Software vs Hardware Implementations for Real-Time Operating Systems

Nicoleta Cristina GAITAN[1], Ioan Ungurean[2]
Faculty of Electrical Engineering and Computer Science, Stefan cel Mare University of Suceava
Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD)
Suceava, Romania

***Abstract*—In the development of the embedded systems a very important role is played by the real-time operating system (RTOS). They provide basic services for multitasking on small microcontrollers and the support to implement the deadlines imposed by critical systems. The RTOS used can have important consequences in the performance of the embedded system. In order to eliminate the overhead generated by RTOS, the RTOS primitives have begun to be implemented in hardware. Such a solution is the nMPRA architecture (Multi Pipeline Register Architecture - n degree of multiplication) that implements in hardware of all primitives of an RTOS. This article makes a comparison between software RTOS and nMPRA systems in terms of response time to an external event. For comparison, we use three of the most commonly used RTOS in developing embedded systems: FreeRTOS, uC/OS-III and Keil RTX. These RTOSs are executed on a microcontroller that works at the same frequency as the implementations of the nMPRA architecture on a FPGA system.**

***Keywords*—Embedded system; real time operating systems; microcontrollers; FPGA**

I. INTRODUCTION

Real-time operating systems (RTOSs) [1] are very important at the development of software applications for embedded systems. They allow the modular design and development of the software application and ensure the deadlines imposed by the critical systems. Two very important features of these operating systems are the predictability and the reliability. The RTOS systems are very important in developing software applications for embedded systems because they provide basic services such as multitasking services for synchronization and communication between tasks [2]. Usually, these RTOSs are used for microcontrollers (MCUs) or processors that do not use virtual memory and that have limited code and data memory [3].

In the selecting of an RTOS for embedded application development, several parameters are considered. For critical applications, there are very important the reliability, predictability and the responsive time to the internal or external events (Worst Case Execution Time – WCET). It should be specified that the response time depends on the implementation mode in RTOS but is closely related to the frequency of operation of the MCU and the way of the assignment of the priorities to these events. In addition to these parameters, licensing costs, code and data memory requirements, RTOS overhead and the expertise/experience in using an RTOS used in previous projects are considered [4].

In an embedded market survey published in 2017, 67% of the embedded projects in progress in 2017 use a form of operating system (RTOS, kernel, software executive). Those who do not use an operating system have specified that they do not need it because the applications being very simple and there are not real time application. This study shows a growing trend in the utilization of the open source operating systems and a downward trend in the utilization of the commercial operating systems from 2012 to 2017. The main reason for choosing a commercial operating system is the real-time capabilities [5].

The most used operating systems in the ongoing projects in 2017 are Embedded Linux (22%), FreeRTOS (20%), OS developed in house (19%), Android (13%), Debian (13%), Ubuntu (11%), Window Embedded 7 (8%), Texas Instruments RTOS (5%), Texas Instruments (DSP/BIOS), Micrium (uC/OS-III) Windows 7 Compact or earlier (5%), Keil (RTX) (4%), Micrium (uC/OS-II) (4%), Wind River (VxWorks) (4%). It can be noticed that for small microcontrollers, with low code and data memory and without virtual memory that can be used to develop hard real-time applications (such as those based on ARM Cortex Mx or ARM Cortex Rx), the most used RTOS are FreeRTOS, Texas Instruments RTOS, Micrium uC/OS-III and uC/OS-II. On the systems that use microprocessors virtual memory systems (such as those based on ARM Cortex Ax), it is possible to develop only the soft real-time applications, which are of the best effort type [5].

Typically, embedded applications are a combination of hard real time and soft real time tasks. In order to help the embedded software developers, MCU manufacturers have begun to provide solutions with one or more MCUs for hard real time tasks (for example ARM Cortex M0) and one or more MCUs or microprocessors (such as ARM Cortex A9) for soft real time application. Examples of such solutions are Sitara System on Chip from Texas Instruments or i.MX 7 Series Application Processors from NXP [6][7].

Since the development of FPGA systems, solutions for the implementation of RTOS in hardware have begun to be designed and developed in order to eliminate the overhead generated by the software RTOSs. These systems are experimental, not being widely adopted by the embedded

developers. An example of such system is the nMPRA architecture presented in [8].

This paper presents a comparison of the performances obtained by the real time operating systems implemented in the software (FreeRTOS, Keil RTX and uC/ OS-III) and the nMPRA architecture that implements in hardware the primitives of a real-time operating system [8]. The comparison is performed in terms of the response time of the task with the highest priority at the trigger of an event expected by this task.

Further, this paper is structured as follows: Section II presents a brief description about software RTOSs; Section III presents some solutions for RTOS primitives implemented in hardware, and in Section IV are presented the experimental results achieved. The conclusions are drawn in Section V.

II. Software Implementation of the RTOSs

RTOSs are operating systems specially designed for embedded applications with real time requirements. For this reason, the overhead generated by the kernel execution is very important because it can interfere with response time to external events. Within these systems, the interrupts handling play an important role. In almost all RTOS interrupts are handled outside the kernel, the task being able to synchronize or receive messages from interrupt service routines [9].

According to the market study presented in, the most used RTOS for small microcontrollers is FreeRTOS. This is open source, providing basic services for multitasking, synchronization, and inter-task communication based on a preemptive kernel with static priorities. An analysis of the performances of this RTOS is presented in [12], where the evolution of FreeRTOS is followed over the course of 10 years. From this study, it can be concluded that FreeRTOS has improved its performance in terms real time facilities [5][10] [11].

Another RTOS widely used is Micrium uC/OS-III. This is a commercial RTOS designed to be used in embedded systems with hard real time requirements based on a preemptive kernel with static priorities. It provides all the services of a multitasking system in terms of synchronization and inter-task communication. It is provided as sources in ANSI-C being ported to a wide range of microcontrollers. His predecessor was Micrium uC/OS-II, which is still used in many projects [5] [9].

A royalty-free real time operating system based on CMSIS-RTOS API is Keil RTX. This was designed for applications based on microcontrollers. In the CMSIS-PACK package for Keil MDK is included the RTX kernel, along with source files and libraries. Keil RTX delivers benefits like task scheduling, multitasking, inter task communication, and shorter ISR system management [13].

A comparison of these RTOS in terms of time for task switching is presented in [3]. These comparisons are made by using three types of synchronization objects: events, semaphores, and mutex. The best performances are obtained for Keil RTX, followed by uC/OS-II and rt-thread. The lowest performance is obtained for FreeRTOS (time is approximately twice as high as Keil RTX). From these tests, it can be seen that the most widely used open source RTOS in embedded projects has much lower performance than commercial solutions represented by Keil RTX and uC/OS-II. This is because licensing costs can be an important criterion in selecting an RTOS.

III. Hardware Implementation of the RTOSs

With the development of FPGAs, a new trend has emerged through the hardware implementation of the primitives of an RTOS. Software RTOS treats interruptions outside the executive, interrupting any task running, which means that they can lead to missed deadlines when there is more than one interruption at the same time. To increase the predictability of the RTOS behavior, they can be implemented in hardware. To this end, the nMPRA architecture (Multi Pipeline Register Architecture - n degree of multiplication) was suggested which, together with nHSE (Hardware Scheduler Engine for n tasks), are a solution of a microcontroller with RTOS implemented in hardware. The nMPRA architecture, together with the nHSE module, is an innovative solution with a response time to external events of 1-3 processor cycles, which means a significant improvement over the software solutions of real-time operating systems or over those of software / hardware hybrids. The nMPRA architecture, defined in [8], uses a 5-stage implementation MIPS pipeline. This is a very strong architecture due to its proprieties, namely: switching between tasks is usually carried out in a single machine cycle or in a maximum of three machine cycles when working with global memory. The system's reaction to an external event will not exceed 1.5 clock cycles if the event is attached to a higher priority task than the current task. The pipeline is not the reset; as a result, there is no need to restore/save the context, due to the multiplication of resources (PC, pipeline registers and registry file). It uses a powerful instruction through which a task can wait for various types of events (time, mutex, event, interrupt, timers for deadlines, etc.). The nMPRA is provided with distributed interrupt controller from which the interrupt inherits the task priority; it supports a static scheduling and has support for the dynamic scheduling of tasks. This architecture has been updated continuously [14][15][16].

An example of such operating systems is the ReconOS that is used for reconfigurable computing. This embedded operating system offers OS services for hardware and software execution. It provides a standardized interface that permit to include hardware accelerators. This solution is hardware-software co-design of a RTOS [17].

Another solution for OS hardware implementation is mosartMCU. It is implemented around a 32-bit RISC-V microcontroller and implements most of the OS directives in hardware. This solution achieves a lower response time for the interrupt handling [18].

µC /OS-III HW-RTOS is a hardware implementation of the µC/OS-III operating system. It implements in hardware the primitives of the µC /OS-III with a significant improvement in performance in terms of response time to internal and external events. It is implemented in R-IN32M3 from Renesas around an ARM Cortex M3 MCU [19].

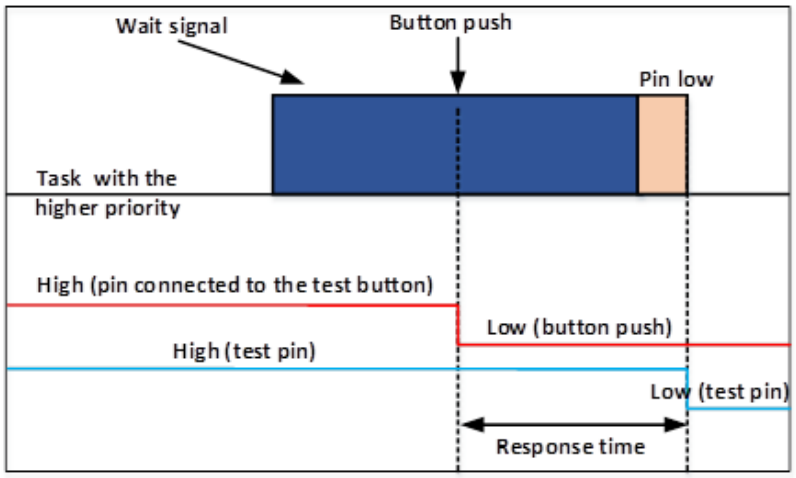

Fig. 1. Time Diagram of Test Applications.

In [20] the authors present the Real-Time Task Manager (RTM) that is an extension for the processor that aims to eliminate the overhead of RTOS primitives. This solution explore the execution in parallel and in hardware the RTOS primitives, function such as scheduling, synchronization primitives, and time management. This solution achieved a decrease of the response time by an order of magnitude.

IV. Performance Comparison

For the experimental tests, it used an implementation of the nMPRA with four task (sCPU) based on an MIPS processor at 33MHz on the Xilinx Zynq-7000 SoC ZC706 Evaluation Kit. For this reason, a microcontroller with the same operating frequency will be used for software operating systems. In this case, it has been used the STM32 NUCLEO-L053R system that is based on the STM32L053R8 ARM Cortex ™ -M0+ MCU, which can be programmed to operate at a frequency of 32MHz.

In this case, it is desirable to measure the response time to an external event. A button connected to a pin port of the microcontroller will generate the external event. This port will generate an external interrupt that must be handled by the task with the higher priority from the system. The higher priority task will remain blocked on the event, and when it goes into run state in response to the event, it will pass a port pin to low. Fig. 1 presents this mode of operation. It can be noted that the task with the highest priority is waiting an event. The event is triggered by pressing the test button (the port at which it is connected will go from high to low). After a time that depends on the operating system and the scheduler operation, the task with the highest priority will pass a second test pins port to the low. Thus, using a two-probe oscilloscope connected to the two ports, the response time to the external event can be measured.

In order to implement these performance tests, we will use three software RTOSs: uC-OS/III, Keil RTX, and FreeRTOS. These are the most used RTOSs for small MCU systems according to the study presented in [5]. The chosen RTOSs systems do not allow the direct attachment of an interrupt to a task, the interrupt service routines being executed outside the kernel. In the present case, for each RTOS, a multi-task application has been developed in which, after the initialization part, the task with the higher priority enters in the waiting state for an event. In the external interrupt service routine for the port that is connected to the test button, the expected event is triggered and the scheduler is executed. The scheduler will select the task with the higher priority for execution. This task will pass to low the second test pin port as soon as it enter in execution state.

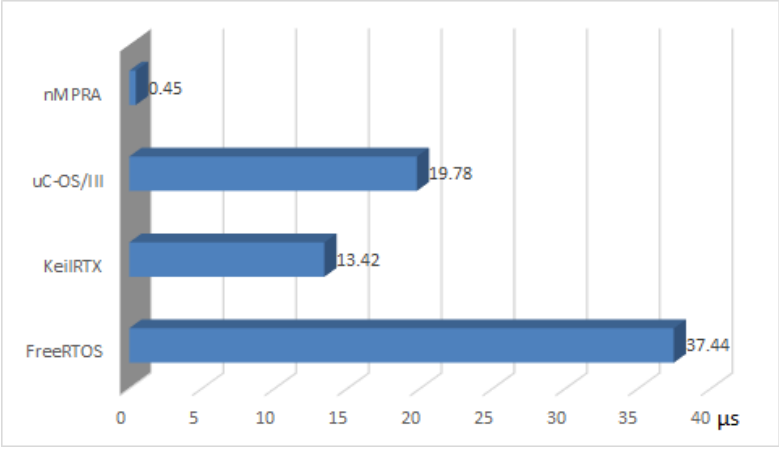

Fig. 2. Response Time to a External Event.

In order to implement this performance test on nMPRA, the external interrupt associated with the pin connected to the test button is attached to the task with the higher priority. After the initialization part, the task with the higher priority will wait the occurrence of an attached event (in this case the external interrupt). When the button is pressed, the external interrupt is triggered and the hardware scheduler will select the task with the higher priority for the execution. This task will pass to low the second test pin port as soon as it enters in execution state.

The results for the four tests (three software RTOSs and one hardware RTOS) are shown in Fig. 2. It can be noticed that the response time for the hardware RTOS is very small compared to software RTOS because the overhead generated by a software RTOS has been eliminated. Although hardware RTOSs have very good performances, they are not used in the industry because they are specialized RTOSs where programming is different and the users are very reluctant, preferring to use software RTOSs that have demonstrated their reliability in previous projects.

V. Conclusions

In this paper, it was presented a comparison between three software RTOSs and one hardware RTOS hardware in terms of response time to an external event. From this comparison, it was observed that the shortest response time is obtained by the RTOS implemented in hardware. This happens because the hardware implementation eliminates the overhead generated by software RTOS. From the tests it can be noticed that the weakest performances are obtained by the open source RTOS. However, FreeRTOS is the most widely used RTOS in embedded projects on small microcontrollers in 2017. Although much better performance are obtained with hardware implementations, the software RTOSs are still used because they are more customized and they were tested in very many projects. They can also be executed on a wide range of microcontrollers.

Acknowledgment

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI-UEFISCDI, Contract no. 220PED/2017, PN-III-P2-2.1-PED-2016-1473, within PNCDI III.

References

[1] Zelenova, S.A. and Zelenov, S.V., 2018. Schedulability Analysis for Strictly Periodic Tasks in RTOS. Programming and Computer Software, 44(3), pp.159-169. https://doi.org/10.1134/S0361768818030076

[2] Wang, K. C. "Models of Embedded Systems." Embedded and Real-Time Operating Systems. Springer, Cham, 2017. 95-111.

[3] I. Ungurean and N. C. Gaitan, "Performance analysis of tasks synchronization for real time operating systems," 2018 International Conference on Development and Application Systems (DAS), Suceava, 2018, pp. 63-66. doi: 10.1109/DAAS.2018.8396072

[4] G. C. Buttazzo. "Hard Real-Time Computing Systems:Predictable Scheduling Algorithms and Applications", Springer Science & Business Media, 2011.

[5] 2017 Embedded Markets Study, Integrating IoT and Advanced Technology Designs, Application Development & Processing Environments, https://m.eet.com/media/1246048/2017-embedded-market-study.pdf

[6] Sitara™ Processors, http://www.ti.com/processors/sitara-arm/overview.html

[7] i.MX 7 Series Applications Processors: Multicore Arm® Cortex®-A7, Cortex-M4, https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/i.mx-applications-processors/i.mx-7-processors:IMX7-SERIES

[8] V. G. Gaitan, N. C. Gaitan and I. Ungurean, "CPU Architecture Based on a Hardware Scheduler and Independent Pipeline Registers," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 9, pp. 1661-1674, Sept. 2015. doi: 10.1109/TVLSI.2014.2346542

[9] LABROSSE, Jean J. uC/OS-III, The Real-Time Kernel, or a High Performance, Scalable, ROMable, Preemptive, Multitasking Kernel for Microprocessors, Microcontrollers & DSPs. Micrium Press, 2009, ISBN:0982337531 9780982337530.

[10] The FreeRTOS™ project website. http://www.freertos.org. Accessed 29 Oct 2018.

[11] FERREIRA, Joao F., et al. Automated verification of the FreeRTOS scheduler in Hip/Sleek. International Journal on Software Tools for Technology Transfer, 2014, 16.4: 381-397

[12] GUAN, Fei, et al. Open source FreeRTOS as a case study in real-time operating system evolution. Journal of Systems and Software, 2016, 118: 19-35.

[13] VIRUTHAMBAL, K., et al. RTOS Based Dynamic Scheduler in Power Quality Applications. International Journal of Scientific Engineering and Technology, 2013, 2.6: 554-559.

[14] I. Zagan and V. G. Găitan, "Implementation of nMPRA CPU architecture based on preemptive hardware scheduler engine and different scheduling algorithms," in IET Computers & Digital Techniques, vol. 11, no. 6, pp. 221-230, 11 2017.

[15] I. Zagan, V. G. Gaitan, "Improving the Performances of the nMPRA Processor using a Custom Interrupt Management Scheduling Policy," Advances in Electrical and Computer Engineering, vol.16, no.4, pp.45-50, 2016, doi:10.4316/AECE.2016.04007

[16] N. C. Gaitan, "Enhanced Interrupt Response Time in the nMPRA based on Embedded Real Time Microcontrollers," Advances in Electrical and Computer Engineering, vol.17, no.3, pp.77-84, 2017, doi:10.4316/AECE.2017.03010

[17] A. Agne et al., "ReconOS: An Operating System Approach for Reconfigurable Computing," in IEEE Micro, vol. 34, no. 1, pp. 60-71, Jan.-Feb. 2014. doi: 10.1109/MM.2013.110

[18] F. Mauroner and M. Baunach, "mosartMCU: Multi-core operating-system-aware real-time microcontroller," 2018 7th Mediterranean Conference on Embedded Computing (MECO), Budva, 2018, pp. 1-4. doi: 10.1109/MECO.2018.8406007

[19] Jean J. Labrosse, "Hardware-Accelerated RTOS: µC/OS-III HW-RTOS and the R-IN32M3", https://www.micrium.com/hardware-accelerated-rtos-%C2%B5cos-iii-hw-rtos-and-the-r-in32m3/

[20] P. Kohout, B. Ganesh and B. Jacob, "Hardware support for real-time operating systems," First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis (IEEE Cat. No.03TH8721), Newport Beach, CA, USA, 2003, pp. 45-51., doi: 10.1109/CODESS.2003.127525