# A Coprocessor for Traffic Scheduling and Schedulability Analysis in FTT-CAN

Ernesto Martins, José Fonseca, Luis Almeida
DET / IEETA – University of Aveiro, Portugal

**The low-processing power microcontrollers used within typical CAN nodes, usually place tight limits on the complexity and flexibility of on-line message scheduling systems. One solution to break this barrier is to transfer the scheduling task to a hardware implementation.**

**A traffic scheduling and schedulability analyser coprocessor is presented in this paper. The coprocessor generates message schedules for the master node CPU of a fieldbus system, leaving it just with the dispatching task. Scheduling can be made to follow one of three different policies. The number of messages to be scheduled and their parameters can be changed dynamically. To support on-line admission control of new messages, the coprocessor implements a schedulability analyser function.**

**The coprocessor was designed to support the FTT-CAN protocol, but it can be adapted to any other fieldbus using centralised scheduling. The description presented here is focused mainly on the coprocessor's overall functionality and interface with the node CPU. To characterise its performance, calculated response times are presented.**

## 1. Introduction

The FTT-CAN protocol (Flexible Time-Triggered communication on CAN) [1] was originally introduced to support time-triggered communication of periodic messages on the CAN network, while providing some operational flexibility in what concerns the on-line changing of message parameters.

The ability to make these changes dynamically is an important feature of modern distributed real-time communication systems, which require an efficient utilization of system resources [2].

All changes made at some point in the message set are reflected on the system with a delay which must be usually upper bounded. If the changes are solicited as a result of a human operation, a reaction time of a few seconds is normally tolerable. On the other hand, if the change request comes from the system itself, as a result of the application of some QoS strategy, a responsiveness of a few milliseconds may be required. This calls for a fast reaction time from the message scheduler.

An efficient software implementation of the scheduler executed in a high-performance CPU can easily fulfil this requirement. However, in cost-sensitive CAN applications (such as in automotive) where nodes are typically powered by low processing power microcontrollers, this level of responsiveness may be hard to achieve.

In addition, to guarantee that changes in the message set don't jeopardise the timeliness properties of the system, a schedulability analysis must be executed beforehand. Whether it is utilisation or response time-based, this analysis requires a level of computation power to be completed within reasonable time bounds, not provided by the aforementioned microcontrollers.

To address these two requirements in the framework of the FTT-CAN protocol, a coprocessor implementing in hardware the scheduling and schedulability analysis functions was developed.

The paper proceeds with a short reference to previously reported scheduling coprocessors, followed by an overview of the FTT-CAN protocol. The coprocessor is then described focusing on its functionality and

interface with the microcontroller. A practical scenario of its utilisation related with autonomous mobile robots is discussed. Finally, its performance is characterised.

## 2. Scheduling Coprocessors

Judging by published work, not much effort has been dedicated in the past to the development of dedicated hardware for scheduling. The SSCoP [3] developed for the Spring kernel and the universal task scheduling coprocessor reported in [4] are two of the few examples that address the problem of task scheduling in real-time operating systems.

In the area of fieldbus traffic scheduling, the Planning Scheduler CoProcessor (PSCoP) [5] developed by our team, was, as far as we know, the first dedicated hardware solution to appear. Being implemented on FPGA technology and running at a mere 12 MHz clock rate, PSCoP exhibited a remarkable performance by being able to generate message-schedules for a fixed time interval, in less than 1% of the time needed to dispatch that same interval.

The coprocessor described in this paper is somehow an evolution of PSCoP, with a whole new functionality, including a schedulability analyser

## 3. FTT-CAN Overview

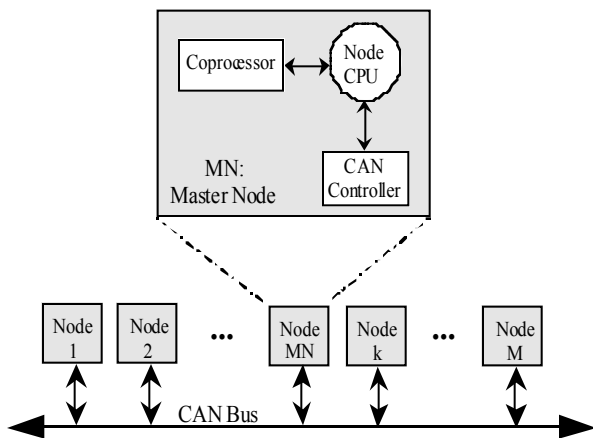The FTT-CAN protocol defines a centralised master node where the properties of all



**Fig. 1** – Typical centralised scheduling fieldbus architecture having a Master Node with a scheduling coprocessor.

synchronous messages in the system are located, and where the scheduler executes.

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

The scheduler schedules message transactions in units of a single EC, called EC-schedules. Each contains the information of which messages should be produced in each particular EC. EC-schedules are 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.

## 4. Scheduling / Schedulability Analyser Coprocessor

The coprocessor works as a slave of a microcontroller in the FTT-CAN master node (see fig. 1). Its function is to take care of scheduling and schedulability analysis while the microcontroller executes EC-schedule dispatching, admission control and node management.

In the scheduling mode the coprocessor generates one EC-schedule at a time. Between the generation of successive EC-schedules the node CPU can change the existing message parameters, as well as add to or delete messages from the set. A high responsiveness is guaranteed by reflecting all changes in the message set right in the next EC-schedule.

Scheduling can be performed according to three static policies: rate monotonic (RM), deadline monotonic (DM) and Priority-based. Operation can be changed dynamically between these policies according to the application requirements.

In the schedulability analysis mode the coprocessor tests the current message set to see if all messages can be produced within their respective deadlines. This analysis can be done between the genera-tion of successive EC-schedules. In fact, as it will be seen later, the coprocessor is fast enough to perform more than one schedu-lability analysis within an elementary cycle.

The joint hardware integration of the scheduling and schedulability analysis functions constitutes a unique feature of this coprocessor in the realm of fieldbus traffic scheduling, mirrored only by the SSCoP coprocessor in the area of task scheduling.

A first prototype of the coprocessor is being implemented in a FPGA supporting up to 32 messages with parameters expressed with a resolution of 8-bits. These figures are appro-priate for a broad range of applications, e.g. the SAE benchmark as described further on.

## 4.1. Basic Architecture

At a high level of description the coproce-ssor architecture doesn't differ much from the one adopted in PSCoP [5]. This is because, despite dealing now with a dynamic schedu-ler, we can still divide the scheduling function in the same two basic activities: 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. In the new coprocessor this work is carried out by the EC-Schedule Builder (ECSB) and the Message Production Timer (MPT), respectively.

Fig. 2 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 that holds the message's period (P), initial phase (Ph), deadline (D) and priority (Pr) parame-ters.

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 requests for allocation in the same EC (from other MPTs) are evaluated, otherwise
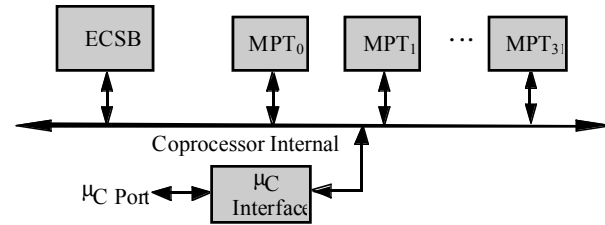


**Fig. 2** - Coprocessor architecture. ECSB- EC-Schedule Builder. MPT - Message Production Timer.

the current EC-schedule is finished.

The schedulability test is based on a response time analysis in which the coprocessor executes internally consecutive scheduling operations, building what is known as the timeline. If the first allocation of each message is made within its deadline, then the schedulability of the set is guaranteed. It can be shown [6] that this simple test constitutes a necessary and sufficient schedulability assessment.

## 4.2. Coprocessor Interface

### 4.2.1- *Software Interface*

Fig. 3 illustrates the coprocessor programmers model with all the registers accessible by the controlling node CPU.

*Message Parameters Register Slots (MPRS)*

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.

The message priority, optionally written to the priority register (PriReg), has a range of 0 (highest priority) to 255. The values written to the phase (PhaReg), period (PerReg) and deadline (DeaReg) registers are expressed in number of ECs. The values written to the message duration register (DurReg) should be expressed in a normalized unit given by NECd = (EC Duration / 255). If a message has a duration of $\Delta t$ time units, its corresponding DurReg register should have the value $\lceil \Delta t / NECd \rceil$. The deadline value written in the DeaReg register is only relevant for schedulability analysis.

**MPRS$_0$**        **MPRS$_0$**        **MPRS$_1$**

| DurReg0 | DurReg1 | | DurReg31 |
|---------|---------|---|----------|
| PriReg0 | PriReg1 | | PriReg31 |
| PerReg0 | PerReg1 | **. . .** | PerReg31 |
| DeaReg0 | DeaReg1 | | DeaReg31 |
| PhaReg0 | PhaReg1 | | PhaReg31 |

7          0 7          0          7          0

**EC-Schedule Register**

31                                                    0

**Control / Status Register**

| Go/Done | SA | NS | AS | INT | n.u. | SP1 | SP0 |
|---------|----|----|----|----|------|-----|-----|

7                                                    0

**Fig. 3** - Coprocessor programmer's model.

## 4.2.1- *Physical Interface*

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. 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.

### 4.3. Using the Coprocessor: A Practical Example

Consider an autonomous robot with several guiding sub-systems such as line tracking, beacon following and target tracking, as well as other application-related sensors and actuators. The robot is equipped with a distributed computer control system based on the FTT-CAN protocol that interconnects its sensors, controllers and actuators. The Master node is supported by the coprocessor just described.

The robot global control follows a flexible approach according to which all activities in the system are executed when necessary, only, and with a quality of service (QoS) depending on the current availability of system resources. In particular, concerning

### *EC-Schedule Register*

At the end of a scheduler operation this 32-bit register contains the schedule for an entire EC: an EC-schedule. Each message allocated in that EC is specified by a 1, in the bit position corresponding to its register slot. In other words, bit number i corresponds to the message assigned to MPRSi. This is the same coding method used in the FTT-CAN trigger message data field (see fig. 4), and it was adopted here in order to minimise the dispatching overhead in the node CPU.

### *Control and Status Register*

This 8-bit register is used by the microcontroller to control the coprocessor and to check its status. Table 1 describes the

## EC-Schedule Register

00010001  00 00000  000001·

Byte 3        Byte 2        Byte 1        Byte 0

To TM data field

Elementary Cycle (

| TM | | SM2 | SM9 | SM2 | SM24 | SM2 | SM1 | TM |

EC Trigger Message

Six scheduled messages:
2, 9, 10, 21, 24 and 28

**Fig. 4** – During dispatching contents of EC-Schedule Register are copied to Trigger Message data field.

function of each bit.



ED[7:0]   Data Bus

EA[7:0]   Address Bus

$\overline{CS}$   Cop. Select

R/$\overline{W}$   Read/Write

$\overline{INT}$   Interrupt

$\overline{RST}$   Reset

CLK   Clock

Coprocessor

**Fig. 5** – Coprocessor external interface.

the communication system, the properties of

the message streams are adapted on-line according to the current needs. For example, consider the line-tracking module. When the robot follows a line, it may adjust its speed according to the current line pattern (curved or straight) or the available bandwidth for the message stream carrying the samples with the robot deviation.

**Tab. 1** – Control/Status Register Bits

| | |
|---|---|
| **Go/Done** | The microcontroller sets this bit to command the coprocessor to generate an EC schedule (if SA=0) or perform a schedulability analysis (if 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 means that the set is not schedulable. |
| **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. |

**Tab. 2** - Scheduling policy control bits.

| SP1 | SP0 | Scheduling Policy |
|---|---|---|
| 0 | 0 | Priority-based |

Suppose that the robot is travelling along a straight line and suddenly faces a sharp curve. Then, two actions can be taken: either the sampling rate is increased and the robot maintains its speed, or the rate cannot be increased and thus the robot must slow down. Increasing the sampling rate implies reducing the period of the respective message stream. Thus, the node CPU requests a schedulability analysis of the

changed message set. If it is schedulable, the change, i.e. reduction in message period, is performed and the robot maintains its speed. Otherwise, the robot slows down.

In general, it's not possible to predict the outcome of the schedulability test. The dynamic activation / deactivation of some sub-systems may lead to deeply varying conditions of bus load. Fig. 6 shows the sequence of interactions that are carried out between the node CPU and the coprocessor in the scenario just described.

First the coprocessor is asked to generate EC-schedules in automatic start mode, using interrupts. Then, when the robot detects the curve, scheduling is suspended, the message period from the line tracking sensors is reduced and a schedulability test is requested. The outcome of this first test indicates a non-schedulable set. In response, the node CPU decides to readjust the message period to some pre-defined intermediate level, and requests a second schedulability test. This test results now in a schedulable condition. EC-scheduling is then resumed with the updated message period.
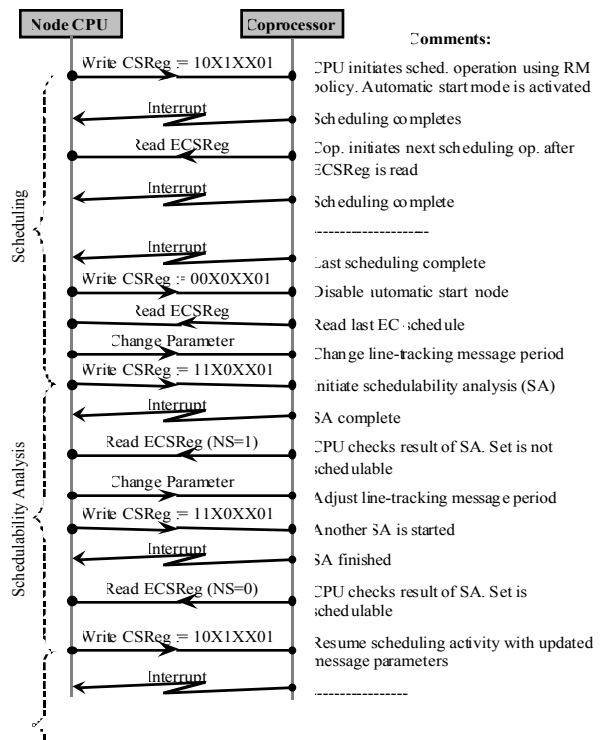


**Fig. 6**. Sequence of interactions between the node CPU and the coprocessor in the

# 5. Performance Evaluation

In this section, we analyse the coprocessor performance to check whether it exhibits the dynamic behaviour required by applications like the one in the previous section. The coprocessor must be fast enough to carry out several schedulability tests and scheduling operations in one EC.

At the time of this writing this coprocessor has not yet been physically implemented, so no measured performance figures are yet available. Nevertheless, to anticipate its performance level we make use of two expressions derived in [7], which quantify the number of clock cycles required by the coprocessor to complete each function: scheduling and schedulability analysis. Since the coprocessor is an all-synchronous design, the results derived from such expressions are believed to be exact.

In addition we consider as workloads selected message subsets from the SAE benchmark. This benchmark is summarized in table 3 that classifies its 53 messages in 6 types, according to their respective periods and deadlines.

## 5.1. Scheduler Function

The basic performance requirement the coprocessor must fulfil is to generate an EC-schedule within the time of an EC. Ideally it should be fast enough in order to spare some EC time for one or more schedulability tests.

The number of clock cycles spent in the generation of an EC-schedule is given by [7]:

$$t_{sch} = 3 + 16 . Nv \qquad (1)$$

where $Nv$ is the number of messages allocated in the EC. To compute the worst case scheduling time, we consider a maximum number of allocations per EC.

**Tab. 3** - SAE message set summary.

| Type | P (ms) | D (ms) | Nº of Msgs |
|------|--------|--------|------------|
| A | 5 | 5 | 8 |
| B | 10 | 10 | 2 |
| C | 50 | 5 | 1 |
| D | 50 | 20 | 30 |
| E | 100 | 100 | 6 |
| F | 1000 | 1000 | 6 |

Given the SAE benchmark message periods, we assume an EC duration of 5ms. Additionally, since all SAE messages have a data area with one byte or less, they all correspond in CAN to single byte messages. In a CAN2.0A fieldbus operating at 500Kbit/s, the smallest duration of these messages is 110μs [8], which means that it is possible for the coprocessor to have all 32 messages allocated in the same EC.

From equation (1), the scheduling time in this worst case scenario is thus computed as 515 clock cycles. If we now consider a low clock rate, e.g. 20MHz, this translates to 25.8μs, which is indeed a very small fraction of the EC time, making almost all the EC available for schedulability analysis.

## 5.2. Coprocessor vs Microcontroller

At this point the key question is: Could we attain this level of performance with an off-the-shelf microprocessor-based solution?

To answer this question we will use data taken from an experimental CAN system, where the scheduler was executed by an 80C592 microcontroller clocked at 11 MHz [9]. For a set of nine messages this micro-controller takes 7.8ms to output a single EC-schedule. If we had, let say, a 200MHz CPU in the master node, which corresponds to a speed-up factor of 18 (ignoring, for simplicity sake, performance gains due to architectural enhancements) we could expect a similar reduction in the scheduling time, to about 430μs which is still much more than what we can achieve with the 20MHz FPGA-based coprocessor. Increasing even further the microprocessor clock will shorten the scheduling time, but will also raise costs and power dissipation problems, which are both sensitive issues in CAN systems.

## 5.3. Schedulability Analyser Function

The performance requirement imposed on the coprocessor in what concerns the schedulability analysis capability is basically the same as for the scheduler function, i.e. it must be completed while the coprocessor is idle between successive scheduling operations.

The maximum time required by the coprocessor to check the schedulability of a message set is given by [7]:

$$t_{sa} = 5 \cdot k + 16 \cdot \sum_{i=1}^{k} N_v(EC_i) \qquad (2)$$

where $k$ is the number of scheduling actions executed as part of the test, and $N_v(EC_i)$ the number of messages allocated in the $i^{th}$ EC-schedule.

To find the highest value that $t_{sa}$ can reach, we need therefore to find the SAE message subset that maximizes $k$ and $N_v(EC_i)$ simultaneously. The maximum possible value for $k$ is:

$$k_{wc} = \max\{D_1, ..., D_N\} \qquad (3)$$

The coprocessor generates all $k_{wc}$ EC-schedules when it is not able to allocate the message with the longest deadline. The maximum $t_{sa}$ will then be obtained in this scenario considering the greatest possible number of message allocations per EC, that is, the highest possible value for $N_v(EC_i)$. Since, from the point of view of CAN, all messages of the SAE benchmark have the same size and we must consider the EC time completely allocated in all $k_{wc}$ ECs, then $N_v(EC_i)$ is constant and given by:

$$N_v(EC_i) = N_v(EC) = \lfloor EC_{SP} / C \rfloor \quad \text{for all } i \qquad (4)$$

where $EC_{SP}$ is the EC time slot allocated to the synchronous phase and C the transmission time of each message.

In table 4 we present four different idealised scenarios where $EC_{SP}$ was selected so that we could have in each EC as much as 8, 9, 10 or 11 messages. For each case we selected a subset from the SAE benchmark in such a way that all generated EC-schedules are fully booked. That turned out possible for the first three cases but not for the fourth. We then considered that a new message of type F (deadline = 1000ms) was added to each subset, and for each case $t_{sa}$ was computed.

Since the bus is saturated in the first three cases, the set becomes not schedulable. To reach that conclusion the coprocessor must build 1000/5 EC-schedules in each case. The time to complete the schedulability test is indicated in the rightmost column of table 4. For the case where $N_v(EC) = 11$, the addition of a new type F message still results

**Tab. 4 -** Schedulability analysis worst-case execution times. (EC=5ms; Bus speed=125 Kbit/s; $C_i = C_{min} = 440\mu s$)

| $EC_{SP}$ (μs) | $N_v$ (EC) | SAE Message Subset | Nº of Msgs | $k$ | $t_{sa}$ (clocks) |
|---|---|---|---|---|---|
| 3700 | 8 | (8xA) | 8 | 200 | 26600 |
| 4000 | 9 | (8xA)+(2xB) | 10 | 200 | 29800 |
| 4500 | 10 | (8xA)+(2xB)+(4xD) +(6xE)+(6xF) | 26 | 200 | 33000 |
| 4900 | 11 | (8xA)+(2xB)+(8xD) +(6xE)+(6xF) | 30 | 15 | 2715 |

in a schedulable set. Since only the first 14 EC-schedules are fully booked, the new message is allocated in EC number 15 which is therefore the last generated by the schedulability test. In this case $t_{sa}$ is significantly lower. For $N_v(EC)$ less than 8 or greater than 11 we will have lower values for $t_{sa}$. The same will happen for bus speeds higher than 125Kbit/s because more messages will fit in each EC.

From table 4 we see that the worst case for $t_{sa}$ corresponds to $N_v(EC) = 10$, and amounts to 33000 clock cycles. For a 20MHz clock this is 1.65ms, which is about one third of the EC time considered in this example (5ms). This result gives a clear indication of the coprocessor performance. In particular it shows the coprocessor capacity to execute schedulability analysis in a substantially less time than the EC duration.

## 6. 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 integrates a timeline–based schedulability analysis, and can be easily explored in other fieldbus systems relying on centralised scheduling.

The coprocessor implements a dynamic scheduler model, allowing all message parameters to be changed on-line with high responsiveness. It has a simple and flexible CPU interface and supports three distinct scheduling policies based on fixed priorities: rate-monotonic, deadline-monotonic and generic priority-based. A first prototype

implemented on a Spartan II series FPGA is almost completed.

The preliminary performance analysis presented revealed excellent results despite the worst-case SAE message subsets considered, and the relatively slow clock rate. In particular it demonstrated that the coprocessor is capable of generating EC-schedules in a very small fraction of the EC duration, leaving virtually all EC time available for schedulability tests. The worst-case execution time of these tests can be computed from the message-set parameters, leaving for the node CPU the decision of whether or not to start a schedulability test, depending on the remaining time left until the next scheduling operation. In any case, even in the most stringent conditions, the analysis has shown that the coprocessor is able to do at least three on-line schedulability tests between successive scheduling actions, for the SAE benchmark.

## References

[1] Almeida, L., Fonseca, J., Fonseca, P.; "A Flexible Time-Triggered Communication System Based on the Controller Area Network" Proc. FeT '99 - Fieldbus Syst. and their Applicat. Conf., Germany, Sept. 1999.

[2] Stankovic, J.A. et al.. Strategic Directions in Real-Time and Embedded Systems. ACM Computing Surveys, 28(4), 1996.

[3] Niehaus D. et. al.. The Spring Scheduling Coprocessor: Design, Use, and Performance. RTSS'93 - IEEE Real-Time Systems Symposium, Raleigh-Durham, North Carolina, USA, 1993.

[4] Hildebrandt J., Golatowski F., Timmermann D.. Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems. ECRTS'99 - Euromicro Conf. on Real-Time Systems, York, England, June 1999.

[5] José Fonseca, Ernesto Martins, Luis Almeida, Paulo Pedreiras and Paulo Neves, "Flexible Time-Triggered Protocol for CAN – New Scheduling and Dispatching Solutions", Proceedings of the 7th International CAN Conference, Amsterdam, October 24/25, 2000.

[6] Almeida, L., Fonseca, J.; "Analysis of a Simple Model for Non-Preemptive Blocking-free Schedu-ling"; RTS 2001 (Euromicro Conference on Real-Time Systems), Delft, The Netherlands, June 2001.

[7] Martins, E., Fonseca, J., "Traffic Scheduling Coprocessor with Schedulability Analysis Capability", Proceedings of the Euromicro Symposium on Digital System Design (DSD2001), Warsaw, Poland, September 4-6, 2001.

[8] Tindell K., Burns A., Wellings A.. Calculating Controller Area Network Message Response Times. DCCS'94 - IFAC Work. on Distributed Computer Control Systems, Toledo, Spain, September 1994.

[9] Almeida, L.; "Flexibility and Timeliness in Fieldbus-Based Real-Time Systems", PhD Thesis, University of Aveiro. Portugal, November 1999.

Ernesto Martins, José Fonseca and Luis Almeida
DET/IEETA, University of Aveiro
Campus Universitário de Santiago
P-3810-193 Aveiro – PORTUGAL
Phone: +351 234 370 373
Fax: +351 234 381 128
Email: {evm, jaf, lda}@det.ua.pt