Proceedings of the 18th International Conference on System Theory, Control and Computing, Sinaia, Romania, October 17-19, 2014

Hardware Event Handling in the Hardware Real-Time Operating Systems

Elena-Eugenia (CIOBANU) MOISUC, Alexandru-Bogdan LARIONESCU, Ioan UNGUREAN
Faculty of Electrical Engineering and Computer Science
Stefan cel Mare University of Suceava
Suceava, Romania
neli.ciobanu@eed.usv.ro, lari@eed.usv.ro, ioanu@eed.usv.ro

Abstract—**An important issue related to Real-Time Operating Systems is the handling of interrupts, timers, mutexes, watchdog timers, synchronization and communication directives as unitary events. In order to obtain a predictable response time for the different types of events that occur simultaneously, it is necessary to have a prioritization mechanism for them. Therefore, this paper proposes a hardware mechanism for handling the events enumerated above, in order to improve the software solution, which is not efficient because it generates delays. The method is implemented in n-tasks Multi Pipeline Register Architecture, that is a microcontroller architecture with Real-Time Operating Systems capabilities implemented in hardware, which allows switching time between tasks of one processor cycle and a response time to events of up to 1.5 processor cycles.**

Keywords—embedded systems; event handling; hardware scheduler; pipeline register; real-time operating systems.

I. Introduction

Today, more and more complex systems are based, in whole or in absolute terms, on the control processor. The fields of use may involve human safety, imposing strict time conditions. Therefore, the development of real-time mechanisms and scheduling of the processes with demanding time conditions is a challenge [1], [2]. In this context, operating systems must provide mechanisms for real-time tasks scheduling to ensure meeting strict time conditions.

In systems where resources are limited, a situation found in most digital control systems (embedded systems), an effective implementation of schedulers is crucial [3]. Real-Time Operating Systems (RTOS) have been designed for real-time physical systems, whose operation depends not only on the logical result of the processing, but also on the time that the results are produced [4]. Thus, time becomes an essential coordinate with impact in all phases of development and exploitation of the whole system [2].

A Real-Time System (RTS) is a system fast enough to guarantee the deadlines for the worst case operating scenario [2]. Scheduling tasks relate to finding reliable solutions for processor assignment for each task, so there is no overlap in their execution during the operation of the system [2]. A parameter that can affect the performance of an RTS is the overhead generated by the operating system. Scheduling and switching context operations can significantly influence the deadlines in critical RTS. This is the reason for the implementation in hardware of the scheduling algorithms instead of using software scheduler. An important feature of hard Real-Time Systems is the determinism and predictability of the critical real-time tasks. The overhead generated by the task context switching operations and the execution rate of the task are just two factors that can cause jitters and missing deadlines in RTS with software schedulers.

Gaitan *et al.* proposed a hardware scheduler architecture [5] integrated into the structure of a CPU with multiplied resources, namely pipeline registers and CPU registers. They also proposed several original hardware static and dynamic task scheduling architectures, unified management of different types of events, access to shared resources, the generation and transfer of messages between tasks that ensure synchronization and communication and a method for interrupt assignment to tasks that allows the control of their behavior.

Multi Pipeline Register Architecture (MPRA) uses a hardware scheduler that is a constituent part of the processor, and its control is via dedicated instructions that are transmitted through the pipeline. The saving task context in classical processor architectures involves saving CPU registers in memory or on the stack, and it can produce jitter and thus may affect the response time of critical RTS. In MPRA processor, the Program Counter (PC) register, pipeline registers and, general registers are multiplied resources, and the memory necessary to implement these registers is directly proportional to the number of tasks from the system. In order to ensure the predictability of RTS, MPRA implements a rigid scheduling scheme which is based on the address priority encoder within the Hardware Scheduler Engine (HSE). However, it leaves enough freedom to implement user desired scheduling algorithms, like Earliest Deadline First (EDF) or Rate Monotone Algorithm (RMA), by a proper ordering of tasks depending on the frequency of execution or deadlines. The MPRA register file has a special implementation which, in addition to context remapping done under the direct control of the HSE, allows the isolation of the procedure calls, so that the user does not have to save the contexts. The described architecture uses only the organizational structure of the Microprocessor without Interlocked Pipeline Stages (MIPS) architecture. The architecture includes the scheduler functionality in a processor functional block and offers the possibility of switching task context in just half of a clock cycle, thus eliminating the disadvantages of the software schedulers. The performance of MPRA does not consist in processing power, but in the speed of switching tasks' contexts and the execution speed of the scheduling algorithm. Studies extend the basic ideas expressed in [6], [7], defining new original functionality for *nHSE* and *nMPRA* (the hardware architecture is presented in fig. 1).

The aim of this paper is to further expand the solution presented in [5] by improving the performances and predictability of the interrupt system and event handlers. The novelty presented in this paper refers to the new hardware implementation of the event selection mechanism using trap registers, the architecture of PC register and the introduction of the *ret esr* (*return from the event service routine*) instruction, to allow automatic handling of events. The solution for the interrupt system presented in this paper has a high degree of flexibility and unlike the software solutions, provide the same response time for all interrupts and events. The solution is applicable for small microcontrollers.

The paper is structured as follows: section II presents the *nMPRA*, section III presents the hardware event handling, section IV presents the *PC* modified architecture, and section V contains the final conclusions.

II. nMPRA

MPRA was transformed into *nMPRA* [5] (fig. 1), by multiplying it *n* times; for each task we have a set of pipeline registers (IFID, IDEX, EXMEM, MEMWB), a Program Counter register and a Banked Register File. Because each task has its own set of pipeline registers and general registers, the context switching operation can be done in one to three CPU cycles, and the response time to external events may be delayed up to 1.5 CPU cycles, the architecture being very fast. An instance of this processor is called "semiCPU" (*sCPU*). All *sCPU* are identical, except *sCPU*$_0$, which will be the only one active after reset, it is the supervisor of the system and it has access to *nMPRA* monitoring registers. *sCPU*$_0$ has the highest priority (0) in the system and this priority cannot be changed. In this architecture, the hardware scheduler HSE [5] (fig. 1) is included in the processor and thus doesn't require additional time for bus arbitration, nor delays the results due to data transfer between the scheduler and processor, and can be directly controlled by instructions sent through the pipeline. Because we have multiple pipeline registers, it causes an isolation of the hardware contexts. The interrupts are considered events that can be assigned to tasks or to the real-time operating systems, and they are treated as the threads [8], not as interrupts in the classical manner.

III. Hardware handling of the events

When multiple events assigned to a task (*sCPU*$_i$) are activated simultaneously, we must find a method to select the handling ordering. After we select this ordering, by using the scheme from fig. 2, it obtains the address of the event service routine associated with the selected event (the event service routine is executed by the *sCPU*$_i$ task).

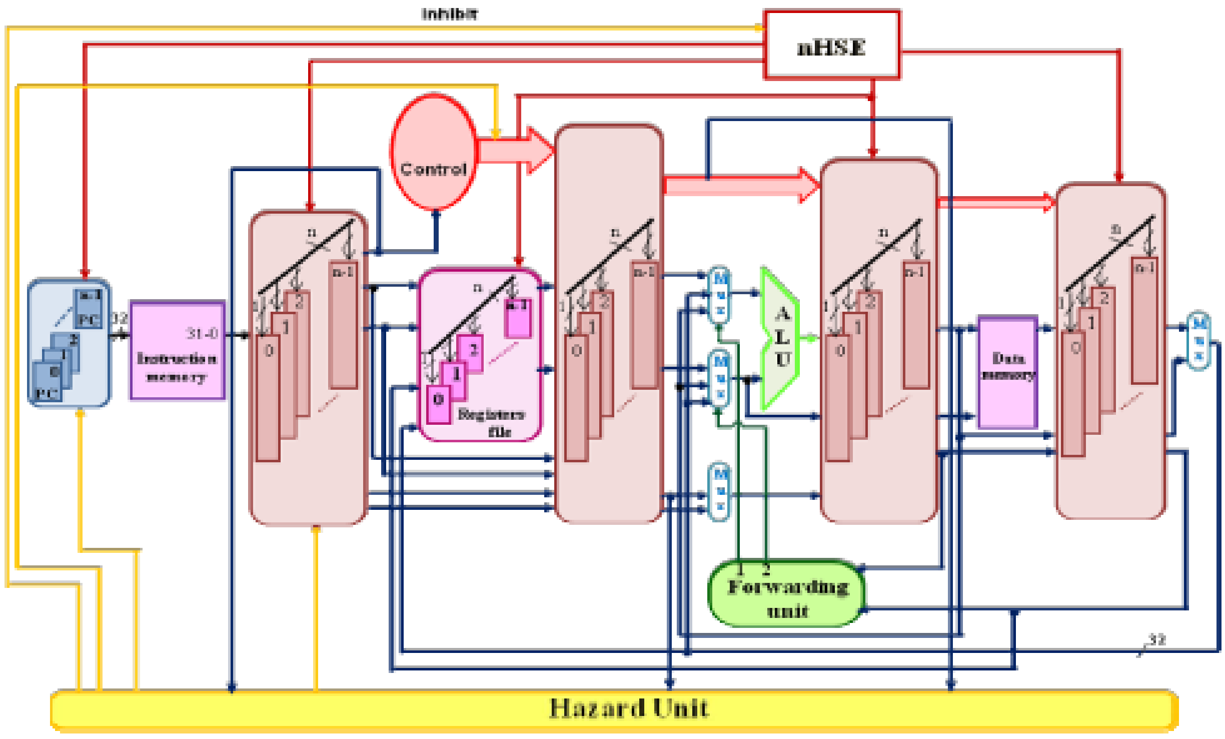


Fig. 1. The *nMPRA* hardware architecture

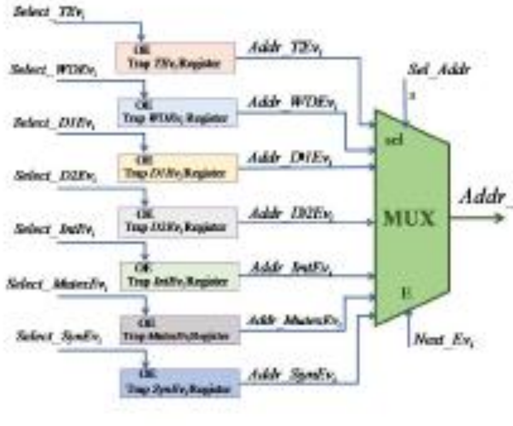


Fig. 2. Selection of the highest priority event handler address

Considering that we have eight types of events in the system, we attach a priority level to each type of event, ranging from 0 to 7. This priority level will be saved into a special register, called the Event Priority Register (*EPR*$_i$), which is attached to every *sCPU*$_i$ (fig. 3). At boot time, *sCPU*$_0$ is the only task active that can initialize all the others *sCPU*$_i$ tasks and can set the priority level of each event associated with every *sCPU*$_i$ in concordance with the specifications of the application.

Fig. 3 presents a prioritization scheme that selects the current active type of event with the highest priority, in order to be served [9]. Depending on the user's demands, the priority level of each type of event can be static and set at the beginning of the application (offline), or it can be dynamically changed during the execution of the application (online). If the selected type of event contains multiple active events, we must make another selection of the event that will be treated first, depending on the type of the event.

Fig. 3 presents the global events prioritization scheme. Considering that in *nMPRA* we have eight types of events, we will need eight schemes for decoding and selecting of the events (see left side of fig. 3). The priorities of the event categories are grouped in the Event Priority Register (*EPR*$_i$) and can store the priorities of the following type of events [5]: *Pri_TEv*$_i$, *Pri_WDEv*$_i$, *Pri_D1Ev*$_i$, *Pri_D2Ev*$_i$, *Pri_IntEv*$_i$, *Pri_MutexEv*$_i$, *Pri_SynEv*$_i$, and *Pri_RunEv*$_i$. The activation signal of the each type of event activates a decoder that generates the priority of the event type accordingly to the priority level stored in the *EPR*$_i$. The output of the priority field is also used for selecting the active input of the MUX multiplexers, which collects the result of the prioritization scheme presented on the right side of fig. 3.

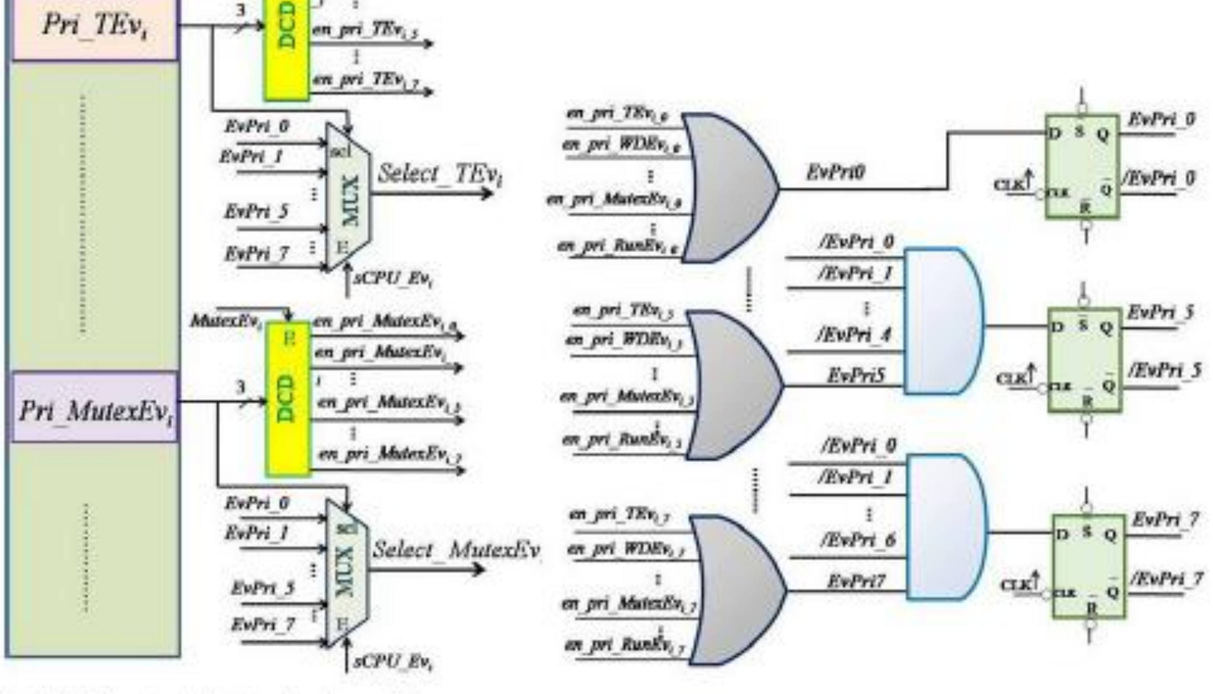
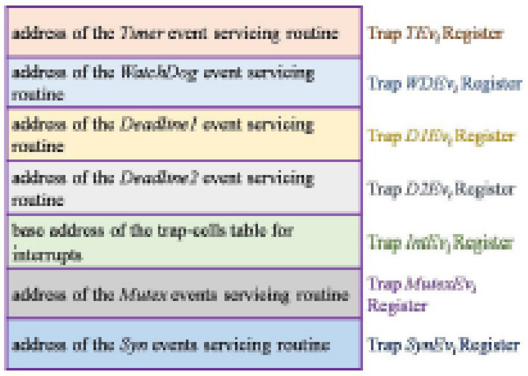


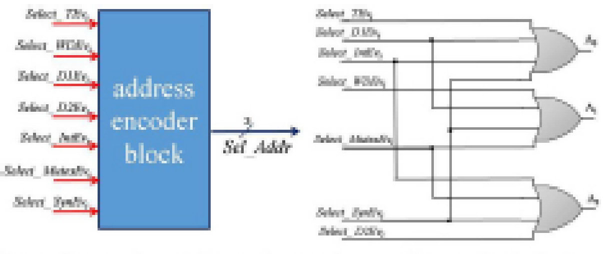
Fig. 3. Global event prioritization scheme (source: [9])



address of the *Timer* event servicing routine	Trap TEv_i Register
address of the *WatchDog* event servicing routine	Trap $WDEv_i$ Register
address of the *Deadline1* event servicing routine	Trap $D1Ev_i$ Register
address of the *Deadline2* event servicing routine	Trap $D2Ev_i$ Register
base address of the trap-cells table for interrupts	Trap $IntEv_i$ Register
address of the *Mutex* events servicing routine	Trap $MutexEv_i$ Register
address of the *Syn* events servicing routine	Trap $SynEv_i$ Register

Fig. 4. Event vector registers

The outputs of the multiplexers are used for selecting an event type to be treated (fig. 5). All the events that are the only ones in their category, like the time-related events (TEv_i, $WDEv_i$, $D1Ev_i$ and $D2Ev_i$), and the ones that are treated globally, through software, have associated one trap register, presented in fig. 5, that points toward their event servicing routine. The addresses of the event service routine are loaded for each task into an event vector register (fig. 4) at startup, by $sCPU_0$ after reset.



$Select\_TEv_i$	$Select\_WDEv_i$	$Select\_D1Ev_i$	$Select\_D2Ev_i$	$Select\_IntEv_i$	$Select\_MutexEv_i$	$Select\_SynEv_i$	A_2	A_1	A_0	
1	0	0	0	0	0	0	0	0	1	1
0	1	0	0	0	0	0	0	1	0	2
0	0	1	0	0	0	0	0	1	1	3
0	0	0	1	0	0	0	1	0	0	4
0	0	0	0	1	0	0	1	0	1	5
0	0	0	0	0	1	0	1	1	0	6
0	0	0	0	0	0	1	1	1	1	7

$$A_0 = Select\_TEv_i + Select\_D1Ev_i + Select\_IntEv_i + Select\_SynEv_i$$
$$A_1 = Select\_WDEv_i + Select\_D1Ev_i + Select\_MutexEv_i + Select\_SynEv_i$$
$$A_2 = Select\_D2Ev_i + Select\_IntEv_i + Select\_MutexEv_i + Select\_SynEv_i$$

Fig. 5. Address selection signals scheme

The trap register of the interrupts contains the base address of the table with trap cells, which memorize the addresses of the interrupt servicing routines. Interrupt handling was analyzed in the context of scheduling models and was proposed an innovative solution to the assignment of events to the tasks (*sCPU*) [8]. Through this method, each interrupt inherits the priority of the associated task. Therefore, the response time to interrupts is more predictable in the context of real-time applications (a task that handle an interrupt can be interrupted only by the interrupts attached to a higher-priority task).

The software solution [8] have as major disadvantage the jitters generated by the test blocks and interrupt service routines when several interrupts are attached to the same $sCPU_i$ and they occur simultaneously.

A hardware solution for the interrupt handling involves extra hardware block that is shown in fig. 6 [8]. The priorities encoder block generates the identifier of the highest priority interrupt when interrupt occurrences. This identifier is multiplied by 4 to calculate the offset in a cells-trap table for interrupts (vector interrupts table). Further, the address of the interrupt service routine is read, and the control is transferred to this routine.

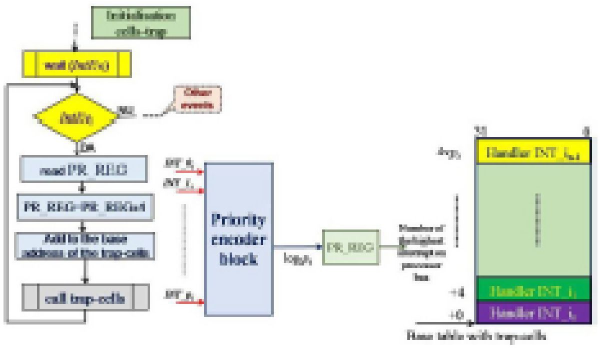


Fig. 6. Hardware interrupts handling (source: [8])

IV. Program Counter Architecture

When an event occurs, the associated task become ready for executions and if it has a higher priority that the task in the execution state then the Program Counter register of the corresponding $sCPU_i$ (PC_i), (see fig. 7) is set by the content of the trap register (the address of the associated routine for the event occurred). In order to implement automatic redirection to the event service routine, the Program Counter (PC_i) should be modified to save internally the return address from an event handler.

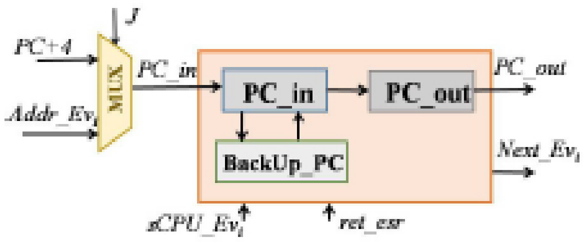


Fig. 7. *Program Counter architecture*

Inside PC_i, there is a register called *BackUp_PC*, which saves the current address from PC_i at the moment when an event occurs, signaled by $sCPU\_Ev_i$ signal. The PC_i will load automatically, using the corresponding trap register, the address of the highest priority active event handler. The return from the event handler is indicated by the execution of the instruction *retesr* (*return from the event service routine*), which determines the activation of the signal *ret_esr*, that indicates to PC_i to load the return address from the *BackUp_PC* register, continuing normal execution of the program. Saving the return address in *BackUp_PC* register determines the deactivation of the $NextEv_i$ signal, indicating that an event is currently being treated and no other event can be serviced until the end of current event handling. After returning from the current event handler, the signal $NextEv_i$ will be reactivated to indicate that the next event can be processed. Fig. 7 shows the structure of PC_i.

V. Conclusions

Global prioritization scheme from [9] is extended in this paper, in order to finish the implementation of the hardware events treating scheme for the *nMPRA* architecture. It is presented the trap registers associated with the event categories and the structure of the Program Counter register (PC_i). The prioritization scheme presented in this paper permits the events handling in hardware. The advantage is reducing the time to detect the event source, and to start the suitable event servicing routine. It even permits the introduction of new events in the system simply by adding the necessary fields in the Task Register (TR_i), Event Status Task Register ($ESTR_i$) and Event Priority Register (EPR_i), updating the global event prioritization scheme and inserting a new trap register for each new event category. The scheme is simple and can be applied to all the events.

As future work, we plan to introduce the optimization of the implementation for mutex events and synchronization and communication primitives.

Acknowledgment

This paper was supported by the project "Sustainable performance in doctoral and post-doctoral research PERFORM–Contract no. POSDRU/159/1.5/S/138963", project co-funded from European Social Fund through Sectorial Operational Program Human Resources 2007-2013.

References

[1] G. Butazzo, "Hard Real-Time Computing Systems", Pavia-Italz: Spinger, 2005.

[2] N.C. Gaitan, " Real-time Acquisition of the Distributed Data by using an Intelligent System", Electronics and Electrical Engineering, Kaunas: Technologija, No. 8, Issue 104, Octomber 2010, ISSN 1392-1215.

[3] M.V. Micea, C.S. Certejan, V. Stangaciu, R. Cioarga, V. Cretu, and E. Petriu, "Inter-Task Communication and Synchronization in the Hard Real-Time Compact", ROSE 2008 – IEEE International Workshop on Robotic and Sensors Environments, Ottawa – Canada, 2008.

[4] G. Butazzo, P. Gai, "Efficient EDF Implementation for Small Embedded Systems", OSPERT 2006- Workshop on Operating Systems Platforms for Embedded Real-Time applications, Dresden- Germany, 2006.

[5] V.G. Gaitan, N.C. Gaitan, I. Ungurean, "CPU Architecture based on a Hardware Scheduler and Independent Pipeline Registers", IEEE Transactions on VLSI System, 2014.

[6] E. Dodiu, V.G. Gaitan, A. Graur, "Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – architecture description", IEEE 35'th Jubilee International Convention on Information and Communication Technology, Electronics and Microelectronics, Croatia, May 2012.

[7] E. Dodiu and V.G. Gaitan, "Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – concept and theory of operation", 2012 IEEE EIT International Conference on Electro-Information Technology, Indianapolis, USA, 6-8 May 2012, ISBN: 978-1-4673-0818-2, ISSN: 2154-0373.

[8] N.C. Gaitan, V.G. Gaitan, E.E. (Ciobanu) Moisuc, "Improving Interrupt Handling in the nMPRA", 12th International Conference on Development and Application Systems, Suceava, Romania, May 15-17, 2014, ISBN: 978-1-4799-5094-2/14.

[9] E.E. (Ciobanu) Moisuc, Al. B. Larionescu, V. G. Gaitan, "Hardware Event Treating in nMPRA", 12th International Conference on Development and Application Systems, Suceava, Romania, May 15-17, 2014, ISBN: 978-1-4799-5094-2/14.

2014 18th International Conference on System Theory, Control and Computing

ICSTCC

Joint Conference

SINTES 18 SACCS 14 SIMSIS 18

PROGRAM & BOOK OF ABSTRACTS

technically co-sponsored by IEEE CSS

ORGANIZERS

October 17-19, 2014
SINAIA - ROMANIA

ICSTCC 2014

18th International Conference on System Theory, Control and Computing

(Joint conference SINTES 18, SACCS 14, SIMSIS 18)

October 17 – 19, 2014
Sinaia, ROMANIA

Editors:
Mihaela Hanako Matcovschi
Lavinia Ferariu
Florin Leon

Organizers:

- **"Gheorghe Asachi" Technical University of Iași**
  Faculty of Automatic Control and Computer Engineering
- **University of Craiova**
  Faculty of Automation, Computers and Electronics
  Automatic Control Research Centre
- **"Dunărea de Jos" University of Galați**
  Faculty of Automatic Control, Computers, Electrical and Electronics Engineering

Technically co-sponsored by

IEEE – CSS Control System Society

Strîmbeanu, Daniel Univ. of Craiova

This paper presents the sensorial system for structure of a backbone hyper-redundant arm with an electro-pneumatically system for position control. A system of cables actuated by DC motors is used for bending. Control system is based on a PIC microcontroller. The position of the robot can be obtained by bending it with the cables and by blocking the position of the elements we need, using the electro-pneumatically system. The major advantage of this type of actuation consists in the fact that the robot can be actuated using a boundary control by cables, the position blocking system for any element being relatively simple. The sensorial system is described and the main features of the global system are presented. The advantages of this sensorial system for this robot architecture are discussed

12:30-12:50 FrA2.5

Hardware Event Handling in the Hardware Real-Time Operating Systems

Moisuc (Ciobanu), Elena-Eugenia	*Stefan cel Mare* Univ. of Suceava
Larionescu, Alexandru-Bogdan	*Stefan cel Mare* Univ. of Suceava
Ungurean, Ioan	*Stefan cel Mare* Univ. of Suceava

An important issue related to Real-Time Operating Systems is the handling of interrupts, timers, mutexes, watchdog timers, synchronization and communication directives as unitary events. In order to obtain a predictable response time for the different types of events that occur simultaneously, it is necessary to have a prioritization mechanism for them. Therefore, this paper proposes a hardware mechanism for handling the events enumerated above, in order to improve the software solution, which is not efficient because it generates delays. The method is implemented in n-tasks Multi Pipeline Register Architecture, that is a microcontroller architecture with Real-Time Operating Systems capabilities implemented in hardware, which allows switching time between tasks of one processor cycle and a response time to events of up to 1.5 processor cycles.

FrA3 Carpati

Fractional Order Modeling and Control (Invited Session)

Chair: Copot, Cosmin	Univ. of Ghent
Co-Chair: Muresan, Cristina Ioana	Tech. Univ. of Cluj-Napoca
Organizer: Copot, Cosmin	Univ. of Ghent
Organizer: Muresan, Cristina Ioana	Tech. Univ. of Cluj-Napoca

11:10-11:30 FrA3.1

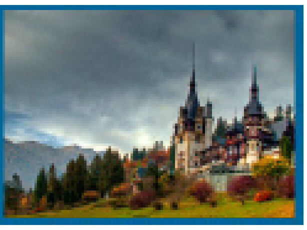
Stabilizing Control Strategies: A Comparison between the Fractional Order Controller and the IMC (I)

Folea, Silviu Tech. Univ. of Cluj-Napoca

18th International Conference on System Theory, Control and Computing

Joint Conference SINTES 18, SACCS 14, SIMSIS 18

17 - 19 October 2014, Sinaia, Romania



Mrs. Elena-Eugenia Moisuc (Ciobanu)
Stefan cel Mare University, Suceava
neli.ciobanu@eed.usv.ro
720229 Suceava
Romania

July 4, 2014

Dear Mrs. Elena-Eugenia Moisuc (Ciobanu),

On behalf of the Program Committee, it gives me great pleasure to invite you to participate in *the 18th International Conference on System Theory, Control and Computing ICSTCC 2014* which will be held at the Rina Sinaia Hotel, Sinaia, ROMANIA, during October 17 - 19, 2014.
The *ICSTCC 2014* is technically co-sponsored by the IEEE Control Systems Society (CSS). The Proceedings will be published in the *IEEE Xplore Digital Library* and will be submitted for indexing in the *Conference Proceedings Citation Index*.

Your paper submitted to the *ICSTCC 2014* has been accepted for presentation by the conference. As indicated in the notification letter sent to you about your paper's acceptance, at least one author of your paper must attend the conference to present the paper. We hope that you will participate in this scientific meeting.
Acceptance of your paper for presentation does not, in any way, financially oblige *ICSTCC 2014* for the expenses incurred by you to travel and attend the conference. If you have any questions, please contact us at icstcc2014@ac.tuiasi.ro.

WARNING: Depending on your citizenship, you may require visa to enter Romania. For additional information about visa and travel authorization, please visit the following website: http://www.mae.ro/en/node/2040

Thank you in advance for your participation. I look forward to seeing you in Sinaia.

Sincerely,
Prof. Mihail Voicu, General Chair of the ICSTCC 2014
[illegible]

Accepted Paper details:

Elena-Eugenia Moisuc (Ciobanu), Alexandru-Bogdan Larionescu, Ioan Ungurean, "Hardware Event Handling in the Hardware Real-Time Operating Systems."