ANNALS OF "DUNAREA DE JOS" UNIVERSITY OF GALATI
MATHEMATICS, PHYSICS, THEORETICAL MECHANICS
FASCICLE II, YEAR VII (XXXVIII) 2015, No. 1

STUDY ON A MICROCONTROLLER ARCHITECTURE IN A HARDWARE REAL-TIME OPERATING SYSTEM

Elena-Eugenia (Ciobanu) Moisuc[1,*], Nicoleta-Cristina Gaitan[1]

Faculty of Electrical Engineering and Computer Science, "Stefan cel Mare" University of Suceava, Romania,
**Corresponding author: neli.ciobanu@eed.usv.ro*

Abstract
A real-time operating system (RTOS) is characterized by high computing power, specialized scheduling algorithm and high frequency clock outage impacting computing power. RTOSs are mainly used for their high response capability compared to the total volume of work they can perform. If certain events such as interrupts, deadlines, timers, mutexes, or watchdog timers occur simultaneously, a strict management to prioritize them is needed. This paper proposes a hardware method of event management; the software scheme is not acceptable because it causes fatal delays in real-time systems. A microcontroller architecture named n-tasks MPRA (Multi Pipeline Register Architecture) is used, which allows switching between tasks of one processor cycle and a response time to events up to 1.5 processor cycles. In this architecture, each task has its own set of pipeline registers.

Keywords: hardware scheduler, interrupt, microprocessor, pipeline register, real-time system

1. INTRODUCTION

A real-time system is a system where the response time during the return of a result is important and is designed for real-time applications [1], [2]. A *Real-Time System* (*RTS*) must ensure deadlines for the worst case operating. Planning tasks relate to solutions to provide a reliable processor for each task [2]. Overlapping operations occurring in task context switching and at the implementation rate of the task can cause jitter and lack deadlines in the RTS with software schedulers.

Gaitan et al. proposed a hardware scheduler architecture *HSE* (*Hardware Scheduler Engine)* [3], static and dynamic, incorporated into the processor with *Multi Pipeline Register Architecture* (*MPRA*). The *Program Counter* (*PC*), pipeline and general registers are multiplied resources and the memory is directly proportional to the number of tasks in the system [4]. *Hardware Scheduler Engine (HSE)* activates or deactivates the interrupts. Because interrupts follow the same regime as the executive tasks, the enabling or disabling executions are done using the same instructions addressed to the tasks. *HSE* uses a unified space for interrupts and priority tasks and planning rule; the high priority tasks cannot be interrupted by interrupts assigned lower priority tasks. This rule supports the need to ensure implementation deadlines for task completion that should provide real-time response to external stimuli.

This study continues the basic ideas of [4] and [5] for *nHSE* and *nMPRA* (hardware architecture is shown in Fig. 1). The solution presented in this paper has a high degree of flexibility and provides the same response time for all interrupts and events, being applicable for small microcontrollers.

This paper presents some summary results of the doctoral research published in international conferences proceedings attended.

The paper is organized as follows: section 2 presents the *nMPRA*, part 3 presents some data about *nHSE*, section 4 includes the treatment of hardware events with a hardware prioritization system, in section 5 the *PC* architecture is presented, and part 6 shows the conclusions.

2. nMPRA

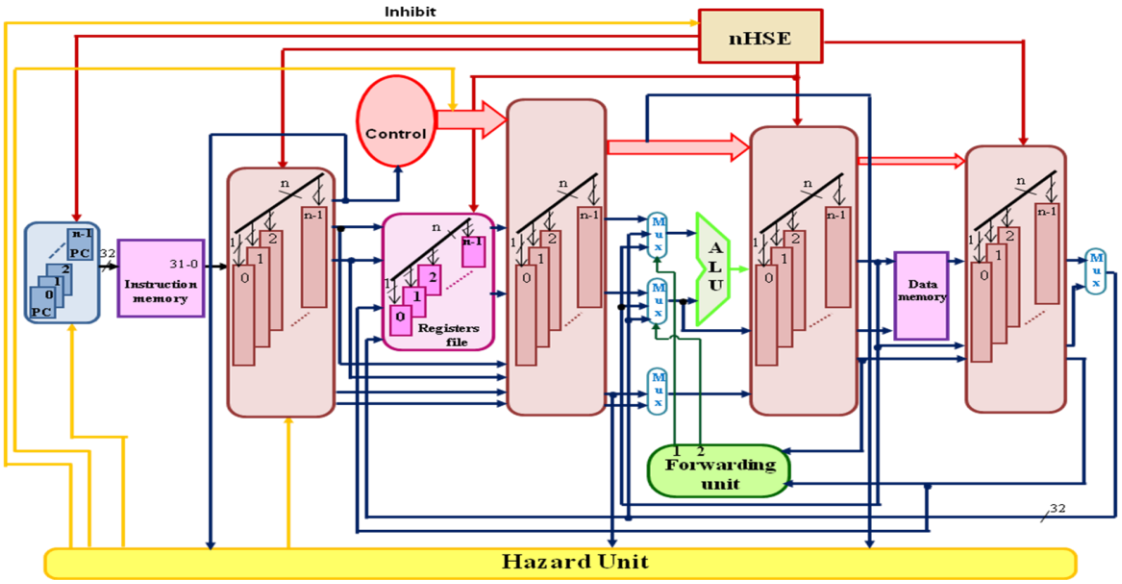


Fig. 1. The nMPRA (source: [7])

The $sCPU_i$ treats a single task. Each instance of task *i* has a set of pipeline registers, a register file, a *PC* register and some control registers. All $sCPU_i$ are identical except $sCPU_0$ that configures and monitors the *nMPRA* registers [7]. The hardware scheduler continually monitors all events addressed to $sCPU_i$, which are timer events (TEv_i), watchdog timer events ($WDEv_i$), deadline events ($D1Ev_i$ and $D2Ev_i$), interrupts attached to the task *i* ($IntEv_i$), mutexes ($MutexEv_i$), synchronization and communication between task events ($SynEv_i$) and self-supporting execution signal for current $sCPU_i$ ($lr\_runs\_CPU_i$).

3. nHSE

In software schedulers, the execution of the scheduler and the time to switch contexts override due dates and decrease the load planning scheme which would have been useful during the execution of tasks assigned. Hardware schedulers aim to relieve the activity of the processor by planning tasks for it.

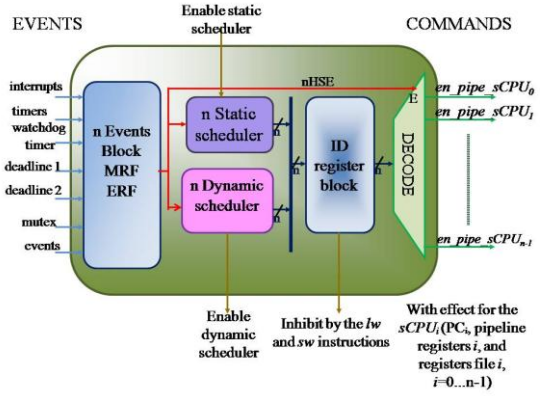


Fig. 2. The nHSE Architecture (source: [5])

MPRA uses an internal scheduler that does not induce inter-processor synchronizations due override nor require additional time for arbitration bus interconnect. *nHSE* (Fig. 2) is a static and dynamic original hardware scheduler which provides unified management for various types of events and a method of attachment of the interrupts to the tasks, allowing control over their behaviour.

4. EVENT HANDLING

The problem to be addressed is the priority of each simultaneously activated event attached to the $sCPU_i$. An *Event Priority Register* (EPR_i) contains the priority level of these 8 types of events. $sCPU_0$ is activated by the *nMPRA* and it sets the priority for the event associated with each $sCPU_i$ according to user requirements.

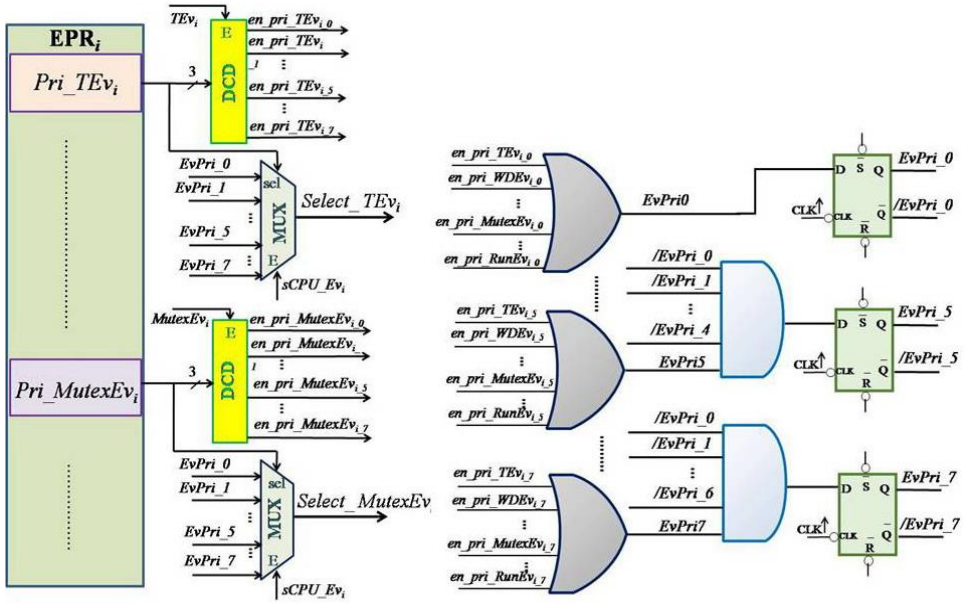
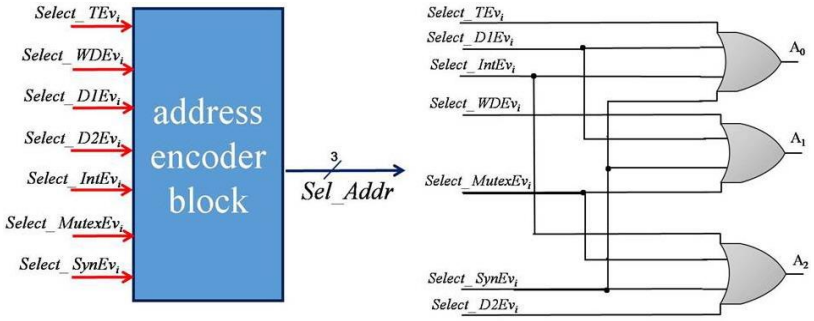


Fig. 4. Global event prioritization scheme (source: [7])

There are 8 types of events, 8 decoders and selection schemes (Fig. 4). The EPR_i register (*Event Priority Register*) stores [7]: *Pri_TEvi*, *Pri_WDEvi*, *Pri_D1Evi*, *Pri_D2Evi*, *Pri_IntEvi*, $Pri\_MutexEv_i$, Pri_SynEv_i, and Pri_RunEv_i. The 8 multiplexers *MUX* collect the outputs of flip-flops D (fig. 4).



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 signal selection scheme (source: [7])

The multiplexers assure the selection of the type of event to be treated (Fig. 5). All events that are unique in their category and which are software globally treated have an associated trap-register shown in Fig. 5. The routine addresses are loaded into an event vector-register (Fig. 6) at boot time by $sCPU_0$ after reset.

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. 6. Event vector registers (source: [7])

Interrupt handling contains a hardware solution for assigning the events to the tasks [5]. Thus, each interrupt inherits the priority of the associated task.

The major disadvantage of the software solution [5] are the jitters generated by the test blocks and the service routines for interrupts when multiple interrupts are attached to the same $sCPU_i$ and occur simultaneously.

In Fig. 7 a hardware solution for interrupt handling is described [5]. There is an encoder of priorities for the interrupts. Multiplying by 4, the movement in a trap-cells table for interrupts is calculated. The interrupt service routine obtains the control.

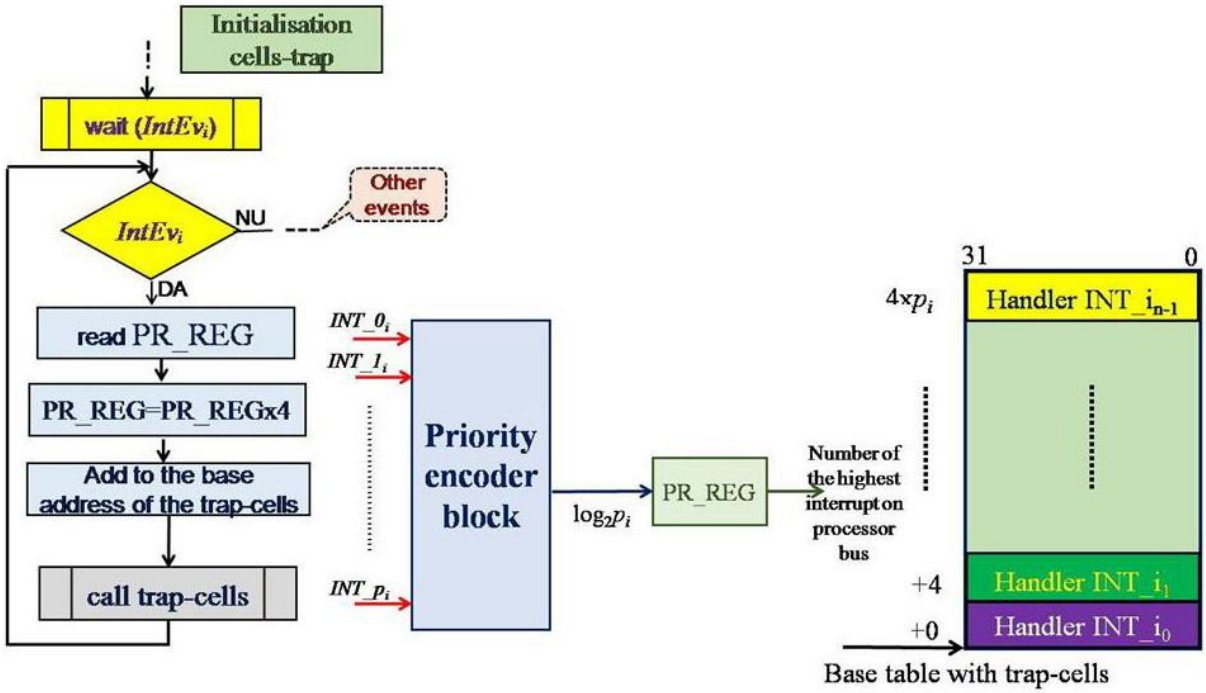


Fig. 7. Hardware interrupt handling (source: [5])

5. PROGRAM COUNTER ARCHITECTURE

In the presented hardware solutions, each $sCPU_i$ has a *Program Counter* (PC_i) register. The task attached to the occurred event becomes ready for execution, and then the trap-register sets the PC_i (Fig. 8). The PC_i was changed to implement automatic redirection to the event handling routines [7].

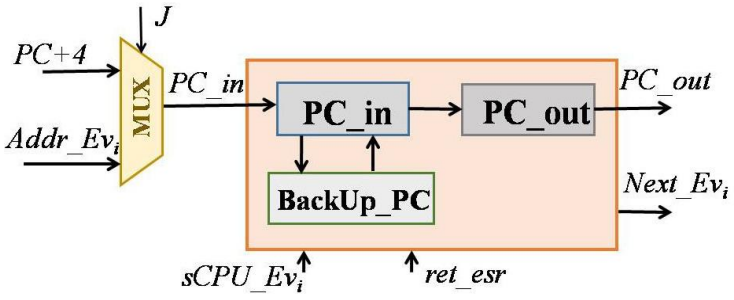


Fig. 8. Program Counter architecture (source: [7])

The *BackUp_PC* is a register that saves the current PC_i address at the reporting of an event by the $sCPU\_Ev_i$ signal. Then, the highest priority address of the active event to be treated automatically uploads in PC_i. The instruction *retesr* (*return from the event service routine*) runs, then the *ret_esr* signal is activated, PC_i uploads the return address from the *BackUp_PC* register, and the normal execution of the program is continued. The $NextEv_i$ signal is disabled, indicating that an event is handled and no other event can be processed until the $NextEv_i$ signal returns from the current event handler [7].

6. CONCLUSIONS

This article presents some team research regarding the management of events [6] and interrupts [5] by implementing effective hardware solutions. The architecture includes scheduler functionality into one functional block of CPU and offers the possibility of switching contexts and tasks in just half clock cycle, eliminating specific disadvantages of software schedulers.

The global prioritization scheme [6] completes the implementation of the scheme for the hardware handling events on *nMPRA*. This scheme can be hardware treating the events. The advantage is the reduced source detection time and start time for corresponding event service routines. The paper shows the trap-registers of the event categories and the structure of the *Program Counter Register* [7].

The survey remains open, especially for improvement to be presented in future articles.

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 the European Social Fund through the Sectorial Operational Program Human Resources Development 2007-2013.

References

1. Gaitan N.C., Real-time Acquisition of the Distributed Data by using an Intelligent System, *Electronics and Electrical Engineering*, Kaunas: Technologija, No. 8, Issue 104, October 2010, ISSN 1392-1215.
2. Gaitan V.G., Gaitan N.C., Ungurean I., CPU Architecture based on a Hardware Scheduler and Independent Pipeline Registers, *IEEE Transactions on VLSI System*, 2014.
3. Dodiu E., Gaitan V.G., Graur A., Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – architecture description, *IEEE 35th Jubilee*

International Convention on Information and Communication Technology, Electronics and Microelectronics, Croatia, May 2012.

4. Dodiu E. and Gaitan V.G., 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.
5. Gaitan N.C., Gaitan V.G., (Ciobanu) Moisuc E.E., 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.
6. (Ciobanu) Moisuc E.E., Larionescu Al.B., Gaitan V.G., 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.
7. (Ciobanu) Moisuc E.E., Larionescu Al.B., Ungurean I., Hardware Event Handling in the Hardware Real-Time, *18th International Conference on System Theory, Control and Computing*, Sinaia, Romania, October 17-19, 2014, ISBN: 978-1-4799-4602-0 ©2014 IEEE.