THE ANNALS OF “DUNAREA DE JOS” UNIVERSITY OF GALATI
FASCICLE IX. METALLURGY AND MATERIALS SCIENCE
N°. 2 - 2015, ISSN 1453 – 083X

DUAL PRIORITY SCHEDULING ALGORITHM USED IN THE nMPRA MICROCONTROLLERS – DYNAMIC SCHEDULER

Lucian ANDRIEȘ, Vasile Gheorghiță GĂITAN
Faculty of Electrical Engineering and Computer Science
“Ștefan cel Mare” University of Suceava
Suceava, Romania
e-mail: andries.lucian2002ro@gmail.com, gaitan@eed.usv.ro

ABSTRACT

This paper is a follow up of an already published paper that described the static scheduler. It deals with a true dynamic scheduling algorithm that is meant to maximize the CPU utilization. The dual priority algorithm is composed of two different scheduling algorithms, earliest deadline first (EDF) and round robin (RR). We have chosen EDF, because it is a dynamic scheduling algorithm, used in real time operating systems, which can be easily implemented in hardware, by improving the nHSE architecture. The new dynamic scheduler algorithm provides a much better CPU utilization, very good switching time for tasks and events within 5 to 8 machine cycles and guarantees that no task will suffer from starvation.

KEYWORDS: real time system, dynamic hardware scheduler, microcontroller, pipeline processor

1. Introduction

In article [1], the author provides a review of the fundamental results of two important scheduling algorithms:

- *Fixed priority (FP - Fixed priority scheduling assumes that the processor will execute the highest priority task among others*).
- *Earliest deadline first* (EDF - Earliest deadline first scheduler is a real time operating system that places processes in a priority queue in order to be scheduled for execution).

The FP algorithm will be assigned a fixed priority that cannot be modified at run time or in the normal operations to each task, while the EDF algorithm is the opposite. Each priority of a task is continuously computed based on the earliest absolute deadline.

Other scheduling algorithms such as *Shortest Remaining Processing timer First* (SRPT) [2] and *Least Laxity first* (LLF) [3] are very powerful and efficient in software, but are difficult to implement in hardware because of the logical ports cost. For this reason we are going to explain in detail only the *EDF* algorithm.

The scheduling algorithms based on FP and EDF, to a certain extent, are good algorithms that can be used in real time operating systems. But in the majority of the commercial real time operating systems, the FP algorithm is implemented due to the simplicity and lower overhead of resources.

The overheads are more visible in software, because EDF can be implemented in different ways that can deal with more or less resources. Each software implementation can lead to more or less CPU utilization.

The hardware resources that are going to be used are not going to increase drastically by EDF, because the Scheduler described in [4] will not be changed at all and the nHSE architecture, described in [6-9], will undergo minor improvements.

The nHSE architecture, described in [8], can support two types of schedulers, namely static or dynamic. So far only the static scheduler has been described and implemented in paper [4], while the dynamic scheduler has been postponed for future implementations.

This paper will present only the theoretical part behind the actual hardware implementation [4]. Based on our experience with the static scheduler, we are convinced that the scheduler will work as intended and the resource utilization will not be much higher.

This paper is organized as follows. The improvements of the nHSE architecture is presented in Section III, the proposed dynamic dual priority algorithm is presented in Section III, followed by the results in Section IV. In the end, in Section V, some conclusions are listed.

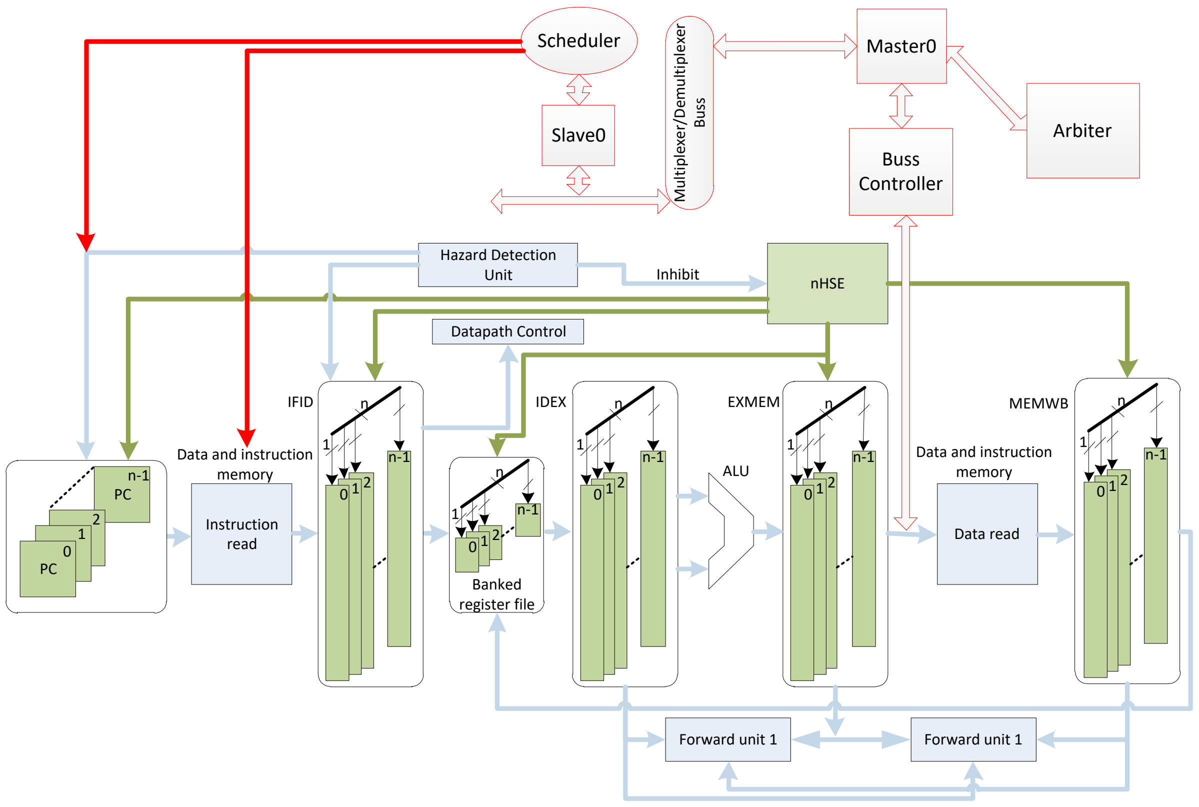

Fig. 1. The nMPRA architecture

2. nMPRA architecture overview

This paper starts with an overview of the nMPRA architecture. In Figure 1, the initial nMRPA architecture [8] is described, while the representations colored in red are the improvements described in [4].

To the current nMPRA architecture [8], was added a slow bus with a *Buss Controller* and a peripheral (Scheduler) which contained the scheduling algorithm (Figure 1).

This new approach optimizes the switching of the hardware tasks in terms of silicon costs and system complexity. During this process, we encounter some synchronization issues, caused by the current architecture of the processor, MIPS with 5 pipeline stages. In order to stop a working task, the Program Counter (PC) must be stopped. The switch process of a task is really simple and is done in two steps.

- Stop the PC from its current task.
- Select the appropriate task to share the resources.

From this description, we can think that the switching of a task can be done in 1 machine cycle, because the stopping of the PC and the selection of the appropriate input / output of the multiplexer / demultiplexer can be done simultaneously.

It could be true if there were no dependencies with the RAM and ALU that were shared. So when the *SelectTask[2..0]* bus (Figure 2) will have a different value, in order to select the new task, the ROM, ALU and ALU will no longer be available for the current task.

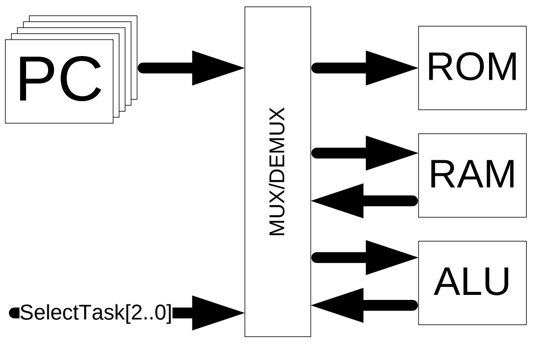

Fig. 2. Simplified nMPRA architecture

We have chosen to wait during 3 machine cycles to solve the synchronization issue. The other 2 machine cycles are used to synchronize the next task program counter address with the ROM memory and the instruction fetch pipeline register because the Scheduler and the program counter use 3 quadrature clock signals.

The whole architecture was designed using VHDL hardware description language (VHDL 93).

Altium Designer 2014 was used, only as a text editor, to include the individual VHDL module and to create the architecture. The design created was simulated using ModelSim Altera Started Edition 10.1d. The only stimuli applied to the microcontroller was 3 quadrature clock signals, while the ROM memory was already initialized with the machine code for all 5 tasks.

The following results were observed, in normal operation, at key points. In the following lines we are going to detail the steps required to do a task switch:

a) *var_process0ready* ((1) in Figure 3) signal represents the event occurring after the activation of the first task. Because no jump instruction is performed, all of the active tasks will be stopped with the help of *processXstall* (X will have values from 0 to 4) signal. The *process1ready* will remain active as a sign that the task is still active. At this moment task1 will be in the ITQ.

b) Wait one clock cycle.

c) *Processactiv* signal selects thread 0 as the active one (3) in Figure 3).

Process0resetstall and *process0startagain* signals are activated ((4) in Figure 3).

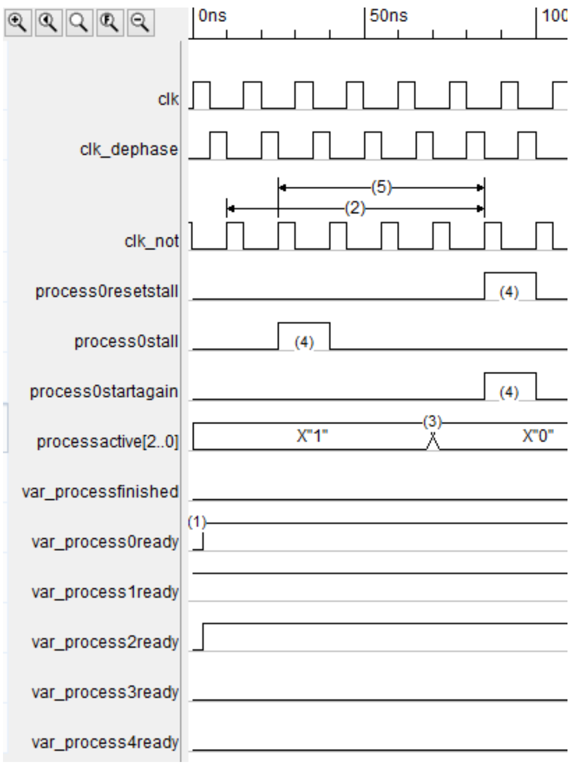

***Fig. 3.** Time for a task switch*

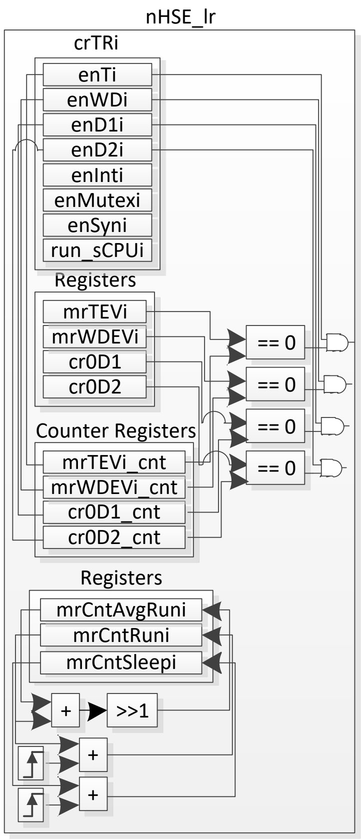

***Fig. 4.** Local nHSE registers*

As we can see in Figure 3 the response time of the scheduler is one machine cycle (between (2) and (5)), the time spent after the task with higher priority becomes active and the starting of the executing code (between (2) and (5) in Figure 3) is done in 5 machine cycles in an interval of 75 ns, where the period of the clock is 15 ns.

3. nHSE Arhitecture Improvements

The architecture of the microcontroller that includes the nMPRA and nHSE architecture firstly described in [8] and then improved in [4] will be improved again in order for the nHSE architecture to make use of the dynamic scheduler.

nMRA architecture stands for n Multi Pipeline Register architecture which means that the most important resources, ROM, RAM, ALU, were shared between multiple hardware tasks.

nHSE architecture stands for "n Hardware Scheduler Engine" and can be used for real time preemptive capabilities of the static and of the dynamic scheduler. It also provides support for:

- Logic events:
  - Interruptions.
  - Events generated by watchdog timers.
  - Events generated by the timer.

  - Events generated by the deadline1 timer.
  - Events generated by the deadline2 timer.
  - Events generated by mutexes.
  - Events generated by synchronization events.
- Static Scheduler: The priorities of n tasks cannot be changed during execution.
- Dynamic Scheduler: The priorities of n tasks can be changed during execution.
- Global and local nHSE registers.

In order to support dynamic scheduling, the nHSE architecture was updated with a new register mrCntAvgRuni, added to the local nHSE registers (Figure 4) for all hardware tasks. The registers will always be equal to the average of the current task execution time. The register mrCntRuni will start to count each machine cycle only after the task has started again. At the end of the task execution the nHSE_lr will add the current value from mrCntRuni to mrCntAvgRun register and shift to the right result, in order to achieve a division by two. The final result will be stored in mrCntAvgRun.

In the following example, we are going to detail the operation performed by nHSE_lr in Table 1:

Table 1. *Average task computation for EDF algorithm*

Step	mrCntRuni	Operation	mrCntAvgRuni
1		mrCntAvgRuni = (mrCntRuni + rCntAvgRuni); mrCntAvgRuni =>> 1;	0
2	500	(500 + 0) >>1	250
3	700	(700 + 250) >>1	475
4	450	(450 + 475) >>1	462
5	900	(900 + 462) >>1	681
6	1000	(1000 + 681) >>1	840
7	1200	(1200 + 840) >>1	1020
8	300	(300 + 1020) >>1	660

As it can be easily seen, the average algorithm has started with the value of the register mrCntAvgRuni equal to 0. If we compute again the same task execution, but this time with floating point, the result will be approximately the same: an arithmetic average will be performed on the number of the processor cycles of the register mrCntAvgRuni from Table 1. The result is (500 + 700 + 450 + 900 + 1000 + 1200 + 200) / 7 = 721.4285714285714 machine cycles.

The difference between the unsigned average and the floating point average is just 61 machine cycles, which can be equal to 12 R type assembler instructions if we consider that an R type instruction is executed in 5 machine cycles.

From the examples above, we can say that the computation error for the unsigned average algorithm is 8.46% which is deducted with the following equation: 100% - (660 * 100) / 721 = 100% - 91.539% = 8.46%

The result 8.46 % indicates the maximum percentage that the unsigned average algorithm can have. The value of the error will decrease as the number of task recurrence increases.

The costs of hardware resources, per hardware task, for a true dynamic dual priority algorithm, are:

- One register of 32 bits.
- An add module;
- A shift to the right by one module.

4. The proposed dynamic dual priority algorithm

This paper is a follow up of the article [4], whose overview is presented in chapter II, where the presented dual priority scheduling algorithm was not a true dynamic scheduler because the dynamic execution of the algorithm was that the priority of the tasks was changed only when one or several tasks did not meet the Round Robin timer (RRB) time constraints.

The algorithm will behave like a true static scheduler only when the following requirements are met:

- The RRB time constrains are met.
- The sum of all concrete task executions is less than the lowest task recurrence.
- No task is going to be promoted to extend the execution tasks class (LTQ).

The main difference between the scheduling algorithm described in article [4] and the algorithm presented in this paper is the use of a true dynamic scheduling algorithm, namely earliest deadline first scheduling (EDF).

In this paper the switching time has not been improved since the last article [4], because we are presenting the other half of the Scheduler. In order to have a better understanding of the algorithm, we are going to reuse and detail some of the information from [4].

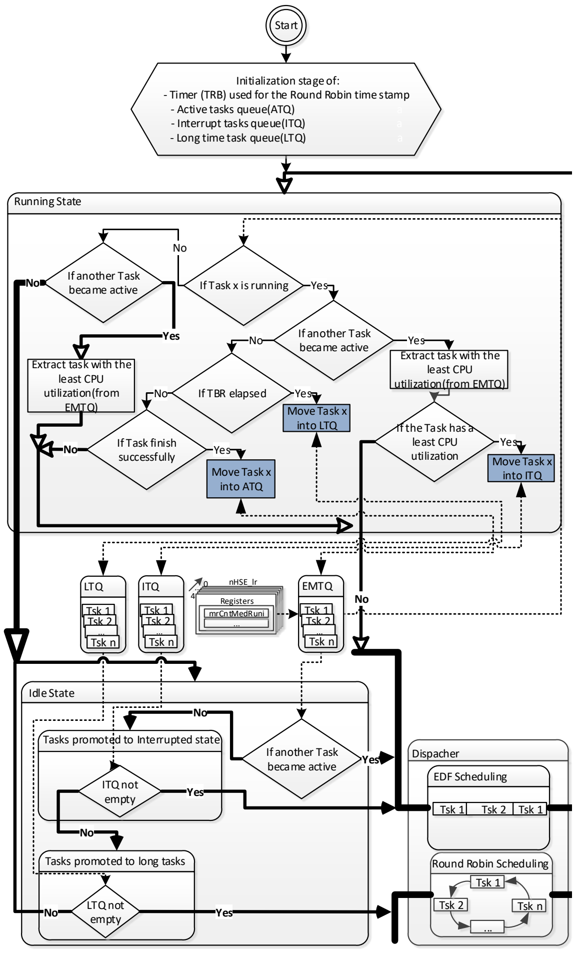
Start
Initialization stage of:
- Timer (TRB) used for the Round Robin time stamp
- Active tasks queue(ATQ)
- Interrupt tasks queue(ITQ)
- Long time task queue(LTQ)
Running State
If another Task became active
If Task x is running
If another Task became active
Extract task with the least CPU utilization(from EMTQ)
If TBR elapsed
Move Task x into LTQ
If the Task has a least CPU utilization
Move Task x into ITQ
If Task finish successfully
Move Task x into ATQ
Yes
No
LTQ
ITQ
nHSE_Ir
Registers
mrCntMedRuni
EMTQ
Tsk 1
Tsk 2
Tsk n
Idle State
Tasks promoted to Interrupted state
ITQ not empty
If another Task became active
Tasks promoted to long tasks
LTQ not empty
Dispacher
EDF Scheduling
Round Robin Scheduling

Fig. 5. The flowchart of a scheduler with a dynamic dual priority algorithm (Figure 3 from [4])

In the following lines, the classes that each task can belong to are presented:

- The class of execution medium time queue, which has the highest priority (the tasks will be inserted by the medium execution time (EMTQ)), will schedule the tasks, based on medium execution time, only in the Running State (RS) of the Scheduler.
- The class of interrupted tasks, which has the second priority (the tasks will be inserted only in the interrupted task queue (ITQ)), will schedule the tasks, based on priorities, only in the Idle State (IS) of the Scheduler.
- The long execution tasks class (these tasks significantly exceed the base period T corresponding to the priority task), which has the least priorities (the tasks will be inserted only in the long task queue (LTQ)), will schedule the tasks, based on ROUND ROBIN (RR) algorithm, only in the Idle State (IS) of the Scheduler.

A global timer, Round Robin timer (TRB), is used to verify if the active time of the current task is not taking too long. If the TRB expires, the current task will be promoted to LTQ and will be scheduled based on RR algorithm.

The TRB must be initialized with the occurrence of the slowest task from the system or less. In Figure 5 is presented the operational flowchart of the dynamic Scheduler (event driven), which is driven by the main clock signal of the processor.

The operation of the Scheduler is exactly the same as in paper [4] whose overview is presented in chapter II, with the main difference that the EDF algorithm will guarantee a better control process unit (CPU) utilization.

5. Conclusions

The current paper presented the true dynamic dual priority algorithm which is the other part of the nHSE architecture, which uses the EDF algorithm for better CPU utilization.

The EDF algorithm will not be efficient in the following cases:

- The execution time of each task is greater than the TRB. In this particular case, only the RR algorithm will be used.
- The execution time of all tasks is less than the time recurrence of the fastest task. This precondition must be achieved in a system where the CPU load is normal.

The EDF algorithm will be efficient by scheduling the tasks whose execution is more likely to be over before another task becomes active, only when the CPU load rises above normal utilization.

The algorithm, shown in Figure 5, will ensure a constant of 5 machine cycles for each switch task and the guaranties that the tasks will be scheduled no matter how difficult the requirements of the system have been.

Acknowledgement

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]. Robert I. Davis, *A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems*, Real-Time

Systems Research Group, Department of Compuyter Science, University of York, York, UK.
[2]. Davis R. I., Burns A., Walker W., *Guaranteeing Timing Constraints Under Shortest Remaining Processing Time Scheduling*. In proceedings of the Euromicro Workshop on Real - Time Systems, p. 88-93, 1997.
[3]. Shaohua Teng, Wei Zhang, Haibin Zhu, Xiufen Fu, Jiangyi Su, Baoliang Cui, *A Least-Laxity-First Scheduling Algorithm of Variable Time Slice for Periodic Tasks*, International Journal of Software Science and Computational Intelligence, Vol. 2, Issue 2, April 2010.
[4]. Lucian Andries, Vasile Gheorghita Gaitan, *Dual Priority Scheduling algorithm used in the nMPRA Microcontrollers*, 18th International Conference on System Theory, Control and Computing, Sinaia, Romania, October 17-19, 2015.
[5]. Lucian Andries, Vasile Gheorghita Gaitan, *Detailed Microcontroller Architecture based on a Hardware Scheduler Engine and Independent Pipeline Registers*, 19th International Conference on System Theory, Control and Computing, Sinaia, Romania, October 17-19, 2014, ISBN 978-1-4799-4602-0, 2014.
[6]. Dodiu E., Gaitan V. G., *Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – concept and theory of operation*, IEEE EIT International Conference on Electro-Information Technology, Indianapolis, IN, USA, 6-8 May 2012.
[7]. 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.
[8]. Gaitan V. G., Gaitan N. C., Ungurean I., *CPU Arhitecture based on a Hardware Scheduler and Independent Pipeline Registers*, IEEE Transactions on VLSI System, 2014, ISSN :1063-8210.
[9]. Gaitan N., Lucian A., *Using Dual Priority Scheduling to Improve the Resource Utilization in the nMPRA Microcontrollers*, IEEE 12th International Conference on Development and Application Systems, Suceava, Romania, May 15-17, 2014.