PSCoP – A Planning Scheduler Coprocessor

E. Martins, P. Neves, J. Fonseca
evm@det.ua.pt; pneves@ua.pt; jaf@det.ua.pt
Dep. Electrónica e Telecomunicações, Universidade de Aveiro, P-3810-193 Aveiro, Portugal

Abstract **- The use of a centralised planning scheduler in fieldbus-based systems requiring real-time operation has proved to be a good compromise between operational flexibility and timeliness guarantees.**

In this paper a preliminary implementation of a hardware scheduling coprocessor based in the planning paradigm is presented. The coprocessor is installed in a special node of the fieldbus, the bus arbiter, and generates scheduling tables to be dispatched by the node CPU. With this solution it is possible to decrease the response time to changes in the system configuration or message parameters of the software-based planning scheduler. This opens the possibility of allowing automatic on-line changes requested by system nodes in addition to the ones requested by human operators, thus improving system reactivity.

In this paper the focus is on the coprocessor's interface with the node CPU and its overall functionality. Initial calculations showing the feasibility of the unit and its expected performance are also derived.

I. INTRODUCTION

The dissemination of embedded fieldbus based distributed systems in real-time applications has triggered a significant research activity on many of the related problems and associated solutions. One of them is the improvement of distributed embedded systems reactivity and flexibility without loosing the timeliness guarantees required for a real-time operation. Some promising results have been studied in [1], concerning the use of a planning scheduler technique in systems based on low-processing power microcontrollers and in fieldbuses such as CAN [2] and FIP [3]. This technique and an associated protocol named FTT-CAN (flexible time-triggered protocol), proposed in [4], can be used to achieve real-time performance in distributed systems based in CAN, keeping a runtime overhead in the nodes that is compatible with the CPUs of most industrial embedded applications. However, a further step towards systems reactivity implies decreasing the response time to required changes. This can be achieved with several solutions, including the use of a specific scheduling coprocessor implemented in hardware.

In this paper, preliminary results concerning the development and use of a scheduling coprocessor in a CAN-based distributed system are presented. Since the coprocessor implements a planning scheduler, the paper starts in section II with a short introduction of this technique. Next, in section III, the motivation to adopt the hardware scheduler solution is briefly discussed and some previous works following the same line are shortly presented. In section IV the coprocessor is described focusing on its functionality and interface with the node CPU. Its internal architecture is then briefly presented. This section also includes some figures showing the feasi-bility of the proposed architecture. The paper concludes, pointing out future improvements to the current architecture.

II. THE PLANNING SCHEDULER

Message scheduling on a fieldbus can be done statically or dynamically. Table driven and priority-based approaches such as the ones in FIP and CAN respectively, fall in the category of static scheduling while dynamic scheduling can be done using planning based or best effort approaches. Although dynamic planning-based schedulers are not com-monly found in current standard fieldbuses, recent work on the subject [5], has shown they could become a good compromise between the static and dynamic approaches.

The planning scheduler and an associated dispatcher can be implemented in fieldbus-based systems imposing an overhead compatible with the low-processing power microprocessors or microcontrollers used as typical nodes' CPUs. Also, it presents some degree of flexibility resulting from the possibility to change, from plan to plan, the message's set, adding or deleting messages or changing their parameters. The underlying concept is the reservation of resources into the future. So, when a new message is accepted, the additional bus bandwidth required is reserved. To do this, the scheduler builds static schedules for consecutive fixed duration periods of time called plans. The static schedules are called plan tables. The creation of a plan table is overlapped with the dispatching of the

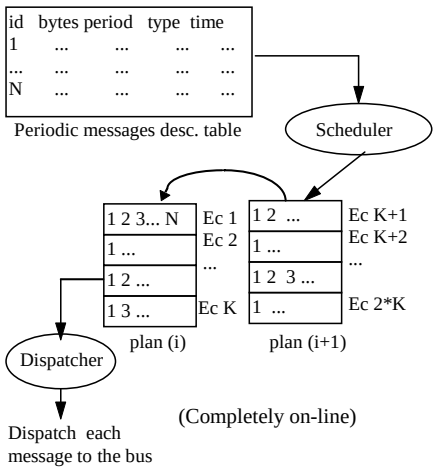

Fig. 1 – The planning scheduler.

previous. In figure 1 the operation of the planning scheduler is illustrated. The dispatcher works with plan i, while the scheduler builds plan i+1.

In common implementations of the planning scheduler, the available bus time is divided in fixed duration time slots called Elementary Cycles (ECs). Each plan includes a fixed number of ECs. Messages' periods are then restricted to an integer multiple of the EC time. Transmission time of the longest message is supposed to be less than the EC duration.

The simple mechanism of this scheduler reduces run-time overhead mainly because it is invoked fewer times. So, comparing with a dynamic scheduler, each time it is invoked, instead of determining the next message to be transmitted, only, it determines all the bus activity, for all the messages, for a certain period of time corresponding to the plan duration.

III. SCHEDULING IN A DEDICATED COPROCESSOR

A. Motivation

Experimental results [6] taken in a CAN-based system where the planning scheduler was implemented supported on a protocol named FTT-CAN (Flexible Time-Triggered) showed the exponential decrease of run-time overhead with the plan duration. Results from the same source have shown also that, for a typical EC duration of, say, 8.9ms, and 20-EC plans, the response time to a request of change in the message set is normally more than adequate when it comes from a human operator. Also, the response time can be reasonable for automatic changes during set-up or upgrade of the system. However, if more dynamic mechanisms are to be thought for the system operation, e.g., changing messages' periodicity to react to a bus overload or to adapt the sampling period of a distributed control system (operation following a QoS - quality of service model), then the response time is clearly insufficient. To overcome this limitation the plan duration should be reduced. Adding to the increased runtime overhead caused by the reduction of the plan, the implementation of automatic procedures to allow on-line changes in the communication parameters will also require relevant processing power at the arbiter node CPU.

Apart from the obvious solution of simply adopting a much more powerful CPU to keep up with all this processing needs, another interesting possibility is to use dedicated hardware to offload the node CPU in the scheduling task. The repetitive nature of the scheduling process, the robustness required for the arbiter node and the desire to reduce strongly the response time to changes led to choose the hardware coprocessor as the first solution to explore. This option was reinforced by the fact that the planning technique makes very easy the exchange of data between the coprocessor and the arbiter CPU, even when the worst case execution time of the scheduling process is not completely determined. The output of the scheduler is, in this case, a list of messages to be produced during several ECs. Although other solutions such as a scheduling coprocessor based in another CPU are yet to be studied in the future, the use of dedicated hardware is presently a good and easy option namely due to the availability of support tools [7].

B. Related Work

While virtually nothing has been reported on specialised hardware for message scheduling in fieldbuses, some recent papers have surfaced describing coprocessors aiming at improving the execution time and predictability of operating system functions.

The Real Time Unit (RTU) reported in [8] is a complete multitasking kernel implemented in an ASIC. It consists of a number of units which handle most of the time-critical functions of a typical real-time kernel. Task scheduling is based on the rate monotonic algorithm. The RTU can handle a maximum of 64 tasks at 8 priority levels, and supports up to 3 application processors. For each processor there is a dedicated ready queue. The prototype described was used in a VME system with 3 CPU boards executing tasks. The interaction between the processors and RTU is through interrupts and registers which makes it easy to use the RTU with different types of processors.

The Spring Scheduling CoProcessor (SSCoP) [9] is a VLSI coprocessor dedicated only to the task of scheduling. It was designed to work together with the Spring kernel and supports also multiple processors. The SSCoP can use different scheduling algorithms, considering shared resour-ce requirements and precedence constraints. The operating system writes the attributes of a set of tasks in the copro-cessors registers. Using these attributes SSCoP tries to build a complete feasible schedule, which, if successfully created, can be read back by the operating system.

Finally, [10] describes a universal scheduling coprocessor for single processor systems. The coprocessor is provided with the task parameters and states, and gives back to the operating system the identification of the task that has to be executed next. The architecture approach is suited for the implementation of nearly every scheduling algorithm that is based on comparison of task parameters. The coprocessor was implemented in FPGA technology and its latest version uses the Enhanced Least-Laxity-First (ELLF) scheduling algorithm and supports up to 32 tasks with a parameter resolution of 16-bits.

IV. THE PLANNING SCHEDULER COPROCESSOR (PSCOP)

The coprocessor currently under development somehow differs from the previous solutions because it directly follows the planning paradigm. PSCoP has then a limited amount of memory to store a scheduler plan i.e. the identification of the messages that must be transmitted each EC of the plan. PSCoP memory is divided in two banks allowing the coprocessor to generate one schedule plan while the CPU dispatches the other.

It the present stage of implementation PSCoP is targetted to work with the FTT-CAN protocol. This simplifies the

interface with the node CPU as explained in the next paragraph. The solution described in *B* presents some degree of scalability since the number of messages can be adapted depending on the operational needs.

A. The Node CPU Interface

To start working, PSCoP needs to be initialised first with the parameters of each variable to be scheduled. These include the variable's period (P), its initial phasing (Ph) and associated transaction duration (C). The parameters of each variable are written by the node CPU in a three register slot within PSCoP's interface. There are as many register slots as the maximum number of variables supported by the coprocessor.

In this experimental version there is no support for explicit deadline or priority parameters. The deadline of all variables is assumed to be the same as their period. Relative priorities are dictated by the allocation of register slots. These are numbered 1 to N and have assigned decreasing priorities. The scheduling priority of a given variable is thus set by mapping its parameters to the appropriate register slot at initialisation time. Clearly, priorities are always static.

The interface includes also an EC register which must be initialised with the elementary cycle duration parameter. A control/status register allows the CPU to start or stop the coprocessor, and provides information about the current state of the scheduling operation.

After instructed to begin PSCoP starts generating schedules. The message schedule for each EC in the plan is presented to the node CPU as an N-bit word which identifies the transactions that must be carried out during that EC. If a transaction of message i is allocated in a given EC, then bit i is set in that EC schedule.

This coding scheme was chosen in view of the FTT-CAN protocol-based experimental system where PSCoP is expected to be used. Since the FTT-CAN trigger message data field uses the same coding principle, the dispatching overhead is thus drastically reduced.

B. Architecture Overview

In devising a hardware structure where the planning scheduler functionality could be mapped, two separate activities were identified within the scheduler algorithm. One of them is performed in the context of each variable and acts basically as a timer, keeping track of the instants when the variable must be produced. The other concerns the placing of transactions in the respective ECs in the plan table. This partitioning of activities inspired the architecture depicted in figure 2. Here, the Variable's Production Timer (VPT) units are responsible for the first activity while the Schedule Plan Builder (SPB) takes care of the second activity.

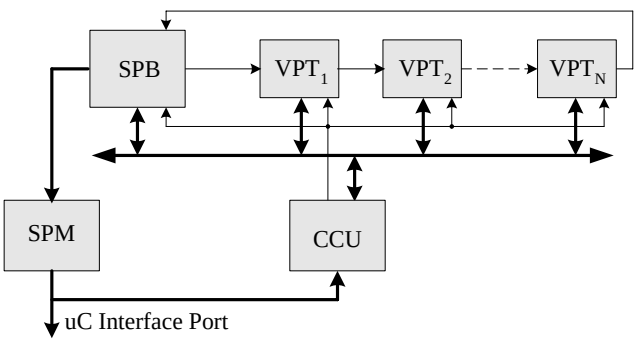

Fig.2 - PSCoP architecture. SPB - Schedule Plan Builder; VPT - Variable's Production Timer; SPM - Schedule Plan Memory; CCU - Configuration Control Unit.

Each variable to be scheduled is allocated to one VPT unit which holds the variable's period (P) and initial phase (Ph) parameters. Global timing information received from the SPB 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 SPB requesting the allocation of the associated transaction. Based on the transactions' duration (C) and on the remaining EC time left, the SPB 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 received, otherwise the current EC schedule is finished and a new one is started.

Because more than one VPT can request allocation in the same EC, a mechanism must exist to help SPB to select which request to serve first. A daisy chain structure similar to the one commonly found in microprocessor-based systems to solve interrupt or bus arbitration, is used with this purpose. The chain signal ripples through VPT_1 down to VPT_N. When a VPT unit raises a request for allocation its chain signal output is deactivated. After this, the unit is allowed to communicate with SPB only if its chain signal input is true, which means that, in a contention situation, the leftmost VPT with a pending request is always the only one with the chain signal input set to true, and therefore the one which can engage communication with SPB.

Besides the VPTs and SPB the PSCoP architecture includes two other functional blocks, the Configuration Control Unit (CCU) and the Schedule Plan Memory (SPM). The former includes control and status registers and provides access to the parameter registers in the VPTs and SPB.

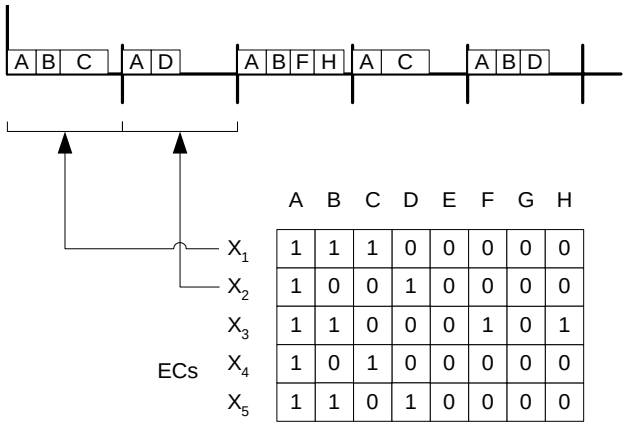

Fig. 3 - EC schedules in the SPM and the corresponding bus transactions. An example showing a 5-EC plan table supporting 8 variables and, above it, the respective timeline diagram.

The SPM unit is where SPB builds the plans with the EC schedules it generates. In the SPM memories an EC schedule is represented by an N-bit word where each bit set represents a specific transaction in that EC. The diagram in figure 3 illustrates the relationship between the transactions placed in EC time slots, and EC schedules in the SPM.

C. Preliminary Feasibility Assessment

The first prototype of PSCoP will be implemented on a XC4010XL FPGA. It will have 64 VPTs and a parameter resolution of 8-bits. The memory banks in SPM will support 20-EC plans, or, in other words, will be 20 x 64-bits FIFO memories. The prototype will be tested on a CAN master node based on a XS40 development kit from XESS® Corporation [11].

At the time of writing the coprocessor is still in the design entry stage, and so simulation results are not yet available. Nevertheless, an accurate estimate of performance was obtained by carrying out a step by step analysis of the various phases of the coprocessor's internal operation, counting the number of clock cycles required by each.

Each variable allocation takes 6 clock cycles. In the end of each EC, another 3 clock cycles are needed to transfer the schedule to the SPM unit and to begin the next schedule. The time taken by PSCoP to build a complete plan with W ECs, t_{sched}, can thus be expressed (in clock cycles) as written below, where $Nv(EC_i)$ is the number of variables allocated in EC_i.

To calculate a worst case scheduling time in our

$$t_{sched} = 3.W + 6.\sum_{i=1}^{W} Nv(EC_i)$$

prototype version, we shall assume a maximum number of allocations in every EC of the plan. For this to occur all messages must have the smallest possible length, which, if we consider CAN2.0A format and a 1Mbit/s data rate, corresponds to a minimum transmission time of 44μs [12]. If we consider an EC duration of 1ms, then we can have at most 22 of these minimum length messages per EC, in every EC. Using the expression above, the scheduling time in this worst case scenario is computed as 2700 clock cycles. Since the FPGA in the development board is clocked at 12MHz, this translates to 0.22ms, or 1.1% of the time taken by the CPU to dispatch an entire plan.

V. CONCLUDING REMARKS AND FUTURE WORK

A coprocessor for traffic scheduling in a field-bus system was described in this paper. Named PSCoP the coprocessor works according to the planning scheduler principle, and builds internally the plan tables in a format which is particularly adapted to the FTT-CAN protocol. Its architecture was defined with a main goal in mind: the design of a simple, working coprocessor which could be implemented in a medium-sized FPGA, and used as an initial testbed to obtain insight on the real performance gains and problems of the architecture. This is expected to allow the identification of the design changes needed to explore the whole benefits of the planning scheduler paradigm.

A first implementation of PSCoP will be available shortly. As shown with the rough performance estimate given for this initial version working at a modest clock rate, PSCoP can easily create a plan table in a small fraction of an EC in a field-bus running at 1Mbit/s. This result is quite encouraging in the development of the coprocessor because it suggests that some of the performance room may be sacrificed in favour of a few design improvements and additional functionality.

At this point it is clear that one of these improvements must concern the arbitration method used to resolve the contention between several VPTs requesting to allocate their transactions in the same EC. In fact the current daisy-chain mechanism, while very simple to implement, strongly compromises the operational flexibility of the planning scheduler. Once the variables are allocated to VPTs it is not possible to change dynamically their priorities. Also, it is not possible to introduce at run-time a new variable with a priority in between the ones already mapped. To get rid of these limitations we are considering to use a self-selection arbitration system in the next version of PSCoP. Since this scheme relies on dynamic priority vectors it will be easy to implement various scheduling policies like RM, EDF or simply priorities-based, and even to switch dynamically between these policies. Another interesting feature to include in this new design will be the possibility to change the plan size while the coprocessor is running.

REFERENCES

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

[2] Bosch, "CAN specification version 2.0 - Tech. Report", Bosch GmbH, Stuttgart, Germany, 1991.

[3] P. Leterrier, "The FIP Protocol", *WorldFip Europe*, 2-4 Rue de Bône, 92160 Antony – France, 1992.

[4] J. Fonseca, L Almeida; "Using a Planning Scheduler in the CAN Network", Proc. ETFA'99 – 7th IEEE Int. Conf. on Emerging Technologies and Factory Automation, Spain, October 1999.

[5] L. Almeida, R. Pasadas, J. Fonseca - "Using The Planning Scheduler to Improve Flexibility in Real-Time Fieldbus Networks" IFAC, Control Eng. Practice Vol. 7, Nº 1, pp. 101-108, Jan. de 1999.

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

[7] Valery Sklyarov et. al.; "Development System for FPGA-Based Digital Circuits", Proc. FCCM'99: IEEE Symp. Field-Prog. Custom Computing Machines, USA, April de 1999.

[8] J. Adomat et. al.; "Real-Time Kernel in Hardware RTU: A Step Towards Deterministic and High-Performance Real-Time Systems"; Proc. of Euromicro RTS '96, L'Aquila, Italy, 1996, pp.164-168.

[9] D. Niehaus et. al.; "The Spring Scheduling Coprocessor: Design, Use, and Performance"; Proc. of the 14th IEEE Real-Time Systems Symposium, USA, 1993, pp.106-111.

[10] J. Hildebrandt, F. Golatowski D. Timmermann; "Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems"; Proc. 11th Euromicro Conf. on Real-Time Systems, England, June, 1999, pp.208-215.

[11] XESS Corporation, URL: http://www.xess.com.

[12] K. Tindell, A. Burns, and A. Wellings; “Calculating Controller Area Network Message Response Times”; Proc. IFAC Workshop on Distributed Computer Control Systems, Toledo, Spain, Sept. 1994.