

A Platform for Safety-Critical Applications

Introduction

There has been an enormous increase of the number of electronic control units over the past few years. According to analyst reports, 90% of all future innovations will be driven by the use of electronics. New functions, such as lane departure warning, automatic emergency braking, traffic sign recognition, and heading control, are increasing the overall complexity of the electronics. In addition, future by-wire systems, such as brake-by-wire, steer-by-wire, and electronic vehicle dynamics control, demand a high level of dependability. That is why the electronic architecture plays a crucial role in the development roadmap of manufacturers and their suppliers.

Nowadays loosely coupled ECUs (Electronic Control Unit) are state-of-the-art. Each ECU is developed with its own internal system structure and there is little integration among different ECUs in this system structure. Today's systems are equipped with up to 70 ECUs. Each ECU acts autonomously and is almost independent of all other ECUs in the network. Information exchange between ECUs is performed via an event-triggered communication system with relatively low communication speed, typically CAN with up to 500 kbit/s. Most ECUs can continue operating even if the communication fails.

Next-generation functions require a paradigm shift from loosely coupled ECUs to an integrated electronic architecture, a distributed safety-critical real-time platform. There are four major reasons for this paradigm shift:

Cost: An integrated architecture allows distributing and adding functions across the network in a cost-optimal way. This lessens the number of ECUs in systems and reduces the cost of the overall system considerably.

Stability and Reliability: Design faults in software and hardware are a major source for instability and low reliability of a complex system. An architecture that supports state consistency can prevent many difficulties of distributed real-time systems, thus facilitating the conditions for the use of the software and reducing the development time.

Safety: New functions, e.g. by-wire and collision avoidance, are safety-relevant. An integrated electronics architecture needs to support the interaction of functions with high and low safety requirements (without the need to validate all functions on the highest safety level).

Multiple Use of Sensors: In an integrated architecture sensors can be used by several systems. This is how faults of individual sensors can be detected and the total number of the sensors and the costs for distributed control can be reduced.

Safety-Critical Applications

In order to be used in commercial applications, an integrated electronics architecture needs to satisfy very high requirements in the areas of safety, availability, and fault tolerance. It needs to support ease of system integration, component upgrades, and the migration of existing systems as well as a great number of models, platforms, and equipments. The architecture must also meet the stringent cost constraints imposed by the market. In the course of this paper it will be discussed that the provision of state consistency is a central property of such an architecture. It is state consistency that helps to fulfill the high requirements of an integrated electronic platform.

The Time-Triggered Architecture (TTA) and its core technology, the time-triggered communication protocol TTP, are specifically designed to meet these requirements and to provide state consistency. For more than 20 years TTP has been developed by the Vienna University of Technology and TTTech in cooperation with industrial partners and leading research institutions. One of the topmost priorities in the development of TTP has been to meet the requirements and cost targets of the industry.

In order to be used for application of the highest safety class, certification is done in accordance with the relevant requirements. The development process and the design of the TTP chip models are documented so as to satisfy the different standards in the aerospace industry. Combined with the relevant documentation and processes of requirements of the semiconductor, this forms the basis for all customers to have their products certified in accordance with the FAA or JAA standards. TTTech's development software is developed in accordance with the FAA standard DO 178B Level A. This guideline belongs to the highest safety standards in the area of software development.

TTP is already used in serial applications for railway systems and special vehicles. Many industrial applications have been announced, e.g. the safety-critical use of TTP in the new Airbus A380. TTP is an open cross-industry standard that is supported by the TTA-Group [1].

The paper is organized as follows: Section 2 introduces the TTA and the TTP protocol. Section 3 accounts for the need of state consistency at the architectural level and explains the core elements of TTP. Section 4 describes how to handle events in the TTA. Section 5 gives a brief comparison of some time-triggered protocols with special emphasis on consistency and handling of time- or event-triggered messages. Section 6 rounds off the article with a conclusion.

The Time-Triggered Architecture

The computer architecture constitutes a blueprint and a framework for the design of a class of computing systems that share a common set of characteristics. The Time-Triggered Architecture (TTA) establishes a frame for data processing in the area of distributed embedded real-time systems with highly reliable applications. It sets up the computing infrastructure for the implementation of applications and provides mechanisms and guide lines to partition large applications into nearly autonomous subsystems along small and well-defined interfaces in order that the complexity of the evolving product can be controlled. Architecture design is thus interface design.

By defining an architectural style that is observed at all component interfaces, the architecture avoids property mismatches at the interfaces and eliminates the need for unproductive "glue" code.

Safety-Critical Applications

A central characteristic of the Time-Triggered Architecture is the handling of (physical) real time as a first-order quantity. The TTA decomposes a large embedded application in nodes (ECUs) and clusters and provides a fault-tolerant global time base of known precision at every node. The TTA takes advantage of the availability of the global time to precisely specify the interfaces among the nodes, to facilitate communication, to guarantee state consistency, to perform prompt error detection, and to support the timeliness of real-time applications.

Structure of the TTA

The basic building block of the TTA is a node. A node comprises a processor with memory, an input-output subsystem, a time-triggered communication controller, an operating system and the relevant application software all in a self-contained unit (possibly on a single silicon die).

Two replicated communication channels connect the nodes to build a cluster. The physical interconnection structure and the communication controllers of all nodes in a cluster form the communication subsystem that is autonomous in the TTA and executes an *a priori* specified periodic TDMA (Time Division Multiple Access) schedule. The communication subsystem reads a data frame with state information from the communication network interface (CNI) at the *a priori* known fetch instant and delivers it to the CNIs of all receiving nodes of the cluster at the *a priori* known delivery instant, overwriting the previous version of this frame. The periodic fetch and delivery instants are contained in a scheduling table within the communication controller called MEDL (Message Descriptor List) consistently known to all communication controllers in a cluster.

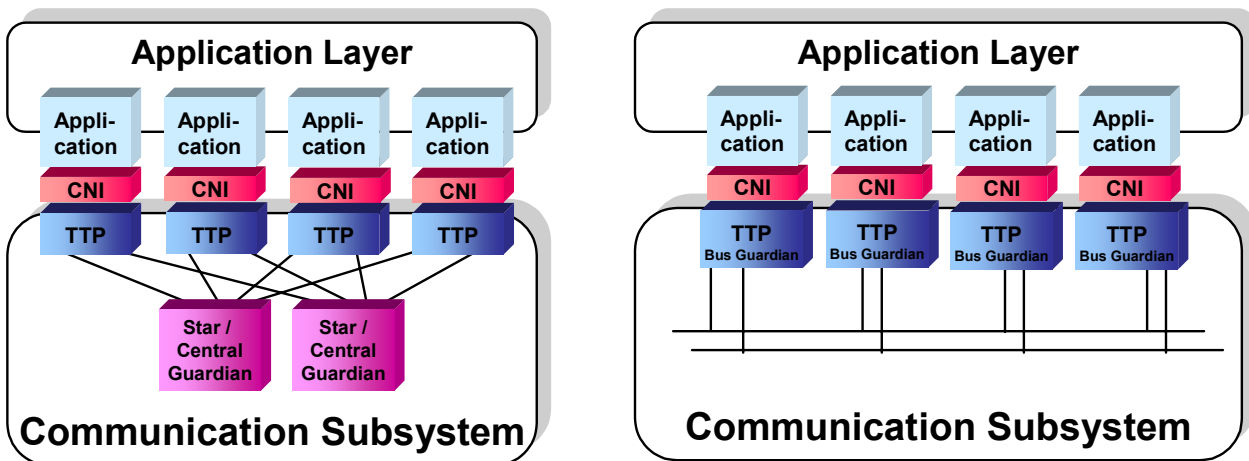


Figure 1. TTA-Star with central bus guardians and TTA-Bus with local bus guardians

At the moment the TTA is implemented on two different network topologies: on a bus topology (*TTA-Bus*) and a star topology (*TTA-Star*). Nodes of the TTA-Bus implementation are equipped with a node-local bus guardian that prevents babbling idiot faults. TTA-Star implements smart central guardians that are shared by all nodes of a cluster. These smart guardians isolate arbitrary nodes failures and support safety-critical applications. As far as the costs are concerned, TTA-Star is very attractive because only one bus guardian per channel is necessary. See figure 1.

Safety-Critical Applications

Design Principles

The following paragraphs discuss briefly the principles that have driven the design of the Time-Triggered Architecture:

Consistent Distributed Computing Platform. The main purpose of the TTA is to provide a consistent distributed computing base to all correct nodes in order that reliable distributed applications can be built with manageable effort. If a node cannot assume that every other correct node works on the same state, then the design of distributed algorithms becomes laborious [3] because the intricate agreement problem has to be solved at the application level. That is why TTP provides consistency support directly at the communication subsystem level.

Unification of Interfaces – Temporal Firewalls. A good architecture must be based on a small number of orthogonal concepts that are reused in many different situations, thereby reducing the effort needed for understanding complex systems. Driven by its time-triggered schedule, TTP is autonomous in carrying data frames from the sender's communication network interface (CNI) to the receiver's CNI. According to the *push-pull* paradigm, the sender can put the information into its local CNI memory while the receiver must get the information out of its local CNI memory. Since no control signals cross the CNI in the TTA (the communication system derives control signals for the *fetch* and *delivery* instants from the progress of global time and its local schedule table MEDL), control-error propagation is eliminated by design. We call an interface that prevents control-error propagation by design a *temporal firewall*.

Composability. In a distributed real-time system the nodes interact via the communication system to provide the emerging real-time services. These emerging services depend on the timely provision of the real-time information at the interfaces of the nodes. An architecture is termed composable in the temporal domain if it guarantees the following four principles [2]:

1. Independent development of nodes
2. Stability of services prior to the integration of a new function
3. Constructive integration of the nodes to generate the emerging services
4. Replica determinism

Scalability. The TTA is intended for the design of complex distributed real-time applications. A complex system that supports many different functions can be constructed most efficiently if the effort needed to understand a particular system function is independent of the system size. Horizontal layering (abstraction) and vertical layering (partitioning) are the means to master the complexity of large systems. In the TTA the CNIs encapsulate a function and make visible only those properties of the environment that are relevant for its operation.

The Time-Triggered Protocol (TTP)

TTP [1] is a fault-tolerant time-triggered protocol that provides the following services:

1. Autonomous fault-tolerant message transport at known times and with minimal jitter between the CNIs of the nodes of a cluster, i.e. the ECUs in a network, by employing a TDMA strategy on replicated communication channels.
2. Fault-tolerant clock synchronization that establishes the global time base without relying on a central time server.
3. Membership service to inform every correct node about the consistency of data transmission. This service can be viewed as a distributed acknowledgment service that informs the application promptly if an error in the communication system has occurred. If state consistency is lost, the application is notified immediately.
4. Clique avoidance to detect faults outside the fault hypothesis, which cannot be tolerated at the protocol level.

In TTP communication is organized into TDMA rounds. A TDMA round is divided into slots. Each node in the communication system has one slot – its sending slot – and must send frames in every round. The frame size allocated to a node can vary from 2 to 240 bytes in length, each frame usually carrying several messages. The cluster cycle is a recurring sequence of TDMA rounds; in different rounds different messages can be transmitted in the frames, but in each cluster cycle the complete set of state messages is repeated. The data is protected by a 24 bit CRC (Cyclic Redundancy Check). The schedule is stored in the message descriptor list (MEDL) within the communication controller. See figure 2.

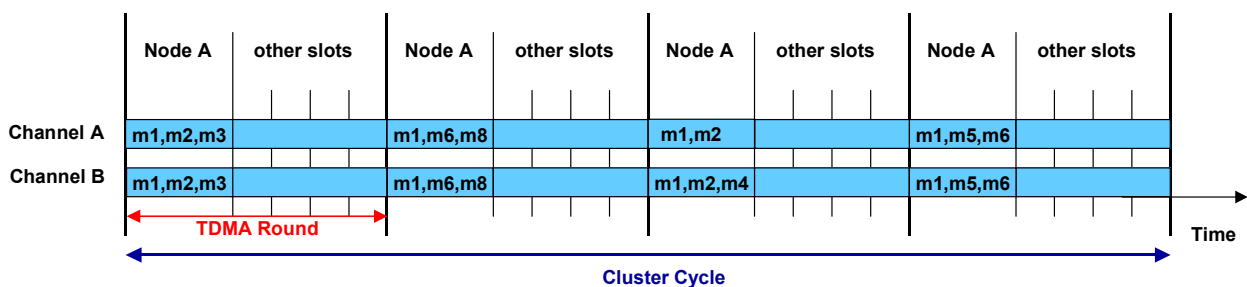


Figure 2. Frames, messages, slots, TDMA round, cluster cycle

The clock synchronization is necessary to provide all nodes with an equivalent time concept. In doing so it makes use of the common knowledge of the send schedule. Each node measures the difference between the *a priori* known expected and the observed arrival time of a correct message to learn about the difference between the sender's clock and the receiver's clock. A fault-tolerant average algorithm needs this information to periodically calculate a correction term for the local clock so that the clock is kept in synchrony with all other clocks of the cluster. The membership service uses a distributed agreement algorithm to determine whether, in case of a failure, the outgoing link of the sender or the incoming link of the receiver has failed.

Safety-Critical Applications

The core algorithms of TTP have been formally verified to prove their correctness. Real tests with multi-million fault injections and heavy ion radiation experiments and experiments with electro-magnetic interferences have been carried out successfully.

Fault Hypothesis and Fault Handling

Provided that the components of a properly configured TTP based system are in different fault containment regions, each can fail in an arbitrary way. Under this assumption the probability of two concurrent independent component failures is small enough to be considered a rare event that can be handled by an appropriate never-give-up (NGU) strategy. However, it should be noted that a very prompt error detection mechanism is needed to ensure that two consecutive single faults are not becoming concurrent.

As for hardware faults, TTP is designed to isolate and tolerate single node faults. By introducing a bus guardian it is guaranteed that a faulty node cannot prevent correct nodes from exchanging data. The bus guardian ensures that a node can only send once in a TDMA round, thereby eliminating the problem of babbling idiots that monopolize the communication medium.

Moreover, TTP implements a never-give-up strategy (NGU) for multiple fault scenarios: if a node detects faults that are not covered by the fault hypothesis, it notifies the application. The application may now decide either to shutdown in fail-safe environments or to restart in fail-operational environments with an agreed consistent state among all nodes of the distributed system.

Fault Tolerance

The mechanisms described above ensure fault tolerance at the communication subsystem level in TTP. These mechanisms of the communication subsystem guarantee that faulty nodes cannot prevent correct nodes from communicating and serve as communication platform for the application. At the application level fault tolerance needs to be implemented by a fault tolerance layer and an appropriate application design. Fault tolerance can be realized by replicating a software subsystem on two fail-silent nodes. Tolerance of a single arbitrary node failure can be ensured by TMR (Triple Modular Redundancy) voting.

Both mechanisms will tolerate single component faults with the respective failure semantics and are thus fit to handle both transient and permanent hardware faults. To re-establish tolerance to single component faults despite the presence of a permanent hardware fault, TTP supports the implementation of transparent hot-stand-by spares.

Fault tolerance is realized by a redundant unit in the network taking over the function of a defective unit, without being noticed at the function level. This is why data consistency is necessary.

Support for Consistency

Data consistency can facilitate the design and development of complex distributed systems considerably. In single node systems consistency can be taken for granted because data written to memory is available to all software subsystems at the same time and all subsystems read the same value, provided that the node is correct. In distributed systems it is no longer justified to assume this kind of consistency. There are two reasons for this: Firstly, message transmission delays that have an effect on the current state must taken

Safety-Critical Applications

into consideration; it is not guaranteed that a message arrives at all receivers at the same point in time. Secondly, individual nodes may fail or messages may get lost.

Design and programming in a distributed system become laborious and difficult, especially for complex applications, if state consistency is not supported at the communication platform level independent from the application. Application-independent support of state consistency is not only capable of relieving the application CPU of executing consistency protocols but, more important, of verifying the correctness of complex consistency algorithm for all applications. This makes it necessary to implement data consistency at the communication controller side of the CNI.

The huge number of possible node failures, communication failures, or differences in message arrival timing and sequence would make the logic of the application subsystem large and complex. Therefore state consistency should be supported as a basic service of the communication subsystem that satisfies the following properties. Provided that the fault hypothesis holds, a system is termed consistent if

1. all correct nodes agree on the same data,
2. all nodes agree on the data sent by a correct sender and
3. all correct subsystems deliver the received value at the same point in time.

It is assumed that a node sends data to a series of receiving nodes. TTA is designed to support communication consistency in the hardware directly on the protocol level. The mechanisms that support consistency are described in the following.

Membership and Acknowledgment

A major philosophy in the design of TTP is that the protocol should transmit data consistently to all correct nodes of the distributed system and that, in case of a failure, the communication system should decide on its own which node is faulty. These properties are achieved by the membership protocol and an acknowledgment mechanism.

Each node of a TTP based cluster maintains a membership list with all nodes that are considered to be correct. This information is updated locally in accordance with successful (or unsuccessful) data transmissions and thus reflects the local view of the receiving node on all other nodes. With each transmission, each receiver sees and checks the sender's membership that is included in the sender's transmission or hidden in the CRC calculation.

Acknowledgment: After transmission, node A seeks acknowledgment from the other nodes in order to determine whether the transmission was accepted at the receiver (on the communication level). This is achieved by checking the membership list of the first (and possibly second) successive sender. If these nodes show node A in their membership list, they state that A's transmission was successfully received. Otherwise, A is informed that the transmission was unsuccessful. Due to the time-triggered principle, re-transmission of the state message is done in the next cycle.

Membership Consistency Check: Due to the strict round-robin scheme of the TDMA round, each node sees and checks the membership lists of all other nodes in one TDMA round. Each sender with a different

Safety-Critical Applications

membership list is assumed as incorrect. This ensures a consistent view of all nodes, which accept each other in the membership, i.e., they communicate successfully with one another.

Clique Avoidance: In order to detect multiple component faults and inconsistencies and to support the never-give-up strategy, the clique avoidance mechanism is active. Before each send operation of a node the algorithm checks if the node is a member of the majority clique. In case the node is in a minority clique, this means that a very rare fault scenario outside the fault-hypothesis has occurred which led to an inconsistency. This condition is signaled to the application software, which can decide whether to initiate fail-stop or fail-operational activities.

The combination of these algorithms together with the common time base established by the clock-synchronization provides communication consistency. This guarantees that all correct nodes receive the same information at the same point in time. TTA thus provides the application software with a very powerful programming model that allows efficient handling complex distributed software systems.

Consistency and Events

In addition to the mechanisms for guaranteed, timely, and consistent transmission of real-time data, TTA also provides mechanisms for the exchange of event data. The need for event data usually arises from on-demand diagnosis, parameter calibration, and debugging.

The TTA event transport mechanism is based on the allocation of dedicated bandwidth for event data. A TTP frame can carry up to 240 bytes. Part of the frame can be used for event messages. The event-triggered messages are piggybacked on TTP frames. This partitioning of each slot in state data, event data, and spare bandwidth for future extensions is depicted in Figure 3.

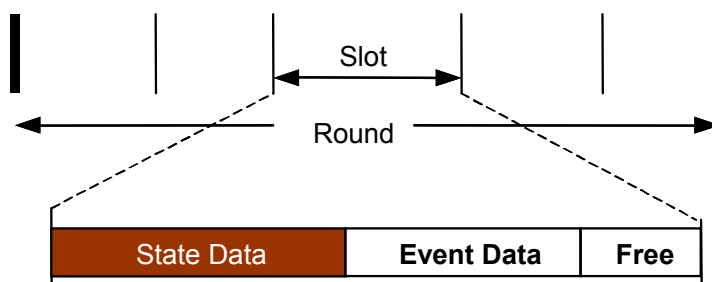


Figure 3. Real-time and on-demand frame partitioning

An important property of this scheme is that the bandwidth for on-demand transmission is arbitrated only among the event messages on each node, not between the event messages of all nodes in the system. This node-local arbitration has the advantage that it ensures full temporal composability. This cannot be attained in case of global bandwidth contention. Furthermore, all fault isolation capabilities of the TTA apply to these dynamically arbitrated channels. A faulty node cannot possibly occupy the channels of some correct node. Furthermore all consistency mechanisms described earlier are also valid for the event messages. This enables TTA to extend full consistency from time-triggered to event-triggered messages.

Safety-Critical Applications

Page 9

Even though the use of a node-local bandwidth is less efficient with respect to the overall bandwidth than the use of a system-wide bandwidth, extensive studies with large manufacturers have shown that the described event transport mechanism satisfies their requirements completely. A standard diagnostic protocol, for example, can be implemented with as little as 0.8% net bandwidth of a 10 Mbit/s TTP system.

In order to support the migration of other software systems, an implementation of the widely used TCP/IP protocol and of the CORBA IIOP protocol on top of the basic TTP communication service is in progress.

Migration Strategy for CAN-Based Software

The generic approach for event-data transmission makes it possible to layer the event-triggered CAN protocol on top of TTP. This provides a clean migration strategy for the large body of CAN based software without the need to re-write the software substantially. In an application a high-speed CAN controller at the maximum speed of 500 kbit/s could be emulated with 5 % net bandwidth on top of a 10 Mbit/s TTP system. A CSMA/CA like order could be established on the basis of the global sparse time base. A hardware based CAN emulation, the registers of which are compatible to widely used CAN modules, has been evaluated.

A Comparison of Time-Triggered Protocols

At the moment there are three different time-triggered protocols in discussion for electronic systems: TTP, TTCAN [4] and FlexRay [5]. Based on publicly available information, this section is to give a brief overview of TTCAN and FlexRay.

TTCAN

Time-triggered CAN is an extension of the well established event-triggered CAN protocol. Communication is based on periodic transmission of a reference message via a time master. This reference message is used to introduce a system wide reference time. Alternatively, the reference message can be triggered by an external event. Based on this reference, a number of so called *exclusive windows* – they are equivalent to the slots in a TDMA system – is defined. Each *exclusive window* is assigned to a specific node, which may send a data frame. In addition, the protocol defines *arbitrating windows*. Within these *arbitrating windows* all nodes on the network are allowed to transmit frames in accordance with the event-triggered CSMA/CA access scheme (which is used by CAN). See figure 4.

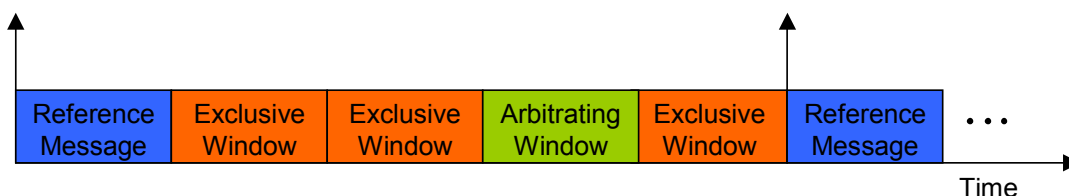


Figure 4. TTCAN time windows

As TTCAN preserves the original CSMA/CA channel access protocol for event-messages, it is inherently limited to a data transmission rate of 1 Mbit/s. Due to the fact that it provides only one communication

Safety-Critical Applications

channel and that the clock-synchronization is executed by a master slave algorithm, TTCAN cannot tolerate arbitrary single component failures.

An interesting feature of CAN is its acknowledgment mechanism, which uses the CSMA/CA principle. The sender transmits an acknowledgment bit at the end of the frame. If the message was received successfully, this is represented by a recessive state on the channel, which is set to the logical *true* condition. If any of the receiving nodes experienced a reception error, the sender changes the state to a dominant channel level, which indicates the logical *false* condition. This mechanism can ensure consistent message delivery for most cases (under specific fault scenarios the acknowledgment bit itself might be received inconsistently).

FlexRay

FlexRay has a time-triggered TDMA access scheme with a mini-slotting protocol for event-triggered transmission. The protocol supports different modes of operations for the clock synchronization. A distributed fault-tolerant mid-point algorithm is supported for the TDMA mode. The distributed mid-point algorithm serves as a reference for a set of TDMA slots with equal length, which is followed by a dynamic segment for events. The slot counter for the mini-slotting protocol is incremented during the dynamic segment. If a node wants to send an event message, it must wait until the slot counter has reached the unique ID which is assigned to the message. Event messages may have different lengths. The advantage of mini-slotting over CSMA/CA is that there is no restriction in communication speed. See figure 5.

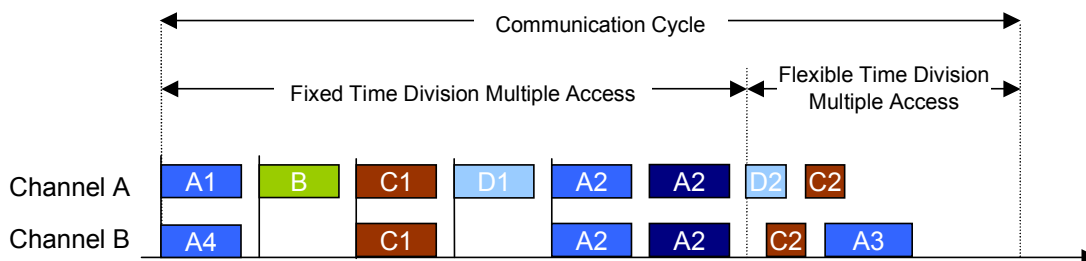


Figure 5. FlexRay communication cycle with static and dynamic part

Similar to TTP, FlexRay supports two redundant communication channels for fault tolerance. As there is no generic fault hypothesis for hardware faults in the communication layer, it is not known which types and frequencies of faults the protocol intends to tolerate. While FlexRay addresses specific hardware faults in the network architecture, networks diagnosis and error recovery are considered application-specific.

While TTP, CAN, and TTCAN have different levels of support for communication data consistency, there is no support for message acknowledgment, consistent message delivery, or the detection of inconsistencies in the communication system of the FlexRay communication protocol.

Safety-Critical Applications

Tabular Comparison Chart

	TTP	TTCAN	FlexRay
Transmission speed	25 Mbit/s; higher speeds possible	1 Mbit/s; higher speeds not possible due to CSMA/CD access	10 Mbit/s planned; higher speeds possible
Maximum frame length for application data	Variable for each node, up to 240 Bytes	Variable for each transmission, up to 8 Bytes	Up to 254 Bytes; for TDMA part all slots must be of equal size, for mini-slotting part individual message length possible;
Clock synchronization	Single fault-tolerant distributed formally verified offset correcting synchronization (with four or more nodes), optional rate correction to external reference time	Master-slave synchronization, cluster internal or driven by external events	Fault-tolerant distributed offset and rate correcting synchronization
Fault hypothesis / Fault tolerance	Tolerates arbitrary single hardware faults, two redundant communication channels	No systematic fault tolerance, no redundant channels	Tolerates many single hardware faults, one / two redundant channels (mixing possible)
Faults detected	Bit errors on the bus, transmit and receive faults	Bit errors on the bus, transmit and receive faults	Bit errors on the bus
Consistency support	Acknowledgment, membership, clique avoidance	Acknowledgment	Protocol checks for consistent cold-start operation but none for consistency during operation
Event handling strategy	Event channel on top of TDMA (CAN emulation)	TDMA plus arbitrating window for CSMA/CA	TDMA plus dynamic segment for mini- slotting
Composability of event messages	Temporally composable	Not temporally composable	Not temporally composable
Availability	Chips since 1998	Chips as engineering samples	Chips as engineering samples

More detailed comparisons of the time-triggered protocols including FlexRay, SAFEbus, Spider, and TTP can be found in [6] and [7].

Conclusion

The Time-Triggered Architecture and its fault-tolerant Time-Triggered Protocol TTP provide a solution for new functions, which address complexity and composability. TTP is based on the TDMA medium access strategy that ensures minimal jitter for the transmission of data. Such protocol functionalities as membership service, bus-guardian, clique avoidance, redundancy on the architecture level, and clock synchronization support development of large complex applications and safety-relevant systems.

The built-in membership service together with the acknowledgment mechanism ensures a consistent data delivery to all correct nodes and prompt error detection. This relieves the applications designer of implementing complex and computationally expensive consistency protocols and of verifying and certifying critical algorithms. The clique avoidance algorithm detects inconsistencies in the system to support a never-give-up strategy. Many of these algorithms have been formally verified.

TTP supports event-triggered communication for diagnosis, debugging, and calibration via event channels. The bandwidth is allocated to event messages. These event channels are temporally composable because the allocation of static bandwidth is done on a per node basis. The full consistency guarantees apply to the event channels, too. A CAN emulation on top of the event channels provides a clean software migration strategy for existing CAN software.

TTA addresses the requirements for next-generation functions. It provides a consistent computing platform, which supports a broad variety of applications, ranging from convenience to safety-critical functions. Several projects in the aerospace and railway industry increase the use of TTP.

References

- [1] TTP/C Specification. <http://www.ttagroup.org>.
- [2] Kopetz H.: Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, 1997.
- [3] Lamport L., Shostak R., Pease M.: The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, 4(3):382-401, 1982.
- [4] Road vehicles – Controller area network (CAN) – Part 4: Time triggered communication; Working Draft ISO 11898-4.
- [5] FlexRay Specification. <http://www.flexray.com>.
- [6] Kopetz H.: A Comparison of TTP/C and FlexRay. Research Report 2001, <http://www.vmars.tuwien.ac.at/frame-papers.html>.
- [7] Rushby J.: A Comparison of Bus Architectures for Safety-Critical Embedded Systems. SRI International. <http://www.csl.sri.com/papers/buscompare>, S. 32-35, 2001. A short version appeared in EMSOFT 2001: Proceedings of the First Workshop on Embedded Software. Ed. by Henzinger T., Kirsch Ch.. Springer-Verlag, S. 306–323, 2001

Safety-Critical Applications

Page 13

Contact

TTTech Computertechnik AG
Schoenbrunner Strasse 7
A-1040 Vienna, Austria
Tel.: +43 1 585 34 34-0
Fax: +43 1 585 34 34-90
E-mail: office@tttech.com
Web: www.tttech.com