

TTPos - The Time-Triggered and Fault-Tolerant RTOS

In the embedded systems market, the importance of composability, dependability, and hard real-time behavior is increasing steadily. At the same time, high volume markets demand substantial cost reductions. Conventional event-triggered solutions are not able to satisfy these requirements. The time-triggered operating system TTPos™ combines a tiny footprint with very fast task switching. TTPos provides time-triggered, priority based, cooperative preemptive scheduling; synchronization to a global time; error detection and application modes. The TTPos kernel is augmented by the node design tool TTPbuild™. TTPbuild supports the design of nodes participating in TTP® clusters. Based on the node design, TTPbuild automatically generates task schedules, the configuration for the operating system, and an optimized fault-tolerance and communication layer.

TIME-TRIGGERED ARCHITECTURE

There are two fundamentally different principles of how to control complex activities of real-time computer systems: time-triggered control and event-triggered control.

In a time-triggered system (TTP cluster), all activities are carried out at certain points in time which are known a priori. Accordingly, all computing nodes in time-triggered systems have a common notion of time, based on synchronized clocks. In contrast, in event-triggered systems all activities are carried out in response to relevant events external to the system.

Characteristics of the Time-Triggered Protocol (TTP)

The Time-Triggered Protocol TTP® is a communication technology developed at Vienna University of Technology and refined in several EU-funded research projects. It addresses the specific requirements of dependable embedded systems and integrates all services needed to design composable fault-tolerant hard real-time systems with minimal effort. TTP provides hard real-time message delivery, extensive error detection mechanisms, distributed clock synchronization, and a distributed membership service.

While the event-triggered paradigm was predominant up to now, the important advantages of a time-triggered architecture are now starting to be recognized:

- **Composability.** The various components of a software system can be developed independently and integrated at a late stage of software development. The smooth and painless integration of components facilitates the management of the ever increasing complexity of embedded real-time systems.
- **Predictable temporal behavior.** The system behavior is controlled by the progression of time based on a periodic pattern making the temporal system behavior predictable. This eases system validation and verification considerably.
- **Diagnosability and testing.** The interface between the components is exactly specified in both the value domain and the temporal domain, permitting the testing of each component in isolation.

Moreover, testing can be carried out without any probe effect.

- **Reusability of components.** In a time-triggered distributed architecture the "glue logic" needed to interconnect existing components into a new system is physically separate from the application software in the component. Existing components may therefore be reused in a new context without any modification of the tested and proven application software.
- **Fault-tolerance.** The replica deterministic behavior of time-triggered architectures supports the implementation of fault-tolerant systems by active redundancy. This facilitates the construction of highly dependable systems.

This technology supports the partitioning of a large system into a set of nearly autonomous subsystems with small and testable interfaces. It is therefore possible to build large and complex systems by integrating independently developed components with minimal integration effort.

Application Domains of TTP

Applications for TTP are highly dependable hard-real-time systems developed for the automotive industry, the aerospace sector, industrial control, building control, public transportation, robotics, and for medical electronics.

A major section of the automotive industry selected Time-Triggered Technology as the technology of choice for future demanding drive-by-wire applications. Being designed as a COTS (commercial off-the-shelf) technology for high volume markets, TTP offers a substantial cost advantage over application-specific solutions.

The Two-Level Design Framework

A development environment for TTP-based systems must support the precise specification of the temporal and functional interfaces between the subsystems of a time-triggered architecture. At the system-level, a system integrator (e.g., an automotive company) defines the subsystem functions and specifies the communication interfaces in the value and time domains precisely. At the subsystem-level, the component supplier retains complete control over all hardware and soft-

FAULT-TOLERANCE

ware design decisions as long as he complies with these interfaces.

The separation of concerns inherent to this two-level design framework translates directly into significant business benefits for all parties involved. The clear definition of responsibilities prevents the omission of essential functions as well as the duplication of efforts resulting in waste and potentially conflicting implementations.

For both system integrator and subsystem suppliers, the delimitation of responsibilities restricts the potential for conflicts and reduces the communication overhead.

For the subsystem supplier, the two-level design framework affords a high degree of independence. The supplier is unrestricted in his design choices, needs less system information, and benefits from a substantial decrease in requirements changes.

For the system integrator, the two-level design framework offers all the benefits of composability: The costs, time, and risk of system integration are reduced and the need for cost-intensive substitutes for temporal composability alleviated. By supporting faster and more precisely estimable turn-around times, the approach permits a shorter time-to-market.

Figure 1 shows TTPechnology's software development environment that implements the described two level design framework. It includes a cluster design tool (TTPplan™) and a monitoring tool (TTPview™) on the cluster-level, as well as node-level tools for downloading (TTPload™) and for fault-tolerance layer generation and operation system configuration (TTPbuild™).

OPERATING SYSTEM REQUIREMENTS

The design of TTPos was determined by the most pressing requirements of the high-volume automotive market, specifically

- **Efficiency.** The operating system must use a minimum of RAM, ROM, and CPU cycles.
- **Robustness.** The operating system must be demonstrably robust and error-free. Any error in the operating system could result in damages of millions of dollars.
- **Support for fault-tolerant hard real-time systems.** The operating system must support error detection and deadline monitoring.
- The operating system must be designed as to fully support the development of **composable, maintainable, and reusable** application software.

Given these requirements, the functionality of the operating system was partitioned into two components

- The run-time (on-line) kernel TTPos.
- The design-time (off-line) component TTPbuild.

The following layer model diagram shows this separation of functions clearly.

OS FTL: Operating System and fault-tolerant layer
TTPftc: TTP Fault-tolerance and communication.

All functions amenable to be done off-line are moved

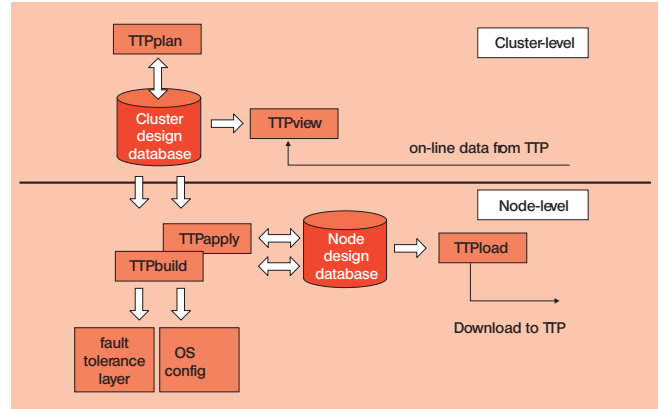


Figure 1. TTP Software Development Environment

out of the run-time kernel and are performed off-line by the node design tool TTPbuild. The kernel itself is thus reduced to bare bones - every ounce of fat trimmed away. In addition to performing most of the functions traditionally done by an operating system kernel, TTPbuild is also charged with globally optimizing the application software by using detailed information about the system design.

TTPos: THE EFFICIENT TIME-TRIGGERED OPERATING SYSTEM

The main responsibility of any operating system is the activation of tasks. TTPos uses a priority based scheduling policy; it supports both a preemptive and a cooperative scheduling discipline.

In TTPos, the entity of scheduling is a task. For each task, a deadline must be specified. TTPos monitors the runtime of each task and checks whether it meets its deadline or not. For performance reasons, TTPos does not enforce deadlines, it only checks them at the termination of the task.

To prevent a total blocking of the system by a high priority task, a watchdog service is provided by the operating system. This watchdog must be periodically updated by a low priority task of the application software. If the watchdog expires, TTPos restarts the system.

To increase the safety of the operating system, TTPos performs an additional consistency check. Every data structure used by TTPos is tagged by a unique 32-bit identifier. Every time information from a data structure is used, this tag is checked. When a corruption of the

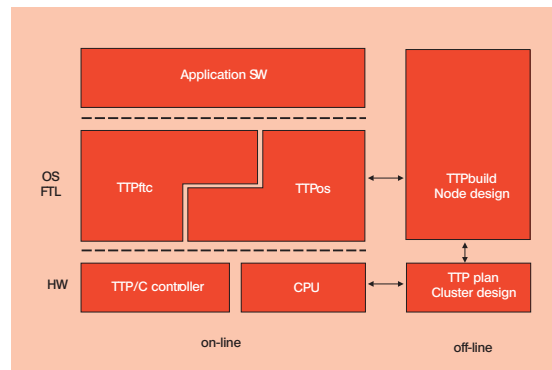


Figure 2. The Layer Model

FAULT-TOLERANCE

data element is detected, TTPos restarts the system.

TTPos supports application modes. An application mode defines a specific set of tasks activated in a pre-defined time schema. By defining more than one application mode, the temporal and or functional behavior of the system can be easily changed by switching between application modes.

TTPos is the world's smallest operating system supporting hard real-time and fault-tolerant applications. Table 1 gives some numbers for the size of RAM in

bytes required by TTPos for typical systems with a single application mode.

Even for a Motorola CPU32 core at 33 Mhz, the task switching time is only about 20 μ s. This includes the complete interrupt latency of the system time interrupt (about 10 μ s) and the runtime of the dispatcher (10 μ s).

TTPBUILD

TTPbuild is the node design tool complementing the TTPos kernel. TTPbuild supports the design of individ-

FAULT-TOLERANCE

Prio.	Tasks	Runtime optimized		RAM optimized	
		Check	No Check	Check	No Check
2	5	78	66	62	50
5	10	141	117	119	95
8	100	204	168	176	149

Prio.: number of priority levels
 Check: TTPos performs tag checks
 No Check: TTPos doesn't perform any tag checks

Table 1. Size of RAM required by TTPos

ual nodes of a TTP/C cluster. The application developer specifies the temporal behavior of the node's tasks and the state messages used for communication between those tasks. Based on this information, TTPbuild

- performs consistency checks on the node design,
- generates a task schedule,
- generates a fault-tolerance and communication layer (FTC layer) and
- generates the operating system configuration for TTPos.

The FTC layer completely hides the TTP/C bus from the application software of the node. For an application task, there is no difference whatsoever between a local message and a message transmitted over the TTP/C bus. This means that the distribution of tasks in a TTP/C cluster can be changed without any changes in the source code of the application tasks - re-running TTPbuild's FTC generator and re-linking the application accommodates such changes.

The FTC layer provides all services necessary for start-up and re-integration of nodes, for the handling of state messages, and for managing the application level membership in a TTP/C cluster.

The message handling done by the FTC includes the mapping of messages to TTP/C frames, the handling of bit messages, the management of replication and redundancy, the replica-deterministic agreement of

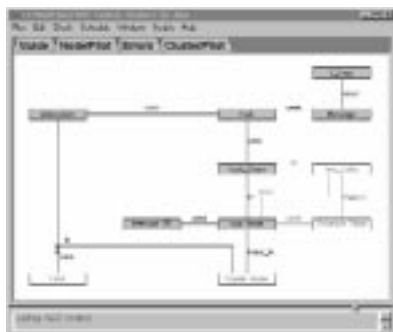


Figure 3. Editing ABS-control

replicated messages, the byte swapping necessary to support nodes with different byte orders on the same TTP/C bus, the necessary synchronization between the node and its associated TTP/C controller, and the handling of sender and receiver status per message.

TTPbuild uses the cluster-level and node-level design information to optimize the FTC layer. As the temporal characteristics of both the message transmission and task activation is known statically, TTPbuild is able to generate the minimum number of tasks to handle the message transmission and can avoid the processing of messages not used by a specific node.

CONCLUSION

TTPos is an extremely efficient RTOS for hard real-time systems. Its features meet the requirements of state-of-the-art highly-dependable, fault-tolerant real-time applications. With the provided tool-support (TTPbuild), developers are able to realize very cost-effective implementations within minimum time-to-market

RELATED LITERATURE

- [1] H. Kopetz. Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers. 1997. ISBN 0-7923-9894-7.
- [2] S. Poledna and G. Kroiss: The Time-Triggered Communication Protocol TTP™/C. Real-Time Magazine 98-4.
- [3] H. Kopetz and G. Grünsteidl. TTP--A Protocol for Fault-Tolerant Real-Time Systems. IEEE Computer, January 1994, pp. 14-23. 1994.
- [4] S. Poledna. Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism. Kluwer Academic Publishers. 1996. ISBN 0-7923-9657-X.
- [5] H. Kopetz. A Comparison of CAN and TTP. Proceedings of the IFAC Distributed Computer Systems Workshop, Como, Italy 1998. ■

Christian Tanzer, senior software architect at TTTech Computertechnik GmbH, received his masters degree in Astronomy from Vienna University in 1988. As a freelance software engineer he acquired extensive experience in industrial control applications.

Before joining TTTech, he developed the off-line tool OLT for the ERCOS real-time operating system as contractor for Bosch. In the X-By-Wire project, Mr. Tanzer extended the OLT to support distributed TTP/C based systems. Since 1990, Mr. Tanzer has acted as a speaker in seminars and workshops on object-oriented software engineering.

Martin Glück, senior operating system designer at TTTech Computertechnik GmbH, acquired in-depth experience in design and implementation of embedded controllers in the automotive sector. Before joining TTTech, he participated in the development of a next generation engine controller at Robert Bosch GmbH, Vienna.