RT-SHADOWS: Real-Time System Hardware for Agnostic and Deterministic OSes Within Softcore

T. Gomes, S. Pinto, P. Garcia, A. Tavares
Centro Algoritmi - University of Minho
{tgomes, sandro.pinto, pgarcia, atavares}@dei.uminho.pt

***Abstract*—Multithreading is a key feature for dealing with the complexity of current generation embedded devices. Real-Time Operating Systems (RTOSes) provide a higher level of abstraction that alleviates design complexity and time-to-market pressure, inducing, however, undesired latencies and unpredictable execution times. Research works have been focusing on improving performance, determinism or agnosticism, but none simultaneously tackled the three metrics.**

This work in progress paper presents RT-SHADOWS, a co-designed architecture that promotes configurability, determinism and agnosticism from the outset. A multi-thread ARM softcore processor was exploited and extended by offloading the scheduling-related features into the hardware layer. The hardware multi-thread support is transparent for the standard RTOS applications and independent of the Operating System (OS). Promising preliminary results demonstrate huge improvements on performance and determinism, with only a small cost on hardware.

***Index Terms*—Real-Time OS, Determinism, Latency, Hardware offloading, FPGA, Multithreading, ARM.**

I. Introduction

Multi-thread architectures are still the best solution for applications with a certain degree of concurrency where multiple threads require high-level of synchronization and communication to perform a real-time task [1]. To deal with the complexity of current embedded systems, RTOSes are used as an abstraction layer on top of the hardware, providing several Application Programming Interfaces (APIs) to simplify and accelerate the development of multi-threaded applications. However, RTOSes induce undesired latencies and unpredictable execution times which consequently increase overhead and contribute to the system's performance degradation [2].

A myriad of approaches try to alleviate this overhead by offloading several OS features into the hardware layer [3], [4], [5]. The drawback of such approaches is that they only support their own OS or their solution only supports a specific OS, requiring a huge porting effort and/or limiting the re-utilization of legacy software. MAPUSOFT [6] is an innovative software-based solution that provides agnosticism between applications and the OS (i.e., endorsing a software-only approach).

From another perspective, other solutions [7], [8], [9], [10] focus on tackling the nondeterminism of real-time systems. As well stated in the literature, OSes (e.g., FreeRTOS and uCOSII) suffer from the rate-monotonic priority inversion [11] which leads to a dual-priority space between threads and interrupts, therefore breaking the deterministic execution of preemptive priority-based RTOSes. Some of the aforementioned works solve this problem by exploiting Commercial off-the-shelf (COTS) hardware only, while others implement specific hardware.

This work in progress paper goes beyond state-of-art presenting a parameterisable, deterministic and agnostic co-designed architecture, which is based on a multi-thread ARM softcore processor. RT-SHADOWS ensures agnosticism by offering hardware multi-thread support independently of the RTOS. Promising preliminary results demonstrate how this solution is able to alleviate the OS overhead through hardware acceleration, ensuring a deterministic and shorter execution time.

II. RT-SHADOWS

Our architecture, depicted in Fig. 1, is based on a Multi-Thread ARM processor. An in-house ARMv5-compliant softcore was modified to provide configurable hardware multi-thread support for real-time systems. RT-SHADOWS is a co-designed hardware/software architecture with parameterisable, deterministic and agnostic features. The number of hardware threads is configurable, depending on the application demands. Currently, our architecture supports 1, 4 or 8 hardware threads configuration.

The hardware multi-thread support is accomplished by replicating task-specific registers (e.g., register-file and status register) and connecting a hardware scheduler into the processor core. The hardware scheduler is implemented as a tightly-coupled ARM Co-processor, ensuring all communications between ARM core and the co-processor are performed in a short and deterministic time. A set of thread management APIs were developed in order to deterministically interface with scheduler, implementing the basic thread management functionalities (e.g., create/delete, resume/suspend, change priority, etc).

RT-SHADOWS presents 2-levels of OS HW/SW transparency. The first-level of transparency is guaranteed on API level, i.e., the applications use the standard RTOSes' APIs. The RTOSes' APIs are wrapped into the RT-SHADOWS APIs in order to interface with the hardware multi-thread support. This means no modifications are required on the OS kernel source. The second-level is given by the generic implementation of the hardware multi-thread support. Every parameter can be configured by software, using ARM coprocessor instructions, which

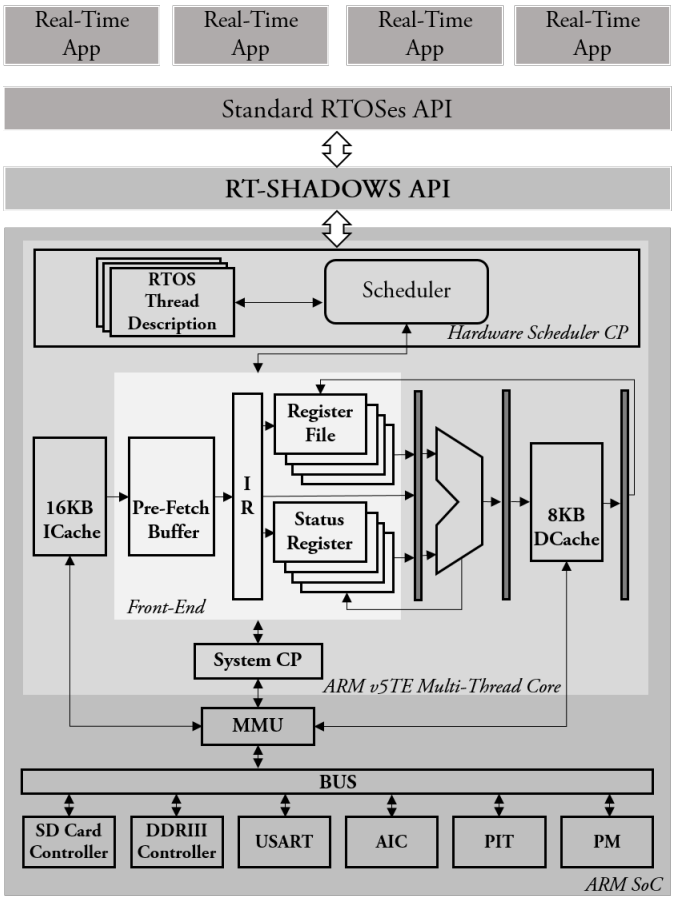

Fig. 1: RT-SHADOWS Top-level Architecture

makes the implementation non-intrusive and independent of the RTOS. In summary, the RTOS's APIs remain intact and only port-specific files of the OS are modified.

Additionally, our architecture is able to solve the rate-monotonic priority inversion using our task-aware interrupt controller presented in [10].

A. Hardware Multi-Thread Support

Depending on the application demands, RT-SHADOWS architecture can provide hardware support to a different number of threads. Each hardware thread owns its register-file bank as well as its own status register ensuring multiple contexts within the same core. The ARM architecture supports multiple executions modes (e.g., IRQ, Supervisor, User, etc) which allows different OSes to run their threads in different modes. In order to speed-up exception processing time, ARM architecture uses banked registers which are only visible depending on the current cpu mode. In order to support multiple OSes, our architecture allows the mode of the banked registers to be software configurable. For instance, FreeRTOS's threads run in system mode while uCOSII's threads run in supervisor mode, therefore the banked registers must be in same mode as the thread's mode used by the OS. The main advantage of this approach, i.e., banked registers per thread, is related with the thread's context-switch. Switching the currently running thread can be done deterministically and with no latency. Thus, several threads may execute concurrently with almost no overhead. The cost of this approach is minimal considering that all the functional units through the pipeline are shared by several threads.

In real-time systems, the system's response time to an interrupt may have significant impact in the system's performance. In order to ensure a short and predictable interrupt handling, a hardware thread is dedicated to the RTOS kernel. Hence, the OS interrupt latency overhead is decreased as no context of the currently running thread must be saved and the interrupt service routine can be executed in shorter time.

B. Hardware Scheduler

The hardware scheduler is in charge of which thread should be running at any instant of time. The main features of the implemented hardware scheduler are: (i) configurable scheduling algorithm depending on the RTOS used; (ii) ability to provide the next thread to execute in one clock cycle; (iii) quick and deterministic communication link between the scheduler and the processor core, since it is implemented as a tightly-coupled coprocessor; and (iv) thread's information is stored in a small and compact array of Thread Control Blocks (TCBs). Currently, the hardware scheduler supports the priority-based scheduling algorithm where the order of the priorities (e.g., ascendant or descendent) can be configured by software. This enables different OSes to configure if low or high priority numbers denote low or high priority threads. This algorithm outputs at any time what is the highest priority thread ready to run. In order to be compatible with different OSes (e.g., FreeRTOS), the scheduler is able to apply a round-robin scheme over the threads which share the highest priority level.

A hardware thread is represented by a TCB. The TCB provides relevant thread's information such as its priority, current state, handler and stack pointer. The hardware scheduler

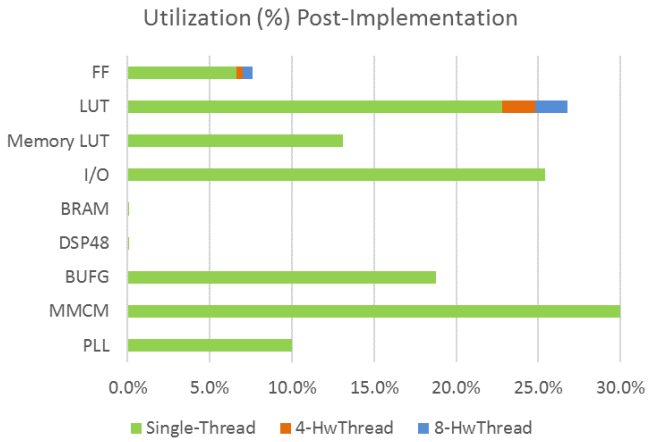

Fig. 2: RT-SHADOWS hardware cost for a different number of threads with hardware multi-thread support

Feature	API	Dispatch	FreeRTOS		μc/OS-II		RT-SHADOWS		
			$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	*ov. (%)*
Create Thread	`xTaskCreate()`	w	30828	33	-	-	2321	5	***-92.5%***
		w/o	27569	30	-	-	2322	5	***-91.6%***
	`OSTaskCreate()`	w	-	-	38282	62	2045	5	***-94.7%***
		w/o	-	-	36572	52	2044	4	***-94.4%***
Delete Thread	`vTaskDelete()`	w	71892	64172	-	-	1445	5	***-98.0%***
		w/o	3852	5	-	-	1445	5	***-62.5%***
	`OSTaskDel()`	w	-	-	10004	80	1228	5	***-87.7%***
		w/o	-	-	7736	12	1228	5	***-84.1%***
Suspend Thread	`vTaskSuspend()`	w	71167	63918	-	-	1445	5	***-98.0%***
		w/o	3301	6	-	-	1445	5	***-56.2%***
	`OSTaskSuspen()`	w	-	-	6868	77	1228	5	***-82.1%***
		w/o	-	-	2411	7	1228	5	***-49.0%***
Resume Thread	`vTaskResume()`	w	6943	10	-	-	1145	5	***-79.2%***
		w/o	4196	7	-	-	1145	5	***-65.6%***
	`OSTaskResume()`	w	-	-	6563	36	1228	5	***-81.3%***
		w/o	-	-	4731	11	1228	4	***-74.0%***
Set Thread Priority	`vTaskPrioritySet()`	w	71281	63659	-	-	1337	5	***-98.1%***
		w/o	6238	1675	-	-	1337	5	***-78.6%***
	`OSTaskChangePrio()`	w	-	-	9433	190	1143	4	***-87.9%***
		w/o	-	-	7438	14	1143	4	***-84.6%***

TABLE I: Comparison between the performance and jitter results in clock cycles for each architecture

provides a full interface with the hardware threads: (i) it allows threads to be created with any priority and any initial state, and to initialize the thread's stack pointer as well as the thread's arguments; (ii) get or set at any time, any of the thread's attributes using a *MCR* or *MRC* instructions for each thread in the system; (iii) it provides an one-instruction access to the data of the currently executing thread in order to perform quicker changes on the running thread data.

III. Preliminary Results

The implemented architecture was evaluated on a Kintex-7 FPGA Embedded Kit (XC7K325T), with a CPU speed of 33 MHz. In order to determine the hardware, performance, determinism and interrupt latency overhead we performed three different kind of experiments: (i) we compared the synthesis hardware results of ARM SoC with and without the hardware multi-thread extension; (ii) we compared FreeRTOS and μc/OS-II APIs running on the correspondent native RTOS using the single-thread ARM core (i.e., without hardware multi-thread support), against the same APIs running on the RT-SHADOWS architecture (i.e., with hardware multi-thread support); and (iii) we measured the interrupt latency on each architecture. MMU, caches and other dynamic architectural features were disabled. Performance Monitoring (PM) unit was used to assess the performance, determinism and interrupt latency overhead (in clock cycles).

Fig. 2 shows the hardware cost of our approach with hardware multi-thread support for four threads and eight threads plus the hardware scheduler. Our approach impacts only on the number of Flip-Flops (FF) and LUTs used. The overhead of four hardware threads is of 0.4% in FFs and 2% in LUTs while for eight threads is of 0.97% in FF and 3.9% in LUTs.

In order to assess the performance (execution time) and determinism (latency variance, known as jitter) of each API, several experiments were conducted. These results translate the mean value of a variation in different parameters: (i) the number of tasks; (ii) the priority of the tasks; and (iii) the gap between consecutive tasks' priority. For instance, each API may have different outcomes depending on different parameters such as the current threads' state or priorities. These parameters were varied in order to evaluate the API in two different scenarios, one where no thread is dispatched after the API is executed and another where the API triggers the dispatching of another thread. In the former scenario, the time measured is given by the time the API starts executing until the last instruction of the API is executed. In the latter, the time is given by the time that the API starts executing until the first instruction of the new thread to execute is issued from memory. Hence, on this time will be included the context-switch operation which is divided in three main parts: (i) save the context of the current thread; (ii) obtain the next thread to execute; and (iii) restore the context of the next thread.

Table I shows the achieved results. As depicted, RT-SHADOWS outperforms the native versions where no hardware multi-thread support is used. Also, the API's nondeterminism is reduced due to high-deterministic APIs provided by our architecture. Nevertheless, a small value of jitter is found on the RT-SHADOWS's APIs which is justified by the use of different clock frequencies between the processor core and the DDR3 memory controller (i.e., the time of a memory access is not fixed). Moreover, the execution time of the hardware-based APIs are independent of any of the varied parameters, such as the number of threads and their priorities.

RT-SHADOWS also improves the OS interrupt overhead which is defined as time between cpu interruption until the first instruction of the corresponding interrupt service routine

FreeRTOS		μc/OS-II		RT-SHADOWS		
$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	***ov. (%)***
1250	3	-	-	436	2	***-65.1%***
-	-	2250	6	1363	5	***-39.4%***

TABLE II: OS interrupt overhead in clock cycles for each architecture

(ISR) is issued from memory [2]. Usually, on a typical RTOS, this encompasses saving the context of the running thread, obtaining the source of the interrupt from the interrupt controller and branch to the ISR. Table II shows the benefits in terms of performance and predictability of using hardware multi-thread support over the native architecture.

IV. Research Roadmap

Work in the near future will proceed through the offloading of other kernel services to hardware. Currently, only a subset of the task management APIs is implemented in hardware. The idea is to guarantee support not only for all the task management APIs of each RTOS, but also for another services such as synchronization, communication, timing, etc. Simultaneously to the extension of kernel services support, we will extend also the list of supported RTOSes. As previously described, our strategy for agnosticism is based on a API mapping strategy, which maps the APIs of existent RTOSes to the APIs of RT-SHADOWS. Each time we want to support a new RTOS, we only need to study the list of APIs of the desired RTOS and develop a new mapping file. This way, RT-SHADOWS transparency levels will be validated by extending the list of supported RTOS such as EmbOS and uTKernel.

After, research will focus not only in the extension of the number of supported hardware threads but also in the scalability of the corresponding implementation approach. Our current solution only supports a maximum number of 8 hardware threads. Increasing this number may not be so trivial as the system may not scale well. Adopting a scalable and configurable strategy will abolish one significant limitation of our current solution. From a different perspective, the possibility of coexistence of software and hardware threads will be also object of study. For systems with limited hardware resources (FPGA or Silicon) or where an increase in the hardware resources can have a negative impact (monetary, area-constraint, power-consumption) this option will not limit our solution. Therefore, if more threads are created than the number of available hardware threads, software threads will be created and dispatched according to their priority.

Finally, research will continue towards the RT-SHADOWS refactoring to allow fine-grain configurations/customizations. The ultimate goal should be to develop a profiling tool that, through a hardware/software co-design methodology, explores the migration of software threads to hardware accordingly to the application demands and constraints.

V. Conclusion

This paper presented our work in progress towards the development of RT-SHADOWS, a co-designed hardware/software architecture that promotes configurability, determinism and agnosticism. We showed how RT-SHADOWS successfully tackled two important metrics of real-time systems such as performance and determinism by migrating the thread scheduler to hardware and providing hardware multi-thread support. Our solution presents very low overhead ratio in terms of area usage and performance improvements. This approach is able to surpass related work by providing an agnostic hardware acceleration solution which is independent of any specific RTOS. Future work will focus on extending the RT-SHADOWS architecture to support new features described in research roadmap section.

VI. Acknowledgements

Tiago Gomes is supported by FCT - Fundação para a Ciência e Tecnologia (Grant SFRH/BD/81682/2011). This work has been also supported by FCT - Fundação para a Ciência e Tecnologia within the Project Scope: PEst-UID/CEC/00319/2013.

References

[1] K. D. Kissell. (2007) Demystifying multithreading and multi-core. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1271568

[2] F. Sheikh and D. Driscoll, "White paper: Mentor graphics - measuring rtos performance: What? why? how?" Tech. Rep., 2011.

[3] D. Andrews, W. Peck, J. Agron, K. Preston, E. Komp, M. Finley, and R. Sass, "hthreads: a hardware/software co-designed multithreaded rtos kernel," in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, vol. 2, Sept 2005, pp. 8 pp.–338.

[4] J. J. Labrosse, "White paper: Hardware-accelerated rtos: c/os-iii hw-rtos and the r-in32m3," Tech. Rep., accessed: 2015-05-13. [Online]. Available: http://micrium.com/hardware-accelerated-rtos-%C2%B5cos-iii-hw-rtos-and-the-r-in32m3/

[5] M. Naotaka, I. Takuya, H. Shinya, T. Hiroaki, and S. Katsunobu, "Arm-based soc with loosely coupled type hardware rtos for industrial network systems," in *Proceedings of the 10th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, ser. OSPERT '14, 2014, pp. 9–16.

[6] Mapusoft, "White paper: Mapusoft os abstractor," Tech. Rep., accessed: 2015-05-13. [Online]. Available: http://www.mapusoft.com/wp-content/uploads/documents/osabstractor_whitepaper.pdf

[7] L. Leyva-del Foyo, P. Mejia-Alvarez, and D. de Niz, "Predictable interrupt management for real time kernels over conventional pc hardware," in *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, April 2006, pp. 14–23.

[8] W. Hofer, D. Lohmann, F. Scheler, and W. Schroder-Preikschat, "Sloth: Threads as interrupts," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, Dec 2009, pp. 204–213.

[9] S. Pinto, J. Pereira, D. Oliveira, F. Alves, E. Qaralleh, M. Ekpanyapong, J. Cabral, and A. Tavares, "Porting sloth system to freertos running on arm cortex-m3," in *Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on*, June 2014, pp. 1888–1893.

[10] T. Gomes, P. Garcia, F. Salgado, J. Monteiro, M. Ekpanyapong, and A. Tavares, "Task-aware interrupt controller: Priority space unification in real-time systems," *Embedded Systems Letters, IEEE*, vol. 7, no. 1, pp. 27–30, March 2015.

[11] F. Scheler, W. Hofer, B. Oechslein, R. Pfister, W. Schröder-Preikschat, and D. Lohmann, "Parallel, hardware-supported interrupt handling in an event-triggered real-time operating system," in *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ser. CASES '09. New York, NY, USA: ACM, 2009, pp. 167–174.