of the interface file system (IFS) and a software UART occupy between 1k and 2 k bytes of ROM and less than 64 bytes of RAM storage, depending on the instruction set of the particular processor. This memory requirement does not include the file space itself, which is application specific. The memory footprint of a low-cost TTP/A that contains all protocol information in ROM and just supports one RODL, as outlined in Figure 5c, requires a few hundred bytes of ROM, depending on the architecture of the processor.

6. CONCLUSIONS

Although both protocols, LIN and TTP/A, address the emerging smart sensor market and have a number of similar characteristics, they contain significantly different functionality. LIN focuses on the basic functions needed to communicate in real-time with a smart sensor. To simplify the sensor implementation, LIN restricts the name space and does not address the issues related to remote diagnosis of smart sensor nodes. In addition to providing efficient real-time communication services, the design of TTP/A has been driven by the objective to establish an architectural framework for on-line diagnosis of smart sensors and plug-and-play capability. This results in a more "function-rich" specification than LIN.

Considering the same bandwidth on the physical channel, the high data-efficiency of the multipartner-round concept of TTP/A results in a better real-time response compared to LIN (see Table 2). However in the area of real-time performance, the main difference between LIN and TTP/A is in the area of protocol jitter. The LIN protocol specification allows a variable interframe gap that can lead to a significant jitter at the end-to-end level. TTP/A requires a constant interfame gap in order the minimize jitter. As a consequence, the jitter in a LIN application is bounded by duration of 40% of a transmission round, while the jitter in TTP/A is less than 1% of the duration of a round.

Both protocols, LIN and TTP/A contain a number of comparable mechanisms for error detection, as shown in Table 3. While TTP/A includes a parity bit in every byte, LIN does not require such a parity bit. There is a significant difference in the protection of the command frame: LIN protects the command frame by a two-bit check field, while TTP/A provides a five-bit check field.

Both, the LIN and the TTP/A specification allow the implementation of simple nodes that do not support the full functionality of the respective protocols. It is thus possible to build low-cost smart sensors with restricted functionality that implement the protocol logic in a state machine. The TTP/A specification supports also the design of generic nodes with the TTP/A logic in ROM memory. It is then possible to configure such a generic node to the given application requirements after system power-up, or even to reconfigure a node during system operation. LIN supports no such functionality

ACKNOWLEDGMENTS

We would like to thank H. Wense, editor of the LIN consortium for constructive comments on an earlier version of this paper and the calculation of the response time of LIN for the scenario depicted in Table 2. This work was supported, in part by the Austrian Ministry of Science, project TTSB and by the European IST project DSOS. The constructive discussions about TTP/A within the TTSB project, in particular with Prof. Färber and his group at the TU Munich and Prof. Göhner and his group at the University of Stuttgart, are warmly acknowledged.

REFERENCES

Deirauer, P. nd B. Woolever (1998). Understanding Smart Devices. *Industrial Computing*. Vol. pp. 47-50.

IEEE (1997). IEEE Standard 1451--A Smart Transducer Interface for Sensors and Actuators,. *IEEE Press*.

ISO89 (1989). International Standard ISO 9141. Geneva, Switzerland. International Organisation for Standardization.

Kopetz, H. (1995). TTP/A -- A Time-Triggered Protocol of Body Electronics Using Standard UARTS. *Proc. SAE World Congress*, Society of Automotive Engineers, SAE Technical Paper 950039. pp. 1-9.

Kopetz, H. (1997). *Real-Time Systems, Design Principles for Distributed Embedded Applications; ISBN: 0-7923-9894-7, Third printing 1999*. Boston. Kluwer Academic Publishers.

Kopetz, H., M. Holzmann, et al. (2000). A Universal Smart Transducer Interface: TTP/A. *Object-Oriented Real-Time Distributed Computing*, Newport Beach, Cal. IEEE Press. pp. 16-23.

LIN (2000). LIN Specification and LIN Press Announcement, SAE World Congress Detroit. www.lin-subbus.org

Nesett, D. (1999). Massively Distributed Systems: Design Issues and Challenges. *Embedded Systems Workshop*, Cambridge, Mass. USENIX Association.

TTP/A (2000). TTP/A Specification of March 30, 2000. www.ttpforum.org

TTTech (2000) WebSite of Time-Triggered Technology www.tttech.org

Wense, H. (2000) Introduction to Local Interconnect Network, SAE World Congress, Detroit, March 2000, Document No. 2000-01-0145, SAE Press