

IMPROVING FLEXIBILITY AND RESPONSIVENESS IN FTT-CAN WITH A SCHEDULING COPROCESSOR

Ernesto Martins, José A. Fonseca
{evm, jaf}@det.ua.pt

DET – IEETA, Universidade de Aveiro
P-3810-193 Aveiro, Portugal

Abstract: Fieldbus distributed systems are increasingly required to accommodate on-line changes in the messages they convey. The FTT-CAN protocol relies on the planning scheduler to achieve this kind of flexibility, but its response time to changes in the message set is severely limited by the plan duration which cannot be set arbitrarily short. This paper presents a scheduling coprocessor solution for FTT-CAN, aimed at improving its flexibility and responsiveness. The coprocessor generates message schedules according to one of three different scheduling policies, includes an on-line schedulability analyser function and can be used also in other fieldbuses using centralised scheduling. The paper focuses on the coprocessor functionality and interface. Preliminary performance figures showing its feasibility are also presented. *Copyright © 2001 IFAC*

Keywords: Real-time communication, fieldbus, traffic control, scheduling, coprocessors.

1. INTRODUCTION

Modern distributed real-time communication systems usually require a certain level of operational flexibility to accommodate on-line configuration changes (Kopetz, 1997). These changes include the addition of new message streams or the changing of existing message parameters, and must be normally preceded by a schedulability analysis to guarantee they don't jeopardise the timeliness properties of the system.

The FTT-CAN (Flexible Time-Triggered communication on CAN) protocol presented by Almeida *et al.* (1999b) at FeT '99 was introduced partially to fulfil this requirement. To achieve a dynamic expression of communication requirements, FTT-CAN relies on a centralised planning technique, known as the planning scheduler (Almeida *et al.*, 1999a), and an on-line admission control protocol (Almeida, 1999c).

The planning scheduler builds on-line static schedules for consecutive periods of time called the

plans. The creation of a plan table is overlapped with the dispatching of the previous. Message parameters can be changed between scheduler invocations, which occur only once each plan. Thus, when compared with a typical dynamic scheduler which is invoked every time unit to determine the highest priority message, the planning scheduler exhibits a considerable lower run-time overhead. This was, in fact, a major reason for its adoption in the FTT-CAN protocol, due to the limited processing power generally available in fieldbus nodes.

All changes made in the message set are reflected on the system with a delay equal to the time to dispatch one to two plans. Thus, the response time is a function of the plan size. If the changes come from a human operating the system, a reaction time of a few seconds is normally tolerable, and thus a large plan size may be used. However, if the change requests come from the system itself (e.g. a device requesting an increase in the sampling frequency of a message), a responsiveness of a few milliseconds may be required, dictating the use of a shorter plan.

With a short plan however, the scheduler must be invoked more often, imposing therefore a higher overhead. If the required response is fast enough, the plan size needed might be so small that the overhead level makes scheduling unfeasible. In a practical situation this may limit the system to accept configuration changes issued by an operator only (Almeida., 1999c).

To address this issue, a new mechanism was recently introduced in FTT-CAN that allows changed message streams to bypass temporarily the scheduler, reducing the response time without changing the plan size (Pedreiras and Almeida, 2000). A different approach to achieve higher levels of reactivity without increasing the protocol complexity, is to transfer the scheduling task to dedicated hardware, leaving the node CPU with just the dispatching task.

This paper presents a traffic scheduling and schedulability analyser coprocessor aimed at improving the flexibility and responsiveness of FTT-CAN-based fieldbus systems. The next section starts with a discussion of the basic approaches considered in the design of such a coprocessor. Next, the basic coprocessor functionality is presented, followed by a short overview of its architecture and an in-depth description of its interface. In the final section, the coprocessor performance is characterised.

2. FTT-CAN BASICS

The transmission of messages in FTT-CAN is carried out in fixed duration time slots called Elementary Cycles (ECs) (see fig. 1). Within each EC there are two consecutive time windows, one dedicated to the transmission of periodic messages (the synchronous window) and another to allow the transmission of normal event-triggered messages (the asynchronous window). In this paper, only the synchronous time-triggered traffic is of concern since it is the only one subject to scheduling.

A centralised scheduler located in the so-called Master Node, holds the properties of all synchronous messages in the system and builds static schedules for fixed duration periods of time. The information of which messages should be produced in each particular EC is broadcasted to all nodes within a special message, known as the Trigger Message. The transmission of the trigger message marks the start of each EC. The nodes that identify themselves as producers of one or more messages, transmit the identified messages in the synchronous window of that EC. Collisions on the bus access are resolved by the native distributed MAC protocol of CAN.

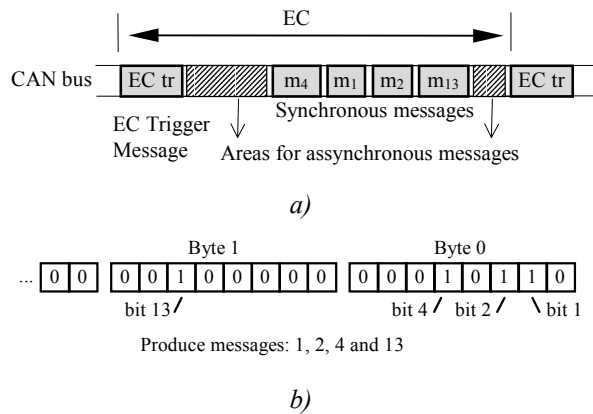


Fig. 1 - a) FTT-CAN Elementary Cycle (EC); b) EC trigger message data field.

3. SCHEDULING IN DEDICATED HARDWARE

Dedicated coprocessors have been used for a long time to improve the execution time and predictability of hard real-time operating system functions, including task scheduling. Two basic technologies have been explored in their implementation: general purpose microprocessors and application specific integrated circuits (ASICs).

Microprocessor-based solutions are inherently more flexible, potentially easier to implement and less costly. Colnaric *et al.* (1998) and Cooling (1994) report on two sampled research projects exploring this approach. ASICs provide arguably the fastest and most deterministic execution time. Compared to the microprocessor solution, their development requires more advanced skills and, typically, a higher total budget. Representative examples of ASIC coprocessors include the RTU (Adomat *et al.*, 1996), which implements all the basic functionality of a multitasking real-time kernel, the SScOP (Niehaus *et al.*, 1993), a specialised scheduling coprocessor with schedulability evaluation capability, and the universal task scheduling coprocessor presented by Hildebrandt *et al.* (1999).

Specifically for traffic scheduling in fieldbus systems little work has been done in the development of dedicated hardware. To do it, we can explore also microprocessor or ASIC-based solutions. However, in this case, an ASIC seems to be preferable because the coprocessor is just needed in the fieldbus nodes to which scheduling may be assigned (the potential master nodes, see fig. 3), and so the cost issue is not a severe drawback.

Moreover, costs can be further limited if the ASIC is implemented using programmable logic devices which are available today in ever-increasing densities and provide more than adequate speeds for this application.

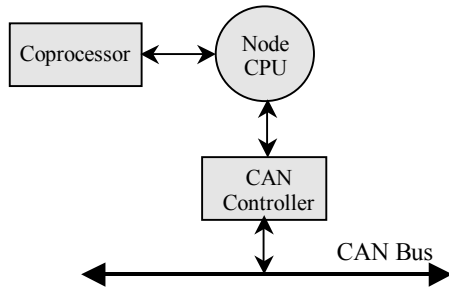


Fig. 2 - CAN arbiter node with scheduling coprocessor.

The Planning Scheduler CoProcessor (PSCoP) (Martins *et al.*, 2000) developed by our team, was the first coprocessor to appear addressing the specific requirements of traffic scheduling in the framework of the FTT-CAN protocol. It was implemented as an ASIC using Field Programmable Gate Array (FPGA) technology.

As its name implies PSCoP works according to the planning scheduler principle, building internally the plan tables which are later read by the dispatcher running in the master node CPU. Running at a mere 12MHz clock rate, it exhibited a remarkable performance level by being able to create a plan table in a small fraction (roughly 1%) of the time taken to dispatch that plan. Building on the lessons learned with PSCoP, a new scheduling coprocessor was defined and is now under development.

4. A SCHEDULING COPROCESSOR WITH SCHEDULABILITY ANALYSIS CAPABILITY

The new coprocessor schedules message transactions in Elementary Cycles (ECs). The bus time is assumed as being divided in ECs, which represent the basic time unit in which message parameters (e.g. periods) are expressed.

The coprocessor builds one EC-schedule at a time. This is one major difference from PSCoP, which, in its current version, includes two plan tables with 16 EC-schedules each. In other words, in the successor of PSCoP, the plan table size is one, which corresponds almost to a fully dynamic scheduler model.

This model was adopted based on the relatively high performance attained by PSCoP, which suggested that scheduling could be done on an EC basis. In this way a significant improvement can be made in the response time, because with a single EC-schedule all changes in the message set are reflected immediately on the next EC-schedule the coprocessor builds.

Besides the scheduling function, the new coprocessor includes also a schedulability analysis capability which was previously executed in software. Every time the message set is changed, an on-line schedulability analysis can be done to check

if the set is still schedulable. Of course, scheduling should be resumed only if the analysis indicated a schedulable set.

Pursuing a goal of maximum flexibility the coprocessor was defined with support for three scheduling policies, namely rate-monotonic, deadline-monotonic, and priority-based. An automatic start mode allows the coprocessor to start building a new EC-schedule after the previous one has been read. The target number of messages to support is 32, and the message parameters are expressed with an 8-bit resolution, which seems to be enough in most applications.

4.1- Coprocessor Architecture Overview

The implementation of the scheduling function in hardware is based on the partitioning of the scheduling algorithm in two basic activities. These are, the action of placement of transactions in the ECs, and the function of keeping track of the instants in time when each message must be produced. A similar partitioning approach, in the context of task scheduling, can be found in the coprocessor described by Hildebrandt *et al.* (1999).

The two activities are carried out by separate units inside the coprocessor. The first is executed by the EC-Schedule Builder (ECSB) and the latter by the Message's Production Timer (MPT). Fig. 3 depicts the coprocessor architecture with one ECSB and 32 MPTs connected through an internal bus. Each message to be scheduled is allocated to one MPT unit which holds the message's period(P), initial phase(Ph), deadline(D) and priority(Pr) parameters. The ECSB builds EC-schedules by accepting requests for allocation from the MPTs. Global timing information received from the ECSB allows all MPTs to be synchronised while keeping track of the EC-schedule currently being generated.

When a MPT detects that the scheduling for a particular EC where its message should be produced has started, it signals the ECSB requesting the allocation of the associated transaction. Based on the transactions' duration (C) and the remaining EC time left, the ECSB unit decides to allocate or reject the transaction. If the transaction is accepted, further

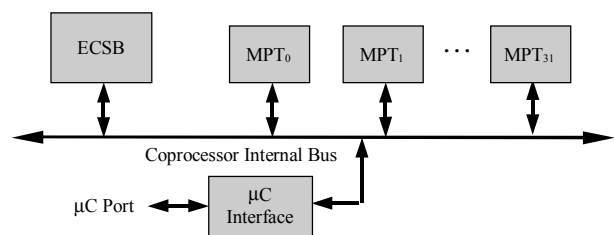


Fig. 3 - Coprocessor architecture. ECSB- EC-Schedule Builder. MPT - Message Production Timer.

requests for allocation in the same EC (from other MPTs) are evaluated, otherwise the current EC-schedule is finished.

Because more than one VPT can request allocation in the same EC, a mechanism exists to decide which request to serve first. This selection mechanism uses priority vectors derived from the message's periods, deadlines or priorities, allowing the implementation of well-known scheduling policies, and to switch dynamically between them if desired.

When testing the schedulability of a message set, what the coprocessor does is to check that the response time experienced by each message is always below its respective deadline value. It can be shown (Almeida, 1999c) that this simple test constitutes a necessary and sufficient schedulability assessment provided that we consider all messages released at time zero (i.e., $Ph = 0$ for all messages). To check individual response times the coprocessor builds what is known as the timeline. This consists basically in constructing the EC-schedules in the normal way, until all messages are allocated at least once. If we guarantee that the first allocation of each message is made within its deadline, then we can assure the schedulability of the set.

4.2- Coprocessor Software Interface

Fig. 4 illustrates the coprocessor programmer's model with all the registers accessible by the controlling node CPU.

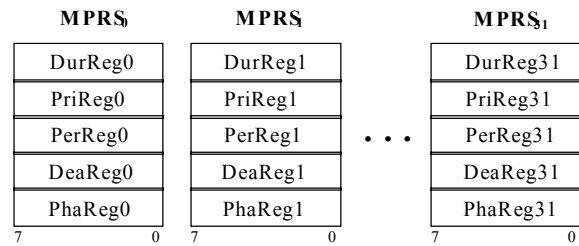
Message Parameters Register Slots (MPRSs); These registers hold the parameters of each message to be scheduled. There are 32 slots of registers for a maximum of 32 messages. Each slot contains 5, 8-bit registers, where the parameters of each message should be written to prior to any scheduling or schedulability analysis operation. A register slot has no message allocated to it if its period register is cleared.

Table 1 indicates the units and ranges of all five parameters. A zero in the priority register corresponds to the highest message priority. The message duration is expressed in a normalised unit given by $NECd = (EC \text{ duration} / 255)$. If a message has a duration of Δt time units, its corresponding DurReg register should have the value $\lceil \Delta t / NECd \rceil$. Since nothing is done to detect missed deadlines during scheduling - the user is expected to use the

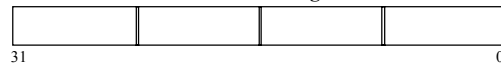
Table 1 - Message parameters registers, units and ranges.

Param.	Priority	Phase	Period	Deadline	Duration
Register	PriReg	PhaReg	PerReg	DeaReg	DurReg
Unit	-		EC		NECd
Range	0 to 255			1 to 255	

Message Parameters Register Slots



EC-Schedule Register



Control / Status Register

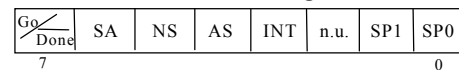


Fig. 4 - Coprocessor programmer's model.

schedulability analysis capability in order to guarantee this never happens - the deadline value written in the DeaReg register is only relevant for schedulability analysis.

EC-Schedule Register (ECSReg); 32-bit register that contains at the end of a scheduler operation, the schedule for an entire EC, called an EC-schedule. Each message allocated in that EC is specified by a 1 in the bit position corresponding to its Message Parameter Register Slot. In this way, bit number i corresponds to the message assigned to $MPRS_i$. This is the same coding method used in the FTT-CAN trigger message data field, and it was adopted here in order to minimise the dispatching overhead in the node CPU.

Control and Status Register (CSReg); This 8-bit register is used to control the coprocessor in scheduler or analyser modes, and to check its status.

Go/Done - The node CPU sets this bit to command the coprocessor to generate an EC-schedule ($SA=0$) or perform a schedulability analysis ($SA=1$). The coprocessor resets this bit when done.

SA - When set, enables the schedulability analysis mode in the coprocessor.

NS - After the coprocessor is instructed to do a schedulability analysis, NS becoming 1 (with $Go/Done = 0$) means that the set is not schedulable. NS is reset with the start of the next coprocessor operation.

AS - When set, enables the automatic start mode, where the coprocessor starts building a new EC-schedule after the less significant byte of ECSReg is read. This mode only works with the scheduler function ($SA = 0$).

INT - Interrupt status bit that reflects the (inverted) state of the external interrupt signal.

SP[0,1] - These two bits specify the scheduling policy to be used by the coprocessor, according to the assignments in Table 2.

Table 2 - Scheduling policy control bits.

SP1	SP0	Scheduling Policy
0	0	Priority-based
0	1	Rate Monotonic
1	0	Deadline Monotonic
1	1	<i>not used</i>

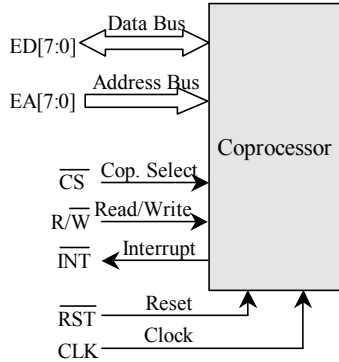


Fig. 5 – Coprocessor external interface.

4.3- Coprocessor Physical Interface and Register Map

The coprocessor has a simple standard peripheral-like 8-bit interface that can be easily connected to a variety of microprocessor or microcontroller families. Fig. 5 shows all the signals and buses of this interface. The active low interrupt signal is asserted when the Go/Done bit is reset in the CSReg, indicating the end of a scheduling or schedulability analysis operation. It is deactivated when either the CSReg or any byte of ECSReg, are read. The 8-bit address bus is used to select each internal register, mapped according to the diagram in fig. 6. Due to the 8-bit data path the ECSReg is divided in four byte registers. The duration registers appear separated from the other registers of the MPRSs, because this allowed us to simplify internal decoding. To prevent unwanted changes in the message set while the coprocessor is active, only the CSReg is accessible when the Go/Done bit is set.

4.4- Using the Coprocessor

To illustrate the coprocessor features described above, an example of the sequence of interactions performed with the node CPU is given in fig. 7. The example highlights the use of on-line schedulability analysis to validate changes made to the message set. It begins with the coprocessor generating a series of EC-schedules in automatic start mode, using interrupts. Then, scheduling is suspended, some message parameters are changed by the node CPU, and a schedulability test is requested. The outcome of this first test indicates a non-schedulable set, which requires the node CPU to readjust some message parameters in order to make the set schedulable again. In the example a second

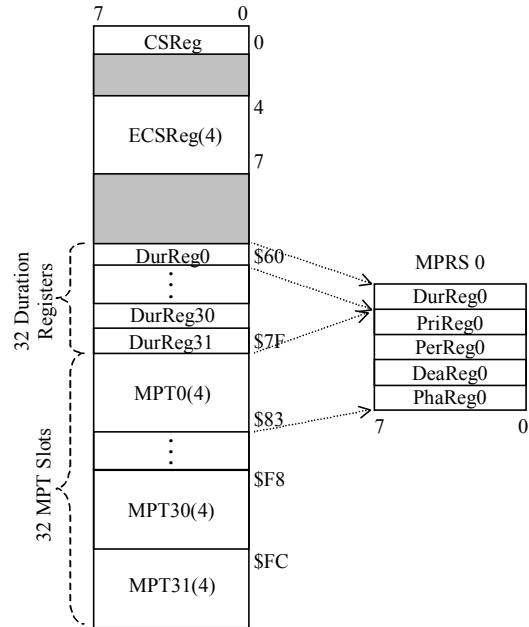


Fig. 6 – Coprocessor register map.

schedulability test is performed after these readjustments, resulting now in a schedulable condition. EC-scheduling is then resumed with the updated message parameters.

In practice on-line schedulability tests are allowed only if they can be done fast enough, or, more precisely, if they don't take more time than the available between consecutive EC-schedules. In the following section we quantify the execution times of

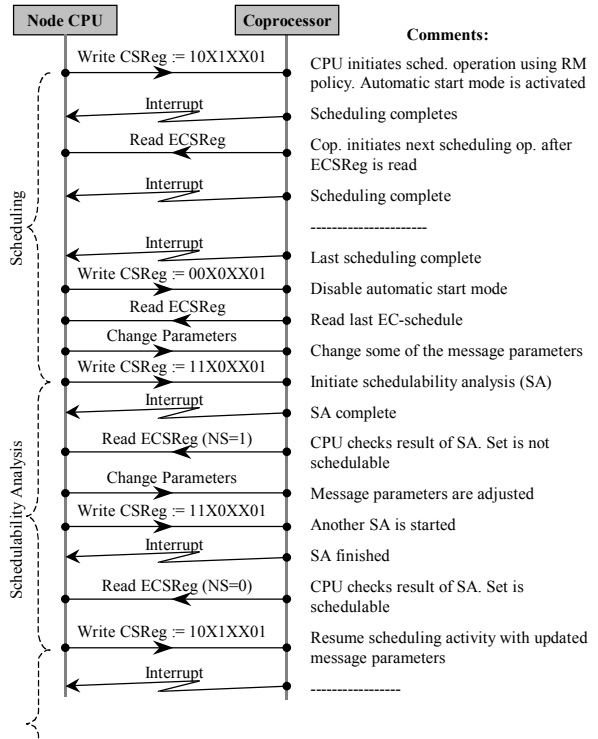


Fig. 7 – A typical sequence of interactions between the node CPU and the coprocessor during scheduling and schedulability analysis.

scheduling and schedulability analysis to show that this dynamic behaviour is indeed possible with the coprocessor.

4.5- Preliminary Performance Evaluation

At the time of writing the coprocessor is still in an early stage of development, with no simulation or real performance figures yet available. Nevertheless an estimate of performance was obtained (Martins and Fonseca, 2001), by analysing the various phases of the coprocessor's internal operation and computing the number of clock cycles required by each. The expressions derived from such analysis will be used next to validate the feasibility of each coprocessor function.

Scheduler Function; The basic performance requirement the coprocessor must fulfil is to generate an EC-schedule within the time of an EC.

The number of clock cycles the coprocessor needs to generate an EC-schedule is given by equation (1), where N_v is the number of messages allocated in that EC.

$$t_{sch} = 3 + 16 \cdot N_v \quad (1)$$

To calculate a worst case scheduling time, we assume a maximum number of allocations in the EC. For this to occur all messages must have the smallest possible length (one single data byte with no stuff bits), which, if we consider CAN2.0A format and a 1Mbit/s data rate, corresponds to a transmission time of 55 μ s (Tindell et al., 1994). Considering an EC duration of 1ms, then we can have at most 18 of these messages allocated in the EC. Using the expression above, the scheduling time in this worst case scenario is computed as 291 clock cycles. If we now consider a low clock rate as, let say, 20MHz, this translates to 14.6 μ s, or about 1.5% of the EC time. Thus, the above stated requirement is easily met even considering the most stringent conditions.

Schedulability Analyser Function; For the schedulability analysis capability to be of practical value, it must be completed while the coprocessor is idle between successive EC-scheduling operations. Because scheduling is literally executed on-the-fly, we can assume that the coprocessor has all the EC time available for schedulability tests.

The time (in clock cycles) required by the coprocessor to check the schedulability of a message set is, at most, given by (Martins and Fonseca, 2001):

$$t_{sa,wc} = 293 \cdot k_{wc} \text{ with } k_{wc} = \max\{D_1, \dots, D_N\} \quad (2)$$

where the D_i 's are the deadlines of each message in the set. With this expression the node CPU can compute the worst case execution time of a schedulability analysis with the current message set,

and, based on the result, decide whether or not it has time to do it. In the example of fig. 7 it was assumed the CPU made this computation before each schedulability test, and, in each case, the value of $t_{sa,wc}$ was below the remaining time left until the end of the current EC. If, in any case, $t_{sa,wc}$ was higher, then that schedulability test would not be performed.

For any given scenario we can also calculate the value of k_{wc} for which there is time to do a schedulability analysis. For example, assume that the CPU can spend at most 90% of the EC time waiting for a schedulability analysis to complete. For a 1ms EC, this represents 0.9ms, or a $t_{sa,wc}$ of 18000 clock cycles. Using expression (2) we arrive at $k_{wc} = 61$, which means that in this worst-case scenario, the coprocessor is able to evaluate within 1ms, the schedulability of a set having messages with deadlines as high as 61ms.

5. CONCLUSION

A coprocessor to support traffic scheduling in the FTT-CAN protocol was described. A special emphasis was given to its overall functionality and user interface. The coprocessor includes a schedulability analysis capability and can be easily explored in other centralised scheduling fieldbus-based systems.

The coprocessor implements a dynamic scheduler model, allowing all message parameters to be changed on-line, with the lowest possible response time. It has a simple and flexible CPU interface and supports three distinct scheduling policies: rate-monotonic, deadline-monotonic and priority-based. A first prototype, possibly implemented on a XC4000-family FPGA with support for up to 32 messages, is under development.

The preliminary performance analysis revealed, on the one hand, excellent results concerning the worst case scheduling time, showing that most of the dispatching time will be available for schedulability tests. On the other hand, the time to execute these tests can vary widely, depending on the deadlines involved, but is nevertheless deterministic. The CPU will decide whether or not to do these tests, based on the time left until the next scheduling operation.

Finally we should note the highly pessimistic assumptions used in the analysis presented. Further validation using more realistic message loads, provided for example by the PSA or SAE benchmarks, should be carried out.

REFERENCES

- Adomat J., et al. (1996). "Real-Time Kernel in Hardware RTU: A Step Towards Deterministic and High-Performance Real-Time Systems"; Proceedings of the Euromicro RTS '96

- Workshop, L'Aquila, Italy, pp.164-168.
- Almeida, L., Pasadas, R., Fonseca, J. (1999a) "Using The Planning Scheduler to Improve Flexibility in Real-Time Fieldbus Networks" IFAC, Control Engineering Practice Vol. 7, N° 1, pp. 101-108, January.
- Almeida, L., Fonseca, J., Fonseca, P. (1999b). "A Flexible Time-Triggered Communication System Based on the Controller Area Network" Proc. FeT '99 - Fieldbus Systems and their Applications Conf., Germany, September.
- Almeida, L. (1999c) "Flexibility and Timeliness in Fieldbus-Based Real-Time Systems", PhD Thesis, University of Aveiro. Portugal, November.
- Colnaric M. et al. (1998). "Implementation of Hard Real-Time Embedded Control Systems"; The Journal of Real-Time Systems, **14**, pp. 293-310.
- Cooling J. E. (1994). "Task Scheduling in Hard Real-Time Embedded Systems using Hardware Co-processors"; Microprocessors and Microsystems, **18**, (10) December, pp. 571-578.
- Hildebrandt J., Golasowski F., Timmermann D. (1999). "Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems"; Proceedings of the 11th Euromicro Conference on Real-Time Systems, England, June 9-11, pp. 208-215.
- Kopetz, H. (1997). "Real-Time Systems Design Principles for Distributed Embedded Applications", Kluwer Academic Publishers.
- Martins E., Neves P., Fonseca J. (2000). "PSCoP – A Planning Scheduler Coprocessor", WIP Proceedings of the IEEE International Workshop on Factory Communication Systems, ISEP Porto, September 6-8, pp.27-30.
- Martins E., Fonseca J. (2001). "Traffic Scheduling Coprocessor with Schedulability Analysis Capability"; DSD '2001 - Euromicro Symposium on Digital Systems Design, Warsaw, 4-6, September.
- Niehaus D. et. al. (1993). "The Spring Scheduling Coprocessor: Design, Use, and Performance"; Proceedings of the 14th IEEE Real-Time Systems Symposium, Raleigh-Durham, North Carolina, pp. 106-111.
- Pedreiras, P., Almeida, L. (2000) "Improving the Responsiveness of the Synchronous Messaging System", DCCS 2000: 16th IFAC Workshop on Distributed Computer Control Systems, Sydney, Australia, 29 November-1 December.
- Tindell K., Burns A., Wellings A. (1994). "Calculating Controller Area Network Message Response Times"; Proceedings of the IFAC Workshop on Distributed Computer Control Systems, Toledo, Spain, September.