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

Ernesto Martins, José Alberto Fonseca

Resumo - **A complexidade e a flexibilidade do sistema de escalonamento on-line de mensagens usado num barramento CAN, é normalmente limitada pelo baixo desempenho dos microcontroladores usados. Uma solução possível para ultrapassar esta limitação consiste em transferir as tarefas de escalonamento para hardware dedicado.**

Este artigo descreve um coprocessador para escalonamento de tráfego e análise de escalonabilidade. O escalonamento pode ser programado segundo três diferentes critérios, e o número de mensagens a escalonar bem como os parâmetros respectivos podem ser alterados dinamicamente. O coprocessador é ainda capaz de analisar a escalonabilidade de um grupo de mensagens, suportando assim mecanismos de admissão controlada. Embora tenha sido desenhado de forma a suportar o protocolo FTT-CAN, pode ser adaptado também a outros barramentos de campo com escalonamento centralizado. Este artigo não apresenta ainda qualquer estimativa do desempenho do coprocessador.

Abstract - **The low-processing power microcontrollers typically used within 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 preliminary version of a traffic scheduling and schedulability analyser coprocessor is proposed in this paper. This coprocessor, which will be implemented on a low-cost FPGA, can generate message schedules for the node CPU, leaving it just with the dispatching task. Adding to this, scheduling can be made to follow one of three different policies, and the number of messages to be scheduled and their respective 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 centralized scheduling.

Since the coprocessor is still in an early design stage, no measures of performance are yet available.

I. INTRODUCTION

The FTT-CAN (Flexible Time-Triggered communication on CAN) protocol presented in [1] and developed further in [2], adopted originally a centralized planning technique to schedule the synchronous traffic on the CAN network.

The planning scheduler [3] was proposed as a means to trade-off between the purely static and dynamic scheduling policies. Specifically it tries to achieve a part of the timeliness characteristics of static scheduling, and some of the flexibility of dynamic scheduling, without having its time overhead. Basically what the planning scheduler does is to build static schedules for consecutive fixed duration periods of time called the plans. The creation of a plan table is overlapped with the dispatching of the previous plan. The bus time is usually divided in fixed duration time slots called the Elementary Cycles (ECs). The EC duration is the basic time unit in which message periods are expressed. Plans are always made up of an integer multiple of ECs.

The plan size affects two important properties of the scheduling system: its run-time overhead and its response time to changes in the message set. Moreover these are affected in opposite directions, as the former decreases with the plan size while the latter increases [2].

The run-time overhead is usually a sensitive issue due to the limited processing power generally available in CAN network nodes, in particular in the FTT-CAN master node where the scheduler executes. In addition, the latter node has to execute on-line admission control procedures which, usually, require relevant processing power. This limitation sets a lower bound on the plan size, beyond which scheduling might not be feasible. Now, the problem arises when this lower bound is still too high to guarantee a satisfactory response time to changes in the message parameters.

In an attempt to decouple the basic scheduler requirements from the capabilities of the node CPU, a specialized coprocessor dedicated to this task was developed. The Planning Scheduler Coprocessor [4], or PSCoP as it was named, was implemented in an FPGA and supports up to 8 messages. The two internal plan tables available in this first version can be configured with up to 16 ECs. The coprocessor is accessed by the CPU as a peripheral, and is capable of scheduling a plan in less than 1% of the time needed to dispatch it [4].

In this paper we propose a new scheduling coprocessor inspired on the same basic PSCoP architecture, but with a whole new functionality, including a schedulability analyser.

The next section starts with a presentation of the coprocessor features, and explains the reasons which led us to include some of them. Section III, which is the main part of this article, gives an overview of the coprocessor

architecture, first by presenting the interface to the microcontroller, and then by discussing the internal architecture with all its functional blocks and execution mechanisms. Finally, section IV describes the present status of this development, and concludes the paper.

II. DEFINING THE COPROCESSOR FEATURES

The Planning Scheduler Coprocessor represented our first effort towards a specialized hardware solution for message scheduling in a fieldbus system. An important result obtained from this experience concerns the speed gap between the coprocessor and the microcontroller, which was found to be much higher than what we thought at first. This suggests that we can introduce additional functionality in the coprocessor (constrained, off course, by the FPGA size) and still manage to have a performance level compatible with the time constraints imposed by the system.

Also based on the high relative performance of PSCoP, it was decided that the new design should work as a dynamic scheduler coprocessor instead of working as a planning scheduler. The notion of Elementary Cycle (EC) was kept, and, instead of building a complete plan table, the coprocessor was thought to build just one EC-schedule at a time.

Having a single EC-schedule output, guarantees that all changes in the message set are accounted for in the next EC-schedule, thus ensuring the highest responsiveness. However, preventing the coprocessor to build several EC-schedules in advance as PSCoP is able to, doesn't allow it to anticipate the missing of deadlines as in the planning scheduler. To solve this problem we introduced a schedulability analysis capability in the coprocessor. Therefore, every time the message set is changed, we can run the analysis to check if the set is still schedulable. Scheduling is resumed only if the analysis indicated a schedulable set.

A response time-based schedulability analysis function can be implemented in the coprocessor using the timeline method [2]. Since this method consists basically in building EC-schedules - exactly what a scheduler coprocessor is supposed to do - its design can be made almost without any additional complexity.

In PSCoP, scheduling is ruled by a static priority-based policy. Pursuing a goal of maximum flexibility, the new coprocessor was defined supporting a few other scheduling policies between which operation can be changed dynamically. Also, besides the period, phase and message duration, there is now explicit support for the deadline and priority parameters. All parameters can be changed during operation.

The following list summarizes the coprocessor features:

- Dynamic scheduler & schedulability analyzer;
- Three scheduling policies: rate monotonic (RM), deadline monotonic (DM), and Priority-based;
- Support for 32 variables with 8-bit parameter resolution;
- Variables parameters are: phase (Ph), period (P), deadline (D), priority (Pr) and message duration (C);
- Scheduling policy, variable set and parameters can all be changed dynamically;
- One, 32-bit, EC-schedule output register.

III. COPROCESSOR ARCHITECTURE

Starting from the feature set described in the previous section, an architecture was defined. In this section we describe such an architecture. Section *A* presents the CPU interface and discusses the coprocessor programmer's model as seen by the node microcontroller. Section *B* describes the architecture at the register-transfer level.

A. Node CPU Interface

A.1. Variable's Parameters Register Slots

These registers (see figure 1) hold the parameters of each variable to be scheduled. There are 32 slots of registers for a maximum of 32 variables. Each slot contains 5, 8-bit registers, where the parameters of each variable should be written to prior to any scheduling or schedulability analysis operation.

The variable's 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. A register slot has no variable allocated to it if its period register is cleared. 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 Δt time units, its

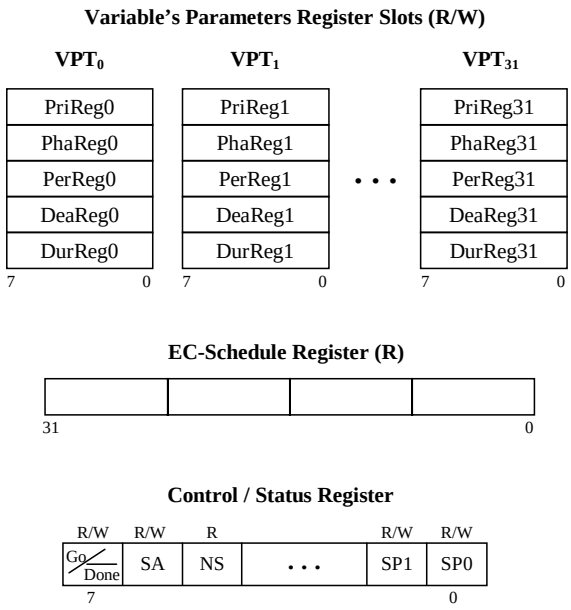

Figure 1 - Coprocessor programmer's model.

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.

A.2. EC-Schedule Register

At the end of a scheduler operation this 32-bit register contains the schedule for an entire EC. Each variable allocated in that EC is specified by a 1, in the bit position corresponding to its register slot. Bit number i corresponds to the variable assigned to register slot i. This is the same coding method used in PSCoP.

A.3. Control and Status Register

This 8-bit register is used by the microcontroller to control the coprocessor. Only 5 bits are used.

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.

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

SP1	SP0	Scheduling Policy
0	0	Priority-based
0	1	Rate Monotonic
1	0	Deadline Monotonic
1	1	not used

Table 1 - Scheduling policy control bits.

B. Architecture Overview

At a high level of description the architecture of the new coprocessor doesn't differ much from the one adopted in PSCoP [5]. This is true because, despite dealing now with a dynamic scheduler, we can still divide the scheduling function in the same two separate activities identified in the planning scheduler algorithm. These are, the action of placement of transactions in the ECs, and the function of keeping track of the instants in time when each variable must be produced. In the new coprocessor this work is carried out by the EC-Schedule Builder (ECSB) and the Variable's Production Timer (VPT), respectively.

Figure 2 depicts the coprocessor architecture with one ECSB and 32 VPTs connected through an internal bus. Each variable to be scheduled is allocated to one VPT unit which holds the variable's period (P), initial phase (Ph), deadline (D) and priority (Pr) parameters. VPTs are thus

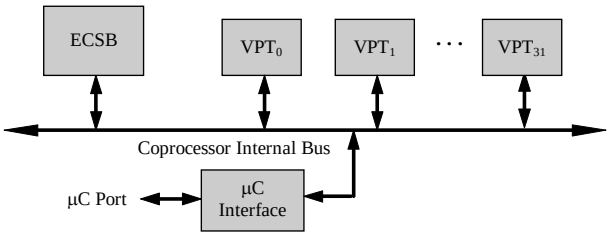

Figure 2 - Coprocessor architecture.
ECSB- EC-Schedule Builder. VPT - Variable's Production Timer.

the physical image of the variable's parameters register slots of the coprocessor programmer's model.

Global timing information received from the ECSB allows all VPTs to be synchronised while keeping track of the EC-schedule currently being generated. When a VPT detects that the scheduling for a particular EC where its variable 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 VPTs) are evaluated, otherwise the current EC-schedule is finished and the Go/Done bit in the Control/Status register is cleared.

B.1. The Variable's Production Timer

The internal structure of each VPT unit is represented in figure 3. The registers PriReg, PerReg and DeaReg of the coprocessor programmer's model (figure 1) are physically implemented in a register file inside the VPT.

Like in PSCoP, there are two presettable decrement counters, the Allocation Counter (AC) and the Deadline Counter (DC). AC is used to keep track of the variable's release time. DC is used in the schedulability analysis to evaluate if the variable can be produced within the specified deadline. Unlike PSCoP, nothing is done here to detect missed deadlines during normal scheduler operation - the user is expected to use the schedulability analysis capability in order to guarantee this never

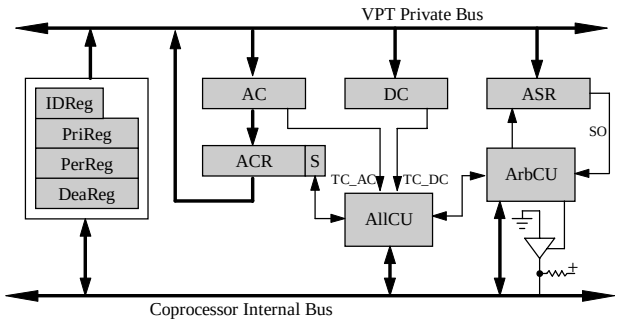

Figure 3 - Variable's Production Timer internal structure.
AC - Allocation Counter; DC - Deadline Counter; ACR - Allocation Counter Register; S - State Flag Register; ArbSReg - ASR - Arbitration Shift Register; AllCU - Allocation Control Unit; ArbCU - Arbitration Control Unit.

happens.

Also, there is no dedicated register for the phase parameter. What is mapped in the PhaReg position of the register slot is the AC counter, which is thus initially set with the phase value as it should be.

The AC counter is decremented by one every time the ECSB starts a new EC-schedule. When AC reaches zero, indicating the variable should be produced, the Allocation Control Unit (AllCU) requests the variable allocation to ECSB.

Because more than one VPT can request allocation in the same EC, a mechanism must exist to decide which request to serve first. In the new coprocessor this selection mechanism is based on a serial self-selection arbiter distributed by all the VPTs. Contention in this kind of arbiter is resolved using dynamic priority vectors which can be derived from message parameters such as their period or priority, making it easy to implement well-known scheduling policies, and to switch dynamically between them as desired. The serial self-selection arbiter is implemented by the Arbitration Shift Register (ArbSReg) and the Arbitration Control Unit (ArbCU) in each VPT, and a common tristate bus contention line.

The VPT and the ECSB have two operating modes: the Scheduler Mode (or S-Mode) and the Analyser Mode (or A-Mode), which correspond, respectively, to the coprocessor functions of scheduler and schedulability analyser.

In the middle of its normal scheduler operation, the coprocessor can be asked to do a schedulability analysis, and then to resume scheduling from the point where it was interrupted. To do this the VPTs must save their state in S-Mode before switching to A-Mode. The state information to save, which will be needed on return from A-Mode, is the current value in AC and a bit indicating whether or not the VPT has a pending request. The resources which hold these two pieces of information during a schedulability analysis are, respectively, the Allocation Counter Register (ACR) and the State Flag Register (S) in figure 3.

B.2. VPT Operating Modes

Figure 4 depicts the operation flowcharts of the VPT in its two modes.

<u>*S-Mode*</u> - In the S-Mode (Fig.4-a), after all registers have been initialized, the VPTs sit idle in state S_{1S}, waiting for ECSB to start building the first EC-schedule. When the Go/Done bit is set in the Control/Status register, ECSB initiates a series of cycles in which it receives the IDs of VPTs requesting the allocation of their variables in the current EC. With a collection of these IDs, ECSB builds the EC-schedule.

The VPT uses its AC counter as shown, with the phase and period parameters, to keep track of the EC where its variable should be allocated. When the processing for that particular EC-schedule begins, the VPT raises a request.

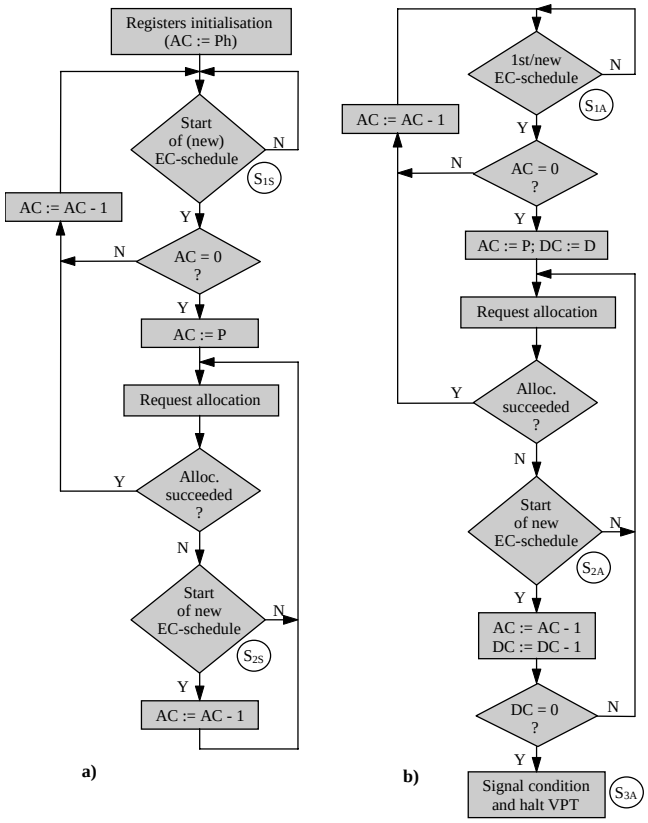

Figure 4 - VPT operation. a)- Flowchart in EC-Scheduler mode (S-Mode); b)- Flowchart in Schedulability Analysis mode (A-Mode).

Next an arbitration cycle decides which among the VPTs requesting allocation, can send its ID to ECSB. For a variable to be successfully allocated, its VPT must not only win the arbitration, but also it must receive a positive acknowledge from ECSB indicating that the variable's transaction fits in the EC (or its remaining time left). The sending of this acknowledge as well as the transmission of the ID, are both part of the arbitration protocol.

Many arbitration cycles may be generated as part of each EC-schedule processing. If a VPT was not able to allocate its variable in the first arbitration cycle, it waits for the next while keeping its request active. Eventually the current EC may be closed before the VPT is able to allocate successfully. In this case the VPT continues asserting its request while waiting for the next EC, where the variable allocation is tried once more.

Between the processing of EC-schedules, VPTs are idle in either states S_{1S} or S_{2S}. In this condition the variable set may be changed by the CPU (the coprocessor idle state is indicated by a zero in bit Go/Done in the Control/Status register). All parameters of existing variables can be changed. In particular, changes in the variable's period are guaranteed to be made without any spurious transient. Variables are deleted by clearing the period register which deactivates the corresponding VPTs. Adding new variables activates other VPTs which, after being initialized, sit idle in state S_{1S}.

A-Mode - As can be seen from figure 4-b, the VPT flowchart in A-Mode is very similar to the S-Mode flowchart. The difference is basically the utilisation of the Deadline Counter (DC).

In testing the schedulability of the message set, what the coprocessor does is to check that the response time experienced by each message is always below its respective deadline value. Theory shows that it is enough to do this response time analysis at time zero only [2]. In other words, if we guarantee that the first allocation of each variable is made within its deadline, then we can assure the schedulability of the set.

To check individual response times we build what is known as the timeline. This consists basically in constructing the EC-schedules in the normal way, until all variables are allocated at least once. If we succeed in doing this without any missed deadline, then the set is guaranteed to be schedulable.

To implement this method, the coprocessor needs little more than the scheduling functionality. In particular the VPT needs just to use the DC counter to monitor the delay between the variable's release and its allocation. If this delay exceeds the variable's deadline, ECSB is signaled.

Transitions between S-Mode and A-Mode - Figure 5 details how the VPTs switch between the two operating modes. Bit SA in the Control/Status register controls these transitions. If SA is zero, the coprocessor is in S-Mode. If SA is set the coprocessor changes to A-Mode, and all VPTs sit idle in S_{1A}. If now the Go/Done bit is set, a schedulability analysis is performed. After the analysis is completed, some VPTs will be in S_{1A}, some in S_{2A}, and, if the set turned out to be non-schedulable, one or more VPTs will end in state S_{3A}. If SA is now cleared, the state of all VPTs prior to the change to A-Mode will be restored, and the coprocessor returns to S-Mode.

If the set turned out to be non-schedulable we might want to change some parameters in the message set and run another schedulability analysis. The changes in the message set (e.g. changes in the parameters, deletion of messages) can also be made with the coprocessor in A-Mode.

To do another schedulability analysis all VPTs must, however, be first initialised in state S_{1A} with AC=0. One

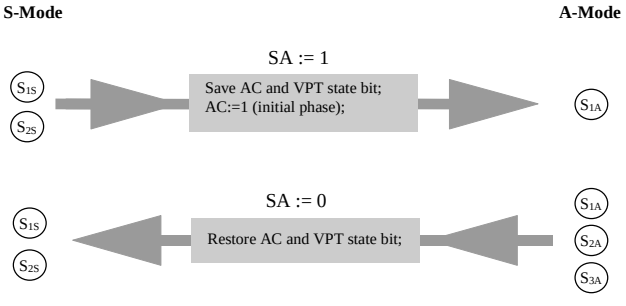

Figure 5 - Transitions between S-Mode and A-Mode in the VPT operation.

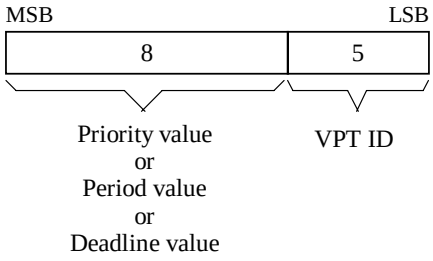

Figure 6 - Arbitration vector.

way to do this is to toggle the coprocessor to the S-Mode and then back to A-Mode using the SA control bit. After this operation all VPTs will be in state S_{1A} ready to start another schedulability analysis.

B.3. Arbitration between VPTs

The self-selection arbitration system serializes the arrival of allocation requests to ECSB, based on the criteria programmed in bits SP[1,0] of the Control/Status Register.

The arbiter relies on a contention bus line with a recessive and a dominant state, through which VPTs with pending requests try to send their respective IDs to the central ECSB. Each VPT uses a different priority vector to contend on this shared bus line. A bit-wise arbitration of all vectors from contending VPTs is serially performed during the arbitration cycle, resulting in the highest vector being transmitted.

Priority vectors have 13 bits as shown in figure 6. The most significant 8 bits are taken from the priority, period or deadline register, as dictated by the scheduling policy in use. The least significant 5 bits are equal to the VPT identifier, which is hardwired in the IDReg register. Using the VPT ID as part of the priority vectors guarantees the uniqueness of all vectors while eliminating the need for a separate transaction to identify the winning VPT.

B.4. The EC-Schedule Builder

This module builds EC-schedules in scheduler mode, and controls the coprocessor operation in analyser mode. Figure 7 shows the ECSB architecture.

As we saw above, the ECSB builds EC-schedules by accepting requests for allocation from the VPTs. The start of processing of a new EC-schedule (in scheduler or analyser mode) is communicated to all VPTs by a global signal, which the VPTs use to decrement their internal counters, and know therefore when they should request the release of their respective variables.

If there are requests, ECSB triggers a first arbitration cycle, at the end of which it acquires the ID of the highest priority VPT. The ID is captured in the register AddLatch, and used as an index to the table MDLut. This is a RAM look-up table where the DurReg registers, holding the

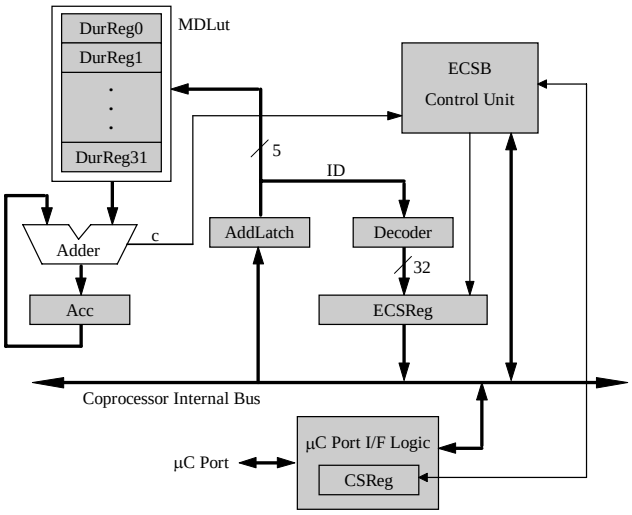

Figure 7 - EC-Schedule Builder architecture and µC Interface. MDLut - Message Duration Look-up table; ECSReg - EC-Schedule Register; CSReg - Control/Status Register.

duration of each transaction, are physically mapped. The transaction duration of the corresponding variable is thus retrieved from this table to check if the transaction fits in the EC, or its remaining free time (if we are already past the first allocation). To check this, ECSB keeps track of the total EC time already booked by previous transactions in the Acc Register. The new transaction time is thus accumulated to the value in this register. Due to the normalization of transaction times, if the time just added exceeds the time left in the EC, this addition results in an overflow condition.

In case there is no overflow, the variable is allocated in the EC. This allocation is recorded in the EC-Schedule Register (ECSReg) by the Decoder module using the ID in the AddLatch, which sets the bit located in the bit position corresponding to the VPT just served. A signal sent at the end of the arbitration cycle, tells the VPT that the allocation was made, allowing it to withdraw its allocation request. In case there are more pending requests from other VPTs, the ECSB initiates another arbitration cycle.

This sequence is repeated until ECSB finds an allocation request whose transaction exceeds the remaining time in the EC. In this case the variable is not allocated, and the EC-schedule is closed. No more arbitration cycles are generated, which means that pending requests are deferred until the next EC-schedule is processed. In scheduler mode the Go/Done bit is reset and the coprocessor goes idle.

In analyser mode the ECSB works basically the same way, with the exception that it does not stop after completing an EC-schedule. In this mode the ECSB simulates the construction of successive EC-schedules. It stops doing this only if a VPT detects the missing of a deadline, or if all VPTs succeed in allocating their variables at least once. The former condition is communicated to ECSB by the assertion of a bus line. It indicates that the set is not schedulable and results in the activation of the NS flag in the Control/Status Register.

The latter condition is detected by ECSB, using the EC-Schedule Register. In analyser mode this register is not cleared between the processing of successive ECs, so that the allocation of all variables is detected when all bits become set. In this case the set is flagged as schedulable by a zero in NS.

Note that, contrary to the VPT modules, the ECSB does not need to save any state information before switching to analyser mode.

IV. PROJECT STATUS AND CONCLUSIONS

In this paper we described a coprocessor capable of traffic scheduling and schedulability analysis. It was conceived as an evolution of PSCoP, and as such it is particularly adapted to the FTT-CAN protocol.

At the time of writing the coprocessor is still in the early stages of development, with only the VPT unit almost past the logic design stage. So, for now, no quantitative performance figures are available yet to assess the coprocessor's feasibility. As soon as all design entry is complete we expect to obtain an estimate of its performance through simulation. Afterwards, a first prototype with all the characteristics described, will be implemented on a XC4010XL FPGA. Hopefully it will include 32 VPTs, depending on the FPGA resources required by each.

For the coprocessor to achieve its goal, it must be able to run the schedulability analysis fast enough, typically in a fraction of the time it takes to dispatch an EC-schedule. In addition, in case the message set is not schedulable, the microcontroller must have time to change whatever it is needed to make it schedulable. Building the timeline implies generating EC-schedules until all variables are allocated at least once. If no deadline is violated in this process, then the set is said to be schedulable. The time to complete the timeline will be dictated by the highest of all deadlines, by the number of variables allocated in each EC and, of course, by the speed of the design itself.

While a worst case for the execution time of this analysis is now difficult to predict, the high performance room obtained for PSCoP, indicates that such a schedulability analysis function has all chances to work within the time constraints imposed by most applications used with fieldbuses.

REFERENCES

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

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

[3] Almeida, L., Pasadas, R., Fonseca, J.; "Using The Planning Scheduler to Improve Flexibility in Real-Time Fieldbus Networks" IFAC,

Control Engineering Practice Vol. 7, Nº 1, pp. 101-108, Janeiro de 1999.

[4] Martins E., Neves P., Fonseca J.; "PSCoP – A Planning Scheduler Coprocessor", WIP Proceedings of the IEEE International Workshop on Factory Communication Systems, ISEP Porto, September 6-8 2000, pp.27-30.

[5] Martins E., Neves P., Fonseca J.; "Architecture of a Fieldbus Message Scheduler Coprocessor based on the Planning Paradigm", submitted for publication in Microprocessors and Microsystems.