

Architecture of a Fieldbus Message Scheduler Coprocessor based on the Planning Paradigm

Ernesto Martins, Paulo Neves, José Fonseca

evm@det.ua.pt, pneves@ua.pt, jaf@det.ua.pt

Electronics and Telecommunications Department / IEETA

University of Aveiro

P-3810-193 Aveiro, Portugal

phone. +351-234-370373 /fax: +351-234-381 128

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. It is particularly well adapted to embedded systems based on low-processing power microcontrollers due to the low overhead it imposes.

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.

The paper includes a short review of the planning technique and a discussion on the motivation to develop the coprocessor as well as on recent similar and related work. The coprocessor architecture and several implementation details such as its interface with the arbiter CPU are presented. The initial calculations showing the feasibility of the unit are also derived, together with the first real implementation of the coprocessor itself.

Keywords: Real-time message scheduling, coprocessors, fieldbuses, digital systems.

1. 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 losing 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 low-power CPUs used in 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 use of a scheduling coprocessor in a CAN-based distributed system are presented. As the coprocessor is, at this moment, specifically developed to implement a planning scheduler, the paper starts with a short presentation of the technique, in section 2. After, in section 3, the motivation to adopt the hardware scheduler solution is briefly discussed and some previous works following the same line are shortly presented, even when they refer to operating systems scheduling and not to message scheduling in a bus. In section 4, the present coprocessor architecture is described either from a global view and for detailed aspects of its main blocks, the schedule plan builder, the variable's production timer and the schedule plan memory. It should be noticed that this version is still limited in some features, namely in the on-line acceptance of changes and in the definition of deadlines which, at this moment, must be equal to the messages periods. This section also includes the first computations showing the feasibility of the proposed architecture. Finally, the paper ends in section 5 with some conclusions and a brief enumeration of the aspects of the coprocessor that are being improved.

2. 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 commonly 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. Finally, temporal response can be guaranteed if an adequate schedulability analysis is used [5] or, at least, it is possible to anticipate situations that can lead to missed deadlines and, thus, take adequate corrective actions. That's why the planning scheduler can result in a good compromise between overhead and flexibility.

The underlying concept behind the planning scheduler 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 previous one. In figure 1 the operation of the planning scheduler is illustrated. The dispatcher is working with plan i, while the scheduler is building plan i+1.

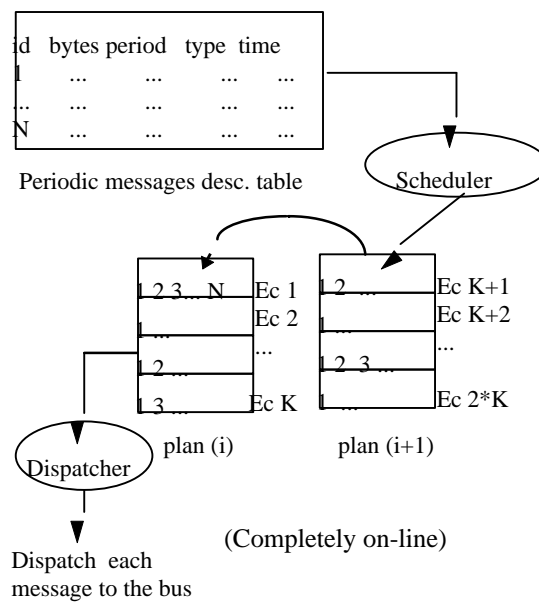


Figure 1 – The planning scheduler

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 (also transmission deadlines) are then restricted to an integer multiple of the EC duration. Transmission time of the longest message is supposed to be less than the EC duration, then

several messages fit, in principle, within an EC.

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. Reducing the plan duration increases the run-time overhead. If the plan duration is increased then the response time of changes in the message set is also increased and flexibility is then reduced.

3. Scheduling in a dedicated Coprocessor

3.1. 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. As an example, with a bus transmission rate of 125Kbits/s, with an EC duration of 8.9ms and with the scheduler implemented in a Philips 80C592 controller, the overhead decreased exponentially from 88% in the worst-case for a plan of 1 EC to 33% for a plan of 20 ECs. Further increase in the plan duration didn't correspond to a significant decrease in the overhead.

The previous results show clearly that the response time to a request of change in the message set (1 or 2 plans of 20 ECs, thus about 180ms in the maximum) is 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.

Besides the runtime overhead due to 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. To overcome this problem three solutions are envisaged. The first one consists in replacing the node CPU by a much more powerful one, keeping all the tasks together. The second and the third require the splitting of the scheduler and of the other tasks such as the admission control and protocol management. Here there are two possibilities: using another CPU as the scheduling coprocessor or implementing it in dedicated hardware.

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. The number of items in the list can be small for a reduced plan duration. The plan duration can be easily adapted to give time for the coprocessor to build the plan in a worst case situation as this just means to reserve more or less memory positions. This advantage will be clearly shown in the discussion of the coprocessor architecture.

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]. Also, there is already some work in the same direction as it will be shown in the following paragraph.

3.2. Related Work

Transferring critical functions of hard real-time systems from the software domain to specialised hardware, is becoming an increasingly hot topic within the scientific community. While virtually nothing has been reported addressing the problem of message scheduling in fieldbuses, some 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 such as semaphores, interrupt handling, event flags and periodic start of tasks. 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. After determining the next task to run, the coprocessor interrupts the CPU forcing a task switch. 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], as the name implies, 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 resource requirements and precedence constraints. The operating system writes the attributes of a set of tasks in the coprocessors registers. Using these attributes SSCoP tries to build a complete feasible schedule, which, if successfully created, can be read back by the operating system. When a new task arrives, a set of the already scheduled but not yet dispatched tasks, together with the new task, is again submitted to SSCoP for schedulability evaluation and rescheduling.

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 (e.g. RM, EDF, LLF). Due to the serial comparison of parameters, scheduling time is independent of the number of tasks. The coprocessor was implemented in FPGA technology and its latest version uses the Enhanced Least-Laxity-First (ELLF) scheduling algorithm (a non-thrashing version of LLF), and supports up to 32 tasks with a parameter resolution of 16-bits.

4. The Planning Scheduler Coprocessor (PSCoP)

Before going into the architecture details of the coprocessor, a first overview of its basic features is in order. 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 3-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.

After instructed to begin, PSCoP starts generating schedules. The results are passed to the node CPU, and consist of one N -bit string per EC; which identifies the transactions that must be carried out during that EC.

4.1. Architecture of PSCoP

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

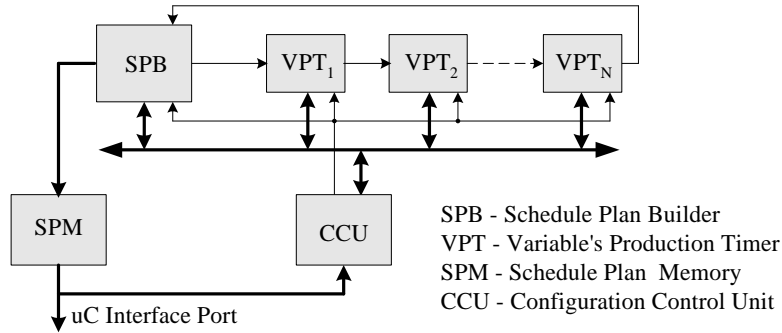


Figure 2 - PSCoP architecture.

Production Timer (VPT) units are responsible for the first activity while the Schedule Plan Builder (SPB) takes care of the second activity.

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 (see figure 3).

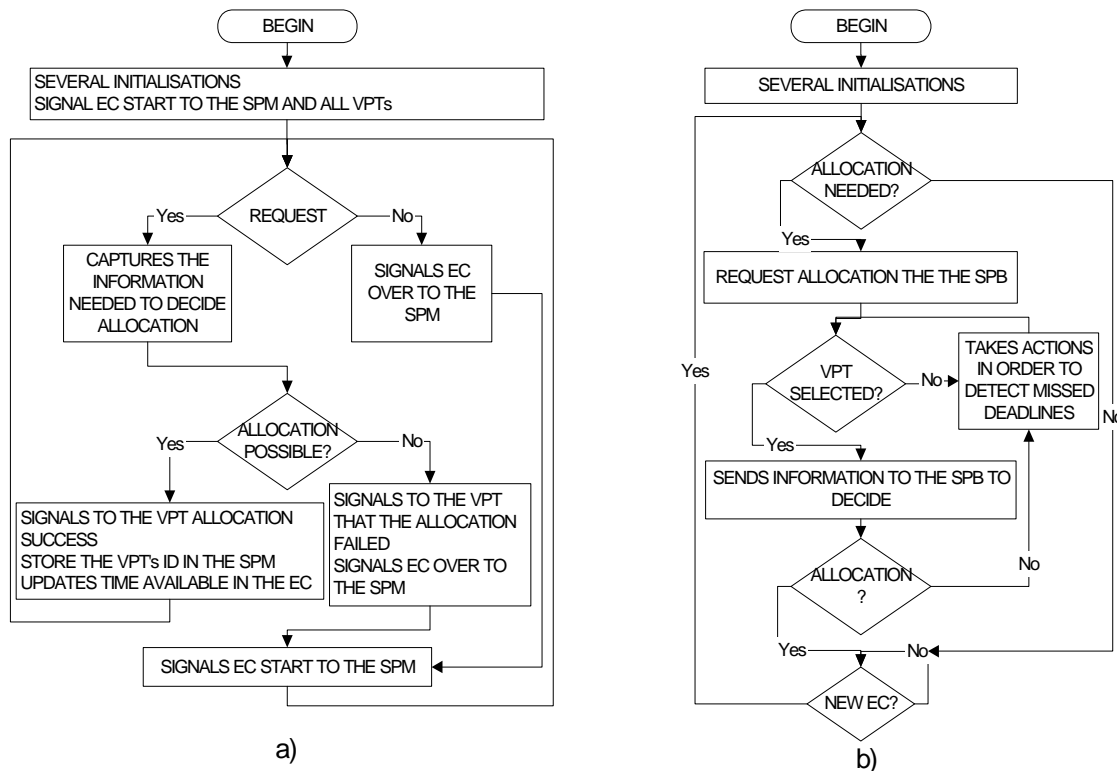


Figure 3 - SPB (a) and VPT (b) operation flowcharts.

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.

The daisy chain structure defines a static priority hierarchy between VPTs, which extends itself to the messages. Thus at configuration time, the highest priority message should be allocated to the VPT closest to the SPB, VPT_1 , while the lowest priority message should be mapped in VPT_N . The resulting message scheduling is determined both by the daisy chain arbitration and the EC time allocation (managed by the 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.

The SPM unit is where SPB builds the plans with the EC schedules it generates, which are latter read by the CPU. It includes two separate plan memories which contain alternately the plan being dispatched and the plan being built. While the SPB is updating one memory, the CPU reads the other.

The following sections describe in detail the coprocessors's main components.

Variable's Production Timer

Figure 4 depicts the internal structure of one VPT unit connected to the internal shared bus of the coprocessor. The registers PhaReg and PerReg hold the variable's initial phase and period, respectively. There are two presetable decrement counters, the Allocation Counter (AC) and the Deadline Counter (DC). The former is used to keep track of the variable's release time, and the latter allows detection of missed deadlines. A state machine-based control unit takes care of the entire operation of the VPT, including arbitration using the daisy-chain mechanism, and the handshake with the SPB.

A clock signal common to all VPTs decrements the AC counter every time the SPB starts a new EC schedule. Initially AC is set with the phase value, and DC with the period. When AC reaches zero, the

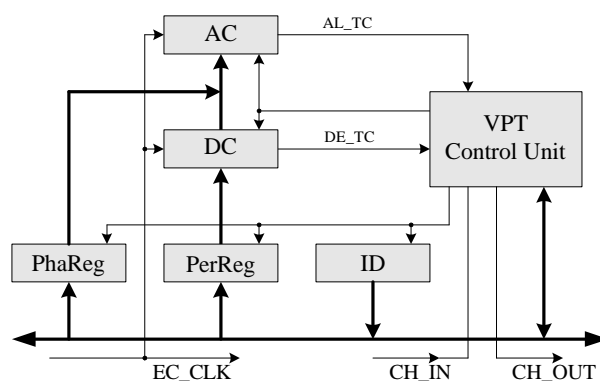


Figure 4: Internal structure of one VPT unit connected to the internal shared bus of the coprocessor.

control unit requests the variable allocation to SPB. If there are more variables to be released on the current EC, the VPT may have to wait for its turn to have the SPB's attention. When its turn arrives the VPT sends its identifier (ID) which is basically its number in the daisy-chain. The SPB unit replies with a positive acknowledge if the variable's transaction fits in the EC, otherwise a negative acknowledge is sent.

If the allocation is successful, DC is copied to AC, and then DC is again initialised with the period. The counters will decrement again with the start of the next EC schedule, and the cycle repeats. On the other hand, if the variable is not allocated, the VPT enters a state where only DC is allowed to decrement, starting on the following EC. Until the VPT is finally able to allocate the variable, the value in DC is the so-called laxity, that is, the number of slack ECs left until the deadline is reached. DC reaching zero means therefore that a deadline has been missed.

The desired behaviour of the coprocessor on a deadline miss depends obviously on the application requirements. In this design, however, it is assumed that such an event leads generally to a catastrophic failure which requires the initialisation of the whole system. Therefore, following a deadline miss, the VPT signals the SPB which, in turn, warns the CPU with an interrupt. The coprocessor is then shutdown.

Schedule Plan Builder

The SPB central unit is where the ECs which make up the plan table are actually built. Its internal structure is shown in figure 5.

The MDLut block is a RAM look-up table used to hold locally the transaction duration parameter (C_i) of each variable. This is the only parameter the SPB needs to know about each variable. All the others are stored in local VPT registers, as we saw before. Another parameter local to SPB is the elementary cycle duration, which is kept in the ECDCReg register.

The SPB builds EC schedules by accepting requests for allocation from the VPTs. An EC schedule is actually a stream of IDs which identify the transactions placed in that EC.

A global EC clock generated in the SPB is used to tell all VPTs that a new EC is about to be

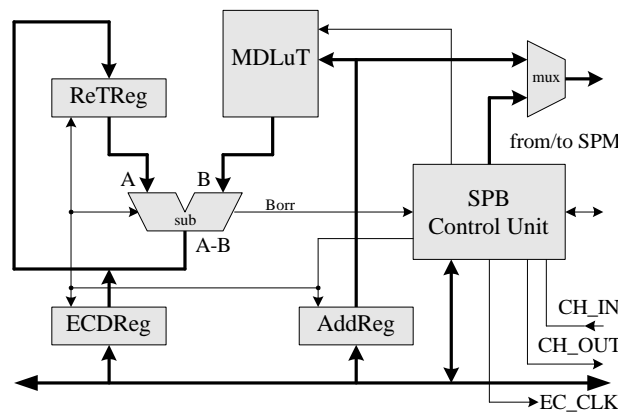


Figure 5 - Internal structure of the Schedule Plan Builder (SPB).

processed. An active transition on this signal, indicates the end of an EC and the start of a new one. VPTs use these transitions to decrement their internal counters, and to know therefore, when they should request the release of their respective variables.

After signalling the start of an EC, the SPB checks if there is a request from any VPT. In case there are multiple requests, the arbitration mechanism selects the highest priority VPT among the competitors, which, then, sends its ID to the SPB. The ID is captured in the register AddReg, and used as an index to the table MDLut. The entry retrieved from the table is the transaction duration of the corresponding variable. This parameter is now used to check if the transaction fits in the EC. To accomplish this, the SPB maintains the remaining free time in the current EC in the register ReTReg, which is initialised at the beginning of every EC with the total EC duration stored in ECdReg. The transaction duration is then subtracted from the contents of ReTReg. If the result is zero or positive ($C_i \leq [\text{RetReg}]$), then the transaction fits in the EC and RetReg is updated with the new time left. A positive acknowledge is sent to the VPT, informing that its transaction was placed in the EC. At this point the VPT withdraws from the daisy chain, allowing the arbitration mechanism to select the next higher priority VPT with a pending request. The previous process now repeats for the next VPT, if any, or the EC schedule is closed and a new one is started.

On the other hand if the subtraction $[\text{RetReg}] - C_i$ returns a negative result, then a negative acknowledge is sent to the VPT, rejecting the transaction allocation. The current EC schedule does not accept any further transactions, and is therefore closed. **This mechanism ensures that the EC duration is never exceeded (transactions are never allowed to cross EC boundaries).** Clearly, before closing the EC schedule, the coprocessor could try to allocate a shorter transaction in the remaining time left in the EC (known as inserted idle time). However this would lead to an undesired priority inversion situation, and so, to avoid it, this portion of EC time is usually left unused.

Figures 6 to 8 illustrate some of the handshake sequences which occur between SPB and VPTs during the actions described above. Figure 6 presents the case where an EC is generated by the SPB and the first VPT needs allocation. As a result, VPT₁ will hold the daisy chain signal and disable it for the other VPTs. In Figure 7 the same VPT uses the data bus and data valid signals to send its ID to the SPB to decide allocation. The SPB signals the decision through a dedicated bus line, and the VPT withdraws itself from the arbitration scheme, allowing the daisy chain signal to ripple to the other VPTs. Finally the timing diagram in figure 8 shows the rejection of a transaction. In this case, the VPT keeps its request signal asserted, signaling that it has still a pending request. The daisy chain is kept blocked.

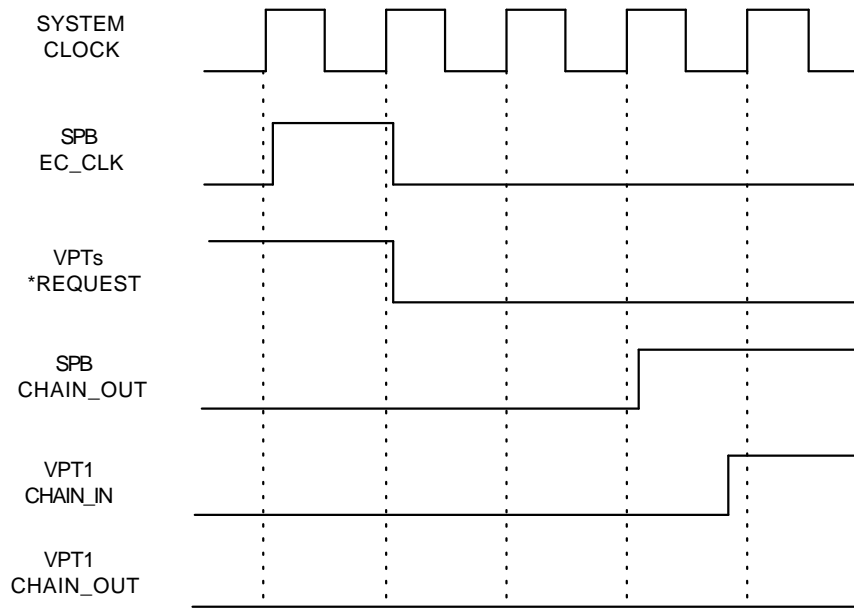


Figure 6 – Generation of a new EC by the SPB and arbitration mechanism with VPT₁ holding the daisy chain.

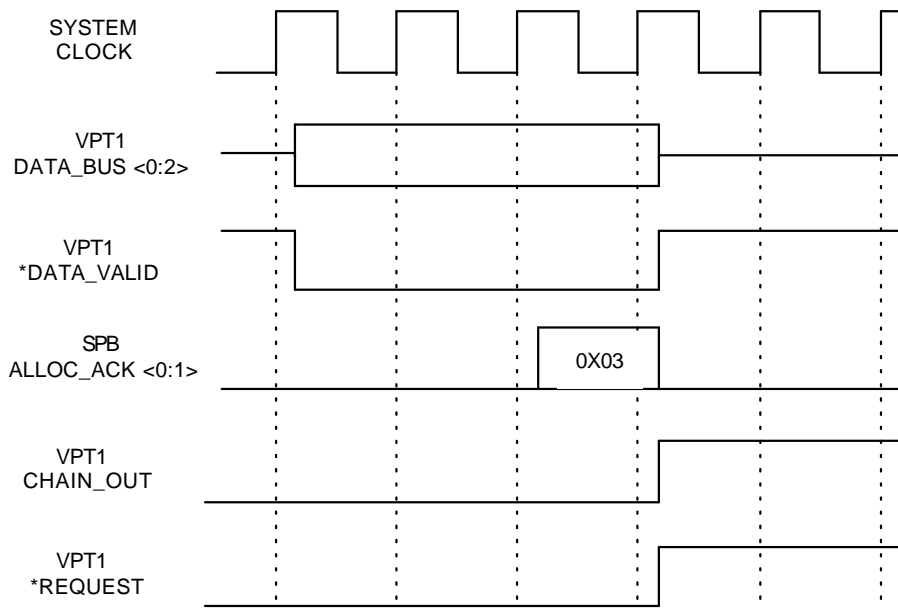


Figure 7 – Acceptance of a transaction: VPT₁ sends identifier and SPB replies with positive acknowledge.

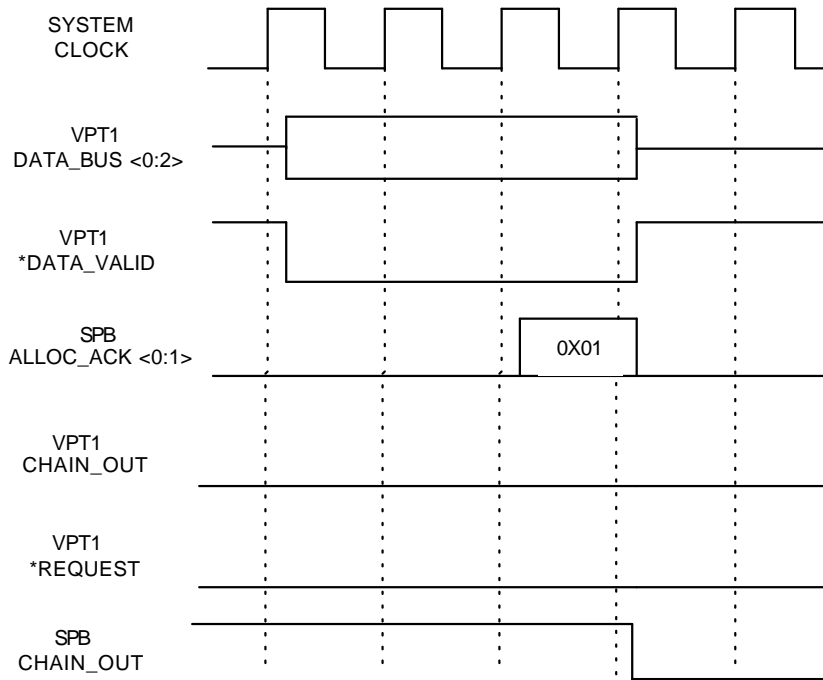


Figure 8 – Rejection of a transaction: VPT₁ sends identifier and SPB replies with negative acknowledge.

Schedule Plan Memory

The EC schedules built by SPB are stored in the SPM unit, which includes two separate memory banks, each with enough capacity to hold an entire plan (see figure 9).

The SPB identifies the transactions allocated in a given EC by the IDs of the corresponding VPTs. When a transaction is allocated, its respective ID is transferred to SPM. A special ID code is used to flag the closing of an EC schedule. In the SPM memories, an EC schedule is represented by an N-bit word, with N being the number of VPTs in PSCoP. The memory requirements of an EC schedule are therefore fixed, independently of the number of transactions placed in that EC. In every word, bit i corresponds to VPT_i. If a transaction corresponding to the variable in VPT_i is allocated in an EC, then bit i is set in that schedule. Otherwise it is kept clear. The diagram in figure 10 illustrates the relationship between the transactions placed in EC time slots, and EC schedules in the SPM.

The SPM receives IDs from the SPB, codes them into an N-bit word, and then writes this word in one of the SPM memories, as soon as a special ID code is received marking the end of the EC schedule.

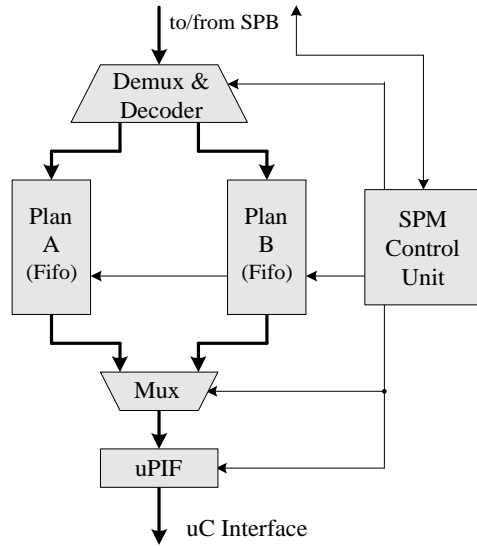


Figure 9- Structure of the Schedule Plan Memory (SPM).

The idea of having two FIFO memory banks is to allow the coprocessor to generate one schedule plan while the CPU dispatches the other. The coprocessor builds the plans much faster than the CPU consumes them. Thus, it is possible to have the CPU reading from a bank while the other bank is already filled with the next plan. In this situation the SPM cannot hold no more EC schedules, and so its control unit commands SPB to stop generating them. The coprocessor remains halted until the CPU resumes reading one bank and switches to the other.

This switching operation is transparent to the CPU. After the last EC schedule from one bank is read, the multiplexer switches the μ PIF unit to the other bank. The CPU can continue reading without ever polling the status register in the CCU. The only exception to this is after initialisation, where the CPU has to wait for PSCoP to complete the first plan.

Finally it should be noted that EC schedules are coded as N-bit words, not only because of the

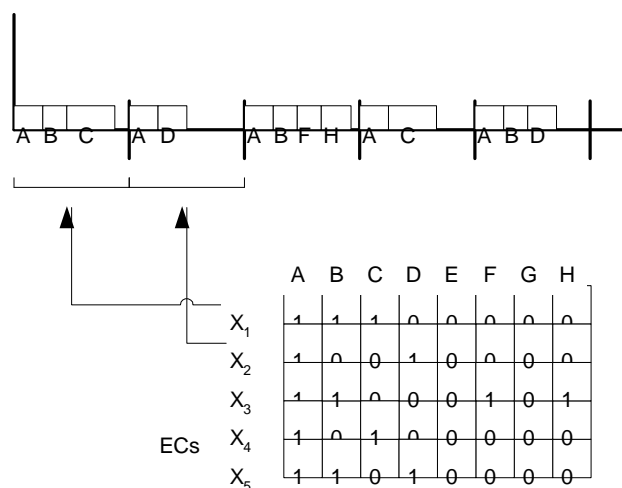


Figure 10 - EC schedules in the SPM and the corresponding bus transactions. This example shows a 5-EC plan table supporting 8 variables and, above it, the respective timeline diagram.

memory advantage this brings, but also because it allows to reduce drastically the CPU post processing overhead in the particular set up where PSCoP is expected to be used. This will be a CAN experimental system where the FTT-CAN [1] protocol is used. Since each N-bit word is already in the form of the FTT-CAN trigger message data field, the CPU load is greatly reduced, minimizing the dispatching overhead.

4.2. Implementation and performance assessment

The first prototype of PSCoP is now implemented on a XC4010XL series FPGA. It has 8 VPTs, a

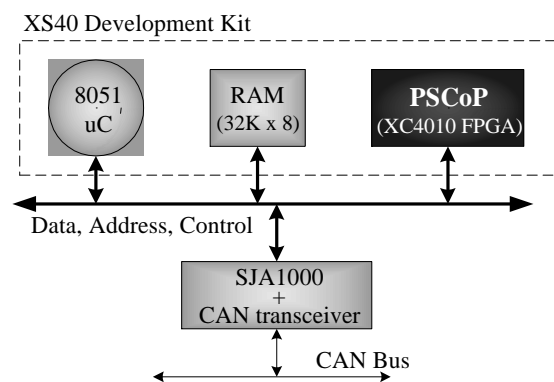


Figure 11 - Development and testing platform of the PSCoP coprocessor.

parameter resolution of 8-bits, and two memory banks in the SPM supporting 16-EC plans (16 x 8-bits FIFO memories). Each VPT occupies a matrix of 5x7 CLBs while the SPB and SPM take together the equivalent of 120 CLBs, for a total of 400 CLBs. The prototype was tested on a CAN master node based on a XS40 development kit from XESS[®] Corporation [11] which, besides having the FPGA clocked at 12MHz, includes an 8051 microcontroller (see figure 11).

Using this test platform the coprocessor performance was characterized by measuring its scheduling execution time as a function of the number of messages. The message sets used were defined in such a way to maximize the coprocessor execution time. This worst-case condition occurs if all messages are allocated simultaneously in all the ECs of the plan, forcing the SPB to generate the highest number of allocations. So to achieve this, the measurements were carried out with homogeneous messages, having zero phase, a period of one EC and a transmission time of a small fraction of the EC duration (so that all messages in the set fit in the same EC).

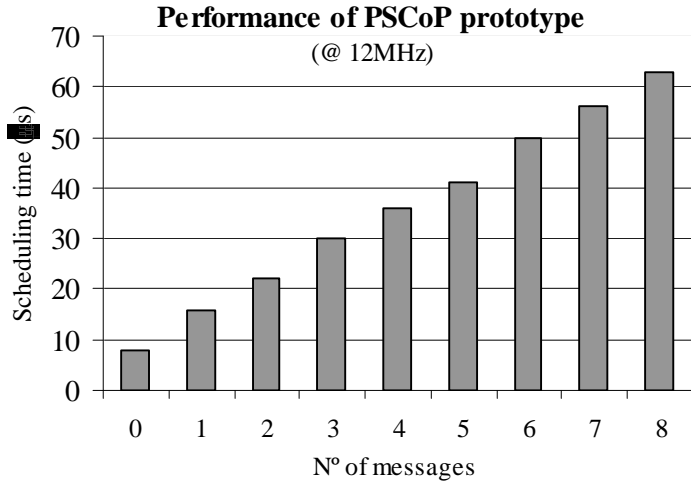


Figure 12 - Worst-case coprocessor execution time vs the number of scheduled messages.

Figure 12 shows the execution times obtained. As it can be seen the time taken by the coprocessor to build a plan grows linearly with the number of messages scheduled, reaching a maximum of 63μsec for an 8-message set. This value should be compared with the time taken by the CPU to dispatch the plan. Considering a EC duration of 1ms (the lowest value typically used in distributed embedded applications), the dispatching time is 16ms, which means that the coprocessor schedules literally on-the-fly.

By analysing the coprocessor internal operation it is possible to obtain an estimate of performance for other implementations supporting more than 8 messages. Since we are dealing with an all-synchronous design we can do this just by counting the number of clock cycles required by the various phases of the coprocessor's operation.

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 .

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

For $W=16$ this equation approximates closely the measured values in figure 12 for $Nv(EC_i)$ between 0 and 8.

We can use now this expression to predict the scheduling time of a larger coprocessor implementation built into a higher capacity FPGA, supporting, for example, 32 messages.

To calculate the worst-case scheduling time of this new version, we assume again a maximum number of allocations in every EC of the plan, now for a 32-message set. Considering again 16-EC plans, the coprocessor execution time is $3.16+6.16.32=3120$ clock cycles. Using the same modest clock rate of 12MHz this translates to 0.26ms, or 1.6% of the time taken by the CPU to dispatch an entire plan.

5. Concluding Remarks and Future Work

A coprocessor for traffic scheduling in a field-bus system was described in this paper. Named PSCoP the coprocessor functions according to the planning scheduler principle, and builds internally the plan tables in a format which is particularly adapted to the FTT-CAN protocol.

The coprocessor 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 test bed 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.

The PSCoP can easily create a plan table in a small fraction of a 1ms elementary cycle. This result is quite encouraging in what concerns future developments 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. In particular scheduling could possibly be done on an EC basis instead of a plan basis which would make the system operation practically fully dynamic. Some research on this topic has already been started.

At this point it is clear that one improvement to consider is 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 the use of 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.

The coprocessor was described using a mix o VHDL, state-graphs and logic gate schematics, and synthesized with the Xilinx[®] Foundation Series software. Its implementation was tested in practice

with the use of many sets of variables, including the worst-case experiment presented here. Within our future developments we intend also to produce a VHDL-only specification of the coprocessor, and then to formally validate its operation.

References

- [1] Almeida, L.; “Flexibility and Timeliness in Fieldbus-Based Real-Time Systems”, PhD Thesis, University of Aveiro. Portugal, November 1999.
- [2] Bosch, “CAN specification version 2.0 - Tech. Report”, Bosch GmbH, Stuttgart, Germany, 1991.
- [3] Leterrier, P. “The FIP Protocol”, *WorldFip Europe*, 2-4 Rue de Bône, 92160 Antony – France, 1992.
- [4] Fonseca, J., Almeida, L.; “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] Almeida, L., Pasadas, R., Fonseca, J.; “Using The Planning Scheduler to Improve Flexibility in Real-Time Fieldbus Networks” IFAC, Control Engineering Practice Vol. 7, N° 1, pp. 101-108, Janeiro de 1999.
- [6] 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.
- [7] Sklyarov, V., et. al.; “Development System for FPGA-Based Digital Circuits”, Proc. FCCM’99: IEEE Symp. Field-Prog. Custom Computing Machines, USA, April de 1999.
- [8] Adomat, J., 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] Niehaus, D., 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] Hildebrandt, J., Golatowski, F. Timmermann, D.; “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>.