An implementation of the flexible spin-lock model in ERIKA Enterprise on a multi-core platform

Sara Afshar[1], Maikel P.W. Verwielen[2], Paolo Gai[3], Moris Behnam[1], Reinder J. Bril[1,2]
[1]Mälardalen University, Västerås, Sweden
[2]Technische Universiteit Eindhoven, Eindhoven, Netherlands
[3] Evidence Srl, Pisa, Italy
Email: {sara.afshar, moris.behnam}@mdh.se, pj@evidence.eu.com, r.j.bril@tue.nl

***Abstract*—Recently, the flexible spin-lock model (FSLM) has been introduced, unifying spin-based and suspension-based resource sharing protocols for real-time multiprocessor platforms by explicitly identifying the spin-lock priority as a parameter. Earlier work focused on the definition of a protocol for FSLM and its corresponding analysis under the assumption that various types of implementation overhead could be ignored.**

In this paper, we briefly describe an implementation of the FSLM for a selected range of spin-lock priorities in the ERIKA Enterprise RTOS as instantiated on an Altera Nios II platform using 4 soft-core processors. Moreover, we present measurement results for the protocol specific overhead of FSLM as well as the natively provided multiprocessor stack resource policy (MSRP). Given these results, we are now in a position to judge when it is advantageous to use either MSRP or FMLP for our system set-up for given global resource access times of tasks.

I. Introduction

In traditional lock-based resource-sharing protocols for real-time multiprocessor platforms, a task that is blocked on a global resource either performs a non-preemptive busy wait, i.e. *spins*, or releases the processor, i.e. *suspends*. The flexible spin-lock model (FSLM) [1] unifies these two traditional approaches. By viewing suspension on a core as spinning on a priority lower than any other priority on a core, the *spin-lock priority* can be treated as a parameter. Spin-based protocols, such as the multiprocessor stack resource policy (MSRP) [15], can be viewed to use the highest priority (HP) as spin-lock priority, and suspension-based protocols, such as the multiprocessor priority ceiling protocol (MPCP) [21], to use the lowest priority (LP). By being able to use an arbitrary priority for spinning rather than the two extremes, the FSLM is expected to improve schedulability.

The resource sharing rules for the FSLM have been defined in [1], assuming partitioned, fixed-priority preemptive scheduling, FIFO-based global-resource queues and both non-nested as well as non-preemptive global resource access, similar to MSRP and MPCP. These rules are complemented with schedulability analysis for specific spin-lock priorities, such as the HP, the LP, and the highest resource ceiling of global resources on a core, also called the ceiling priority (CP). Initial simulation results based on the developed theory [2] confirm the expectations with respect to improved schedulability. In particular, CP turned out to significantly improve schedulability compared to HP. The schedulability analysis developed in [1] does not take implementation overhead into account, however. The simulation results may therefore be biased.

In this paper, we present an implementation of the FSLM for a selected range of spin-lock priorities [26], in particular the range from CP until HP in Erika Enterprise [13], as instantiated on an Altera DE0 board from Terasic [24] using 4 soft-core processors. Erika Enterprise is a free of charge, open-source real-time operating system (RTOS) implementation, which was originally developed for small-scale OSEK/VDX [18] compatible embedded systems for the automotive market. Erika Enterprise has been ported to the Altera Nios II environment [11], supporting multiple soft-cores. We have ported Erika Enterprise to the Altera DE0 board. Based on our implementation, we compare the overhead of HP, as originally implemented in Erika Enterprise, and CP.

The remainder of this paper is organized as follows. In Section II, we briefly present related work. Next, in Section III, we present our real-time scheduling model and system. Sections IV and V describe the design and implementation of FSLM in Erika Enterprise. Section VI describes the experiments performed and briefly presents the measurement results. We conclude the paper in Section VII.

II. Related Work

In [17], two-phase waiting algorithms [19] are investigated through analysis and experiments, with the aim to minimize the cost of synchronization in large-scale multiprocessors. A two-phase waiting protocol is a combination of a spin-based and a suspension-based protocol. A task first spins for a statically determined amount of time, and subsequently blocks if further waiting is required. The MIT Alewife distributed-memory multiprocessor [3], which supports a shared-memory programming model, has been used for experimental measurements. The paper suggests to use knowledge about wait-time characteristics and the cost of blocking (i.e. the context-switching overhead) to set the maximum spinning time.

In [14], an experimental evaluation of MPCP and MSRP is presented based on a Janus dual-processor architecture. For

This work is supported by the Swedish Foundation for Strategic Research via the research program PRESS, the Swedish Knowledge Foundation and ARTEMIS Joint Undertaking project EMC2 (grant agreement 621429).

random period generation of tasks the results show MSRP to be better than MPCP, although the results are not conclusive. For a more application-specific architecture representing a typical automotive application, MSRP has shown to clearly perform better. Moreover, they observed that MSRP is significantly simpler to implement, has lower overhead, and can achieve RAM memory optimization. Similar to this work, interrupt-based inter-processor mechanisms have been used for communication among tasks on different processors and atomic test-and-set mechanisms have been used for shared memory.

A first implementation of the PCP [22], SRP [5], M-PCP (an extension of PCP for multiprocessors), D-PCP [20] (a variant of MPCP used for distributed systems) and FMLP [6] synchronization protocols has been discussed in [7]. FMLP uses suspension-based mechanism for access to long resources and spin-based mechanism for access to short resources. A LITMUS^{RT} [10] platform has been selected for implementation which is a real-time extension of Linux operating system. In [8] a schedulability comparison has been made among MPCP, D-PCP and FMLP considering runtime overheads on LITMUS^{RT}. The experiments showed that the spin-based FMLP variant always had the best performance. The results confirmed their earlier results in [9] regarding preferability of spin-based approach to suspension-based approach under EDF scheduling.

This work complements earlier work by evaluating preemptable spinning, as supported by FSLM, through experimental measurements.

III. Scheduling model and system

In this section we describe our real-time scheduling model, the Altera DE0 board and development environment, and the Erika Enterprise and accompanying tool-suite RT-Druid.

A. Real-time scheduling model

We assume a set $\mathcal{P}$ of m identical cores $P_0, \ldots, P_{m-1}$, a set $\mathcal{T}$ of n sporadic tasks $\tau_0, \ldots, \tau_{n-1}$, and a set $\mathcal{R}$ of resources other than cores used by tasks. Tasks are statically allocated to cores, assigned unique priorities on each core, and scheduled using fixed-priority pre-emptive scheduling. Tasks do not suspend themselves.

Resources are categorized as *private*, *local*, or *global* based on task usage and task allocation. Private resources are used by a single task. Local resources are used by multiple tasks, and all those tasks are allocated to the same core. Global resources are also used by multiple tasks, but that set of tasks is allocated to at least two different cores. In this paper, the focus will be on global resources. Example 1 illustrates a configuration with a global resource.

Example 1. *Consider a set $\mathcal{P}$ of two cores P_0 and P_1, a set $\mathcal{T}$ of 4 tasks $\tau_0, \ldots \tau_3$, and a singleton set $\mathcal{R}$ of one resource R. As also indicated in Table I, R is used by tasks τ_0, τ_1, and τ_3. Task τ_3 is allocated to core P_0 and tasks τ_0, τ_1, and τ_2 to P_1. As a result, R becomes a global resource.*

	resource usage	allocation
τ_3	R	P_0
τ_2		P_1
τ_1	R	P_1
τ_0	R	P_1

TABLE I: Resource usage and allocation of tasks of $\mathcal{T}$.

Moreover, we assume that the priority π_i of task τ_i is higher than the priority π_j of task τ_j if and only if $i > j$. An activation of a task is also called a *job*. We assume constrained deadlines, i.e. deadlines of tasks equal or smaller than their periods.

For FSLM, we assume FIFO-based resource queues and both non-nested as well as non-preemptive global resource access, similar to MSRP and MPCP. When a task is blocked on a global resource, it will perform a busy-wait on a core-specific spin-lock priority. That spin-lock priority is determined statically, and may range from the lowest to the highest priority on the core. In this paper, we assume the spin-lock priority is taken from the range $[CP, HP]$, where HP represents the highest priority on the core and CP represents the highest resource ceiling of the global resources used on that core. Example 2 illustrates FSLM for the configuration described in Example 1.

Example 2. *For the configuration of Example 1, the highest resource ceiling of the global resources used on core P_0 is equal to the priority π_3 of task τ_3. Similarly, the highest resource ceiling on P_1 is equal to the priority π_1 of task τ_1.*

For the same arrival pattern of tasks, Figure 1 illustrates FSLM for two different spin-lock priority assignments; one conform MSRP (Figure 1(a)), i.e. using HP, and one using CP on each core (Figure 1(b)). Because task τ_3 accesses the global resource R in the time interval $[1, 8)$, task τ_0 starts spinning upon its resource request to R at time 3 for both cases. Spinning is performed non-preemptively for MSRP (Figure 1(a)), i.e. using HP, and preemptively when using CP (Figure 1(b)). Using HP, τ_2 is blocked from its arrival at time 6 until task τ_0 releases the global resource R at time 12. Conversely, using CP, task τ_2 can preempt τ_0 at time 6 during spinning. Task τ_2 can execute till time 8, when τ_3 releases R, τ_0 is granted R, and τ_0 subsequently accesses R till time 12. When task τ_0 releases R at time 12, τ_2 is resumed.

This example shows that tasks with a priority higher than the spin-lock priority, e.g. τ_2 on P_1, experience less blocking due to global resource arbitration under FSLM using CP than using HP as spin-lock priority.

By restricting the range to $[CP, HP]$, the protocol maintains two attractive properties of MSRP. Firstly, at any moment in time, at most one job on a core can have a pending request for or access to a global resource. As a result, a job that is spinning on a global resource will have to wait for at most $m-1$ jobs on remote cores. Consequently, the length of any global resource queue, even the sum of the length of all global resource queues, is at most $m-1$. Secondly, any job of a task

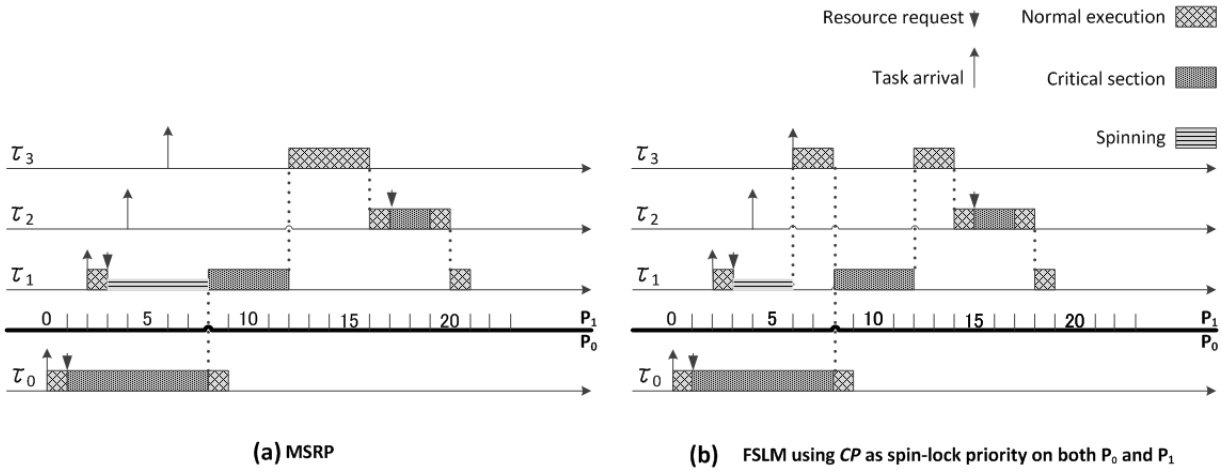


Fig. 1: Timelines for the same arrival pattern of tasks of $\mathcal{T}$, illustrating the FSLM for an assignment of (a) HP (conform MSRP) and (b) CP to the spin-lock priorities of each core.

on a core can be blocked at most once due to global resource requests of lower priority tasks on that core. Another attractive property of MSRP, i.e. the ability to use a single stack for all tasks on a core, is no longer maintained, however, as illustrated by the preemption of task τ_2 by τ_0 in Figure 1(b) at time 8.

B. Altera DE0 board and development environment

The Altera DE0 development and education board is equipped with the Altera Cyclone III 3C16 field-programmable gate array (FPGA) device, which offers 15,408 logical elements (LEs). The FPGA device can be configured by means of Altera's Quartus II Web Edition Software and Altera's Nios II Embedded design suite.

Using Altera's tools, we created a hardware design consisting of 4 Nios II processors (cores) and added internal (RAM) and external (SDRAM) memory, a mutex (to support mutual exclusive access), inter-core interrupt communication between every pair of cores, and performance counters (to enable high-resolution measurements) to the design, amongst others.

The resulting multi-core platform can communicate through a shared memory interconnect [23] and via inter-core interrupts. The connections for the inter-core interrupts are illustrated in Figure 2.

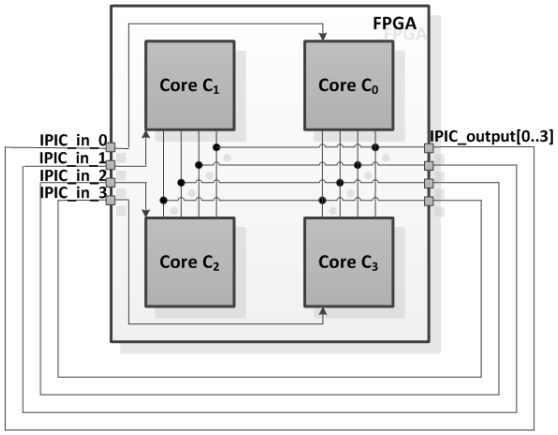


Fig. 2: The connections for inter-core interrupts

C. Erika Enterprise and RT-Druid

As mentioned above, Erika Enterprise was originally developed for OSEK/VDX-based systems. We used the multi-core extension [11] of the so-called "multistack" configuration of the "BCC2" conformance class of the OO (OSEK OS) kernel [13] of Erika Enterprise. RT-Druid [12] is a tool-suite developed for Erika Enterprise providing a system modeler, code-generator plugins for the open-source Eclipse framework [25] and schedulability analysis plug-ins. The RT-Druid Modeler is used for configuring both the application as well as Erika Enterprise, using the OSEK Implementation Language (OIL).

Both Erika Enterprise as well as RT-Druid have been extended for multiprocessor systems. To that end, the standard OIL has been extended to facilitate allocation of tasks to cores, amongst others.

Below, we first briefly describe the structure of a multi-core Erika Enterprise and its mapping on the Altera DE0 board. Next, we describe some key characteristics of Erika Enterprise.

1) Structure and mapping: The multi-core Erika Enterprise is a kernel-layer on top of Altera's hardware abstraction layer (HAL); see Figure 3. For our instantiation, the kernel-layer consists of approximately 20 standard files and 3 files generated per core by RT-Druid. The input for RT-Druid is a `CONFIG.OIL` file. The actual application is described by a set op files in the API-layer next to the `CONFIG.OIL` file.

2) Characteristics of Erika Enterprise: Erika Enterprise supports MSRP. To that end, it maintains a data structure in shared memory. When a task is busy waiting for a global resource, it spins on, i.e. polls, data in shared memory using the G-T algorithm [16].

Erika Enterprise also support event-based communication between cores using inter-core interrupts. As an example, a remote activation of a task can be accomplished through a

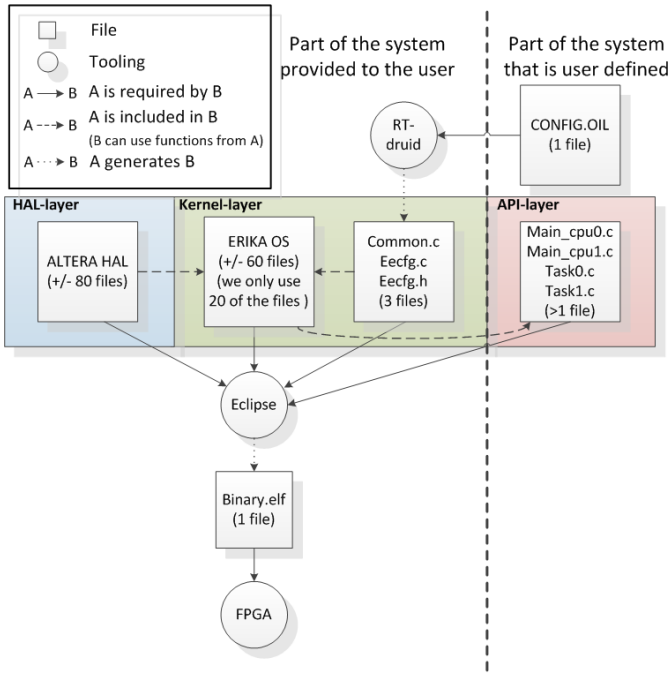


Fig. 3: Erika Enterprise, Altera's HAL, application and mapping

so-called *remote notification* (RN). The sending core builds an RN message in shared memory and subsequently raises an interrupt at the receiving core. The interrupt handler of the receiving core inspects and processes the RN message asynchronously. Message buffers for RN require mutual exclusion.

IV. Design of the Flexible Spin-Lock Model

For the implementation of FSLM, five main aspects need to be considered:

1) Static selection of the spin-lock priority per core;
2) Dynamic change of the system ceiling to the spin-lock priority when a task blocks on a global resource;
3) Notification of a (blocked) task on a remote core that a global resource became available, when applicable;
4) Preemption of a higher priority task upon global resource access, when applicable.
5) Resumption of the preempted, higher priority task, when applicable.

We consider each of these aspects in more detail below.

A. Selection of spin-lock priorities

On each core where one or more tasks use a global resource, a spin-lock priority must be selected. In this paper, we only consider spin-lock priorities from the range $[CP, HP]$, and we therefore need to derive CP and HP from the system configuration. RT-Druid therefore needs to be extended with means (*i*) to determine CP and HP from the `CONFIG.OIL` file, (*ii*) to interact with a user to allow selection of spin-lock priorities per core, and (*iii*) to configure the kernel-layer of the RTOS with the spin-lock priorities.

B. Blocking on a global resource

When a task blocks on a global resource, the system ceiling on that core is raised to the spin-lock priority and the task starts spinning. This is illustrated in Figure 1(b) at time 3. Raising the system ceiling upon blocking is similar to the regular behavior upon a local resource access, which is based on the stack resource policy (SRP) [5].

C. Notification of a (blocked) task

Unlike MSRP, a task may be preempted during spinning, as illustrated in Figure 1(b) at time 6. As a result, the blocked task may not be aware that the global resource is released and becomes available. The design therefore has to be adapted from a polling-approach by the spinning task to a notification-approach by the releasing task. The existing remote notification mechanism present in Erika Enterprise can be used for FSLM as well. The first blocked job in the FIFO-queue of a global resource R, if any, will therefore be notified upon release of R by means of an interrupt, as illustrated in Figure 1(b) at time 8.

D. Preemption of the preempting task

Whenever a task τ_s spinning on a global resource is preempted by a task τ_p with a higher priority than the spin-lock priority, the preempting task τ_p must be preempted when the τ_s is granted the resource, as illustrated in Figure 1(b) at time 8. Although this gives rise to a preemption that disallows tasks to use a single stack, this behavior is supported by Erika Enterprise.

E. Resumption of the preempting task

When the task releases a global resource, it is checked whether or not the task preempted a task with a higher priority than the spin-lock was executing at the moment the resource was granted. In the former case, the preempted task is resumed, as illustrated in Figure 1(b) at time 12. In any case, the system ceiling is adapted, removing the traces of the request and access to the global resource.

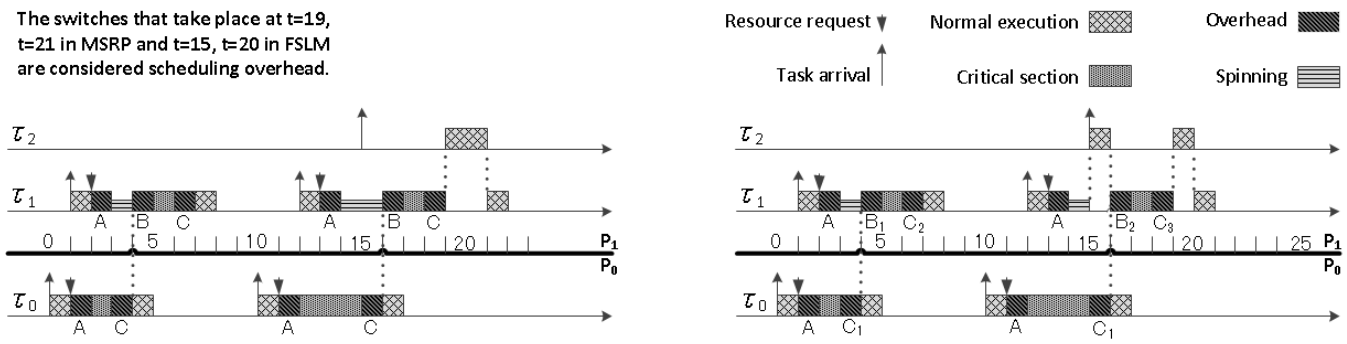
V. Implementation of the Flexible Spin-Lock Model

In this section, we first briefly present the implementation of MSRP in Erika Enterprise. Next, we will present the necessary changes for the generalization of MSRP to FSLM for the restricted range $[CP, HP]$ of spin-lock priorities.

A. Existing Implementation of MSRP in Erika Enterprise

In MSRP, a task requiring access to a global resource busy waits non-preemptively until (*i*) it is the first in line (first-in-first-out) waiting for the resource and (*ii*) the resource is free. Because spinning in MSRP is non-preemptive, at most one task per core can spin on a global resource. It is therefore also possible to associate a FIFO-queue of cores with every global resource.

To implement MSRP, Erika Enterprise essentially maintains a distributed polling-bit queue for each global resource (G-T algorithm [16]), i.e. a (non-empty) FIFO queue of polling



(a) MSRP

(b) FSLM using CP as spin-lock priority on both P0 and P1

Fig. 4: Synchronization protocol specific overhead under MSRP and FSLM

bits used by cores that want to access that global resource. The polling bits are stored as global data and the addresses of queue elements are stored in local data. To enable access to the queue, the tail of the queue is also stored as global data, containing the address of the global polling bit that needs to be inspected by the next core that requires access to the global resource. Access to the tail of the queue requires mutual exclusion. Releasing a global resource requires toggling the related polling bit only.

B. From MSRP to FSLM in Erika Enterprise

In the MSRP implementation of Erika Enterprise, a task that is accessing a global resource is unaware of the fact that another task on a remote core may (or may not) have requested the same resource, i.e. it is unaware of successors in the polling-bit queue. A task that is waiting for a global resource to become available is aware of the task in front of it in the polling-bit queue.

To facilitate notification for FSLM, the releasing task must know which task/core needs to be granted the global resource. The "knowledge" of the order in the queue must therefore become bi-directional. Rather than using a global polling-bit, we therefore used a global field representing both locked and unlocked as well as the task to be notified upon release of the global resource, if any. This global field requires mutual exclusive access. To reduce contention on shared data, we implemented an additional local bit for spinning.

Upon a global resource release, a task first checks whether or not tasks are blocked on that resource. The resource is subsequently released. In case tasks are blocked, the first in line, i.e. the successor of the releasing task, is notified through a dedicated remote notification (RN).

When an RN is received, it is first checked if the task blocked on the global resource is still spinning or has been preempted by a (or actually one or more) task(s) with a higher priority than the spin-lock priority. In the former case, the system ceiling is raised to reflect non-preemptive execution and the local bit is toggled, enabling the spinning task to access the global resource. In the latter case, the currently executing task must be preempted in addition, and the blocked task allowed to continue.

When a task has released a global resource, it has to check whether or not its access to the global resource induced the preemption of a task. In the former case, the preempted task (or any other task with a yet higher priority), is allowed to be resumed (or started).

For the original implementation of MSRP, a single low-level spin-lock is used for both the access to the shared data structures for global resources as well the RN message buffers. For the implementation of FSLM we added a low-level spin-lock, allowing parallel access to these two types of shared data.

VI. Experimental Evaluations

We performed a comparative evaluation of the implementation of MSRP and FSLM by measuring the overhead of both protocols. Overhead occurs at three specific moments during the protocols (see also Figure 4), i.e.

A) upon global resource request,
B) when the access to a global resource is granted, and
C) when a global resource is released.

A global resource request requires mutual exclusive access to the shared data structures for global resources for both MSRP and FSLM. Because all m cores may simultaneously perform a request to that data, a core may have to wait on $m-1$ other cores before it is granted access. Under FSLM, the release of a global resource also requires mutual exclusive access to that shared data. Under MSRP, releasing a resource only requires toggling a bit.

Under FSLM, releasing a resource may require the submission of a RN, and therefor mutual exclusive access to the RN message buffers. Similarly, access to the RN message buffers is required when a global resource is granted to a task while it is waiting. Upon release, all cores, except the waiting core(s), may require access to the RN message buffers, i.e. at most $m-1$. Upon access, all cores may require access to the RN message buffers.

Measurements using performance counter cores [4] were performed for two scenarios, one without preemption during spinning (from time 0 until time 8 in Figures 4(a) and 4(b)) and one with preemption during spinning (from time 10 onwards

in Figures 4(a) and 4(b)). We have repeated the experiments 100 times. The measurement results are given in Table II.

	MSRP		FSLM	
Request	A	$160 + 79^{(a)}$	A	$189 + 146^{(a)}$
Access	B	18	B_1	$127 + 538^{(b)}$
			B_2	$140 + 538^{(b)} + 700^{(c)}$
Release	C	255	C_1	$322 + 94^{(a)} + 560^{(b)}$
			C_2	$255 + 94^{(a)}$
			C_3	$255 + 94^{(a)} + 700^{(c)}$

TABLE II: Measurement results in cycles. The superscripts (a) and (b) are added to the values of the worst-case critical section length for access to shared data structures for global resources and to the RN message buffers, respectively. The superscript (c) denotes context-switching overhead.

From these results, we conclude that the overhead for a global resource request is roughly the same for MSRP and FSLM. Compared to MSRP, the overhead for a global resource access and a global resource release is significantly higher for FSLM, however. As indicated in the table, this is mainly due to additional logic, reading and writing RN message buffers, and the additional context switches.

As described in Section III, FSLM reduces the blocking time due to spinning for tasks with a higher priority than the spin-lock priority. Based on our measurements, we are now in a position to determine when to use MSRP or FSLM for those tasks. The sum of the additional overheads for MSRP is 512 cycles, whereas this sum for FSLM is $2,762$ cycles, which corresponds to $10\mu s$ and $55\mu s$ on our $50MHz$ platform. The break-even is therefore when the sum of the remote global resource access times of tasks on our multi-core platform exceeds $2,250$ cycles, or $45\mu s$.

VII. Conclusions

In this paper, we presented an implementation of FSLM in Erika Enterprise on an Altera Nios II platform and a comparative evaluation of the protocol specific overheads of the native MSRP supported by Erika Enterprise and FSLM. Our experiments reveal that the overhead of global resource access and global resource release is significantly increased for FSLM. Based on these results, we are now in a position to judge when it is advantageous to use either MSRP or FSLM for such a system set-up for given resource access times.

References

[1] S. Afshar, M. Behnam, R. Bril, and T. Nolte. Flexible spin-lock model for resource sharing in multiprocessor real-time systems. In 9^{th} *IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 41–51, June 2014.

[2] S. Afshar, M. Behnam, R. J. Bril, and T. Nolte. On per processor spin-lock priority for partitioned multiprocessor real-time systems. Technical Report 3874, Available: http://www.es.mdh.se/publications/3874-, Mälardalen University Sweden, 2014.

[3] A. Agarwal, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, K. Kurihara, B.-H. Lim, G. Maa, and D. Nussbaum. The MIT Alewife Machine: A large-scale distributed-memory multiprocessor. In M. Dubois and S. Thakkar, editors, *Scalable Shared Memory Multiprocessors*, pages 239–261. Springer US, Boston, MA, 1992.

[4] Altera. Quartus II 8.1 handbook, Volume 5: Embedded peripherals, Chapter 29: Performance counter core. Technical Report https://www.altera.com/en_US/pdfs/literature/hb/qts/archives/quartusii_handbook_8.1.pdf, November 2008.

[5] T. Baker. Stack-based scheduling of real-time processes. *Journal of Real-Time Systems*, 3(1):67–99, March 1991.

[6] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A flexible real-time locking protocol for multiprocessors. In 13^{th} *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 47–56, Aug. 2007.

[7] B. Brandenburg and J. Anderson. An implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP real-time synchronization protocols in LITMUS^{RT}. In 14^{th} *IEEE Intl. Conf. on Embedded and Real-Time Computing Sys. and Applications (RTCSA)*, pages 185–194, Aug. 2008.

[8] B. B. Brandenburg and J. H. Anderson. A comparison of the M-PCP, D-PCP, and FMLP on LITMUS^{RT}. In 12^{th} *International Conference On Principles of Distributed Systems (OPODIS)*, pages 105–124, Dec. 2008.

[9] B. B. Brandenburg, J. M. Calandrino, A. Block, H. Leontyev, and J. H. Anderson. Real-time synchronization on multiprocessors: To block or not to block, to suspend or spin? In 14^{th} *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 342–353, April 2008.

[10] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson. LITMUS^{RT}: A testbed for empirically comparing real-time multiprocessor schedulers. In 27^{th} *IEEE International Real-Time Systems Symposium (RTSS)*, pages 111–126, Dec 2006.

[11] Evidence S.r.l. Erika Enterprise Manual for the Altera Nios II target - the multicore RTOS on FPGAs (version 1.2.3). Technical Report http://download.tuxfamily.org/erika/webdownload/manuals_pdf/arch_nios2_1_2_3.pdf, Evidence S.r.l., Pisa, Italy, December 2012.

[12] Evidence S.r.l. RT-Druid reference manual - A tool for the design of embedded real-time systems (version: 1.5.0). Technical Report http://download.tuxfamily.org/erika/webdownload/manuals_pdf/rtdruid_refman_1_5.0.pdf, Evidence S.r.l., Pisa, Italy, December 2012.

[13] P. Gai, E. Bini, G. Lipari, M. D. Natale, and L. Abeni. Architecture for a portable open source real time kernel environment. In 2^{nd} *Real-Time Linux Workshop and Hand's on Real-Time Linux Tutorial*, 2000.

[14] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform. In 9^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 189–198, May 2003.

[15] P. Gai, G. Lipari, and M. Di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In 22^{nd} *IEEE Real-Time Systems Symposium (RTSS)*, pages 73–83, Dec. 2001.

[16] G. Graunke and S. Thakkar. Synchronization algorithms for shared-memory multiprocessors. *IEEE Computer*, 23(6):60–69, June 1990.

[17] B.-H. Lim and A. Agarwal. Waiting algorithms for synchronization in large-scale multiprocessors. *ACM Transactions on Computer Systems*, 11(3):253–294, August 1993.

[18] OSEK group. OSEK/VDX operating system. Technical report, February 2005. [Online], Available: http://portal.osek-vdx.org/files/pdf/specs/os223.pdf.

[19] J. Ousterhout. Scheduling techniques for concurrent systems. In 3^{rd} *IEEE International Conference on Distributed Computing Systems*, Sep. 2014.

[20] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[21] R. Rajkumar, L. Sha, and J. Lehoczky. Real-time synchronization protocols for multiprocessors. In 19^{th} *Real-Time Systems Symposium (RTSS)*, pages 259–269, Dec. 1988.

[22] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, Sep. 1990.

[23] A. S. Tanenbaum. *Structured Computer Organization (5^{th} Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.

[24] Terasic16. Altera DE0 board. http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=364, Last visited: May 2016.

[25] The Eclipse Foundation. eclipse. Technical Report http://www.eclipse.org/.

[26] M. P. Verwielen. Performance of resource access protocols. Eindhoven University of Technology (TU/e), MSc-thesis, June 2016.