
THE FTT-CAN PROTOCOL FOR FLEXIBILITY IN SAFETY-CRITICAL SYSTEMS

A NEW COMMUNICATION PROTOCOL FOR DISTRIBUTED EMBEDDED SYSTEMS ATTEMPTS TO FIND A COMPROMISE BETWEEN THE OFTEN-OPPOSING GOALS OF SYSTEM FLEXIBILITY AND SAFETY.

Joaquim Ferreira
Instituto Politécnico de
Castelo Branco

Paulo Pedreiras
Luís Almeida
José Alberto Fonseca
Universidade de Aveiro

..... Flexibility and safety are often considered conflicting concepts¹ because flexibility implies dealing with changing requirements that can, in turn, produce unpredictable and possibly unsafe operating scenarios. Therefore, some in the automotive and avionic system design industry believe² that a safety-critical system implies a fully static system in which all operating conditions are completely defined at pre-runtime. However, flexibility supports evolving requirements, simplifies maintenance and repair, and improves efficiency in system resources. The issue, then, becomes how to find a compromise achieving flexibility without jeopardizing system safety.

Achieving this compromise is particularly important in safety-critical systems that demand resource efficiency. For example, heavy pressure exists to reduce cost in automotive distributed computer control systems. Here, the communication infrastructure deserves particular attention because of the current trend toward encapsulating single functions in separate nodes. This fully distributed scenario has several advantages:

- dependability, with easy replication of nodes and definition of error-containment regions;

- composability, because the system is built by integrating nodes that constitute independent subsystems;
- scalability, through easy addition of new nodes to support new functionality; and
- maintainability, because of the architecture's modularity and easy node replacement.

However, a high level of distribution also leads to increased traffic on the network, which calls for bandwidth efficiency to meet temporal and throughput requirements.

Time vs. event

When reasoning about the efficient use of the network bandwidth, it is necessary to consider time- and event-triggered communication paradigms. In the automotive industry, the move toward the time-triggered paradigm seems clear with such system standard candidates TTP/C³ (time-triggered protocol—for class C safety requirements), Time-Triggered Controller Area Network (TT-CAN),⁴ and FlexRay.⁵

According to the time-triggered paradigm, all communication occurs at predetermined times. Hence, transmission schedules are predetermined, leading to more foreseeable temporal behavior. This is particularly suited to

the transmission of periodic message streams. Moreover, by setting an adequate relative phasing between message streams, designers can control the collisions at medium-access level, eliminating, or bounding, network-induced mutual interference. This feature leads to composability with respect to the temporal behavior because elimination of mutual interference allows the temporal behavior of a subsystem when isolated, to remain unaffected upon integration in the global distributed system. Relative phasing control also provides improved temporal behavior under heavy traffic conditions. The feature enables better packing of transmissions in the communication system. This means shorter worst-case response times to communication requests and more efficient bandwidth utilization. Finally, the time-triggered paradigm supports detection of connectivity failures at the receiver because message-arrival times are known across the system.

Despite these positive properties, the time-triggered paradigm also has a downside: It is inefficient concerning average network utilization when one or more nodes generate messages in a sporadic way and require fast response. For example, with alarms, event-triggered communication is more efficient because senders transmit only in response to significant state changes, that is, events.

Of course, the event-triggered paradigm also has a downside. It does not support composability with respect to the temporal behavior because the asynchrony of node transmissions leads to mutual interference whenever nodes are integrated in the system. In addition, the requirements for worst-case communication can become higher when one considers the situation in which all nodes attempt to communicate simultaneously. Thus, either we must use more bandwidth or the response times to communication requests will be longer. Finally, the remaining nodes will not immediately detect a fail-silent failure, for example, caused by a system shutdown in one node. To allow remote detection of these failures, event-triggered systems typically use a heartbeat—a timed mechanism using coarse synchronization.

Therefore, from the point of view of efficient bandwidth use, the time-triggered paradigm works better for periodic communication; the

event-triggered paradigm for sporadic communication. However, many practical applications of distributed embedded control, such as automotive systems, require the exchange of information of both a periodic and sporadic nature. The former is typically associated with control loops and the latter with alarms and management. Although these two types of traffic can be conveyed over fully event-triggered systems—such as typical CAN-based systems, or fully time-triggered systems, such as TTP/C—network efficiency suggests using a combination of the two paradigms. Such a combination must enforce temporal isolation between both types of traffic. Otherwise, the asynchrony of the event-triggered traffic will spoil the properties of the time-triggered traffic because of mutual interference. A typical way to achieve temporal isolation is to assign nonoverlapping time windows exclusively to the transmission of each type of traffic. Both TT-CAN and FlexRay support such a combination.

Finally, as the name suggests, time drives time-triggered communication. Thus, a coherent notion of time must be enforced across the system. Since nodes are asynchronous by nature because of drift and limited accuracy of local clocks, they require specific systemwide synchronization mechanisms. These, in turn, consume further bandwidth, although normally just a small amount. Conversely, event-triggered communication does not explicitly require a global notion of time or systemwide synchronization mechanisms.

Flexibility and safety

Generically, the term *flexibility* means the ability to change form. Since the term can apply to many different areas, we need to define exactly to which form we are referring. For example, concerning embedded communication systems, we can refer to such aspects as physical media, topology, bit encoding, live insertion support, mode changes support and rapid download of new application software, and dynamic communication requirements support. The number of options the system supports determines its degree of flexibility.

Among these aspects, flexibility in dynamic communication requirements brings up higher concerns regarding safety. This is because a change in communication requirements could lead to a network overload and

consequent timing failures. Nevertheless, limiting the extent of changes to guarantee the timely behavior of the network is possible.

Currently, most distributed control systems used in automobiles are not safety-critical because they merely enhance the performance of conventional mechanical systems (for example, antilock braking systems and assisted steering). However, in the near future, fault-tolerant computer-control systems that use active redundancy and can operate in the presence of faults will replace the conventional mechanical link between driver and actuators. These systems, typically called X-by-wire, will indeed be safety-critical.

Despite eventual safety-critical requirements, network-traffic-parameter flexibility remains important for greater network resource efficiency. In this context, we consider event-triggered communication systems flexible, that is, they react promptly to communication requests issued at any instant and to instantaneous (variable) communication requirements. Conversely, time-triggered systems, such as TTP/C, are not so flexible; communication occurs at predefined instants exclusively. These systems don't take into account runtime variations in application communication requirements. However, some degree of flexibility can be achieved with TTP/C, since it allows dynamic changes within a set of predefined operational modes. Nevertheless, all modes must be based on the same time-division multiple access (TDMA) round and be preconfigured in all nodes. TTP/C also supports the dynamic addition of nodes and messages if designers have reserved the required space in the TDMA round and cluster cycle. This, however, keeps extra bandwidth allocated, even when not immediately needed, making it bandwidth inefficient. Therefore, the flexibility of TTP/C in terms of support to dynamic communication requirements is still limited.

The systems that combine both event- and time-triggered paradigms present an intermediate level of flexibility due to the event-triggered part (this holds true for both TT-CAN and FlexRay). FlexRay makes the claim of flexibility for this very reason. Notice, however, that in both systems, the time-triggered traffic requires pre-runtime static definition and is thus inflexible. This inflexibility

decreases using the same technique as applied in TTP/C (reserving extra bandwidth at design time for runtime use in accommodating new messages).

A new protocol

We propose the flexible time-triggered communication on CAN (FTT-CAN) protocol⁶ to overcome inflexible handling of time-triggered traffic in distributed computer-control systems. FTT-CAN combines both time- and event-triggered paradigms in an efficient way, taking advantage of the underlying medium-access control of CAN. The main feature of the protocol supports online changes to time-triggered communication requirements, such as adding new messages, removing existing ones, or adapting their parameters such as period. The protocol includes a dynamic traffic scheduler that takes into account current communication requirements. An online admission control rejects any change request that might compromise the continued timely behavior of the system.

The main aim of the protocol is to support efficient network utilization when subsystems undergo changes in operating modes. It does this by sharing network bandwidth. Each subsystem only uses the bandwidth that it currently needs. The higher efficiency of this approach is obvious when several subsystems do not operate simultaneously, for example, with cruise control and antilock braking in a car.

The dynamic traffic scheduler supports extra flexibility in two ways: The scheduling policy can change online, for example as a response to an overload, and omitted messages, due to node failures or transmission errors, can be rescheduled within the respective deadlines using backward error-recovery techniques. TTP/C, TT-CAN, or FlexRay do not support any of these FTT-CAN features because of their static approach to managing time-triggered traffic. Table 1 compares efficiency for both types of traffic and for dynamic time-triggered communication requirements between FTT-CAN and the other protocols.

TTP/C receives the lowest score for efficient handling of event-triggered traffic because it handles this traffic type in the same way as time-triggered traffic, using dedicated time windows called *event channels*. If there is no message to convey, the system wastes that win-

dow of bus time. FlexRay uses defined bus windows shared by all nodes to send the event-triggered traffic, making it more efficient than TTP/C. FlexRay, however, uses a collision-free medium-access mechanism, which is equivalent to assigning different wait times to asynchronous messages according to their priority. This mechanism can result in substantial bus idle time

when there are ready-to-send, yet low-priority messages. TT-CAN and FTT-CAN also allow event-triggered traffic to share defined bus windows. Moreover, they both take advantage of the deterministic collision resolution mechanism of CAN, which assigns priorities embedded in message identifiers and lets nodes try to immediately send messages within the allowed windows. This mechanism handles event-triggered traffic more efficiently and flexibly than those used in TTP/C and FlexRay.

For efficiency in handling time-triggered traffic, the figures become inverted. TTP/C receives the highest score because it uses a highly packed format with several messages piggybacked in a single frame, and it doesn't use explicit message identifiers or addresses. Messages are identified by the time at which they are transmitted. On average, TTP/C supports data efficiencies (the ratio between time to transmit data bits and bus time) between 60 and 80 percent, with a theoretical limit of around 95 percent. In the current version of FlexRay,⁵ a new maximum data payload of 246 bytes per frame leads to data efficiency figures close to those of TTP/C. As for CAN-based systems, because of its maximum of 8 bytes per frame, average efficiency figures are between 30 and 40 percent, with a theoretical limit of around 50 percent.

FTT-CAN

We first presented the original idea underneath the FTT-CAN protocol at the 19th IEEE Real-Time Systems Symposium.⁶ The main features of the protocol are

- flexible handling of time-triggered traffic;
- support for on-the-fly changes both on the message set and scheduling policy;
- online admission control of change

Table 1. Traffic and communication handling efficiency in different protocols.

Protocol	Relative efficiency		
	Event-triggered traffic	Time-triggered traffic	Dynamic time-triggered communication requirements support
TTP/C	Lower	Higher	Lower
FlexRay	Medium	Higher	Lower
TT-CAN	Higher	Lower	Lower
FTT-CAN	Higher	Lower	Higher

- requests for the time-triggered traffic;
- indication of temporal accuracy of time-triggered messages;
- support for different types of traffic (time triggered, event triggered, hard real-time, soft real-time and non-real-time);
- temporal isolation, that is, time and event-triggered traffic do not disturb each other; and
- efficient use of network bandwidth.

As the name suggests, we built the FTT-CAN protocol on top of CAN. We realized, however, that we could abstract away the underlying network, leading to the so-called network-independent FTT paradigm. The application of this paradigm to different networks results in a set of protocols of which FTT-CAN is an example. Recently, we applied the same paradigm to Ethernet, leading to FTT-Ethernet.⁷

Any protocol following the FTT paradigm must fulfill the properties referred to previously in the list. We achieve this compliance by enforcing a synchronized cyclic framework based on four main features: centralized synchronization, centralized traffic scheduling, master/multislave transmission control, and a producer-distributor-consumer cooperation model.

Centralized synchronization builds, in a simple way, a coherent notion of time across the system. The bus time is organized as an infinite sequence of fixed duration windows called elementary cycles (ECs)—the duration is defined at pre-runtime. A special trigger message (TM) indicates the start of each EC. A particular master node broadcasts the TM, which synchronizes all nodes in the system. The duration of the EC sets the temporal res-

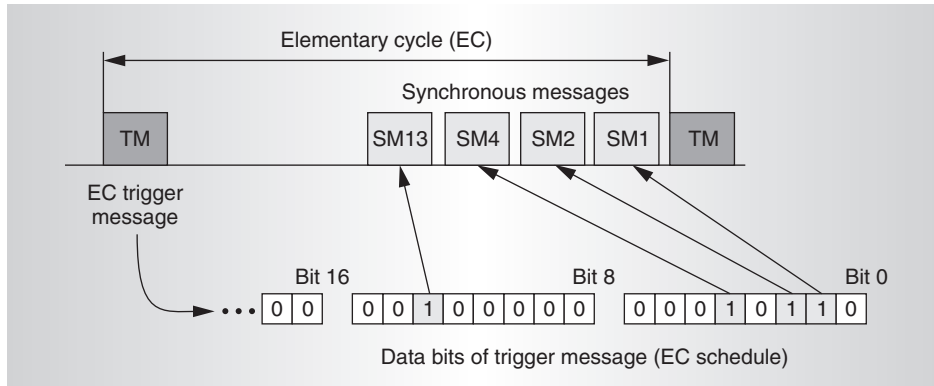


Figure 1. Master/multislave access control. Slaves produce synchronous messages according to an elementary-cycle schedule conveyed by the trigger message. If the x data bit is 1, then message x is produced in this EC; if it is 0, then message x is not produced.

olution of the time-triggered part of the communication system. Consequently, the transmission periods of the traffic are integer multiples of the EC duration.

Centralized traffic scheduling allows locating both the communication requirements and the message scheduling policy in one master node, facilitating online changes to both and achieving a high level of flexibility. Such centralization also facilitates implementation of online admission control in the master node, guaranteeing traffic timeliness for changing communication requirements.

Master/multislave transmission control allows enforcing the traffic timeliness in the bus and achieving a high-bandwidth efficiency. In the first aspect, typical of master-slave transmission control, the master explicitly tells each slave when to transmit, thus enforcing traffic timeliness. The second aspect results from the protocol using a master-slave transmission control on a per (elementary) cycle rather than a per message basis. A single master message triggers several slave messages in a single EC, thus reducing the number of control messages and consequently increasing bandwidth efficiency. The TM data field conveys the so-called EC schedule, that is, the messages that must be transmitted within the respective EC, as Figure 1 shows. These messages are transmitted synchronously within the EC's framework.

As opposed to typical time-triggered systems, a master/multislave system does not explicitly need a global distributed time base. The master node tells the nodes when to transmit time-triggered messages. The system uses a centralized

time base within the master only, to trigger transmissions via the EC schedule broadcast in the TMs. Furthermore, the centralized scheduling and master/multislave features facilitate complex or costly scheduling policies implementation, with no impact on any node except the master. All other nodes promptly follow the EC schedules. For example, we have efficiently implemented earliest-deadline-first scheduling of messages using the FTT paradigm.⁸

Finally, the time-triggered communication in FTT-CAN follows the producer-distributor-consumer cooperation model,⁹ such as the periodic traffic in the world factory instrumentation protocol (WorldFIP). The nodes that generate data relevant for other nodes transmit (produce) the data when the master—that is, the distributor—tells them. The nodes that need a particular datum—for example, the temperature of sensor X , or the pressure of sensor Y —scan the network traffic for the information and receive it, that is, consume it, once detected. This model inherently supports one-to-many communication and requires source addressing, which is the native addressing mechanism in CAN. With respect to the combination of time- and event-triggered traffic, the EC is divided into two distinct phases—one for each type of traffic—called synchronous and asynchronous windows. The event-triggered traffic is asynchronous because the application can request the respective transmission at any time regardless of the EC framework.

These FTT-CAN features are independent of the underlying network and, thus, are present in any FTT protocol. There are, however, several network-specific features concerning the traffic organization and timeliness enforcement within each EC and the support for event-triggered traffic. In this particular case, the FTT-CAN protocol takes advantage of the CAN-prioritized, distributed media access control (MAC) with its nondestructive bitwise arbitration mechanism. For example, the synchronous messages scheduled for a given EC can be triggered simultaneously

within the respective synchronous window. It is up to the distributed MAC of the CAN to resolve collisions and serialize message transmissions.

For concerns about event-triggered (asynchronous) traffic, the situation is similar. Slaves with pending requests might try to transmit immediately, but only within the asynchronous window of each EC, shown in Figure 2. This scheme, similar to the arbitration windows in TT-CAN, allows a very efficient combination of time- and event-triggered traffic, resulting in low communication overhead and shorter response times.

As seen in Figure 2, the synchronous window is processed after the asynchronous one, close to the end of the EC. As a result, the start of the synchronous window varies depending on the length of the messages scheduled for that EC. The TM also conveys the instant in which the synchronous window of each EC must start, relative to the reception of the TM. Nevertheless, the protocol establishes a maximum duration for the synchronous windows and, correspondingly, a maximum bandwidth for that type of traffic. This reserves minimum bandwidth for the asynchronous traffic.

To enforce a strict temporal isolation between synchronous and asynchronous traffic, FTT-CAN prevents the start of transmissions that won't complete within the respective window. We achieve this by removing from the network controller transmission buffer any pending request that cannot complete within that interval, keeping it in the transmission queue. As a consequence, a short amount of idle time might appear at the end of the asynchronous window (α in Figure 2). Variation in the stuff bits used in the physical encoding of CAN messages might make a short amount of idle time appear at the end of the synchronous window.

Master node

A synchronous requirement table (SRT) residing in the master node holds the communication requirements of time-triggered traffic. Each SRT entry contains the description of one periodic (synchronous) message stream. These parameters include identification; data byte length; maximum transmission time; period length; deadline; relative phasing; message stream value to the application; attributes indi-

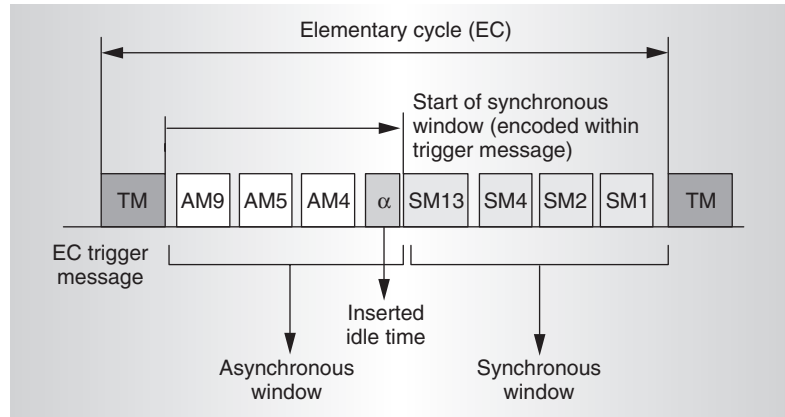


Figure 2. The double phase elementary cycle with synchronous and asynchronous windows.

cating acceptable operations over the message stream; and a list, or range, of specified parameter values. These attributes or ranges limit flexibility on a per stream basis. They can indicate that no online changes are accepted for a given stream (static stream), or that the stream properties might be changed online (dynamic stream), such as adapting its period or relative phasing. In this case, the adapted values must always be within the list or range specified.

The central component within the master internal architecture, depicted in Figure 3 (next page), is the SRT. The traffic scheduler executes online, recurrently scanning the SRT to build the EC schedules broadcast on the network. Therefore, the scheduler takes into account any change in the SRT when next invoked. This makes the system highly flexible for dynamic time-triggered communication requirements. All change requests, however, go through the online admission control block before acceptance in the SRT. This enforces timely behavior of the communication system and assures that changes don't go beyond the flexibility constraints described by the attributes.

As stated previously, the scheduler executes online, with runtime computing costs directly related to the scheduling policy. The scheduler can execute in low-processing-power microcontrollers (for example, a demonstrator was built using Philips 80C592 controllers). When the scheduling incurs high computing costs, we use either a more powerful CPU or a special-purpose scheduling coprocessor. The respective cost penalty is confined, however, to the master node.

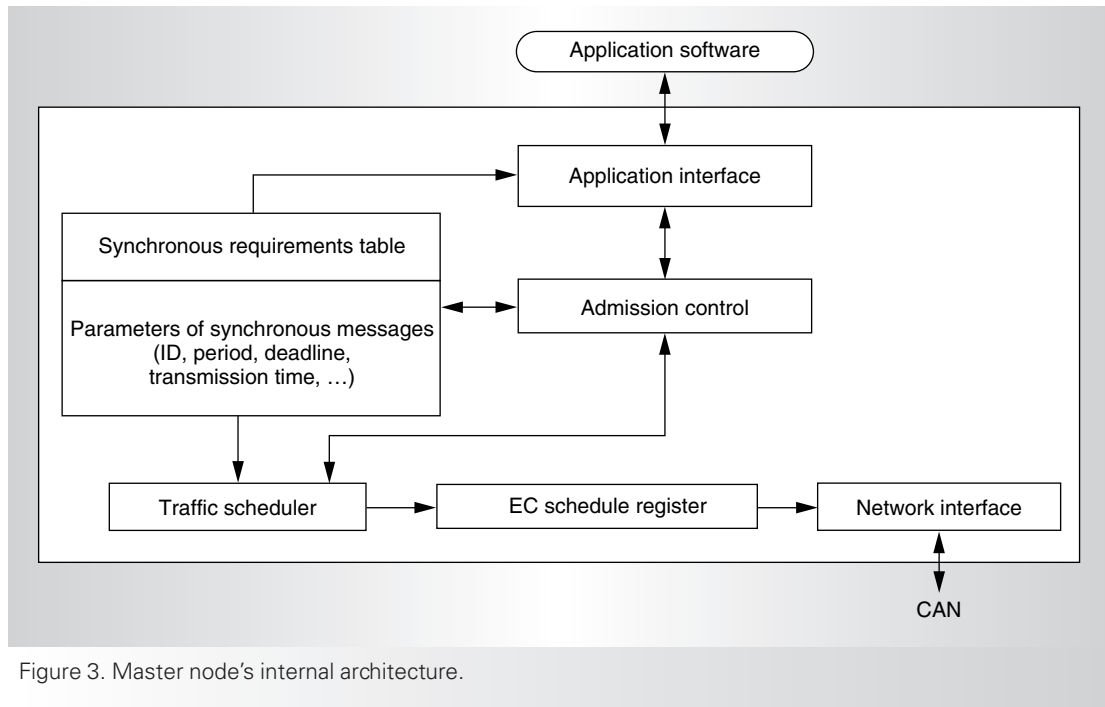


Figure 3. Master node's internal architecture.

Two subsystems—the synchronous messaging system (SMS) and the asynchronous messaging system (AMS)—manage the communication services delivered to the application by FTT-CAN. The SMS offers services based on the producer-distributor-consumer model and handles the synchronous traffic with autonomous control, that is, the network interface exclusively carries out transmission and reception of messages without intervention from the application software. Shared buffers pass data to and from the network. The AMS offers send-and-receive basic services only. The AMS follows an external control approach according to which the application directly initiates the transmission. The SMS delivers services to manage the SRT, making use of special-purpose asynchronous messages.

Fault hypothesis

We need to thoroughly determine the impact of network errors and node failures to use FTT-CAN in safety-critical applications. We use fault tolerance techniques to limit that impact, which, however, increases system cost. The particular configuration of the FTT-CAN design lets the designer determine the degree of complexity or fault tolerance.

The fault hypothesis that we consider in the protocol takes into account physical faults, such

as those related to electromagnetic interference, defects, or damage. Moreover, as is common in all bus-based systems, the hypothesis does not tolerate partitioning unless the bus is replicated. In addition, we consider every node in the system as fail-silent—that is, it transmits either correctly or not at all. A node can be fail-silent in the time domain where transmissions occur at the right instants only, or in the value domain where messages contain correct values only. FTT-CAN enforces fail silence in the time domain in all nodes by using bus guardians: autonomous devices with respect to the node host processor that enforce adequate timing in the node transmissions. Upon reception of a TM, FTT-CAN bus guardians synchronize their internal timers, decode the trigger-message contents (duration of synchronous and asynchronous windows and EC schedule), and block transmissions with the wrong timing. In the value domain, the protocol considers fail silence in the master node only. In this case, the scheduler and the SRT are replicated internally. Whenever the EC schedule built by the scheduler replica does not match the one built by the primary scheduler, no TM is generated. The remaining nodes are application specific—if this fail-silent behavior in the value domain is desired, it must also be implemented within the node.

In a replicated bus scenario, each node attaches to both redundant buses using independent network controllers. Every transmission carried out on both buses occurs with the automatic retransmission upon network errors disabled. The system considers a transmission successful if at least one of the transmissions in both controllers was successful. On the reception side, all nodes receive messages from all redundant media and discard the replicas by comparing the identifiers.

Transmission errors

Electromagnetic interference, bad connectors, or cabling short circuits can cause transmission errors. When this happens, the original CAN protocol triggers an automatic retransmission. This feature provides reliable communication. Timeliness, however, suffers because the retransmission takes place independently of the temporal validity of the respective message.

In FTT-CAN, unless replicated buses are used, there is no need to disable the automatic retransmission since its duration is limited by the duration of the window where the retransmission takes place. All transmission activity is suspended at the end of each window in every EC, including retransmissions. This leads to error confinement within both subsystems—any error in the SMS does not affect the AMS and vice versa. Within each subsystem, the designer can allocate extra time to cope with errors as forecast by an appropriate error model¹⁰ (this is fault-tolerant scheduling). Possible errors, however, beyond the error model might occur. An active mechanism based on the master node handles these errors according to a never-give-up strategy. A bus monitoring mechanism looks for missing synchronous messages. If the respective producer node did not send a particular message instance, the master tries to reschedule the message for future ECs¹¹ within the message deadline, in a best-effort approach. The TM always transmits in single-shot mode. If any error occurs during the TM's transmission, it will not effectively broadcast, resulting in an omission handled with master replication.

Replication and synchronization

The reception of the TM synchronizes the entire FTT-CAN-based distributed system.

When no such message occurs, a temporary loss of connectivity occurs. One or more backup masters prevent such a situation. These masters monitor the network looking for TMs. Whenever the next TM is delayed more than a given tolerance, the backup masters try to transmit the missing TM. The first backup master that succeeds becomes the primary master. All others receive the recovered TM and become, or remain, backup masters. In cases of transmission of an error-corrupted TM, the system considers it an omission and both active and backup masters will retry transmission.

Fundamental to this design is the synchronization between primary and backup masters. Since schedulers running on the masters are dynamic, it must be guaranteed that in each EC they generate similar schedules. Thus, in every EC, all backup masters compare their schedules with the one conveyed by the TM. Moreover, they also compare a short cyclic sequence number encoded in the TM. If a backup detects an inconsistency, it issues a synchronization request to the primary. This causes the current primary master to download the SRT and the relative phasing information necessary to resume scheduling synchronously. This process might take a few ECs depending on the size of the SRT and current network utilization. During this time, the resynchronizing backup remains inactive.

At startup, a relatively simple synchronization procedure assures that only one master becomes the primary one. Any starting (or restarting) master waits for a preconfigured time, that is, an initial period lasting for a few ECs, where it listens for a TM. If one is detected, this indicates that the system is running and that a primary master is working. Thus, the starting master issues a synchronization request and enters backup mode as soon as the synchronization process completes. If no TM occurs during the initial listen period, the starting master assumes that it is the first master activated, starts the scheduler, and tries to enter primary mode by transmitting a TM. However, other starting masters might also compete simultaneously to become primary. Therefore, all of the starting masters check for successful transmission. The master that succeeds becomes the primary. The others receive the transmitted TM, issue a synchronization

request, and enter backup mode.

Finally, a membership service enables the system to know which backup masters are present as well as their operational status. This service uses a regular query, broadcast by the currently active primary master, to which all operational backup masters respond with information about their status. This service uses special-purpose asynchronous messages whose transmission frequency is setup at pre-runtime.

The adoption of CAN to support the FTT paradigm has several advantages. It greatly simplifies efficient handling of event-triggered traffic because of the deterministic collision resolution mechanism embedded in the respective MAC. Moreover, CAN network controllers and cabling are relatively inexpensive, an important aspect in the automotive industry. And last, but not the least, the relatively robust physical layer with respect to error detection and tolerance of physical faults lets FTT-CAN systems operate in harsh environments.

Nevertheless, the maximum standardized transmission speed on CAN is 1 megabit per second, which limits available bandwidth and puts more emphasis on efficient bandwidth utilization. TT-CAN suffers from the same limitation, but both FlexRay and TTP/C use considerably higher transmission speeds. FlexRay's transmission speed is 10 Mbps; TTP/C's is 2 to 25 Mbps. Of course, these larger bandwidths allow a more relaxed approach toward efficient bandwidth utilization. Nevertheless, despite being a controversial topic within the automotive systems design industry, with the current trend of increasing the number of subsystems in a car, even those bandwidths will probably become scarce in the near future.

Therefore, to cope with an ever-growing bandwidth demand, we have recently proposed a new FTT-based protocol that is supported on Ethernet (FTT-Ethernet⁷). Such as FTT-CAN, this protocol shares the properties inherent to the FTT paradigm. It enforces a collision-free MAC on Ethernet and allows taking advantage of scalable transmission speeds, low-cost network adapters, and hierarchical topologies based on hubs and switches. Safety issues are being currently addressed. Once completed, the FTT-Ethernet protocol will support more bandwidth-demanding applications such as

multimedia systems, computer vision, navigation systems, and broadband Internet access, all integrated in a flexible and safe communication framework. MICRO

References

1. H. Kopetz, *Real-Time Systems Design Principles for Distributed Embedded Applications*, Kluwer Academic, Boston, 1997.
2. *Minimal Operational Performance Standards for Avionics Computer Resources*, ARINC RTCASC-182/EUROCAE WG-48, RTCA, Washington, D.C., 1999.
3. *TTP/C Protocol*, version 0.5, TTTech Computertechnik, Vienna, 1999.
4. *ISO/CD11898-4, Road Vehicles—Controller Area Network (CAN)—Part 4: Time-Triggered Communication*, Int'l Organization for Standardization, Geneva, 2000.
5. F. Bogenberger, B. Müller, and T. Führer, "Protocol Overview," *Proc. FlexRay Int'l Workshop*, 2002; <http://www.flexray.com/html/infws402.htm> (current July 2002).
6. L. Almeida, J. Fonseca, and P. Fonseca, "Flexible Time-Triggered Communication on a Controller Area Network," *Proc. Work-In-Progress Session, 19th IEEE Real-Time Systems Symp. (RTSS)*, 1998; <http://www.cse.unl.edu/rtss98wip/proceedings/> (current July 2002).
7. P. Pedreiras, L. Almeida, and P. Gai, "The FTT-Ethernet Protocol: Merging Flexibility, Timeliness and Efficiency," *Proc. 14th Euromicro Conf. Real-Time Systems*, IEEE CS Press, Los Alamitos, Calif., 2002, pp. 152-160.
8. P. Pedreiras and L. Almeida, "A Practical Approach to EDF Scheduling on CAN," *Proc. IEEE Workshop Real-Time Distributed Embedded Systems (WRTDES)*, 2001; <http://rtde.cs.tamu.edu> (current July 2002).
9. J.P. Thomesse and M. Leon-Chavez, "Main Paradigms as a Basis for Current Fieldbus Concepts," *Proc. Int'l Conf. Fieldbus Technology (FeT 99)*, Springer-Verlag, Vienna, 1999, pp. 2-15.
10. N. Navet and Y.-Q. Song, "Design of Reliable Real-Time Applications Distributed over CAN (Controller Area Network)," *Proc. IFAC Symp. Information Control in Manufacturing (INCOM)*, Elsevier Science, Oxford, England, 1998, pp. 391-396.
11. J. Ferreira et al., "FTT-CAN Error Confinement," *Proc 4th IFAC Conf. Fieldbus*

Technology (FeT), Elsevier Science, Oxford, England, 2001, pp. 8-15.

Joaquim Ferreira is currently a PhD student at the Universidade de Aveiro, Portugal. He is also *professor adjunto* at the Instituto Politécnico de Castelo Branco, Portugal. His research interests include real-time communication systems, fieldbuses, fault tolerance, and digital design. He has a *licenciatura* degree in electronics and telecommunications engineering and an MSc in electrical engineering from the Universidade de Aveiro, Portugal.

Paulo Pedreiras is a PhD student at the Universidade de Aveiro, Portugal. His research interests include real-time communication systems, scheduling, fieldbuses, and real-time operating systems. He has a *licenciatura* degree in electronics and telecommunications engineering from the Universidade de Aveiro, Portugal.

Luís Almeida is assistant professor in the Department of Electronics and Telecommunications at the Universidade de Aveiro, Portugal. His research interests include real-time networks for distributed industrial/embedded systems and navigation control for mobile robots. He has a *licenciatura* degree in electronics and telecommunications engineering and a PhD in electrical engineering from the Universidade de Aveiro, Portugal. He is an IEEE Computer Society member.

José Alberto Fonseca is an associate professor in the Department of Electronics and Telecommunications at the University of Aveiro, Portugal. His research interests include embedded systems, distributed systems, and industrial communications. He has a *licenciatura* degree in electronics and telecommunications engineering and a PhD in electrical engineering from the University of Aveiro, Portugal. He is a member of the IEEE Industrial Electronics Society.

Direct questions and comments about this article to Luís Almeida, Departamento de Electrónica e Telecomunicações, Universidade de Aveiro, Campo Universitário de Santiago, 3810-193, Aveiro, Portugal; lida@det.ua.pt.

Coming Next Issue

SEPTEMBER-OCTOBER
2002

Systems on a Chip

Moore's law is enabling the design of multi-million transistor SOCs. Luciano Lavagno of the Politecnico di Torino presents articles on complexity issues faced by designers and architects related to integrating these SOCs. Topics will include

- Chain: A Delay-Insensitive Chip Area Interconnect,
- Coping with Latency in SOC Design,
- Eclipse: A Scalable High-Performance Computing Solution for Networks on a Chip,
- Octagon: A High-Performance Interconnect Architecture for Networking Systems on Chips,
- A Hierarchical Test Methodology for Systems on a Chip,
- Efficient Construction of Aliasing-Free Compaction Circuitries, and
- A New Polymorphous Computing Fabric.

IEEE Micro

serves your interests

IEEE
micro
The magazine for chip and silicon systems designers