Using a Hardware Coprocessor for Message Scheduling in Fieldbus-based Distributed Systems

José A. Fonseca, Ernesto V. Martins

Universidade de Aveiro / IEETA
Depto de Electrónica e Telecomunicações
3810 Aveiro, Portugal
jaf@det.ua.pt, evm@det.ua.pt

Paulo Neves

Escola Superior de Tecnologia
Instituto Politécnico de Castelo Branco
6000 Castelo Branco, Portugal
pneves@est.ipcb.pt

Abstract

Fieldbus based distributed embedded systems used in real-time applications tend to be inflexible in what concerns changing operational parameters on-line. Recent techniques such as the planning scheduler can avoid this problem but do not show adequate responsiveness for automatic negotiation of parameter values. In this paper the use of ASIC based coprocessors for message scheduling is proposed to solve the problem. Such coprocessors can be used in the arbiter nodes of systems based on widely used producer-consumer fieldbuses like WorldFIP and CAN. A prototype built with a Xilinx FPGA is presented. First performance results are shown and analyzed. They demonstrate that the device is able to achieve the expected performance and also point to the possibility of evolution to an almost dynamic scheduling approach.

1. Introduction

Fieldbus based distributed embedded systems (DES) have become quite used in many applications in the automotive industries, avionics, industrial automation and domains where real-time performance is required. One of the drawbacks of many implementations of these systems is the lack of flexibility they usually show in order to keep timeliness guarantees during operation. If a system parameter must be modified, e.g., the period of a variable, the operation must be stopped and the configuration will only be changed after an off-line analysis has determined that there isn't any risk of jeopardizing the real-time performance.

Some recent studies [1] show that DES flexibility and, also, reactivity, can be improved by the use of a planning scheduler technique without loosing the timeliness guarantees required for real-time operation. This applies even when DES's nodes are based on low-processing power micro-controllers as it is the case of systems based in fieldbuses such as CAN [2]. In fact, the planning technique and an associated protocol named FTT-CAN (flexible time-triggered protocol), proposed in [4], can be easily implemented in a typical node of such DESs, keeping a runtime overhead compatible with the CPUs of most applications.

Besides creating flexibility, 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 specific scheduling coprocessors. In this paper, this last solution is analysed and some results concerning the use of a scheduling coprocessor in a CAN-based distributed system are presented. Also, some on-going work with further improvement of performance is briefly discussed.

2. Centralized dispatching in DES solutions

A typical fieldbus architecture is presented in figure 1. Several nodes are connected through the fieldbus and the MAC – Medium access control must have adequate characteristics to be able to guarantee all the timing message transmission constraints of the overall system.

In many DES solutions the MAC is centralized. This means that one of the nodes will decide when a particular message transmission occurs. This is the case, for example, of FIP [3] and of the already mentioned FTT-CAN protocol. This specific node, usually called arbiter, uses a dispatching table in which the traffic pattern of the DES is stored. In FIP the dispatching table in the arbiter contains the description of the periodic traffic for a specific time interval called the macro-cycle. This interval corresponds to the least common multiple of the periods of all the periodic messages used in the DES (or an integer multiple of this number). In the case of FTT-CAN, the dispatching table contains the description of the traffic (in principle periodic) during a fixed time interval called the plan. A scheduler must then be used to generate the

successive dispatching tables as explained in the next section of the paper. This scheduler will be, in principle, a task which is executed in the arbiter node to avoid the problem of loading the dispatching table from anywhere outside it.
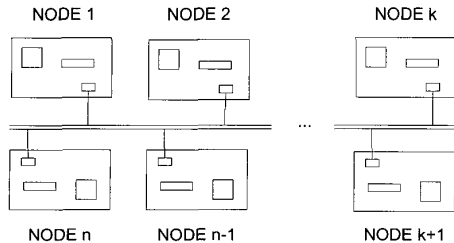

Figure 1. Typical fieldbus architecture.

In what concerns hardware, the typical node of a fieldbus system (figure 2) is based in a microprocessor or micro-controller. The option for low power µC/µP is common as it turns out to be cost-effective. The arbiter node can show a similar architecture if, as in FTT-CAN, it executes the scheduler and dispatching functions in software. However, as it is shown in [5], the scheduler poses a significant overhead to the node CPU. This problem can be solved increasing the interval associated to the dispatching table, giving more time for the scheduler to execute. The consequence is a reduction in the system responsiveness.
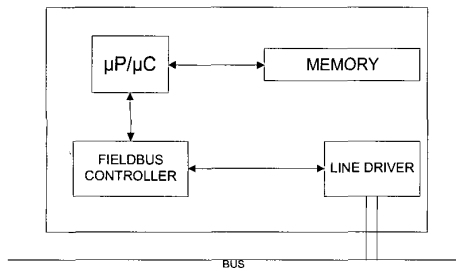

Figure 2. Typical node of a fieldbus system.

A possible solution to obtain flexibility and responsiveness is the use of a dedicated coprocessor for some of the arbiter functions, in special for the scheduler function. This frees the node CPU from the most processing intensive task. Then, a typical off-the-shelf low-power CPU can still be used without jeopardizing performance. Also, as the number of arbiter or potential arbiter nodes is small, its increased complexity has not a strong impact on the overall system cost.

The idea of using a specialized coprocessor seems to be new for the problem of message

scheduling in fieldbuses. However, several other papers have addressed the improvement of execution time and predictability of operating system functions. The real time unit reported in [6] is a rate monotonic based multitasking kernel implemented in an ASIC. The Spring Scheduling VLSI co-processor (SSCoP) [7], works with the Spring Kernel [8], using different scheduling algorithms. In [9] a universal scheduling coprocessor for single processor systems is described.

Returning to fieldbus based systems, and in what concerns the arbiter node architecture, two solutions are envisaged for the dedicated coprocessor: a second µC/µP or an ASIC. The decision between them is not immediate as both present advantages and drawbacks. The µC/µP approach is easy to implement with available tools and it may have a low cost (considering simple devices). However, it also incurs in some problems as the low level of predictability in what concerns worst case execution time of the scheduler operations and the lower dependability when compared with an ASIC. This last problem is enlarged due to the number of sub-systems used with the µC/µP (memory, decoding, interrupt, ...).

On the other side, the ASIC-based solution also starts to present some initial drawbacks such as the difficulty of getting easy and low cost development tools, the skills needed to work it and the final price of the coprocessor unit. However, some features seem promising: the possibility to obtain very low execution times, enabling operation even with pessimistic worst-case timing calculations and the easy synchronization in the read and write accesses performed by the arbiter node CPU.

Also, some drawbacks, like the price are not so important due to the low number of arbiter nodes required in each system and to the progressive price lowering that ASIC-based solutions have been facing along the last years.
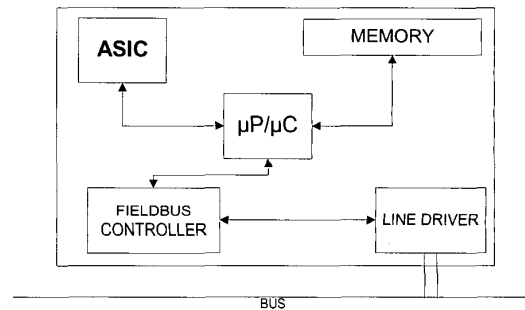

Figure 3. Arbiter node architecture.

In the present work, the ASIC approach is used for the scheduling function, mainly because of the expected gains in the timing response and in the predictability expected for the coprocessor performance. This approach leads to the arbiter node architecture depicted in figure 3.

From the commercial solutions available to implement the ASIC, the choice went to a Xylinx FPGA-based coprocessor due to several reasons such as the possibility of fast programming and reprogramming of the device (without the need for removing it from the system), the availability of software programming tools [10] and the reasonably low cost of the devices.

3. The planning scheduler

The coprocessor described in this paper is based on the planning scheduler [5] whose 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 previous one. In figure 4 the operation of the planning scheduler is illustrated. The dispatcher is working with plan i, while the scheduler is building plan i+1.
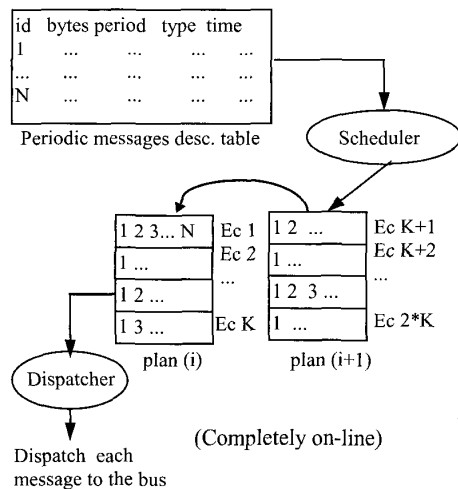

Figure 4. The planning scheduler.

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. Several messages fit, then , within an EC.

The simple mechanism of this scheduler reduces run-time overhead in the node CPU mainly because it is invoked fewer times. So, comparing with a dynamic scheduler, each time it is invoked, instead of determining only the next message to be transmitted, 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 responsiveness is then reduced.

So, if the scheduler execution time is lowered, it is possible to achieve an adequate responsiveness for automatic changes in the message set [5], keeping all the flexibility and simple implementation achieved by the use of the planning scheduler. This can be done recurring to the coprocessor described below.

4. The Planning Scheduler Coprocessor (PSCoP)

4.1. Architectural Overview

The hardware coprocessor developed to implement the planning scheduler, called PSCoP, needs to be initialized with the parameters of each message to be scheduled. These include the message's period (P), its initial phasing (Ph) and associated transmission time (C). The parameters of each message 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 messages supported by the coprocessor, 8 in the current version.

Presently, there is no support in PSCoP for explicit deadline or priority parameters. The deadline of all messages is assumed to be equal to their period. Relative priorities are dictated by the allocation of register slots. These are numbered 0 to 7 and have assigned decreasing priorities. The priority of a given message depends on the register slot where its parameters are stored at initialization time. Priorities are then always static.

After instructed to begin, PSCoP starts generating schedules. The results are passed to the node CPU, and consist of one byte per EC, in which each bit identifies a transaction that must be carried out during that interval.

The scheduler algorithm includes two separate activities. One of them, performed in the context of each message, consists in keeping track of the ECs 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 is reflected in the architecture depicted in figure 5. Here, the Message's Production Timer (MPT) units are responsible for the first activity while the Schedule Plan Builder (SPB) takes care of the second activity.
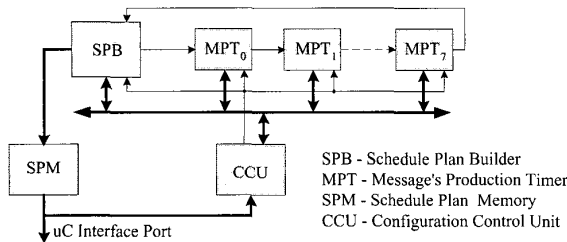

Figure 5. PSCoP architecture.

Each message to be scheduled is allocated to one MPT unit which holds the variable's period (P) and initial phase (Ph) parameters. Global timing information received from the SPB allows all MPTs to be synchronized while keeping track of the EC schedule currently being generated. The MPT uses a timer to determine the ECs when the respective message is ready to be transmitted. When the MPT detects the ready condition, 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 MPTs) are received, otherwise the current EC schedule is finished and a new one is started.

Because more than one MPT 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 MPT0 down to MPT7. When a MPT 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 MPT 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 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 MPTs and SPB.

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 a byte where each bit set represents a specific transaction in that EC. The diagram in figure 6 illustrates the relationship between the transactions placed in EC time slots, and EC schedules in the SPM.

An initial feasibility study and preliminary presentation of PSCoP architecture can be found in [11]. Details about the architecture as well as a full description of PSCoP operation principles may be obtained in [12].
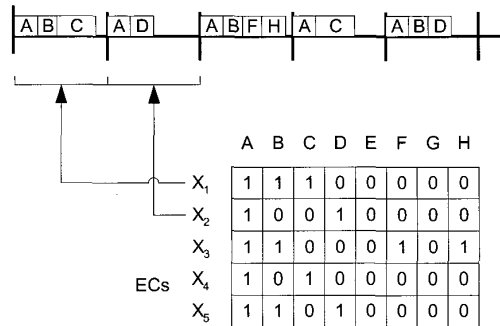

Figure 6. EC schedules and corresponding bus transactions in the timeline diagram.

4.2. Some Implementation Details

PSCoP prototype was implemented on a XC4010XL series FPGA. It has 8 MPTs and a parameter resolution of 8-bits. The memory banks in SPM support 16-EC plans (they are thus 16-bytes FIFO memories). The overall system, depicted in figure 7, uses a XS40 development kit from XESS Corporation [13]. Besides the FPGA, it includes a 8051 microcontroller which is used in many industrial embedded systems. The prototype becomes a CAN arbiter node by adding a SJA1000 [14] controller and an additional transceiver.

The XC4010 FPGA in the prototype was in practice fully used to create the scheduler with room for 8 messages. Some figures about the FPGA resources occupied in this project are the following:

- 400 CLBs (100%), 232 CLB flip-flops and 160 CLB latches;
- 641 4-input look-up tables and 231 3-input LUTs;
- 24 16x1 RAMs;

- 44 IOBs and 2 clock IOBs ;
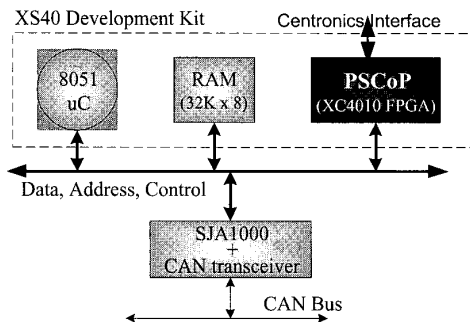
The design equivalent gate count is 8533 gates.

Figure 7. Development and testing plat-form for the PSCoP coprocessor.

In order to bound the signal delays within the FPGA and thus achieve an adequate execution time for the scheduler, it was necessary to impose some constraints to the location of the coprocessor blocks referred in the previous section. This was the case of every MPT to which absolute placement constraints were applied. Also, some blocks of the SPB and SPM (like the control units) were constrained in relative positions, in order to occupy a pre-defined CLB structure.

Figure 8. Placement of the PSCoP blocks in the FPGA die.

Figure 8 shows the placement of the PSCoP blocks in the FPGA array. It can be noticed the regular structure adopted for the MPTs' position. This was done to bound the delay in the daisy chain signal which travels from the SPB through all the MPTs and returns back to the SPB. The overall path delay can reach 44ns, which is more than half a clock period when operating at 12MHz, and may therefore introduce a delay of an additional clock period.

4.3 Measures of the coprocessor performance

The most important performance assessment of the coprocessor comes from the time taken to build a plan schedule. Therefore, measures were taken of the elapsed time between when the run command issued by the microcontroller to the coprocessor and the activation by this unit of the signal that indicates the end of the plan construction.

The coprocessor execution time should depend on the particular message set. In consequence, it was decided to take measurements with 9 different message sets, containing from 0 to 8 messages. This approach allows to observe the variation of the PSCoP execution time with the number of messages. All sets were defined in a way that all the messages are allocated in all the ECs of the plan, which is, in principle, a worst case situation for the coprocessor operation.

To do this, the requirements table of the message set has the specified number of messages, with the respective parameters identical for every one:

- The initial phase is always 0.
- The period is always 1EC. As a result, the deadline is also always 1 EC.
- The transmission time and the EC duration are set so that it is possible to have simultaneous release of all messages in every EC.

# Messages	0	1	2	3	4	5	6	7	8
Exec. Time (μsec)	8	16	22	30	36	41	50	56	63

Table 1. PScoP performance results @ 12MHz.

Table 1 shows the values of the coprocessor execution time for the referred experiments. As it can be seen, the maximum execution time is 63μsec for a set with eight messages. Considering that the worst case transmission time of a CAN 2.0A message at a 1Mbit/sec transmission rate (as in ISO-11898 [15]) is between 53 and 130 μsec [16], for a length of the data field of 0 and 8 bytes respectively, and that the EC must, in principle, have room for several messages, it is obvious that this execution time is an order of magnitude below the values needed when the planning approach is used.

Also, on a preliminary analysis that needs further validation, it seems that the execution time grows linearly with the number of messages in the set. If this result is confirmed, it means that a set with, e.g., 20 messages would be scheduled in a time interval of the order of 200 μsec. This means that with this coprocessor the scheduling could possibly be done on an EC basis instead of a plan basis. The system operation will then become practically fully dynamic.

5. Conclusions and Future Work

A coprocessor for traffic scheduling in fieldbus-based DESs was described in this paper and the first performance results were presented and briefly discussed. The coprocessor can operate with an off-the-shelf microcontroller in the arbiter node of DESs using, e.g., FIP or CAN fieldbuses.

The results demonstrate that the device is more than adequate to be used in flexible DESs built around a planning scheduler. The low values of the scheduling operation execution time will enable to develop systems with very good responsiveness as soon as a future version of the coprocessor with the possibility of changing parameters during operation is available.

Besides this last improvement in the coprocessor operation and the expansion of the maximum number of messages, work in progress includes a new version in which different scheduling policies like rate monotonic, deadline monotonic or priority based can be used and switched dynamically. In this version a response time-based schedulability analysis function based on the timeline method [1] will also be available.

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] Philippe Leterrier, "The FIP Protocol", WorldFip Europe, Antony, France, 1992.

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

[5] L. Almeida, R. Pasadas, J.A. Fonseca - "Using The Planning Scheduler to Improve Flexibility in Real-Time Fieldbus Networks" *IFAC Control Engineering Practice* Vol. 7, N. 1, pp. 101-108, January 1999.

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

[7] D. Niehaus et. al.; "The Spring Scheduling Coprocessor: Design, Use, and Performance"; *Proc. 14th IEEE Real-Time Systems Symposium*, North Carolina, U.S.A, 1993, pp.106-111.

[8] J. Stankovic, K. Ramamritham, "The Spring Kernel: a new Paradigm for Real-Time Systems", *IEEE Software*, May 1991.

[9] J. Hildebrandt, F. Golatowski, D. Timmermann; "Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems"; *Proc. 11th Euromicro Conf. Real-Time Systems*, England, June 9-11, 1999, pp.208-215.

[10] V. Sklyarov et. al.; "Development System for FPGA-Based Digital Circuits", *Proc. FCCM'99: IEEE Symposium on Field-Programmable Custom Computing Machines*, USA, April 1999.

[11] E. Martins, P. Neves, J. Fonseca - "PSCoP – A Planning Scheduler Coprocessor", *Proc. WIP Session - WFCS'2000 – 3rd IEEE Int. Workshop on Factory Communication Systems*, Porto, Portugal, 5-8 Sept. 2000.

[12] E. Martins, P. Neves, J. Fonseca – "Architecture of a Fieldbus Message Scheduler Coprocessor based on the Planning Paradigm", submitted to *Microprocessors and Microsystems*.

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

[14] SJA1000 Stand-alone CAN Controller, Preliminary specification, Philips Semiconductors, 1999.

[15] ISO/DIS 11898, "Road Vehicles – Interchange of Digital Information - Controller Area Network (CAN)", Feb. 1992.

[16] K. Tindell, H. Hanssom, A. Wellings; "Analysing Real-Time Communications: Controller Area Network (CAN)"; *Proc. RTSS'94 – Real-Time Systems Symposium*, Dec. 1994.

Glossary of acronyms

ASIC - Application-Specific Integrated Circuit
CAN – Controller Area Network
CCU – Configuration Control Unit
CLB – Configurable Logic Block
DES - Distributed Embedded Systems
EC – Elementary Cycle
FIP – Factory Instrumentation Protocol
FTT-CAN – Flexible Time-Triggered protocol on CAN
FPGA – Field Programmable Gate Array
LUT – Look-Up Table
MPT – Message's Production Timer
PSCoP – Planning Scheduler CoProcessor
SPB – Schedule Plan Builder
SPM – Schedule Plan Memory