Bringing Hardware Multithreading to the Real-Time Domain

RT-SHADOWS: **R**eal-**T**ime **S**ystem **H**ardware for **A**gnostic and **D**eterministic **O**Ses **W**ithin **S**oftcore

Tiago Gomes, Paulo Garcia, Sandro Pinto, João Monteiro, Adriano Tavares

***Abstract*—The emergence of hardware multi-thread (HW-MT) architectures increased the performance of MT applications. However, traditional HW-MT architectures are not suitable to Real-Time Operating Systems as their performance-oriented scheduling algorithm may conflict with RTOS software scheduling.**

This letter presents RT-SHADOWS, a portable architecture which provides a unified hardware-software scheduling, bringing the benefits of HW-MT to the RTOS domain. We show that tightly-coupled real-time compliant hardware integration achieves throughput benefits, maintaining the RTOS scheduling policy intact while increasing the predictability of RTOSes. Our solution shows on average, speed-ups between 3 and 4 times over the native versions with very low area usage/performance overhead ratio, due to its minimal cost (2% of extra slices per hardware-supported thread). This work surpasses related work by providing a complete and agnostic hardware solution which is independent of any specific RTOS.

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

I. Introduction

Real-Time Operating Systems (RTOSes) aid designers in simplifying and expediting the development of multi-threaded applications by providing several Application Programming Interfaces (APIs) at the cost of performance overhead and reduced predictability [1]. Research [2][3][4] towards alleviating this overhead by migrating different functionalities (mainly RTOS schedulers) into hardware has been performed. Usually, a hardware scheduler is implemented in FPGA-fabric and loosely-coupled to the processor using a commercially available bus [2][3][5]. Although these projects focus on hardware acceleration, there is no concern about portability to other RTOSes, which limits legacy software re-use. These projects fail to get the attention of the industrial community [6], either because these solutions do not cover a wide spectrum of RTOSes or are too complex and RTOS-dependent, which requires in-depth knowledge of the RTOS architecture from software developers. MAPUSOFT [7] is a software-based solution to provide agnosticism between applications and the OSes, while SEOS [6] is a hardware solution focusing on adaptability for other RTOSes; however, acceleration is only based on a hardware scheduler. To the best of the authors' knowledge, no research has applied hardware multi-threading (HW-MT) to existent RTOSes, allowing legacy applications to benefit from shorter and deterministic context-switch and interrupt handling.

HW-MT has been established as an architectural feature to maximize throughput in a processor. The idea is to maintain the processor running at maximum throughput by executing several hardware-supported threads. HW-MT can increase throughput up to 25% [8] by hiding idle times such as memory latencies or branch penalties [9]. A HW-MT architecture typically encompasses per thread replication of the architectural units on a processor (e.g., register-file, status register, program counter, etc.) and a hardware scheduler in charge of managing threads execution flow. This approach has been proved to offer better processor utilization (throughput) when compared to single-threaded cores [10]. However, traditional HW-MT architectures are applied to Server/Desktop applications [8] and are not suitable for RTOSes commonly used in embedded systems. HW-MT schedulers use their own thread scheduling algorithm (e.g., BMT, IMT or SMT) to explore chip utilization. This hardware level scheduling diverges from RTOS software scheduling, resulting in a hierarchical scheduling policy which breaks the expected RTOS execution flow and established static analysis methods; e.g., MIPS32 SMT-based HW-MT support [4] requires equivalent SMT software scheduler. Fig. 1 depicts the two levels of hierarchical scheduling found in traditional HW-MT architectures.

We present a HW-MT solution which provides throughput benefits, maintaining the RTOS scheduling policy intact and increasing the predictability of RTOSes. To accomplish this, we have developed RT-SHADOWS, a highly-portable multi-threaded softcore processor which offers hardware multi-thread support portable across RTOSes, as well as higher performance predictable behaviour. Our approach unifies RTOS scheduling and hardware-based thread scheduling, implementing a holistic multi-thread scheduling (Fig. 1) which leverages the advantages of HW-MT to the real time world. By integrating the RTOS scheduler with the processor scheduler in a tightly-coupled fashion, the performance advantages of HW-MT are achieved without sacrificing (often improving) deterministic execution nor invalidating established static analysis methods.

The main contribution of this letter is the implementation of a hardware multi-threading architecture to cope with real-

This work was supported by the FCT within the Project Scope:PEst-UID/CEC/00319/2013. The work of T. Gomes was supported by the FCT, Fundação para a Ciência e Tecnologia (Grant SFRH/BD/81682/2011).

T. Gomes, P. Garcia, S. Pinto, J. Monteiro and A. Tavares are with Centro Algoritmi, University of Minho, Portugal (e-mail: {tiago.a.gomes, paulo.garcia, sandro.pinto, joao.monteiro, adriano.tavares}@algoritmi.uminho.pt).

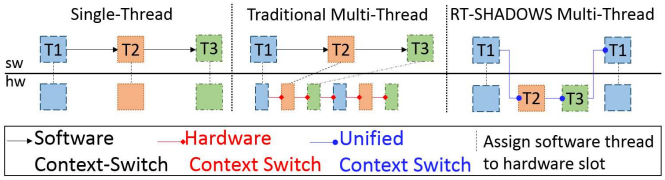


Fig. 1: Scheduling on single-thread, traditional multi-thread and RT-SHADOWS architectures

time applications. The main features of our system architecture are: (1) Unified HW/SW Multi-Threading Support; (2) Deterministic Tightly-Coupled Processor Scheduler; (3) APIs Agnosticism; (4) High Portability; (5) Short and Deterministic Interrupt Handling [11].

II. Problem Description

Traditional HW-MT architectures apply their own scheduling algorithm (e.g., BMT, IMT or SMT) to manage threads execution flow in order to achieve maximum performance. Furthermore, RTOSes also apply their own scheduling algorithm which results in a hierarchical scheduling conflict as shown in Fig. 2, which depicts an example where the IMT scheduling algorithm would change the expected behaviour of an RTOS execution flow. Running 2 threads in round-robin scheme (i.e., both have the same priority), it is expected that T1 starts executing first and takes the mutex (Fig. 2 (a)). After the time-slice assigned to T1 is finished, T2 starts executing. As the mutex is already taken by T1, T2 will fail to take it. In Fig. 2 (b) we are applying a round-robin scheme but using an IMT policy at hardware level (i.e., at each clock cycle a different thread is dispatched). With IMT scheduling, T2 may take the mutex before T1 (Fig. 2 (b)) which would modify the expected execution flow. RT-SHADOWS takes advantages of HW-MT architectures using current RTOS solutions by unifying the two scheduling strategies.

III. RT-SHADOWS Architecture Description

An in-house ARMv5-compliant softcore was extended with new micro-architectural features to provide parameterisable, deterministic and agnostic HW-MT support. Fig. 3 depicts RT-SHADOWS architecture. The number of hardware-supported

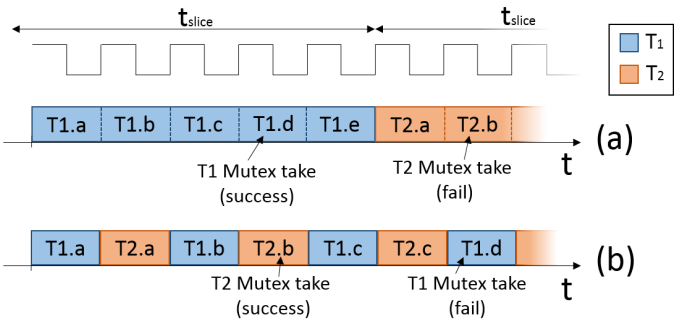


Fig. 2: Scheduling conflict; (a) RTOS scheduling policy (b) IMT scheduling policy

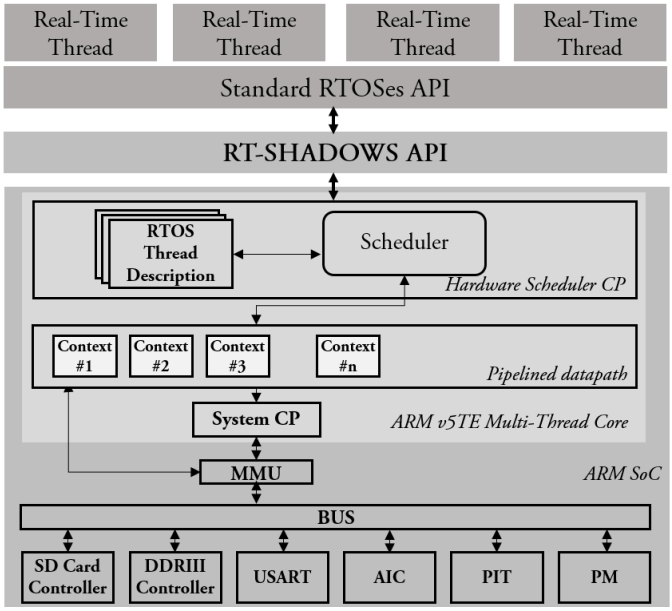


Fig. 3: RT-SHADOWS top-level architecture

threads is configurable up to 128 threads, depending on the application demands. For area-constrained platforms, RT-SHADOWS allows the use of regular software threads if the number of application's threads is greater than the number of hardware-supported ones, ensuring scalability. Also, the use of delay timers and synchronization methods (e.g., mutexes and semaphores) is optional. RT-SHADOWS offers a set of thread management and synchronization APIs commonly used in RTOSes. These are application-transparent, i.e., applications use the standard RTOSes APIs, wrapped into RT-SHADOWS APIs, in order to interface with the HW-MT support. In summary, only OS port-specific files are modified and no modifications are required on the OS kernel source, ensuring all the standard APIs remain intact.

A. Hardware Multi-Threading Support

Interrupt processing and RTOS services are the most important aspects that define RTOS performance. Each hardware-supported thread has its own registers, allowing short and deterministic switching of multiple contexts within the core. The ARM architecture supports multiple execution modes (e.g., IRQ, Supervisor, User, etc) which different RTOSes can leverage. In order to speed-up exception handling time, ARM uses banked registers for each mode. To support multiple RTOSes, our architecture allows banked registers mode to be software configurable; e.g., FreeRTOS's threads run in system mode while uCOSII's threads run in supervisor mode, with banked registers' mode matching RTOS's specification. In order to ensure a short and predictable interrupt response time, a hardware-supported thread is dedicated to the kernel. Hence, the RTOS interrupt latency overhead is decreased as no context of the currently running thread must be saved. Additionally, our architecture is able to solve the rate-monotonic priority inversion found in many RTOSes using our task-aware interrupt controller presented in [11].

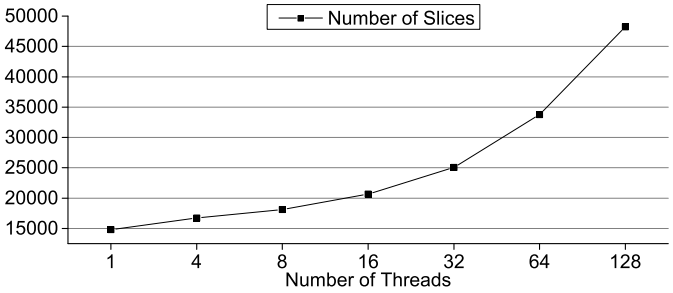


Fig. 4: RT-SHADOWS hardware cost for a different number of hardware-supported threads over the single-thread version

B. Unified Scheduler

The unified processor scheduler is implemented as a tightly-coupled ARM co-processor. In contrast to loosely-coupled schedulers, usually connected to a bus where several peripheral devices may compete for access, communication between the core and the co-processor is performed using MCR/MRC instructions, ensuring a short and deterministic communication link. This processor scheduler offers a high level of software configurability: (1) configurable scheduling algorithm (optional round-robin scheme); (2) configurable order of thread priorities (e.g., ascending or descending), enabling RTOSes to configure if low or high priority numbers denote low or high priority threads; and (3) ARM mode of the banked registers. A compact Thread Control Block (TCB) is used to store the thread's information such as its priority, current state, handler and stack pointer.

C. Delay Timers and Synchronization Mechanism

Delay Timers are used by RTOSes to block the currently running thread by a specific amount of time. This usually implies a timer for each thread, unblocking it as soon as its timer expires. In order to optimize the hardware cost of supporting hardware-supported threads delay a different approach was implemented. There is a single system timer in

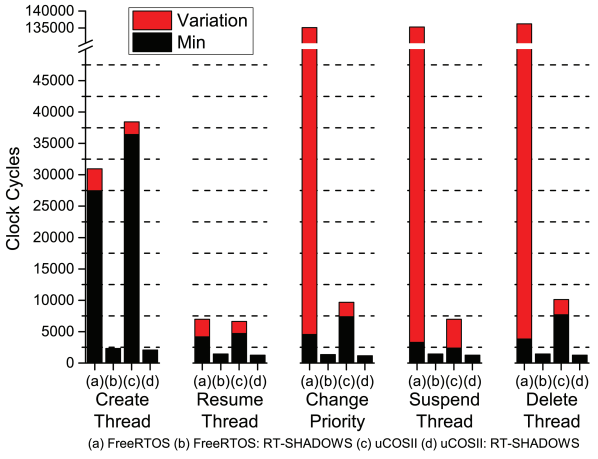
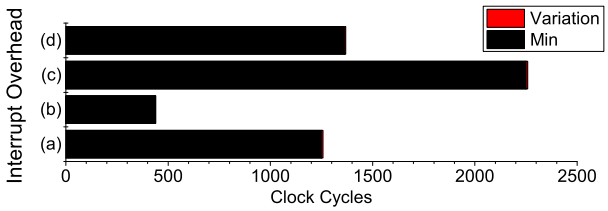


Fig. 5: Comparison between the performance and jitter results in clock cycles for each architecture



(a) FreeRTOS (b) FreeRTOS: RT-SHADOWS (c) uCOSII (d) uCOSII: RT-SHADOWS

Fig. 6: RTOS interrupt overhead in clock cycles for each architecture

charge of counting OS ticks. Each thread has a time-stamp register containing the number of ticks for which it must wait until the system timer overpasses the time-stamp. Hence, delays can be implemented as extra registers plus comparators (one per thread) instead of using an array of timers.

Semaphores and mutexes allow threads to wait for a specific event or guarantee an exclusive access to a resource. Usually, a thread can wait for an event or resource for a period of time in which the thread will be blocked until the event or resource is available. Taking advantage of available hardware, Delay Timers are used to manage the blocking time, supporting mutexes and semaphores in hardware with almost no hardware cost.

IV. Results and Evaluation

To evaluate our solution in terms of performance and determinism two different experiments were conducted. Experiment IV-A assesses APIs and shows the benefits of the hardware multi-thread extensions over the native RTOS execution. Experiment IV-B runs Thread-Metric Benchmark Suite in order to evaluate how RT-SHADOWS alleviates RTOS overhead. Both experiments were validated on a Kintex-7 FPGA Embedded Kit (XC7K325T). Fig. 4 shows the hardware cost of our approach. Our architecture requires around 2 percent more chip space to support an extra hardware-supported thread.

A. API Evaluation

Several measurements were conducted in order to assess minimum execution time (Min) and latency variance (Variation), of the most common APIs. Fig. 5 presents the results on each architecture: (a) FreeRTOS native version; (b) FreeRTOS with multi-threading extensions (c) uCOSII native version; and (d) uCOSII with multi-threading extensions. A particular API may have different outcomes depending on multiple parameters such as the current threads' states or priorities. Hence, these results translate the values obtained from the variations of these different parameters: (1) number of threads; (2) thread's priority; (3) consecutive threads' priority gap and (4) whether the API triggers a context-switch. These variations encompassed several corner cases. As depicted, configurations with multi-threading extensions outperform the native versions. Both performance and determinism are significantly increased. RT-SHADOWS reduces the overhead from 56% up to 98%. Also, RT-SHADOWS shows that the variation of the

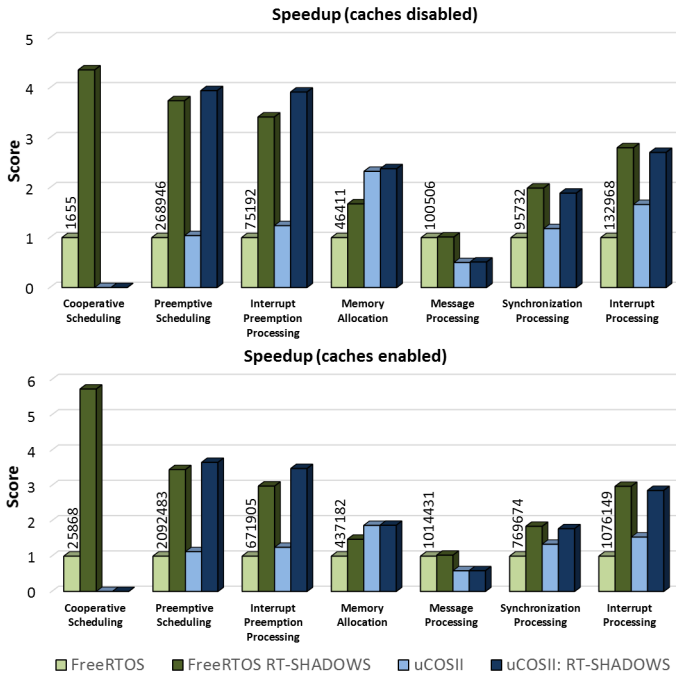


Fig. 7: Speed-up results running Thread-Metric Benchmark with caches enabled

aforementioned parameters does not interfere with hardware-based APIs' execution time, i.e., the hardware latency is constant no matter how many threads are hardware-supported or created and their priorities. It is also noticeable that uCOS natively has better determinism than FreeRTOS. The dispatching of threads with a huge gap between their priorities is the main factor behind FreeRTOS indeterminism. Fig. 6 depicts OS interrupt overhead of each configuration. This overhead is measured as the time between CPU interruption until the first instruction of the corresponding interrupt service routine is issued from memory [1]. RT-SHADOWS is able to attend an interrupt request in shorter time than the native versions.

B. Thread-Metric Evaluation

The Thread-Metric Benchmark Suite is a benchmark that measures RTOS real-time performance developed by Express Logic Inc. [12]. The suite consists of 7 benchmarks, each evaluating interrupt processing and RTOS services. Each benchmark's score represents the RTOS impact on the running application, i.e., the greater the score the smaller the impact. These experiments were conducted with a clock frequency of 33 Mhz, 10 ms periodic timer, with and without caches enabled and the IAR compiler with no optimization. We executed the benchmark on the FreeRTOS and uCOSII native versions and compared them over RT-SHADOWS architecture. Fig. 7 shows how our architecture outperforms the native versions. Specially on benchmarks where context-switch and interrupt handling are exacerbated, the RT-SHADOWS is able to show speed-ups between 3 and 4 times. On the memory and message specific benchmarks, RT-SHADOWS can still present speed-ups due to the gain obtained on the periodic context-switch. There is no results for uCOSII running the cooperative scheduling since this algorithm is not supported. Our system outperforms the native version with and without caches enabled.

V. Conclusion

This letter described RT-SHADOWS, a co-designed hardware/software architecture which implements a holistic HW-MT solution, promoting configurability, determinism, performance and portability. We showed how such a holistic HW-MT approach can be applied to RTOSes solutions without the need for refactoring legacy-software. Our solution outperforms native solutions in terms of performance and determinism. RT-SHADOWS presents very low area usage/performance overhead ratio, due to its minimal cost (2% extra slices per hardware-supported thread). This work surpasses related work by providing a complete and agnostic hardware solution which is also RTOS-agnostic. Future work will encompass the development of new features and refactoring RT-SHADOWS to allow fine-grained configurations/customizations. The ultimate goal will be to develop a profiling tool, that through a hardware/software co-design methodology, explores the migration of software threads to hardware according to the application and hardware platform demands and constraints.

References

[1] F. Sheikh and D. Driscoll, "White paper: Mentor Graphics - Measuring RTOS Performance: What? Why? How? ," Tech. Rep., 2011.

[2] 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.

[3] J. J. Labrosse, "White paper: Hardware-accelerated rtos: c/os-iii hw-rtos and the r-in32m3," Tech. Rep., accessed: 2015-08-03. [Online]. Available: http://micrium.com/hardware-accelerated-rtos-%C2%B5cos-iii-hw-rtos-and-the-r-in32m3/

[4] A. Oliveira, L. Almeida, and A. de Brito Ferrari, "The arpa-mt embedded smt processor and its rtos hardware accelerator," *Industrial Electronics, IEEE Transactions on*, vol. 58, no. 3, pp. 890–904, March 2011.

[5] I. Bahri, M. Benkhelifa, and E. Monmasson, "Hw-sw real-time operating system for ac drive applications," in *Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), 2012 International Symposium on*, June 2012, pp. 194–199.

[6] S. E. Ong, S. C. Lee, N. Ali, and F. Hussin, "SEOS: Hardware Implementation of Real-Time Operating System for Adaptability," in *Computing and Networking (CANDAR), 2013 First International Symposium on*, Dec 2013, pp. 612–616.

[7] Mapusoft, "White paper: Mapusoft os abstractor," Tech. Rep., accessed: 2015-08-03. [Online]. Available: http://www.mapusoft.com/wp-content/uploads/documents/osabstractor_whitepaper.pdf

[8] D. Koufaty and D. Marr, "Hyperthreading technology in the netburst microarchitecture," *Micro, IEEE*, vol. 23, no. 2, pp. 56–65, March 2003.

[9] A. Sodan, J. Machina, A. Deshmeh, K. Macnaughton, and B. Esbaugh, "Parallelism via Multithreaded and Multicore CPUs," *Computer*, vol. 43, no. 3, pp. 24–32, March 2010.

[10] R. Dimond, O. Mencer, and W. Luk, "Custard - a customisable threaded fpga soft processor and tools," in *Field Programmable Logic and Applications, 2005. International Conference on*, Aug 2005, pp. 1–6.

[11] 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.

[12] I. Express Logic. Thread-Metric Benchmark Suite. [Online]. Available: http://rtos.com/downloads/articles_and_white_papers-1/