The Design of Large Real-Time Systems: The Time-Triggered Approach

H. Kopetz M. Braun C. Ebner A. Krueger
D. Millinger R. Nossal A. Schedl

Institut fuer Technische Informatik
Technische Universitaet Wien, Austria
email: hk@vmars.tuwien.ac.at

Abstract

The time-triggered(TT) architecture approach supports the spatial partitioning of a large, distributed real-time system into a set of autonomous subsystems with small control-free data-sharing interfaces between them. This paper presents such a TT architecture and gives a detailed description of the interface between an autonomous time-triggered communication subsystem based on the TTP protocol and the host computer within a node of this architecture. This interface acts as a temporal firewall that eliminates the possibility of control error propagation from one subsystem to another subsystem. It thus facilitates the independent development and validation of the subsystems and supports the composability of the distributed architecture with respect to timeliness, validation, and certification.

1 Introduction

The dramatic improvements in the cost/performance ratio of microelectronics devices over the past ten years have opened large new markets for the utilization of real-time computer technology. Many of these new applications, e.g., the computer control of core vehicle functions, such as engines, transmissions, and brakes in the mass market of automotive electronics have high dependability requirements.

The notion of dependability covers the nonfunctional attributes of a computer system that relate to the quality of service a system delivers over an extended period of time. In the context of hard real-time systems, all five measures of dependability, i.e., reliability, safety, availability, maintainability, and security [Laprie, 1992] are relevant. The most critical of these measures is safety: the probability that a system will not fail in a catastrophic failure mode. Safety is a system issue that depends on the proper operation of both, the hardware and the software. Whereas a number of techniques are known to mask the consequences of random hardware failures, the problems of avoiding, detecting, and handling design failures in the software (and possibly in the hardware) have not been solved satisfactorily up to now [Littlewood & Strigini, 1995].

Design failures have their origin in the unmanaged complexity of a design. Although there is no universally accepted measure of the complexity of a design available, there is general agreement [Rushby, 1993] that the following system attributes increase the complexity of a design:

- the requirement to meet strict deadlines,
- the system size, often expressed in the number of source code instructions,
- the coordination, synchronization and timing of concurrently executing tasks, and
- the error detection and redundancy management in fault-tolerant systems.

Many hard real-time applications excel in all four of these attributes. Whereas the first of these attributes, the requirement to meet strict deadlines, is fundamental to any hard real-time system and cannot be avoided, it is conceivable to devise an architecture that helps to effectively reduce the impact of the other three system characteristics on the complexity of a real-time application.

It is the objective of this paper to show how a properly designed time-triggered architecture will guarantee the deadlines and reduce the complexity of a real-time application caused by the system size, the coordination and synchronization of the concurrently executing tasks, and the redundancy management.

This paper is organized as follows. In the next section we present a model of a real-time system that supports the reasoning about the temporal accuracy of real-time data. Chapter three discusses the characteristics of time-triggered architectures, presents the fundamentals of the time-triggered protocol TTP, and explains how a time-triggered architecture reduces the complexity of the application software by partitioning a distributed application into a set of autonomous subsystems bounded by stable internal data sharing interfaces, the Message Base Interfaces (MBI). Section four describes the properties of the MBI in detail. Section five, the last section before the conclusion, is devoted to the discussion of implementation concerns.

2 A Real-Time System Model

2.1 System Structure

A real-time application can be decomposed into a set of subsystems, called *clusters*, e.g., a controlled object (i.e., the machine that is to be controlled), the controlling computer system, and the operator. The controlling computer system has to react to stimuli from the controlled object within an interval of real-time that is dictated by the dynamics of the controlled object.

We make the following assumptions:

(1) The controlling computer system (Fig. 1) consists of at least one set of self-contained computers, the *nodes*, which communicate via a Local Area Network using the TTP Protocol [Kopetz & Grünsteidl, 1994]. We call such a subsystem a *computational cluster*.

(2) If there is more than one computational cluster present, then the computational clusters communicate with each other via so-called *gateway nodes* that are logical members of both clusters.

(3) The nodes are *fail-silent* and messages which are mutilated by the communication system can be detected and discarded. Nodes have a deterministic behavior to support the implementation of fault tolerance. Node failures and communication failures can be masked by replicating the nodes and grouping them into Fault-Tolerant Units (FTU) and by replicating the message transmission in space and/or time.

(4) All clocks in the nodes are synchronized such that an approximate global time base of sufficiently small granularity is available in the every node of the distributed computer system [Kopetz & Ochsenreiter, 1987]

(5) Some nodes, the *interface nodes*, support a connection to the intelligent instrumentation, i.e., the sensors and actuators, which are at the interface to the controlled object.

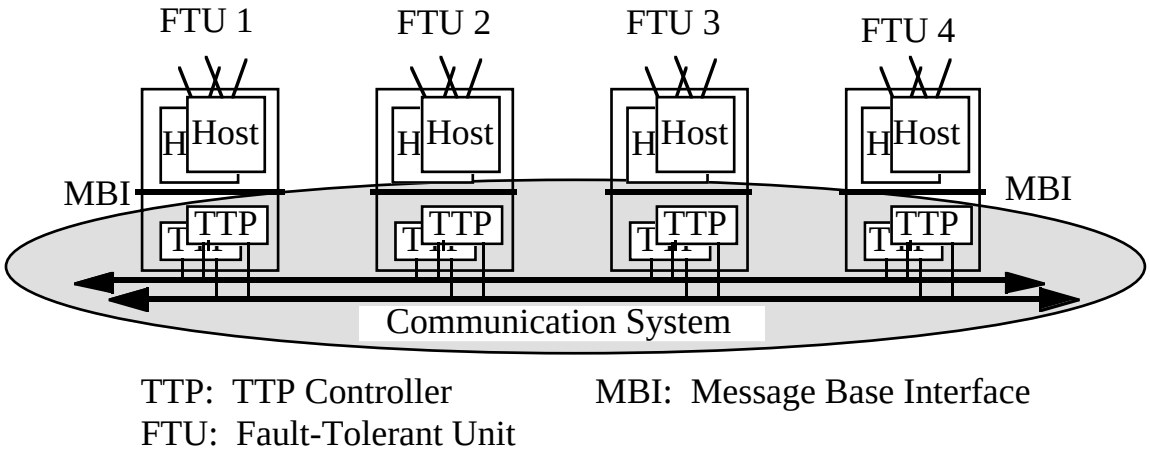


Fig. 1.: Structure of a Distributed Computer System

2.2 Sensors and Actuators

Sensors and actuators are needed to interface a real-time computer system to the world of the physical process via interface nodes. In the given model no interrupt signals are associated with the operation of a sensor or an actuator.

Sensors

We call a *physical sensor* with an associated microcontroller an *intelligent sensor*. It is the purpose of an intelligent sensor to encapsulate and hide the detailed technical interface of the physical sensor from the host computer and to provide a preprocessed abstract high level sensor reading to the host computer.

In our model the node interface to a sensor consists of an input data register, an optional time register, and a control line. The input data register contains a current value of a physical quantity. If a time register is provided then the contents of the time register denotes the point in time when the value in the input data register has been measured (we call this a *timed* input). Whenever the control-line is activated by the host, the contents of the input data register (and the time register) are transferred to the node computer. In a time-triggered system it is possible to synchronize statically the point in time when a sensor reads a state variable in the controlled object with the point in time when the interface node accesses the input data register in the sensor device.

Actuators

An actuator is a device that contains an output data register, an optional time register, a control line, and an energy source acting on an output device to achieve the desired effect in the environment. In case there is no time-register present, a new value written into the output data register causes the immediate release of energy from the energy source such

that the physical device causing the desired effect can start to act. If there is a time register, the energy source is activated at the point in time contained in the time register (we call this a *timed* output). The completion of the output operation will be delayed according to the known transfer function of the output device.

2.3 RT Entities and Observations

From the point of view of the control system designer, the behavior of a real-time application can be modeled by a set of computational processes operating on representations of *real-time (RT) entities*. A RT-entity [Kopetz & Kim, 1990] is a state variable of interest for the given purpose, either in the controlled object (an external RT-entity) or in the computer system (an internal RT-entity). A RT-entity has a time-dependent internal state. Examples of RT-entities are the temperature of a particular vessel (external RT-entity), the speed of a vehicle (external RT-entity), or the intended position of a specific control valve (internal RT-entity).

Every RT-entity is in the *sphere of control* of a subsystem (e.g., in the controlled object or in a particular node of a distributed computer system) that establishes the value of this RT-entity at any point in real-time. From outside its sphere of control, the state of an RT-entity can only be observed, but not modified. We call the atomic tuple

O = <entity name,value,point of observation>

an *observation* O(e,v,t) of an RT-entity e with value v at time t. The value v of an observation represents an estimation of the state of the RT-entity at time t.

Observations of the controlled object taken at a particular point in time form the basis of computer internal *RT-images* of the observed RT-entities that are transmitted to, manipulated by, and stored within *RT-objects* in the nodes of the computer system. This transportation and manipulation of the observations take a certain span of real-time, determined by the delay characteristics of the communication protocols and the processing speed within the nodes. Thus an RT-image that is available to the client of a RT-object in a remote node at a particular point in time t_{use} is already aged by this span of real-time. We call the maximum allowable length of this time-span, i.e., the difference between the latest point of use of an RT-image and the point in time that the RT-entity had this value (normally the point of observation), the *temporal accuracy* of the RT-image. An observation that is inaccurate, either in the domain of value or in the domain of time, can cause a failure in the service of the real-time computer system.

The patterns of change of an RT-entity, i.e., the dynamics of the RT-entity, determine the response time requirements on the computer system. If the response time of the computer system is much longer than the time between significant changes in the state of the RT-entity, then it is difficult to perform any real-time control of the controlled object. Real-time control is only feasible if the speed of the computer system matches the dynamics of the controlled object.

3 The Time-Triggered Design Approach

In a time-triggered architecture all system activities are initiated by the progression of a globally synchronized time. There is only one control signal in the system, the periodic clock signal, which partitions the continuum of time into a sequence of equidistant granules. It is assumed that all clocks are synchronized and every observation of the controlled object is timestamped with this synchronized time. The granularity of the

global time must be chosen such that the temporal order of any two observations taken anywhere in a distributed time triggered system can be reestablished from their time stamps [Kopetz, 1992].

The time-triggered paradigm of a real-time system is based on a distinctive view of the world: the observer (the computer system) is not driven by the events that happen in its environment, but decides autonomously when to look at the world. It is therefore impossible to overload a time-triggered observer!

A TT system takes a snapshot of the world, an observation, at recurring predetermined points in time determined by the current value of its synchronized local clock. This snapshot is disseminated within the computer system by the communication protocol to update the RT-images in the remote RT-objects with minimal delay. The semantics of the periodic messages transported in a TT system is a state-message semantics, i.e., a new version of a message overwrites the previous version and a message is not consumed on reading. This semantics is well suited to handle the transport of the RT-images of the state variables used in control applications. The state message semantics provides a predefined constant load on the communication system and eliminates the delicate problem of dynamic buffer management.

3.1 Temporal Firewalls

TT architectures delegate a high degree of autonomy to their subsystems. TT systems are not pushed by the events that happen in the environment and which are outside the sphere of control of the computer system. In a TT system the temporal control remains always inside the controlling subsystem. Since the subsystems derive their control signals autonomously from the progression of time, the interfaces can be designed as data-sharing interfaces without any control signal passing the interface. We call such an interface that blocks any control error propagation a *temporal firewall*. Three of these temporal firewalls within a distributed time-triggered architecture deserve special mention:

(1) the interface between the controlled object and the computer system,

(2) the interface between the application software within a node and the communication system, the message-base interface (MBI), and

(3) the gateway interface between clusters.

In a TT architecture the interface between the sensors and actuators in the controlled object and the computer system is a data sharing interface free of any control signals. A TT architecture does not have to make any assumption about the proper generation of control signals (interrupt signals) by an external subsystem that is outside its sphere of control. There is no possibility for a malicious external device to upset the computer by generating unspecified sporadic interrupts.

The *Message Base Interface (MBI)*, which will be discussed at length in the remainder of this paper, separates the communication within a cluster from the processing within a node. It is a data sharing interface that hides the communication system from the application software within the node and vice versa behind a memory abstraction implemented in a dual ported random access memory (DPRAM) that is accessible by both, the communication controller and the host CPU that executes the application tasks. The communication controller autonomously updates the images of the state variables of the MBI. Whenever an application task needs a timely image, it just reads the specified DPRAM area.

If the temporal accuracy of the real-time data is longer than the longest time interval between the point of observation and the point of use of this data element, then the application process at the receiving end can always assume that the real-time data in the MBI is temporally valid. If the data is *phase sensitive*, i.e., the above cited condition is not satisfied, then the receiving task must be synchronized with the sending task on the basis of the a priori known receive time of the data and/or a state estimation task must be executed at the receiving end to improve the temporal accuracy of the real-time image. Since there is no control signal crossing the MBI other than the synchronized ticks of the clock, the control of any one of the subsystems (communication controller, host CPU) cannot be interrupted by a failure in the control of the other subsystem.

The gateway interface between clusters implements the relative views of interacting clusters. It is an application specific data-sharing interface. The gateway interface software collects the information in the sending cluster that is relevant for the receiver and transforms it to a representation that conforms to the information representation expected by the receiving cluster. There are no control signals passing through a gateway component, i.e., a gateway does not reduce the autonomy of control of the interconnected clusters.

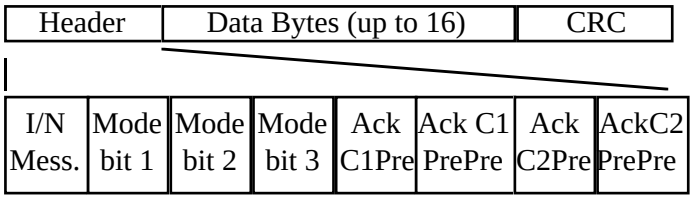
3.2 The Time-Triggered Communication Protocol

Communication between the nodes of a Time-Triggered Architecture (TTA) is realized by the time-triggered protocol (TTP) in the communication subsystem. TTP is an integrated time-triggered protocol that provides prompt transmission of messages with high data efficiency, a responsive membership service, a fault-tolerant clock synchronization service, mode change support, error detection with short latency and distributed redundancy management. The interested reader is referred to [Kopetz & Grünsteidl, 1994] to find a comprehensive description of the protocol.

Principle of Operation

Fig.2 presents the format of a TTP frame on the network. We distinguish two frame types, the N-frames (normal frames) used during normal operation and the I-frames (initialization frames) used for system initialization. The header byte of an N-frame contains four fields: the first bit to identify the message type, then a three bit mode change field, and a four bit acknowledgment field that contains acknowledgment information about the receipt of the messages from the predecessor and the prepredecessor. The I-frame contains the controller state (see below) in its data field. The attributes of the messages sent and received by the protocol are described in a static configuration data structure, the so-called message descriptor list (MEDL) that resides at the controller. According to this list the TTP controller periodically and autonomously reads the messages to be sent by the node from the MBI of the sender and writes received messages to the MBI of the receiver.

N Message:



I-Message:

Fig. 2: Frame Formats of TTP

In a time-triggered architecture much information about the behavior of the system, e.g., which node has to send what type of message at a particular point in time of a sparse time base, is known a priori --- at design time --- to all nodes of the ensemble. TTP makes best use of this a priori information to reduce the number and size of messages, e.g., by retrieving the message identification from the a priori known time of receiving a message. Access to the transmission medium is controlled by a static TDMA scheme. Each node is allowed to send messages only during a predetermined time span, called its TDMA slot. Depending on the degree of replication, an *FTU slot* will contain one, two, or three TDMA slots. The sequence of the periodic TDMA slots is called a *TDMA cycle*.

In a time-triggered architecture, a receiver can only interpret the frame sent by a sender if sender and receiver agree about the controller state at the time of sending and receiving. In TTP this controller state (the C-state) consists of three fields: the mode, the time and the membership. The mode field contains the identification of the current operational mode of the system. The time field contains the global time at the beginning of the current FTU slot. The membership field reveals which FTUs have been active and which FTUs have been inactive at their last membership point. To enforce C-state agreement between a sender and a receiver the CRC of a normal message is calculated over the message contents concatenated with the local C-state. In case the C-state of the sender is different from the C-state of the receiver, any message sent by the sender will be discarded by the receiver.

Fault Handling Strategy

The fault handling strategy of TTP is based on a two level approach: At the system level the protocol must detect and manage clean node failures promptly. At the component level it must be ensured that all internal failure modes of a node are mapped into a clean node failure at the network interface of the node. At the system level TTP assumes that all nodes exhibit a fail-silent behavior in the temporal domain, i.e., a node either sends (1) a correct message (correct operation) at the specified point in time or (2) no message (crash failure or omission failure) or (3) a detectably incorrect message at the correct point in time. A detectably incorrect message, e.g., a message with a CRC field such that the corruption of the message can be detected by recalculating the CRC polynomial, must be discarded by the receiver. At a higher level of abstraction we can therefore assume that a message is either correct or no message has arrived. We call any failure of the above indicated type a "clean node failure". Note that a difference in the C-state of the sender from that of the receiver will also result in a "missing message" because of the CRC calculation technique explained above.

To tolerate the loss of messages, every FTU can send multiple copies of the same logical message on the physical network. If an FTU consists of two actively redundant nodes and every node is connected to two redundant communication channels, then four physical copies of every message --- one on each channel from each node of the FTU --- can be transmitted. Up to three of these four messages can be corrupted without any effect on the application. In any case, there will be only a single correct message presented to the application software at the MBI interface, i.e., the redundancy is hidden from the application. TTP does not support the retransmission of corrupted messages because retransmissions have a negative effect on the temporal properties of a real-time system.

High Error Detection Coverage (HEDC) Mode. TTP provides a high error detection coverage mode [Kopetz, 1994] to increase the error detection coverage of a node by adding a 16 bit end-to-end CRC to the application data. This 16 bit end-to-end CRC is applied in addition to the communication CRC. To avoid the possibility that an incorrect message is selected by the operating system (this failure mode has been observed in fault injection experiments), the expected send time and a unique message key is concatenated with the message before the 16 bit end-to-end CRC is calculated (Fig.3) by the application task.

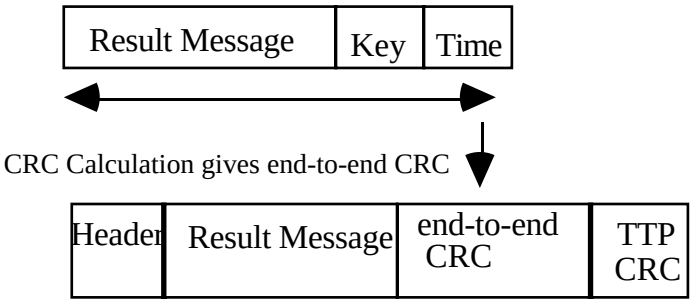


Fig.3: End-to-End CRC Calculation in Class C Messages

This end-to-end CRC field is added to the message and checked by the TTP controller of the receiving node as well as at the application level of the receiver. For this check, the controller must be provided with the necessary information. The (static) key value is stored locally in the MEDL. The value of the (dynamic) C-state time used for the CRC generation is calculated at the sender and receiver relative to the point in time when the message is expected to arrive at the controller of the receiving node. The TTP controller at the receiver takes the current time for its end-to-end CRC check. Fig.4 illustrates this principle.

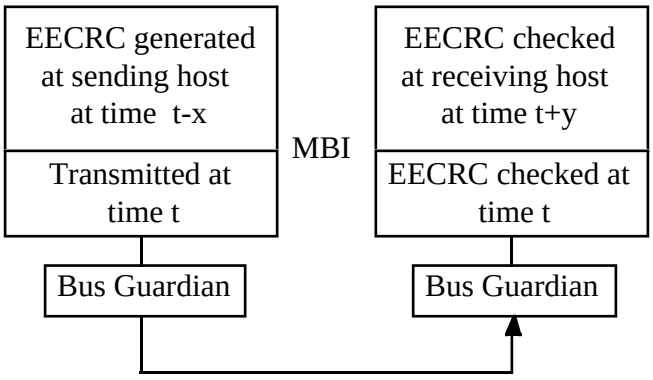


Fig 4: Time for the EECRC check

It not only avoids the costly storing of additional configuration data at the TTP controller, but also decouples the sender's and the receivers' data structures. If the generation time of the message changes, the only data structure affected is that of the sender (the time value x will change), and vice versa for the receivers.

The end-to-end error detection mechanism makes sure that a transient error corrupting a message between the point in time when a message has been generated by the application software at the sending node and the point in time when a message is used by the application software at the receiving node is detectable. The effectiveness of this end-to-end CRC has been confirmed by extensive fault-injection experiments [Karlsson, Folkesson, Arlat, Crouzet, & Leber, 1995]

Babbling Idiot Avoidance. A critical failure mode of a fail-silent node is the spontaneous transmission of senseless messages at arbitrary points in time. This failure mode is often called "babbling idiot" failure. Whereas the end-to-end CRC mechanism is sufficient to detect this failure mode in the value domain, an additional mechanism is needed to detect this failure mode in the time domain. The temporal regularity of TTP is used to implement a failure detection mechanism for this failure mode. Since every node has a fixed position in the TDMA send sequence, the "right to transmit" is only given (by a separate hardware bus guardian, see Fig. 4) in the allocated TDMA slot of the node. If a node tries to send a message outside its TDMA slot, the bus guardian will terminate the operation of the node and initiate a node self test and recovery.

3.3 Coping with the Complexity of a Design

In this section we investigate how a time-triggered architecture copes with the complexity of a design. We start by analyzing how the problem of system size is handled.

The Impact of System Size

System complexity increases more than linear with the number of elements and the intensity of the interactions among elements, i.e., with the system size. The most successful approach to cope with the complexity of large systems is the introduction of system structure: the definition of subsystems (or parts) with high inner connectivity contrasted by weak interactions between these subsystems.

We distinguish between two kinds of structuring: horizontal and vertical. Horizontal structuring (or layering) is related to the process of abstraction, of finding a reduced representation of the system which serves the given purpose. Many software engineering techniques (e.g., structured programming, virtual machines) propose one or another form of horizontal structuring. Vertical structuring is related to the process of partitioning a large system into a number of fairly independent subsystems with small and stable interfaces between these subsystems.

The time-triggered architecture approach strongly supports the spatial partitioning of a large distributed real-time system into a set of autonomous subsystems with small control-free data-sharing interfaces between them. It adheres to the old architectural principle: *form follows function* [Rechtin, 1992] that can only be implemented in a distributed computer architecture, where the function of a node and the hardware required to realize this function become an inseparable unit. This unity of form and function with sovereign control simplifies the problem of testing, certification, diagnosis, and redundancy management.

We have introduced two types of subsystems: the communication subsystem and the processing subsystem within a node. The communication subsystem, based on TTP, acts autonomously and predictably without any control input from the nodes. It has its own data structures, (the MEDL), and can be tested in isolation from the application software within the nodes.

The MBI between a node and the communication subsystem resembles closely the sensor/actuator interface between an interface node and the controlled object. Each node in a time-triggered architecture views the rest of the system (the MBI) in a way that an autonomous computer views its environment. The software within a node can be tested and certified in isolation against a simulation model of the MBI. Because of the sparse timebase of the TTA, the simulation model can generate the precise behavior of the MBI in the value and time domain.

The Task Coordination Problem

The complexity of the synchronization and the task coordination in distributed real-time systems has its origin in the data dependency of the temporal control structures and the dense time. By basing the generation of the static temporal control structures on the *maximum* execution time of a program and the *maximum* execution time of the communication protocol, the data dependency of the temporal control structures is eliminated in TT architectures. It is thus possible to validate the temporal control structures in isolation from the actual data. This approach simplifies the validation effort drastically while, at the same time, it reduces the achievable resource utilization (since the maximum and not the average execution time is used in the schedule).

From the temporal point of view, TT architectures partition the dense continuum of time into a sequence of equivalence intervals that start and end at the gridpoints of a space-time lattice ("sparse timebase" as introduced in [Kopetz, 1992]). The distance between these gridpoints is determined by the precision of the clock synchronization. By observing the environment and performing output actions only at the gridpoints of this space-time lattice, the temporal order of I/O events can be maintained throughout the whole distributed system and the potential input space can be reduced significantly.

Guarantee of the Deadlines

The off-line search for a satisfying schedule can be considered as a constructive schedulability test. Although the scheduling problem itself is NP complete, the validation of the solution, i.e., the test whether a proposed schedules meets all deadlines and observes all specified task dependencies (i.e., the precedence and mutual exclusion constraints) is simple. In a TT architecture the deadlines of all tasks are predictable and deterministic with respect to the specified gridpoint of the space-time lattice, i.e., within the precision of the global clock synchronization.

Redundancy Management

The replica determinism of a time-triggered architecture establishes the foundation for the transparent implementation of fault-tolerance. Provided the programmer of application tasks adheres to the rules that are required for the generation of replica determinate application code [Poledna, 1994], the replication of processing nodes and communication channels can be handled within the communication system. It is thus hidden by the MBI that is accessible to the application software.

4 The Message Base Interface

The Message Base Interface (MBI) is the most important interface within a time-triggered architecture. It is a data-sharing interface that binds together the communication subsystem with the host computers and eliminates any means of control error propagation across this interface.

4.1 Structure of the Interface

The design of the MBI as a data sharing interface is reflected by its structure: It consists primarily of data fields. We distinguish two different parts:

(1) The *Status/Control Area (SCA)* contains system related information. It provides a facility for the TTP controller and the host CPU to communicate with each other via dedicated data fields. Moreover, protocol information like status and control information, the C-State and the current view of the external reference time are exchanged using the SCA.

(2) The *Message Area* contains the messages sent or received by the node, including a control byte for each message.

Status/Control Area

The message base status/control area is a memory area of the DPRAM containing control and status information that has to be shared between the TTP controller and the host CPU. The memory layout of the message base status/control area's registers is shown in Fig. 5.

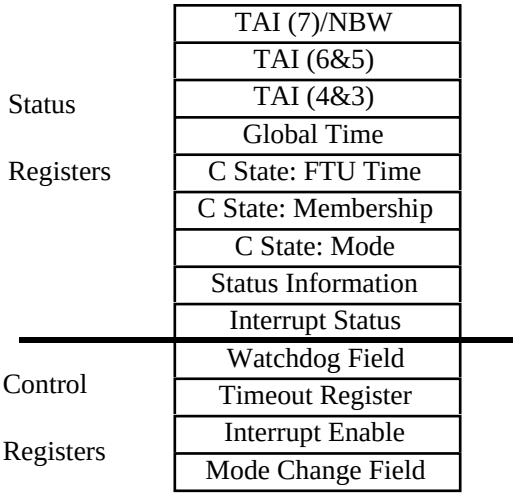


Fig. 5: Status/Control Area of the Message Base Interface

Status Registers written by the controller:

TAI . In systems with external clock synchronization, this six byte field (five data bytes, one status byte for the NBW protocol [Kopetz & Reisinger, 1993]) contains the current time (TAI) in the format specified in Fig.6.

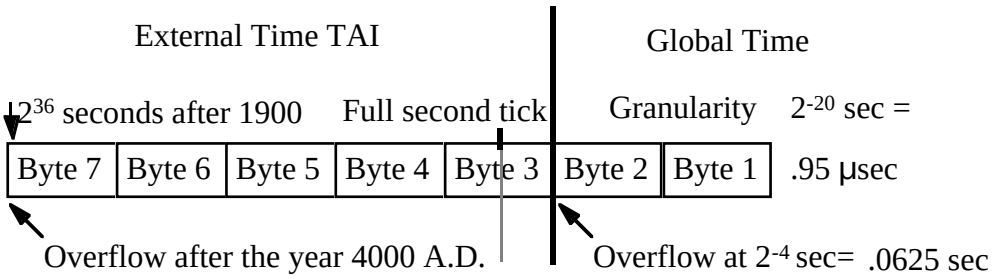


Fig. 6: Time format at the MBI

Global Time. This sixteen bit register contains the current global time of the node with the specified granularity of 2^{-20} seconds.

C-State . This field contains the current C-state of the TTP protocol. It includes the current time, the current membership field and the current mode (including the current MEDL entry). The granularity of the C-state time is equal to the length of an FTU slot, i.e., its value is increased only when a new FTU slot starts.

Status Information. This field contains information about the current status of the protocol execution. It is written by the TTP controller and read by the host operating system.

Interrupt Status. This register contains information about the interrupt(s) that was (were) last raised by the TTP controller. It is read by the host CPU after an interrupt was raised. Its contents is a bitmap where each bit represents an interrupt source. Reading this register acknowledges the interrupt.

Control Registers written by the host:

Watchdog Field. This field, updated periodically by the host CPU, contains the inversion of the global time with a maximum latency that is specified in the MEDL. The controller checks this field to determine if the host CPU is still alive.

Timeout register. This register contains a global time value written by the host CPU. When the value of the global time counter (register Global Time 0) reaches this value, the TTP controller raises an interrupt.

Interrupt Enable. The layout of this register is the same as the interrupt status register's. Each bit can be used to separately enable/disable the corresponding interrupt type.

Mode Change Field. The control message contains information about a requested mode change the host CPU wants to communicate to the TTP controller. It is therefore written by the host CPU and read by the TTP controller.

The interrupt registers are introduced for the purpose of interface control and are described in section 5.1.

Message Area

In addition to the data contained in the messages, the message area also contains a status byte and an optional end-to-end CRC for each message. The length of the end-to-end CRC is 16 bits. The structure of a message in the message area is shown in Fig. 7.

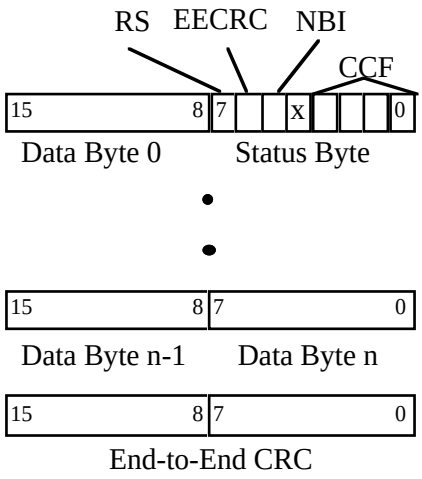


Fig. 7: Message Base Entry

The Status Byte of each message in the message area contains the following fields:

Receive Status (RS).This flag determines if the last reception of the message was valid, i.e. at least one of the redundant messages was received correctly. If it is not set, the data bytes contain an old value of the message.

End-to-End CRC invalid (EECRC). This flag is set if the end-to-end CRC of all redundant messages was invalid.

Noisy Transmission (N) or Bus Idle (BI). If the last reception of the message was valid (indicated by RS), this flag is set if noise could be detected during the transmission by the sampling logic. In this case the bit will only be set if the reception of all redundant messages was noisy. If no valid message could be received, this bit signals if the bus(ses) were absolutely idle during this/these slot(s). In this case the bit is only set if all busses were idle.

Concurrency Control Field (CCF). This field is used by the NBW protocol [Kopetz & Reisinger, 1993] when accessing a message. Given the CCF size of four bits, a read access to a message can be interrupted up to eight times (16 different values, two per access) before a wrap-around occurs.

4.2 Consistent Data Transfer

Since the MBI is based on a 16-bit architecture, the consistency of single-word data transfers is guaranteed by the hardware arbitration. In the following we will describe how the consistency of a multi-word data transfer is realized at the MBI.

It is essential to analyze the MBI's access pattern, i.e., the sequence and frequency of read and write accesses from either side. The regularity of a time-triggered communication scheme restricts the access to the MBI to well-defined intervals known to both the host CPU and the TTP controller a priori. The host CPU will perform many read accesses --- a node receives more messages than it sends --- and only a few write operations in each TDMA cycle. The opposite is true for the TTP controller: it performs more write than read operations since received messages have to be written to the MBI.

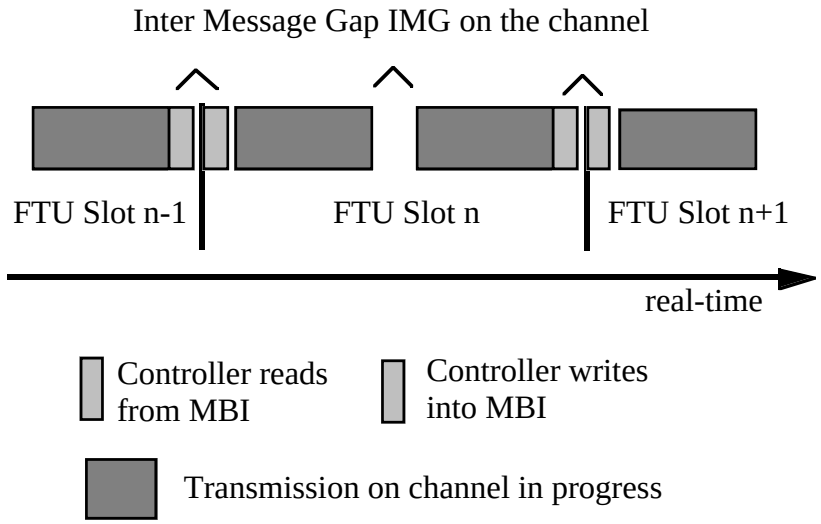


Fig.8: Timing at the Message Base Interface

Fig. 8 shows a typical FTU slot. The time between the transmission of messages, which is required for protocol tasks, is called the *Inter Message Gap* (IMG). During the intermessage gap at the beginning and at the end of a FTU slot the communication controller reads from and writes to the MBI. In the rest of the FTU slots the two messages are transmitted on the channel, with a short intermessage gap inbetween.

Controller to Host.

The data transfer from the TTP controller to the host CPU is under the control of the current MEDL and consists of copying one message from the receive buffer of the TTP controller to the DPRAM. The selection of the message out of the set of redundant ones depends on whether

- the CRC check was OK and
- the end-to-end CRC of HEDC messages was also OK (see below).

Besides the message data, the status byte containing status information on the message reception has to be assembled and copied to the DPRAM. The status byte and the data field of a message are written to the MBI during the above-mentioned update interval. If the host CPU derives its read access intervals from the global time base then access conflicts can be avoided by a properly planned schedule without any explicit online synchronization.

If on the other hand the host CPU accesses the MBI at arbitrary points in time, the non-blocking write protocol NBW is used. This protocol enables the host CPU to detect any write operation of the TTP controller that occurs while it is reading a message. In this case, the read operation of the host has to be repeated. The TTP controller is never delayed while accessing the MBI.

Host to Controller.

To synchronize the host's write operation with the TTP controller, the host has to be aware about the times when the TTP controller will read a particular field from the MBI. The interface control mechanisms, described in section 5, help to simplify the synchronization of the host operating system with the operation of the TTP controller. Although the NBW protocol is only intended to be used for the data transfer from the controller to the host, it provides a useful error detection mechanism for data transfer in the other direction. Therefore, the control byte is also provided for the messages sent by the controller.

4.3 Redundancy Management

Redundancy management comprises all mechanisms required to control the reconfiguration of nodes in a redundant architecture. In the following, these mechanisms and their influence on the MBI will be discussed.

The host CPU is not concerned with any of the redundancy mechanisms employed to achieve a higher level of fault tolerance (e.g., replicated SRUs and busses). This implies that only one correct message out of the set of redundant messages has to be copied to the message base. Information on the reception status of every message is available in the Status Information field of the SCA.

The system architecture described in the previous section supports the high error detection coverage (HEDC) mode to increase the error detection coverage. The end-to-end CRC (in addition to the normal CRC) is generated in the application task of the host CPU. It is important to note that the end-to-end CRC is only valid within a cluster. If the message has to be passed to another cluster via a gateway, the gateway has to check the old end-to-end CRC and calculate a new one.

Many fault tolerant systems do not make the assumption of fail-silence in the value domain. Instead, three nodes operate in active redundancy. Their results are compared and a correct value is determined by voting. Triple modular redundancy (TMR) can be implemented with nodes using the MBI without changing this interface. The only requested change concerns the implementation of the communication controller software. For the host CPU, this change of the redundancy scheme is transparent.

4.4 Mode Changes

Mode changes are a powerful mechanism to dynamically change the temporal control structure of a cluster. A mode change can influence the control flow in the whole system, causing a potential danger that an erroneous host CPU disturbs the function of the whole system. Therefore mode changes have to be handled with great care and should be avoided during the normal operation of safety critical applications. In safety critical applications mode changes are only supported during the system startup phase, where they are necessary. Once the system has begun executing the application, mode changes are disabled. This can be achieved by setting the proper parameters in the MEDL.

To initiate a mode change, the host CPU has to write the number of the new mode into the *Mode Change Field* of the status/control area. Before assembling a message to be sent, the TTP controller reads this information. It checks if this mode change is permitted after this FTU slot according to its static configuration data in the MEDL and, if allowed, sets the corresponding bit in the TTP frame header. Upon reception of a message, the frame header is evaluated by the receiving controller. To inform the host CPU of a requested mode change, a flag in the *status register* of the MBI's SCA is set. If enabled, an interrupt to the host CPU will be raised at the end of the current FTU slot.

5 Implementation Considerations

5.1 Interface Control

In the prototype implementation of the time-triggered system MARS [Kopetz, Damm, Koza, Mulazzani, Schwabl, Senft, et al., 1989] the periodic synchronized clock signal always raised an interrupt at the host computer. This mechanism caused a substantial overhead for the run time system in the host. We therefore decided to give the host CPU the capability to control the occurrence of the time interrupts by setting the appropriate bits in the Interrupt Enable Register of the SCA.

The MBI provides programmable time interrupts in case the host operating system needs to be synchronized with the global time or some other significant events detected by the controller. A controller time interrupt can only be raised at grid points of the synchronized time grid. At which gridpoints this interrupt will be raised is determined by the interface control that has been selected by the host. There are several conditions that can trigger a time interrupt. The MBI allows to request an interrupt after the following events:

(1) Global Timer. The host CPU can write a time value into the timeout register of the MBI. When the synchronized global time reaches the value contained in this register, an interrupt to the host controller is raised. The host controller interrupt service routine will usually perform a table lookup to invoke the proper application task and then it will usually write a new time value into the timeout register.

(2) Message Reception. According to the contents of the MEDL the reception of a particular message may trigger an interrupt to the host CPU. Since message transmission under the control of TTP is driven by the progression of time, the reception of a message marks a significant, predefined point in time.

(3) Mode Change. A host CPU can enable an interrupt at the end of the FTU slot during which a mode change has been detected by the controller.

(4) Membership Change. As with mode changes, changes to the current membership take place at predefined points in time and can be programmed to cause an interrupt.

The interrupt sources may be turned off by the host CPU. For this purpose the MBI provides the interrupt enable register which allows the host CPU to selectively and dynamically enable or disable specific interrupt sources. Information on the source of a specific interrupt is contained in the interrupt status register, which is part of the MBI's status/control area. Upon an interrupt request, the host CPU has to read this register, thereby resetting all flags and acknowledging the interrupt.

This time interrupt control provides the flexibility needed to synchronize the actions of the operating system of the host with the global time.

5.2 Electrical and Mechanical Specification

The electrical and mechanical specification of our implementation of the MBI conforms to the industry standard IndustryPack™ Logic Interface Specification [Greenspring, 1989]

5.3 Current Implementations

In 1991 a prototype of the time-triggered architecture MARS was implemented. This prototype contained two 68070 processors, one for the communication and the other for the application tasks [Reisinger, Steininger, & Leber, 1995]. In the last four years extensive fault-injection experiments were performed on this board in the context of the ESPRIT project PDCS. During these experiments we gained a deep insight into the performance aspects of the operating system and the behavior of the fault tolerance mechanisms [Karlsson, et al., 1995].

Based on these insights we are now designing a new version of the time-triggered architecture with a special focus on the clean implementation (in software and hardware) of the MBI. In a first step it is planned to build a small board TTP controller by using the Motorola 68332 microcontroller. This microcontroller has an on-chip Time Processing Unit (TPU) that will be used to support the clock synchronization. In a second step our VLSI department is going to implement a single chip version of this TTP controller that will be hardware/software compatible to the board implementation of the first step.

6 Conclusion

The processing nodes of a any distributed architecture are neither omniscient nor clairvoyant. Therefore it cannot be expected that optimal decision are made, even if a system is based on dynamic scheduling. The question, whether local online dynamic decision are better or worse than global offline static decisions depends strongly on the regularity of a particular application. Since real-time control applications tend to be highly regular and require a predictable temporal performance, we conjecture that TT architectures provide a good solution for this problem domain.

The key issue in the design of large systems relates to the recursive structuring of the system into its subsystems and the specification of the interfaces between these subsystems. From the architectural point of view, these interfaces should enforce a minimal coupling between the subsystems in order to make the architecture intelligible and to reduce the possibility of error propagation across a subsystem interface. In this paper we have proposed such a concrete interface for a time-triggered architecture, the message base interface MBI, that separates the communication subsystem from the processing subsystems in a distributed fault-tolerant real-time architecture. Since this interface is free of temporal control signals, the possibility of control error propagation between subsystems is eliminated. The simple and stable interfaces that are established between the subsystems of a TT architecture effectively support the design, validation and integration of large real-time systems.

References

Greenspring (1989). *Industry Pack Logic Interface Specification*, Menlo Park, Cal. Greenspring Computers Inc.

Karlsson, J., Folkesson, P., Arlat, J., Crouzet, Y., & Leber, G. (1995). Integration and Comparison of Three Physical Fault Injection Techniques. In B. Randell, J. L. Laprie, H. Kopetz, & B. Littlewood (Ed.), *Predictably Dependable Computing Systems* Heidelberg: Springer Verlag.

Kopetz, H. (1992). Sparse Time versus Dense Time in Distributed Real-Time Systems. In *Proc. 14th Int. Conf. on Distributed Computing Systems*, (pp. 460-467). Yokohama, Japan: IEEE Press.

Kopetz, H. (1994). Fault-Management in the Time-Triggered Protocol. In *Proc. SAE World Congress*, SAE SP 1012 (pp. 77-84). SAE Press.

Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C., & Zainlinger, R. (1989). Distributed Fault-Tolerant Real-Time Systems: The MARS approach. *IEEE Micro, 9*(1), 25-40.

Kopetz, H., & Grünsteidl, G. (1994). TTP-A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer, 24*(1), 22-66.

Kopetz, H., & Kim, K. (1990). Temporal Uncertainties in Interactions among Real-Time Objects. In *Proc. 9th Symposium on Reliable Distributed Systems*, (pp. 165-174). Huntsville, AL, USA: IEEE Computer Society Press.

Kopetz, H., & Ochsenreiter, W. (1987). Clock Synchronisation in Distributed Real-Time Systems. *IEEE Trans. Computers, 36*(8), 933-940.

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

Laprie, J. C. (Ed.). (1992). *Dependability: Basic concepts and terminology - in English, French, German, German and Japanese*. Vienna, Austria: Springer-Verlag.

Littlewood, B., & Strigini, L. (1995). Validation of Ultradependability for Software Based Systems. In B. Randell, J. L. Laprie, H. Kopetz, & B. Littlewood (Ed.), *Predictably Dependable Computing Systems* (pp. 473-493). Heidelberg: Springer Verlag.

Poledna, S. (1994). *Replica Determinism in Fault-Tolerant Real-Time Systems*. PhD, Technical University of Vienna.

Rechtin, E. (1992). *Systems Architecting, Creating and Building Complex Systems*. Englewood Cliffs: Prentice Hall.

Reisinger, J., Steininger, A., & Leber, G. (1995). The PDCS Implementation of MARS Hardware and Software. In B. Randell, J. L. Laprie, H. Kopetz, & B. Littlewood (Ed.), *Predictably Dependable Computing Systems* (pp. 209-224). Heidelberg: Springer Verlag.

Rushby, J. (1993). *Formal Methods and the Certification of Critical Systems* No. SRI-CSL-93-07). Computer Science Lab, SRI.