A Universal Smart Transducer Interface: TTP/A

H. Kopetz
M. Holzmann
W. Elmenreich
hk@vmars.tuwien.ac.at

Insitut für Technische Informatik
Technische Universität Wien
Austria

Abstract

The primary goal of a universal smart transducer interface is the provision of a framework that helps to reduce the complexity of large distributed real-time systems by introducing precisely specified (in the value domain and in the temporal domain) and small interfaces between smart transducers and their users. This paper presents a universal smart transducer interface that can be implemented on top of different real-time communication systems. It integrates a time-triggered communication protocol with an interface file system that provides the sources and sinks for the exchanged information. The final section discusses an implementation of this interface on a low cost (less than 1 $) commercial off the shelf microcontroller.

1. Introduction

The point-to-point connection of process input/output devices to a control system is expensive, both from the installation point of view and from the engineering point of view. One approach to reduce these costs is the introduction of the emerging smart transducer technology. A smart transducer is an intelligent subsystem consisting of a sensing or actuating device (sometimes already an integrated device on a silicon die), a micro-controller with the necessary software, and a communication network interface (CNI) to a field bus. A properly designed CNI of a smart transducer node should present a standardized high-level view of the sensor and encapsulate the idiosyncrasies of the particular sensing element within the smart transducer node.

Such a smart-transducer CNI should be understandable, temporally predictable and implementable in available low-cost microcontrollers. To improve the understandability, the externally visible interface of a smart transducer should be designed around few already familiar concepts. The transducer interface must be generic, i.e., the interfaces of most of the available transducer should be expressible within the model. Ideally, the same basic concepts--and as a consequence the same application software at the user's side--should suffice to communicate with the majority of the available sensing and actuating devices. Since the bandwidth and response time requirement of various transducers may differ by orders of magnitude, the model should be flexible to accommodate different communication speeds and different media access protocols, e.g., a simple single wire UART channel as well as a high-speed fiberoptic channel.

Since more than ten years it has been recognized that a standardized real-time communication network, a fieldbus, to replace and enhance the existing 4-20 mA analog signal standard would be an enabling technology beneficial to the industrial instrumentation business as a whole. However, many vendors were reluctant to support such a single common standard in fear of losing some of their competitive advantages. As a consequence, a number of different mostly incompatible fieldbus solutions (see [1]) have been developed and promoted. In 1992 the two large rival fieldbus groups ISP (Interoperable Systems Project supported by Fisher-Rosemount, Siemens, Yokogawa, and others) and the WorldFIP (supported by Honeywell, Bailey, and others) introduced two competing interim fieldbus standards. In 1994 these two rival groups merged to form the Fieldbus Foundation (FF). It is the stated objective of the FF to develop a single interoperable fieldbus standard in cooperation with the IEC (International Standard Organization) and the ISA (Instrumentation Society of America). This new fieldbus standard IEC/ISA SP 50 should integrate different types of control instruments and support, as far as possible, existing interfaces. In the mean-time the Control Area Network CAN was developed by the automotive industry to reduce the wiring harness within a car. From the functional point of view, the CAN bus can deliver a communication service that is closely related to that of the field bus,

although with limited temporal predictability. The immense size of the automotive market has led to the appearance of low-cost highly integrated CAN chips that are being used by a number of companies in the factory and process automation market. Many of the cited efforts to create a standardized field-bus have focused on the issues of reliable communication and wiring, but have neglected the higher level issues that must be addressed if interoperability or interchangeability of devices and subsystems must be achieved.

It is the objective of this paper to present a generic interface of a smart transducer that can be used to connect many different concrete sensor and actuator subsystems within the same conceptional framework and can be implemented on diverse communication systems. The rest of the paper is organized as follows: Section 2 explores the abstract properties of an interface and discusses why the 4-20 mA current loop was highly successful. Section 3 presents the generic TTP/A protocol [3], a temporally predictable field-bus protocol that can be implemented on different physical layers. Section 4 discusses the three shared code spaces that must be provided to enable an information exchange across an interface: a common name space, provided by an interface file system (IFS), a time space, and a value space. Section 5 presents an implementation of this interface on a low-cost microcontroller and an UART bus. The paper ends with the conclusions in Section 6.

2. What is an Interface?

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 smallest area of concern is a cluster, consisting of a set of sensor, actuator, and processing nodes connected by a broadbus communication medium. The set of all nodes of the cluster must thus share the same concepts and must agree on common code spaces.

For example, the interface between a driver and a car for the purpose of braking is the brake pedal. There are two relevant state variables associated with the brake pedal at this interface: the pressure applied to the brake pedal by the driver and the tactile feedback, the resistance, provided by the brake pedal back to the driver. The position of the brake pedal in the car identifies these state variables uniquely to both interfacing subsystems. The temporal association between sending the information (e.g., by the driver) and receiving the information (e.g., by the car) is implicit, because of the mechanical connection of the two subsystems. The value space of the state variables is the domain of pressures that can be applied. In some braking system, the temporal sequence of states, i.e., the speed with which the brake pedal is pressed by the driver, is another important input. If a driver starts to press the brake pedal quickly, the braking system assumes that she/he intends to brake hard. In an electronic braking system this information is sometimes used to initiate the appropriate braking action before the full pressure is applied to the pedal, thus gaining a few valuable milliseconds in the response time of the braking system. In hydraulic braking systems, the tactile feedback is a sideproduct of the hydraulic mechanisms. In an electronic "brake-by-wire" system, the force of the tactile feedback must be calculated and actuated by an actuator node in order to give the driver the relevant tactile feedback about the consequences of the braking action according an accustomed, sometimes non-cognitive, model of the brake in the mind of the driver.

2.1 Observations

In the abstract, the purpose of a smart sensor interface is the timely exchange of "observations" of real-time entities between the engaged subsystems across the provided interfaces. An RT entity is a state variable of interest that has a name and a value. An observation [3] 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 point on the "time space" 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. Communication across an interface is only possible, if the code spaces for names, instants, and values and the referenced concepts are shared by all engaged subsystems. One key task in the development of a generic smart transducer model is concerned with the specification of these code spaces and the meaning of the referenced elements.

If the value of an observation does not change over the time interval of interest, we call this observation a *timeless observation.* Timeless observations can be represented by tuples, consisting only of an entity name and the associated value.

2.2 Why was the 4-20 mA Current Loop Interface so Successful?

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 RT entities is formed by the set of interfacing wires, each wire denoting one particular RT entity. 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. 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 standard realizes a multicast topology by supporting the installation of many receivers in the current loop. Such a multicast topology is needed in many automation systems, where an observed signal must be distributed to a number of independent receivers, e.g., an operator display, a controller, and a computer system interface.

2.3 Smart Sensors

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 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 microelectronic 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 [4]:

(i) Electrically weak non-linear sensor signals that originate from an MEMS sensor can be generated, conditioned, transformed into digital form, and calibrated on a single silicon die without any noise pickup from long external signal transmission lines [2].

(ii) It is possible to locally monitor the sensor operation and thus simplify the diagnostics. 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 its own documentation on silicon.

(iv) The internal complexity of the smart-sensor hardware and software and the internal sensor failure modes can be hidden from the user by a well-designed fully specified smart sensor interface that provides just those services that the user is interested in. Thus, 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. Furthermore, information about sensor performance and diagnostic information must be stored in the sensor and accessed during maintenance. A generic smart sensor interface must thus provide a standardized name space for all these data elements.

3. The Generic TTP/A Protocol

The generic TTP/A communication protocol is a time-triggered protocol for the communication among smart transducer nodes within a cluster. It is controlled by an active master who establishes the common time base within the cluster. In case the master fails, a secondary master can take over control. In TTP/A it is assumed that every node has a unique personal identification number, e.g., an eight byte integer that is used to assign a (short) logical name to the node after power up. The scope of the logical name is a single cluster.

In TTP/A the communication is organized into *rounds*. A round consists of one or more frames. Between any two frames there is an *interframe gap*, the length of which is an implementation specific parameter. A frame is a sequence of bytes transmitted by a single node. From the point of view of the protocol, each round is independent of all other rounds. Any two rounds are separated by at least an *interround gap* that is significantly longer than the interframe gap. To simplify startup and recovery, there is no protocol state stored between rounds. Every round has a name, the *round name*, that identifies the round. The structure and duration of every round is static and specified *a priori*, i.e., it is common knowledge to all nodes of a cluster.

A round starts with a special frame from the master, the *fireworks frame* that has characteristic features. The arrival of the fireworks frame from the master is a synchronization event that starts a common time-base for this round in each node. The fireworks frame contains the round name and can carry additional data. According to the specification of the selected round, the fireworks frame is followed by *data frames* of specified lengths from the specified nodes.

TTP/A distinguishes between three types of rounds

(i) A *broadcast round* that just consists of the fireworks frame sent by the active master.

(ii) A *master-slave round* that consists of two frames, the fireworks frame from the master containing, among others, the address of a slave and a response frame from the slave. The main purpose of a master

slave round is the reading and writing of data into a file of the distributed TTP/A file system.

(iii) A *multipartner round* that starts with the fireworks frame from the master and continues with data frames from the specified nodes. Multipartner rounds are periodic and are used to update real-time images.

The specification of a multipartner round is called a round descriptor list (RODL). A RODL can be viewed as a set of related files, one in each participating node, that identify the file address of the exchanged data and specify the point in time (relative to the start of the round) when this node has to become active (read or write data or perform some action).

During the normal TTP/A operation there is a regular sequence of multipartner rounds and master-slave (or broadcast) rounds (Figure 1). The periodic multipartner rounds exchange current real-time observations, while the sporadic master-slave rounds access a TTP/A file, if required.

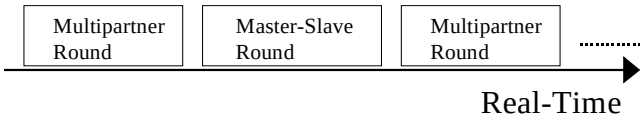

Figure 1: Traffic on the TTP/A Bus

The TTP/A master contains at least two interfaces, one to the TTP/A bus and the other to the environment of the sensor subsystem. In the time-triggered architecture, this latter interface conforms to the communication network interface (CNI) of the TTP/C specification [5], such that the same access mechanisms (software tools) can be used to access sensor data from the local node and sensor data from a remote node.

4. The Shared Code Spaces

Communication is only possible, if the interfacing subsystems share the concepts and representation of the data items in the interface. To exchange observations across a smart sensor interface, agreement on the name space, the time space, and the value domains must be provided.

4.1 The Name Space

The TTP/A protocol integrates the communication between nodes and the storage of the communicated data within each node by providing a distributed interface file system (IFS) of a cluster that acts as a source and sink of the data exchanged among the nodes. This distributed file system is the collection of the local interface file systems of each node. The IFS supplies the shared and structured name space for the data-elements that are exchanged among the nodes of the cluster. It provides the stable intermediate structure that serves as the basis for the design of higher-level protocols that assign meanings to the contents of IFS file records. This meaning can be assigned either formally or informally by a documentation file. For example, a documentation file with informal information may contain (in verbal form) that

> "this sensor is a temperature sensor that provides the current temperature in record *a1* of file *f1*. A lower alarm limit has to be written into record *a2* . Diagnostic information about the sensor can be found in file *f2*. ".

In the future we plan to formalize this information to support a "plug-and-play" capability of TTP/A sensors.

File Structure

The IFS is structured into a set of index-sequential files with constant record length and a static file structure. The address of any record in the IFS of a cluster is composed of the following fields

<file name><node name><record number>

The record is the smallest unit that can be addressed in an IFS. Since all records of a file have the same length, a file can be viewed as a matrix of bytes. The last byte of every record is a horizontal check byte. The last record of every file is the vertical check record. The file system thus contains enough redundancy to correct a single error in the file. This characteristic is used to perform periodic file checks that monitor the integrity of the files and, if desired, correct single bit flips. These periodic file checks improve the reliability of the file system significantly. The horizontal check byte is also used when a record is transmitted in a frame.

File Operations and File Types

There are three file operations defined in the IFS: *read*, *write*, and *execute*. *Read* reads a record from a file, *write* writes a record into a file, and *execute* executes a file. The file operation code--*op-code* for short--can be coded into a two-bit field.

There are four different file types in the IFS: *read-only documentation files*, *input-output files*, *RODL files* and *special command files*. Read-only documentation files contain the documentation about a node. Input-output files are normally used to store observations and parameters. The RODL files contain the RODLs and special command files contain executable program modules that can be executed (e.g., a JAVA applet). The transmitted data of a round form the input parameters for such an execution.

The following Table 1 indicates which operations may be performed on which file type:

Table 1: IFS File Types and File Operations

File type / Operation	Read	Write	Execute
Documentation file	X		
Input/Output file	X	X	X
RODL File	X	X	X
Command file	X	X	X

The TTP/A file system guarantees that any single *read* or *write* operation of a file record is atomic. This implies that after a file *write* the horizontal check byte and the check record must be updated atomically. If a user needs a level of atomicity beyond the single file record, he/she has to design his/her own concurrency control protocol, for example an NBW protocol [6] that uses one record as a concurrency control field.

RODL Files

A RODL file is a distributed file that contains the specification of a RODL for a particular round. A RODL file consists of a collection of (sub)files, one in each participating node. The RODL file name is also the round name. The master can initiate the execution of a RODL by naming, within the fireworks frame, the appropriate RODL file name in the file execute command.

A record of a RODL file has the following structure

<round position><op-code><file-name><length><file record address>

The *round position* tells the node at what position in the round an action (*read*,*write*, or *execute*) is required. The *op-code* field specifies the action. The *file-name* field identifies the file that is involved in the action. The length field specifies the length of the data frame and the file-record address specifies, in case of a read or write action, which record is involved in the action.

Since the individual RODL subfiles can be addressed and manipulated just as any other files, the programming of a new RODL can be performed with the available file operations without having to introduce any new concepts or mechanisms.

4.2 The Time Space

In TTP/A, the point of occurrence of an event is recorded by recording the state of the local clock at the instant of event occurrence. In order to economize the representation of the continuously flowing time, only an interval of time around "now", the current time, can be expressed in the slave-node local TTP/A time space. This is in agreement with the strategy of TTP/A to reduce the internal state of a slave as far as possible. In TTP/A, the epoch of the time scale starts anew with the arrival of every fireworks frame of a multipartner round. The fireworks frame also contains synchronizing information to be able to synchronize the rate of the clock of the receiving slave to the rate of the master. This is important if "on-chip" oscillators are in use at the slaves, since these "on-chip" oscillators have a bad long-term stability. The granularity of the time-scale is chosen such that the duration of a granule is an integer fraction of the physical second. It is thus possible to express time values by the fractions of seconds that expired in the commonly agreed epoch.

Since two succeeding epochs can be partially overlapping, it is necessary to provide a mechanism to identify which one of the overlapping epochs has been used for the time measurement. For this purpose, an *alternating time bit* that classifies each epoch as either an even epoch or an odd epoch is provided. By making this alternating time bit part of the time value it is clear whether the time-measurement refers to the last even epoch or the last odd epoch. The alternating time bit is part of the fireworks frame that initiates an epoch.

4.3 The Value Domain

It is recommended to use the encoding schemes proposed in this section for encoding the data values transmitted in TTP/A networks. The use of these value domains will improve interoperability. Since TTP/A is intended to be used in very small sensor nodes utmost care has been taken to design an *efficient* coding scheme. In addition to the code space for values, TTP/A provides a code space for in-band error messages and out-of-band confidence markers.

Confidence markers are introduced to give a smart sensor the capability to express its confidence [7] in a delivered value. In multi-sensor systems, where more than one sensor observe a physical quantity, the confidence marker can be manipulated on the basis of comparing multiple independent direct or indirect observations of the same physical quantity. The confidence marker is a half-byte that provides space for sixteen confidence classes ("0000" for highest confidence and "1111" for 'no confidence'). The confidence marker has to be transmitted out-of-band, since it is produced in addition (and not instead) of a data value.

The sequence of data items contained in a frame is called a message. TTP/A distinguishes between three types of messages, *restricted* messages, *unrestricted* and *free* messages. The restricted message reserves the codes binary '1111 0000" to '1111 1111' of the first byte for in-band error codes to support the transmission of error information within the data bytes. This implies that in a restricted message the first byte may not contain a data code value that is equal or larger than binary '1111 0000'.

An unrestricted message starts with a special first byte which contains in the first half byte the confidence code and in the second half byte an error code. If an error code is set, the confidence marker in the first half-byte must be '1111' (meaning no confidence). If the confidence marker has any other value than '1111' the error code must be '0000' (meaning no error). All other bytes of an unrestricted message are application specific.

There are no rules that restrict the data coding of free messages. There is no in-band error code in free messages. The detailed error codes and data formats can be found in the TTP/A specification document.

5. UART TTP/A Implementation

In this section we describe a concrete implementation of the generic TTP/A protocol on a low-cost (cost of less than 1 US $) 8 bit microcontroller that uses an industry standard UART communication channel. Since the 8 bit microcontroller implementation of TTP/A should be executable on very small microcontrollers, the design of an efficient coding schema for the name spaces is of importance.

5.1 Communication System

As described in Section 3, the communication of TTP/A is organized into rounds. Every round starts with the fireworks frame from the master. The fireworks frame of a multipartner round consists of two UART bytes, the fireworks frame of a master-slave round consists of four UART bytes. The temporal distance between the first and the second byte in the fireworks frame is significantly longer than the temporal distance between the other bytes of a round, in order to generate a characteristic feature of the fireworks frame in the temporal domain. In order to produce a characteristic feature in the value domain, the first byte of the fireworks frame has even parity, while all other bytes of the round (except the synchronization pattern, see Section 5.2) have odd parity.

For the byte oriented data transmission, a standard UART format with the following characteristics has been chosen (for all bytes other than the first byte of the fireworks frame):

One start bit, 8 data bits, one parity bit and one stop bit, odd parity. Between two consecutive UART bytes there is an inter frame gap (IFG) of 1 bit cell. A frame and the following IFG form a 12 bit cell long slot (11 bit UART frame + 1 bit IFG). Thus a message of one byte is transmitted in a slot of 12 bitcells.

The selected bus interface conforms to the ISO K Line serial link bus interface (ISO-9141). The physical layer is a single-wire UART channel with a nominal bit rate of 10 kbit/s. Higher UART speeds require a different physical layer.

5.2 Synchronization

Start Up Synchronization: To support low-cost microcontrollers with imprecise on-chip or R/C oscillators, a start up synchronization is required after the cold start of a node. Start up synchronization means adjustment of the speed of the local clock of such nodes, to enable UART communication. Differences up to 50% from nominal frequency can be compensated. The TTP/A master can be configured to send synchronization patterns to enable nodes with an imprecise oscillator to adjust the speed of their local clocks. For a synchronized node, the synchronization pattern reads as binary '0110 0110' with even parity. The synchronization pattern is unique, because the value binary '0110 0110' is forbidden in the first byte of a fireworks frame.

Continuous Clock Synchronization: The local clocks of the nodes are re-synchronized with the reception of the first two bytes of the fireworks frame at the beginning of each round. The state correction of the clock is performed at the instant of reception of the second byte of the fireworks frame: every node sets its local clock to zero. Rate correction is done by measuring the interval between the instants of reception of the first two UART bytes of the fireworks frame. The master sends these two bytes with an *a priori* known temporal distance, providing information about its local clock to the slaves.

5.3 Namespaces

We have tried to map all names into single or multiple byte objects, such that these objects can be communicated efficiently by an UART protocol and stored in a single byte memory location of an 8-bit microcontroller architecture.

File-operation: The generic TTP/A protocol requires 3 file operations: *read*, *write*, and *execute*. The operation code is assigned to the first 2 bits of one byte.

File-name: It has been decided to provide a name space for filenames of 64 interface files in each node. This requires a six-bit name field. The byte that carries the File-operation code can also contain this six bit file-name field, such that a single byte contains file operation and file name.

Node-name: It has been decided to provide a name-space for 256 different nodes in a cluster. The node name can thus be represented in a single byte.

Record-number: It has been decided to limit an IFS file to 256 records. The record number can thus be presented in a single byte as well.

Record identification It is thus possible to express an operation on any record within a cluster with three bytes: First byte: file operation/file name, Second byte: record number, Third byte: node-name. If an operation is directed to a local file of a node, a two-byte record identification is sufficient (the node name is implicit).

RODL entry: A RODL entry contains 4 bytes: one byte for the identification of the round position of the data, one byte for the op-code and file name, one byte for the frame length and one byte for the record number in the selected local file.

Record length: A constant record length of 4 bytes has been chosen for the IFS files

Fireworks frame: The fireworks frame of a multipartner round consists of two bytes: the first byte contains the file operation/file name of the selected RODL file and the second byte contains an exclusive-or checksum over the first byte. The fireworks frame of a master-slave round consists of four bytes: the first three bytes contain a *record identification*, the fourth byte contains an exclusive-or checksum over the first three bytes.

File write operation: A file write operation consists of 9 bytes: four bytes operation code, identification of the record and exclusive-or checksum over the first three bytes, 4 bytes data, and one byte horizontal check field containing an exclusive-or checksum over the 4 data bytes.

File read operation: A file read operation consists of two frames with 9 bytes in total. The first frame consisting of four bytes identifies the data, the second frame contains four data bytes and the final byte is the horizontal check field.

5.4 UART TTP/A Implementation Experience

Figure 2 depicts a typical TTP configuration. The five nodes A,B,C,D, and E are TTP/C nodes that construct the fault-tolerant global time base of a TTP/C cluster. The nodes D and E are the masters of the two TTP/A buses. The implementation of the TTP/A protocol requires two parts, a master and a slave part. The master part of the protocol was implemented on a Motorola MC68376 on TTTech's TTP-Node-boards (Node E and node F of Fig. 2). In addition to the TTP/A interface, these TTP-Node-boards contain a communication network interface to TTP/C, such that the global time established in the TTP/C cluster can be used as a synchronized timesource for both TTP/A clusters.

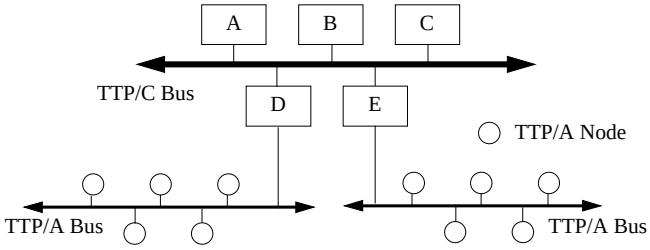

Figure 2: TTP/C and two TTP/A clusters.

The slave part of the protocol was implemented on the ATMEL AT90S2313, an 8 bit RISC microcontroller. The ATMEL AT90S2313 has 2K flash memory for program storage, 128 bytes EEPROM, 32x8 bit internal registers and additional 128 bytes of SRAM. No external memory was necessary to build up the node. The AT90S2313 does not provide an internal RC oscillator, but the algorithm for start up synchronization was implemented for evaluation purposes. The UART was implemented in software.

Code Size: The code for the protocol takes about 500 instructions, whereof 75 instructions are used for the UART implementation. An instruction needs 2 byte of flash memory, so 1 K of flash memory is used for protocol code, 1 K remains for application code and the storage of IFS files. The RODLs are located in SRAM. Initialization parameters like node identifier and standard RODLs are located in EEPROM. I/O Files are mapped to sensor/actuator values but could also be buffered in RAM.

Performance: The performance of a TTP/A system depends on the number of nodes in the cluster, the length of messages and the speed of the bus. Normally the round sequence as shown in Figure 1 is executed. The periodic multipartner round is used to transmit the newest version of the real-time data. The response time for the RT data can be guaranteed. The sporadic master slave rounds between the multi-partner rounds are normally used to read and write to the IFS files in a cluster.

With the current implementation we have achieved a response time of 31 msec for the real-time data on the 10 kbit/sec UART bus, assuming 4 nodes which have two data bytes in each frame. Within this 28 msec interval there is also space for one master-slave round of 9 bytes between two subsequent multipartner rounds.

Table 1 shows the calculated response times in milliseconds for different TTA configuration, assuming a two frames with 1 byte each for the RT data of each node.

Table 2: Response Times for Different TTP/A configurations

Speed \ nodes	4	8	16	32
9.6 kbits/sec	28	38	58	98
100 kbits/sec	2.7	3.6	5.5	9.4
500 kbits/sec	0.53	0.72	1.11	1.88

In order to support the higher transmission speeds, a more powerful microcontroller with an hardware UART interface is required.

6. Conclusions

The low-level programming of the input/output routines for the diverse I/O devices is cumbersome and error prone. The specification of a universal smart transducer interfaces makes it possible to hide the idiosyncrasies of the various I/O devices behind such an interface and to provide to the programmer a unified view of the I/O devices. Such a universal smart transducer interface must guarantee predictable real-time performance and provide the flexibility to communicate all types of data among smart transducer devices. In this paper we have proposed to integrate the communication services with an interface file system (IFS) to provide such a standardized interface.

The presented interface file systems (IFS) provides a common name space for the data items that are exchanged among the transducer nodes and a master node in a distributed real time system. It establishes a stable intermediate structure that is a solid base for many new services. In the future we plan to built higher level services by the standardization and automatic interpretation of the contents of IFS documentation files.

The implementation of the IFS on a very small microcontroller has shown that it is economically feasible to assign local intelligence even to low-cost I/O devices and thus establish the foundation for a new effective style of I/O programming.

Acknowledgments

This work was supported, in part by the Austrian Ministry of Science, project TTSB and by the European IST project DSOS.

References

[1] AB, P. A. (1999). Catching the Bus. http://www.iol.ie/readout/~fieldbus.

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

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

[4] Kopetz, H. (1999).Do Current Technology Trends Enforce a Paradigm Shift in the Industrial Automation Market? Closing Keynote at the 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 99), Barcelona, Spain, October 18-22, 1999.

[5] Kopetz, H. (1999). Specification of the TTP/C Protocol. TTTech, A 1040 Wien, Schönbrunnerstraße 13.

[6] Kopetz, H. and J. Reisinger (1993). The Non-Blocking Write Protocol NBW: A Solution to a Real-Time Synchronisation Problem. *Proc. 14th Real-Time Systems Symposium*, Raleigh-Durham, North Carolina. pp.

[7] Parhami, B. (1991). A Data-Driven Dependability Assurance Scheme with Applications to Data and Design Diversity. *Dependable Computing for Critical Applications* A. Avizienis and J. C. Laprie Ed. Vienna. Springer Verlag. pp. 257-282.