A dual shared stack for FSLM in Erika Enterprise

A dual shared stack for FSLM in Erika Enterprise

S.Muthu N. Balasubramanian[1], Sara Afshar[2], Paolo Gai[3], Moris Behnam[2], Reinder J. Bril[1,2]
[1]Technische Universiteit Eindhoven, Eindhoven, Netherlands
[2]Mälardalen University, Västerås, Sweden
[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 multi-core platforms. Unlike the multiprocessor stack resource policy (MSRP), FSLM doesn't allow tasks on a core to share a single stack, however.**

In this paper, we present a hypothesis claiming that for a restricted range of spin-lock priorities, FSLM requires only two stacks. We briefly describe our implementation of a dual stack for FSLM in the Erika Enterprise RTOS as instantiated on an Altera Nios II platform using 4 soft-core processors.

I. Introduction

The flexible spin-lock protocol (FSLM) introduced in [1] unifies spin-based and suspension-based resource sharing protocols for real-time multi-core platforms by explicitly identifying the spin-lock priority as a parameter per core. FSLM may significantly improve schedulability of systems compared to traditional protocols [2], such as the spin-based multiprocessor resource policy (MSRP) [7] and the suspension-based multiprocessor priority ceiling protocol (MPCP) [10]. Unfortunately, this improved schedulability comes at a cost. In particular, the attractive property of MSRP for resource-constrained embedded real-time multiprocessor systems that tasks can share a single stack is no longer maintained.

In this paper, we claim that for a restricted range of spin-lock priorities, FSLM requires only two shared stacks. In addition, we briefly describe an extension of an existing implementation of FSLM [3] with a dual stack in Erika Enterprise [6], as instantiated on an Altera DE0 board from Terasic [11] 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 [8] compatible embedded systems for the automotive market, and natively supports MSRP.

II. A hypothesis for a dual shared stack in FSLM

The resource sharing rules for FSLM have been defined in [1], assuming partitioned, fixed-priority preemptive scheduling (P-FPPS), first-in-first-out (FIFO)-based global resource queues and both non-nested as well as non-pre-emptive global resource access, similar to MSRP and MPCP. These rules are complemented with schedulability analysis for arbitrary but fixed spin-lock priorities per core in [2].

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).

Unfortunately, a single shared stack on a core is not feasible for FSLM whenever the spin-lock priority on that core is not the highest priority. As an example, consider a task τ_i that is preempted during spinning on a global resource R by a task τ_j. When τ_i is granted access to R, it is immediately allowed to execute its critical section, resulting in a preemption of τ_j, which implies interleaved executions of τ_i and τ_j.

Assuming $\pi_{P_k}^{\max}$ and $\pi_{P_k}^{\text{spin}}$ to denote the highest priority and the spin-lock-priority on a core P_k, respectively, and $\pi_{P_k}^{\text{G}}$ to denote the highest resource ceiling of global resources on P_k, we postulate the following hypothesis for FSLM.

Hypothesis 1. *For a spin-lock priority $\pi_{P_k}^{\text{spin}}$ in the range $[\pi_{P_k}^{\text{G}}, \pi_{P_k}^{\max})$, FSLM allows the set of tasks on core P_k to require only two stacks, i.e. tasks with a priority at most $\pi_{P_k}^{\text{spin}}$ share one stack and the other tasks share another stack.*

III. Implementation of dual shared stack in Erika

Erika Enterprise utilizes RT-Druid [5], a dedicated tool-suite providing a system modeler, code-generator plugins for the open-source Eclipse framework [11] and schedulability analysis plugins. The code-generator plugins of RT-Druid are used for configuring both the application and the Erika Enterprise kernel based on a user defined configuration file written in the OSEK Implementation Language (OIL) [9]. The OIL language, which is text based, is used to specify the RTOS objects and properties such as tasks, resources and alarms that are static and specified during compilation.

A. Task stack properties in Erika

We used the multi-core extension [4] with the "multistack" configuration of the "BCC2" conformance class of the OO (OSEK OS) kernel [6] of Erika Enterprise. Various objects such as CPU object, OS object and Task object in the OIL file allow for the specification of attributes that configure the system and application as required. The OIL file allows for the specification of the stack memory at the OS and task level.

1) `CPU_DATA` *section:* The OS object is usually contained within the CPU object and is used for configuring the OS attributes. In a typical multi-core system, it contains a `CPU_DATA` section for every individual core in the system. The `MULTI_STACK` attribute in that section is used to define whether or not the system supports multiple stacks for the application tasks. The attribute can be set to `TRUE` or `FALSE`.

We used a Nios-II [4] based hardware design, which allows Erika to schedule the main() function as a background task

on a DUMMY_STACK. Hence, for a configuration without MULTI_STACK support, the background and the application tasks use the DUMMY_STACK as a shared common stack.

2) Task object: The task object is also contained within the CPU object and is used for configuring the task properties. The STACK attribute in the task object is used to specify if the task uses a dedicated private stack or a shared stack. The attribute can be set to PRIVATE or SHARED. Setting a private stack also allows for the specification of an individual stack size using the SYS_SIZE attribute.

3) Configuration file generation: RT-Druid generates configuration files for each core in the system based on the attributes specified in the OIL file. The eecfg.c and eecfg.h files contain the stack definition for the core.

```
extern int __alt_stack_pointer;
#define __RTD_SYS_STACK_ADDRESS \
    (int)(&__alt_stack_pointer)

    EE_UREG EE_hal_thread_tos[EE_MAX_TASK+1] = {
        0,    /* dummy*/
        0,    /* task0*/
        0,    /* task1*/
        1,    /* task2*/
        2     /* task3*/
    };

    struct EE_TOS EE_nios2_system_tos[3] = {
        {(EE_ADDR)(__RTD_SYS_STACK_ADDRESS             )},
        {(EE_ADDR)(__RTD_SYS_STACK_ADDRESS- 3584  -36 )},
        {(EE_ADDR)(__RTD_SYS_STACK_ADDRESS- 3840  -36 )}
    };

    EE_UREG EE_hal_active_tos = 0; /* dummy */
```

Listing 1: Example auto-generated multi-stack configuration

Listing 1 shows the auto-generated stack configuration in eecfg.c for a core with 4 tasks. This configuration has been created with tasks 0 and 1 sharing a stack while tasks 2 and 3 have private stacks. Since Erika by default allows for the specification of only one shared stack per core, as seen in the listing, all other tasks that are not a part of the default shared stack must currently be allocated their own private stacks. Hence, in the given example, there are 3 stacks in total.

B. Adding support for dual shared stack

Since the auto-generation of dual shared stacks in Erika is currently unsupported, the generated configuration files have to be modified manually to incorporate dual shared stacks for FSLM. In order to leverage the existing single shared stack per core support in Erika in the process, the following method is employed to create dual shared stack:

- Since the tasks with priorities at most $\pi_{P_k}^{\text{spin}}$ can share a stack, their STACK attribute are set to SHARED. This ensures that these tasks are automatically assigned to the default common shared stack (DUMMY_STACK).
- Tasks with priorities higher than $\pi_{P_k}^{\text{spin}}$ are defined with STACK attribute set to PRIVATE, resulting in the auto-generation of individual stacks for this group of tasks.
- After the completion of auto code generation by RT-Druid, the stack properties of the group of private tasks in the eecfg.c file are modified such that a second shared stack is created; see Listing 2 for our example.

```
extern int __alt_stack_pointer;
#define __RTD_SYS_STACK_ADDRESS \
    (int)(&__alt_stack_pointer)

    EE_UREG EE_hal_thread_tos[EE_MAX_TASK+1] = {
        0,    /* dummy*/
        0,    /* task0*/
        0,    /* task1*/
        1,    /* task2*/
        1     /* task3*/
    };

    struct EE_TOS EE_nios2_system_tos[2] = {
        {(EE_ADDR)(__RTD_SYS_STACK_ADDRESS             )},
        {(EE_ADDR)(__RTD_SYS_STACK_ADDRESS- 3584  -36 )}
    };

    EE_UREG EE_hal_active_tos = 0; /* dummy */
```

Listing 2: Example modified dual stack configuration

Note that in Listing 2 the stack related data structures EE_hal_thread_tos and EE_nios2_system_tos are modified such that only one of the multiple private stacks remain while all the tasks in the second group utilize that stack, making it a second shared stack.

IV. Conclusion and Future Work

In this paper, we presented a hypothesis stating that for a restricted range of spin-lock priorities, FSLM requires only two stacks. Next, we briefly presented our implementation of a dual stack for FSLM in the Erika Enterprise RTOS.

As future work, we plan to (*i*) prove our hypothesis for FSLM, (*ii*) improve on our existing implementation of FSLM in Erika Enterprise, and (*iii*) develop schedulability analysis incorporating implementation overheads of FSLM.

References

[1] S. Afshar, M. Behnam, R. Bril, and T. Nolte. Flexible spin-lock model for resource sharing in multiprocessor real-time systems. In *Proc. 9th IEEE SIES*, pages 41–51, June 2014.

[2] S. Afshar, M. Behnam, R. J. Bril, and T. Nolte. An optimal spin-lock priority assignment algorithm for real-time multi-core systems. In *Proc. 23rd IEEE RTCSA (to appear)*, 2017.

[3] S. Afshar, M. P. W. Verwielen, P. Gai, M. Behnam, and R. J. Bril. An implementation of the flexible spin-lock model in Erika Enterprise on a multi-core platform. In *Proc. 12th OSPERT*, pages 55–60, July 2016.

[4] 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.

[5] 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.

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

[7] 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 *Proc. 22nd IEEE RTSS*, pages 73–83, Dec. 2001.

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

[9] OSEK/VDX Consortium. OIL: OSEK Implementation Language. Technical report, July 2004.

[10] R. Rajkumar, L. Sha, and J. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proc. 19th IEEE RTSS*, pages 259–269, Dec. 1988.

[11] The Eclipse Foundation. eclipse. Technical Report http://www.eclipse.org/, April 2017.