ORK+/XtratuM: An Open Partitioning Platform for Ada*

Ángel Esquinas[1], Juan Zamorano[1], Juan A. de la Puente[1], Miguel Masmano[2], Ismael Ripoll[2], and Alfons Crespo[2]

[1] Universidad Politécnica de Madrid (UPM), E-28040 Madrid, Spain
aesquina@datsi.fi.upm.es,jzamora@fi.upm.es,jpuente@dit.upm.es
[2] Universidad Politécnica de Valencia (UPV), E-46022 Valencia, Spain
mmasmano@ai2.upv.es, {iripoll,alfons}@disca.upv.es

Abstract. The ARINC 653 standard defines an Integrated Modular Avionics (IMA) architecture for building complex systems consisting of several real-time applications with different levels of criticality running in the same hardware platform. Applications execute in *partitions* that are isolated from each other in the temporal and spatial (i.e. storage) domains. The standard defines an architecture and an applications program interface (API) for an operating system or *application executive* (APEX) supporting these concepts.

This paper describes an implementation of a partitioning platform for Ada based on a similar approach. The platform is built with two components: the XtratuM hypervisor, which supports multiple virtual machines on a single computer, and the Open Ravenscar Kernel (ORK+), a small, reliable real-time kernel supporting the Ada Ravenscar tasking profile. This combination provides an open-source platform that enables high-integrity Ada applications to share the same computer board with other, possibly less critical, applications.

Keywords: Ada 2005, real-time systems, high-integrity systems, integrated modular avionics, partitioned systems, ORK, Ravenscar profile.

1 Introduction

Current avionic systems are often composed of several applications that may have different levels of criticality. In such kind of systems, applications must be isolated from each other, so that their integrity is not compromised by failures occurring in other applications. A common approach to isolation has been based on using a *federated* architecture, i.e. on allocating different applications to different computers. However, the growth in the number of applications and the increasing processing power of embedded computers have opened the way to *integrated* architectures, in which several applications are executed on a single computer platform. In this case, alternate mechanisms must be put in place in order to isolate applications

* This work has been partly funded by the Spanish Ministry of Science, project TIN2008-06766-C03 (RT-MODEL).

A. Romanovsky and T. Vardanega (Eds.): Ada-Europe 2011, LNCS 6652, pp. 160–173, 2011.

from each other. A common approach is to provide a number of *logical partitions* on each computer platform, in such a way that each partition is allocated a share of processor time, memory space, and other resources. Partitions are thus isolated from each other both in the temporal and spatial domains. Temporal isolation implies that a partition does not use more processor time than allocated, and spatial isolation means that software running in a partition does not read or write into memory space allocated to other partitions.

Partitioning has been successfully implemented in the aeronautics domain by the so-called Integrated Modular Avionics (IMA) concept [15]. The IMA architecture requires a specialized operating system layer that provides temporal and spatial isolation between partitions. The ARINC 653 standard [3] defines an architecture and an applications program interface (API) for such an operating system or *application executive* (APEX), in ARINC terms.

Temporal isolation is achieved by using a two-level scheduling scheme. A global *partition scheduler* allocates processor time to partitions according to a static cyclic schedule, where partitions run in turn for the duration of a fixed slice of time (see figure 1). The global scheduler is a variant of a static cyclic executive, while the local schedulers are priority-based. Spatial isolation between partitions is provided by implementing a separate address space for each partition, in a similar way as process address spaces are protected from each other in conventional operating systems.

It should be noted that the Ada 2005 advanced real-time mechanisms allow other approaches to time and space partitioning, which may be simpler to implement and provide more flexibility in scheduling real-time tasks [14,18]. However, there is a strong demand for IMA systems in industry, and support for such

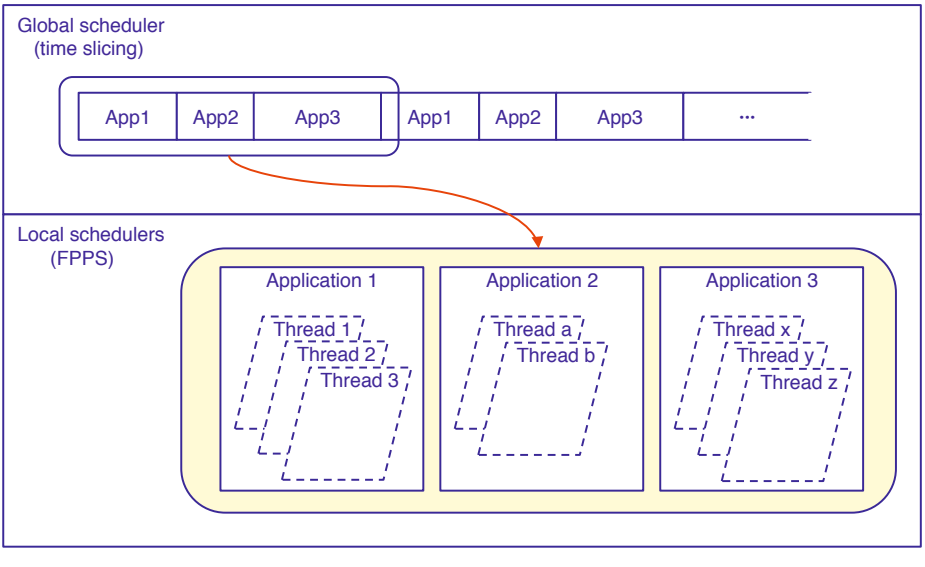

Fig. 1. Hierarchical scheduling architecture

architectures must be made available to Ada software developers as well. Indeed, there are multiple industrial ARINC 653 implementations available from a variety of vendors, and the standard has been successfully used in a number of commercial and military avionics systems. However, there is currently no open source platform available which can be used to build partitioned systems. This paper shows how an open-source platform following the IMA approach can be built by combining the XtratuM hypervisor [12] with the Ada 2005 version of the Open Ravenscar Kernel (ORK+) [19]. The hardware platform is LEON2 [8], a radiation-hardened implementation of the SPARC V8 architecture [16] that is commonly used in European space systems.

The rest of the paper is organized as follows: Section 2 introduces the architecture and the main features of the XtratuM hypervisor. The architecture of ORK+ is described in Section 3. Section 4 describes the approach and some issues that arose during the porting of ORK+ to run on top of XtratuM. A preliminary evaluation of the performance of the platform compared to ORK+ running directly on a bare board is included in Section 5. Related work is summarized in Section 6. Finally, some conclusions about the resulting partitioning platform are drawn and plans for the near future are exposed in Section 7.

2 Overview of XtratuM

XtratuM [12] is an open-source hypervisor that has been designed to meet safety critical real-time requirements in embedded systems. Its most relevant features are:

- Bare machine (type 1) hypervisor.
- Para-virtualization. The virtual machine interface is similar, but not identical, to the underlaying hardware.
- Dedicated devices: some devices can be directly and exclusively managed by a designated partition.
- Strong temporal isolation by enforcing a static cyclic plan to execute partition in a major temporal frame.
- Strong spatial isolation by allocation partitions at specified memory regions that cannot be accessed by other partitions.
- Safe partition execution: partitions are executed in processor user mode, whereas the hypervisor is executed in privileged processor mode.
- Fine-grained resource allocation via a configuration file that specifies the resources available on the board and the way they are allocated to partitions.
- Robust inter-partition communication mechanisms based on sampling and queuing ports.
- Some restricted services can only be used by *system* partitions, not by *normal* partitions (the default).
- Fault management model. Faults are detected and handled by the hypervisor, as a consequence of a system trap or a hypervisor-generated event.

XtratuM provides a virtual machine interface that is close to the native hardware. The virtual machine interface gives access to the system resources: CPU registers, clock, timer, memory, interrupts, etc., through a set of system calls

(*hypercalls*). Each virtual machine defines a partition that can contain either a bare machine application or an operating system on top which applications can run. An operating system that runs in a partition has to be *para-virtualized*, which implies that some parts of the operating system hardware abstraction layer (HAL) have to be replaced with the corresponding hypercalls.

The Xtratum architecture is shown in figure 2. The figure shows several partitions based on ORK+/XtratuM, and an additional partition based on a bare machine C code running directly on top of XtratuM. Other configurations are also possible. In the figure, only one partition is defined as a *system* partition, while the other ones are *normal* or user partitions. In general, several partitions can be configured as system partitions.

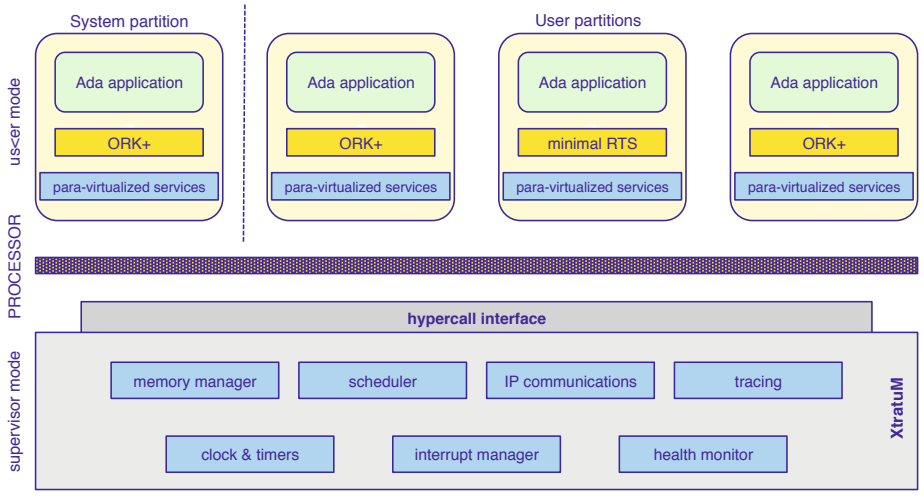

Fig. 2. XtratuM architecture

The services provided by XtratuM are summarized in table 1. As shown in the table, some services are constrained or partially constrained to be used only in system partitions. As an example, partition management is restricted in such a way that any partition can halt, stop, resume, reset, or shutdown itself, but only system partitions can execute these actions on other partitions. Likewise, only system partitions have access to the health monitor events in order to analyse errors detected at run time.

XtratuM provides some additional services for managing specific hardware resources that depend heavily on the processor architecture. Table 2 shows the specific services for the SPARC V8 architecture.

XtratuM version 2.2 is currently being used by CNES (Centre National d'Études Spatiales, France) as a time and space partitioning (TSP) based solution for building highly generic and reusable on-board payload software for space applications [1,2].

Table 1. XtratuM general hypercalls

Group of services	Hypercalls	Partition
Clock management	get clock; define timers	normal
IRQ Management	enable/disable IRQs, mask/unmask IRQs	normal
IP Communication	create ports read/receive/write/send messages	normal
IO management	read/write IO	normal
Partition management	mode change halt/reset/resume/ suspend/shutdown partitions	system
Health monitoring management	read/seek/status HM events	system
Audit facilities	read/status	system

Table 2. XtratuM SPARC V8 specific hypercalls

SPARC V8 hypercalls
XM_sparcv8_atomic_add
XM_sparcv8_atomic_and
XM_sparcv8_atomic_or
XM_sparcv8_flush_regwin
XM_sparcv8_get_flags
XM_sparcv8_inport
XM_sparcv8_iret
XM_sparcv8_outport
XM_sparcv8_set_flags

3 Overview of ORK+

ORK [19] is an open-source real-time kernel which provides full compliance with the Ada Ravenscar profile [4, D.15] on embedded computers. The kernel has a reduced size and complexity, and has been carefully designed to allow developers to build reliable software systems for embedded applications on LEON-based computers. It is integrated with a cross-compilation system based on GNAT[1]. The current version, ORK+, includes support for the new Ada 2005 timing features, such as execution-time clocks and timers, group budgets and timing events. Restricted support for execution-time timers and group budgets is provided, despite not being part of the Ravenscar profile, as these mechanisms have been found useful to ensure some temporal properties at run time [13].

The kernel functions can be grouped as follows:

- Task management, including task creation, synchronization, and scheduling.
- Time services, including absolute delays, real-time clock, execution time clocks and timers and timing events.

[1] `http://www.dit.upm.es/ork`

– Interrupt handling, including attaching a protected parameter procedure to a hardware interrupt vector, dealing with interrupt priorities and initializing vectors in the interrupt table.

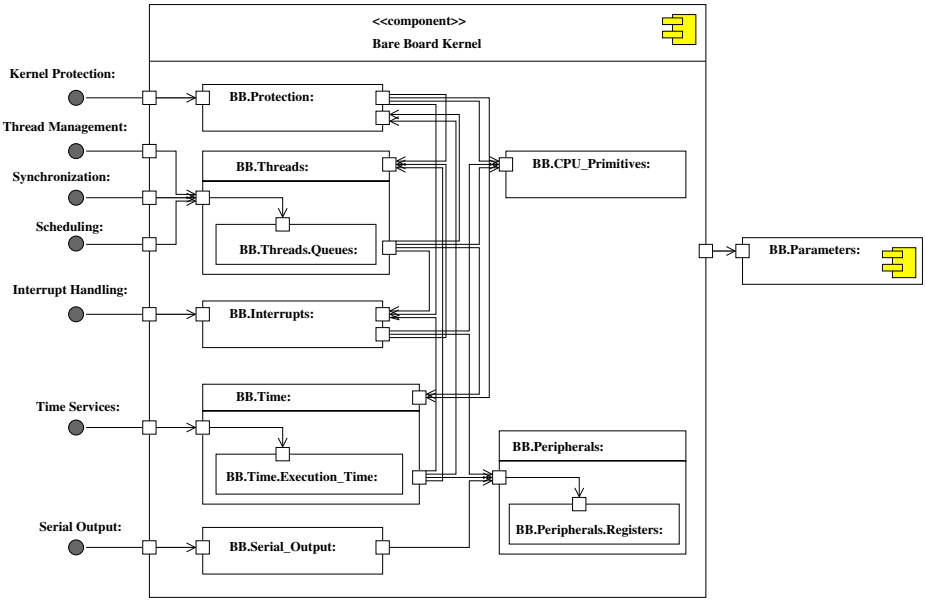

Fig. 3. ORK+ architecture

The kernel components implementing the above functions are depicted in figure 3. Most of the components are fully independent from the underlying platform. The lower-level components, which are hardware-dependent —namely BB.CPU_Primitives, BB.Time, BB.Peripherals, and BB.Interrupts— have platform-independent specifications. Consequently, only the implementations of these components have to be rewritten in order to port the kernel to the XtratuM virtual machine interface. Overall, 1398 out of 7316 lines of code have been modified. This figure accounts for the XtratuM interface as well as low-level assembly routines that are part of the implementation of the above packages.

4 Porting ORK+ to XtratuM

4.1 Adapting the XtratuM Interface

As a general rule, the kernel components that closely depend on the processor characteristics have to be re-implemented in order to port ORK+ to different hardware platforms. In the case of XtratuM, the porting strategy is slightly different, as the question is not to port the kernel to a different processor, but to the virtual processor interface provided by the hypervisor for a partition. As previously stated, the virtual processor is similar, but not identical, to the

original processor architecture. Therefore, the processor-dependent components of ORK+ have to be para-virtualized, i.e. some of the processor resources have to be accessed by means of the Xtratum hypercalls.

ORK+ is written mostly in Ada, except for a few low-level routines that are written in SPARC assembly language. On its side, the XtratuM native interface is coded in C, which is its main implementation language. Therefore, an Ada interface to XtratuM has to be built as a first step so that the ORK+ code can use the XtratuM hypercalls. To this purpose, the standard Interfaces.C package and pragma Import [4, Ap. B] have been used to write a new kernel package, named System.BB.XtratuM, which provides the hypercall interface. The parts of this package that are related to interrupt support are shown in listing 1 as an example.

4.2 CPU Management

The context switch operation is one of the first things to be re-implemented when porting a kernel to a different architecture, as it is highly dependent on the processor. In this case, the SPARC architecture includes a set of register windows that are especially complex to handle [16]. At the lowest level, XtratuM provides some basic support for this feature, including register window underflow and overflow trap handlers. However, the basic context switch operations have still to be provided by ORK+ in order to support Ada tasking at the application level.

An important difference when running ORK+ on top of XtratuM with respect to the original, bare machine implementation, is that now the kernel runs in user processor mode, as all XtratuM partitions do. As a result, the ORK+ context switch routine cannot access privileged processor registers, which are only available to the hypervisor, running in supervisor mode. Consequently, the ORK+ context switch routine has been rewritten so that all the assembly language code referencing privileged processor registers uses specific SPARC V8 hypercalls (see table 2). These hypercalls have a lightweight implementation in order to keep context switches and other low-level operations efficient.

4.3 Interrupt Support

XtratuM virtualizes the 16 interrupt sources of the SPARC architecture, and defines 32 additional virtual interrupt sources that are intended to be used for Xtratum services. For instance, XM_VT_HW_TIMER is a virtual interrupt source for a software elapsed-time timer defined by the hypervisor. Therefore, the number of interrupt sources in XtratuM is greater than in the original SPARC architecture. This has been reflected in the kernel System.BB.Interrupts package, as well as the standard Ada.Interrupts and Ada.Interrupt.Names packages, so that application code can attach protected parameterless procedures to all interrupt sources, including the XtratuM virtual interrupts.

XtratuM does not support priorities for interrupt sources. Therefore, all the interrupt sources have the same priority, as it is customary for hypervisor and

Listing 1. Interrupt support in the Xtratum Ada API

```
pragma Restrictions (No_Elaboration_Code);
with Interfaces.C;

package System.BB.Xtratum is
   pragma Preelaborate;

   use type Interfaces.C.unsigned;

   --------------------
   -- INTERRUPTS --
   --------------------
   XM_VT_EXT_FIRST : constant Interfaces.C.unsigned := 32;

   XM_VT_EXT_HW_TIMER : constant Interfaces.C.unsigned := 0 + XM_VT_EXT_FIRST;

   XM_VT_EXT_EXEC_TIMER : constant Interfaces.C.unsigned := 1 +
                          XM_VT_EXT_FIRST;
   procedure Disable_Interrupts;
   pragma Import (C, Disable_Interrupts, "XM_disable_irqs");
   -- All external interrupts (asynchronous traps) are disabled

   procedure Enable_Interrupts;
   pragma Import (C, Enable_Interrupts, "XM_enable_irqs");
   -- Interrupts are enabled

   function Mask_IRQ (Irq : Interfaces.C.unsigned) return Integer;
   pragma Import (C, Mask_IRQ, "XM_mask_irq");
   -- Mask an interrupt
   -- [1 .. 15] Hardware Interrupts. Traps from 0x11 to 0x1F
   -- [31 .. 63] Extended Interrupts. Interrupts trigger the traps from
   --            0x100 to 0x11F.

   function Unmask_IRQ (Irq : Interfaces.C.unsigned) return Integer;
   pragma Import (C, Unmask_IRQ, "XM_unmask_irq");

   ...

end System.BB.Xtratum;
```

operating systems. Consequently, ORK+ has been modified so that the System.Interrupt_Priority range has only one value.

4.4 Time Services

ORK+ provides direct support for the Ravenscar profile time services, i.e. Ada.Real_Time.Clock, absolute delays, global timing events, and execution-time clocks.

It also supports execution-time timers and group budgets, as an extension to the Ravenscar profile. The original implementation of these time services is based on two hardware timers: a periodic timer and a single-shot timer [19].

It must be noted that hardware timers are indeed elapsed-time timers. However, the XtratuM hypervisor, as it is common in partitioned systems, has a dual concept of time: in addition to the common notion of elapsed real-time, there is the notion of *partition-time*, which only advances when a partition is scheduled. Accordingly, Xtratum provides two kinds of software timers, as well as two kinds of clocks: elapsed-time clocks and timers, and partition-time clocks and timers.

The real-time mechanisms, i.e. Ada.Real_Time.Clock, absolute delays, and global timing events, are implemented in ORK+/XtratuM in a similar way to the bare machine version, i.e. by using the elapsed-time clock and timer. However, execution-time clocks cannot be implemented in the same way. Since the hypervisor switches the running partition without giving any notice to the software running in the partitions, implementing execution-time clocks on elapsed-time timers would also account for the time the partition is not running. In order to avoid this inconvenience, all execution-time mechanisms, i.e. execution-time clocks and timers, as well as group budgets, are implemented using partition time timers.

5 Performance Evaluation

5.1 General Approach

Possible losses in performance are a key issue when running critical real-time software on a partitioned system. We have carried out some experiments in order to quantify the loss of performance incurred by an application running on an ORK+/XtratuM partition with respect to using ORK+ on a bare LEON2 computer. To this purpose, we have developed a set of scenarios in order to evaluate which is the performance loss incurred by the hypervisor layer when different partition sets are defined.

There are two possible approaches to performance evaluation in this context:

- Direct measurements: the hypervisor is instrumented with breakpoints at the hypercall entry/exit points so that clock registers can be dumped in order to compute the execution time of every service.
- Indirect measurements: the application code executes some workload. The difference in the amount of work that is performed when running in a partition compared to what is done on the bare board provides a measurement of the effect of the hypervisor on the application.

We have opted for the indirect measurements approach as it is simpler to implement and does not require any special equipment nor instrumenting the hypervisor code. In order to get a good accuracy in the estimation of the overhead introduced by the hypervisor, an extremely simple workload, consisting on incrementing a counter whose value is sampled at periodic time intervals, has been

used. The total increment in an interval provides a good estimation of the number of instructions executed in it. It should be noted that the duration of the slot assigned to the partition must be long enough to guarantee that XtratuM will not perform a partition switch during the interval, in order to avoid additional overheads on the measurement.

The platform used for the evaluation is the TSIM/LEON SPARC simulator, version 2.0.6 (professional version) running at 80 MHz with 4 MB of RAM and separate instruction and data caches of 4 KB. During the evaluation the LEON2 processor was running at 2.25 CPIs (clock cycles per instruction), which means a performance of 35 MIPS. This is a typical value for current space systems.

5.2 Scenario Description

The evaluation scenario consists of three tasks:

- *Counter* task: a background, low priority task that continously increases a counter. The counter value is global and can be read by other tasks.
- *Timer* task: a periodic task with an intermediate priority level. The period of the task is calculated so as to generate a specified number of preemptions of the counter task in a reader task period.
- *Reader* task: a high priority service that periodically reads the counter value and stores the increment in the period. In the experiments the task period has been set to 1 second.

Figure 4 shows an execution diagram for the above tasks. This scenario has been executed several times for different values of the number of preemptions incurred by the counter task in a period of the reader task. The values used for the evaluation go from 1 to 1000 preemptions per second, which correspond to timer task periods between 1000 and 1 milliseconds. This scenario has been executed in a slot in the XtratuM schedule which is longer than the total duration of the experiment, so that no additional interference due to partition context switch is incurred.

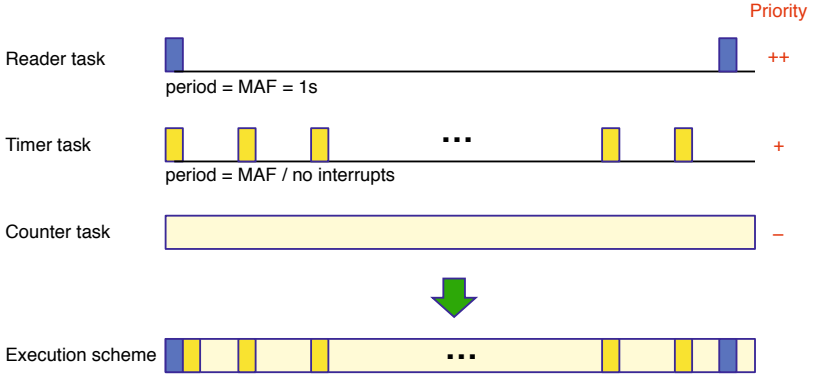

Fig. 4. Evaluation scenario

Table 3 shows the results of the evaluation for ORK+ running on a native LEON2 platform, and ORK+ running in a XtratuM partition. The numbers are the average, maximum, and minimum count values in a reader task period over a 50-second interval, which is the total duration of the experiment. The results are shown for different values of the timer task period (TTP).

Table 3. Evaluation results

ORK+ on native LEON2							
TTP	1000	500	100	50	10	5	1
AVG	9999113	9998618	9994117	9988498	9943509	9887269	9444166
MAX	9999131	9998642	9994141	9988522	9943520	9887280	9445368
MIN	9999046	9998558	9994056	9988436	9943462	9887222	9437370
DIF	0	495	4996	10615	55604	111844	554947
PL1	0,000%	0,004%	0,049%	0,106%	0,556%	1,118%	5,549%
ORK+ on XtratuM							
TTP	1000	500	100	50	10	5	1
AVG	9997222	9994785	9975228	9950780	9755212	9510748	7555050
MAX	9997235	9994791	9975236	9950790	9755216	9510760	7555056
MIN	9997185	9994750	9975195	9950749	9755176	9510719	7555019
DIF	0	2437	21994	46442	242010	486474	2442172
PL2	0,000%	0,024%	0,219%	0,464%	2,42%	4,866%	24,428%
PL3	0,02%	0,03%	0,19%	0,38%	1,89%	3,80%	20,00%

The first column, for a TTP of 1000 ms, provides a basic value for the performance, as there are no preemptions of the counter task in a reader task period. For the rest of the TTP values, the difference (DIF) between the average values and the reference value provides an indication of the performance losses (PL) of the counter task due to the task switching overhead of the timer task. The following performance loss indicators are shown:

- *PL1*: performance loss for ORK+ with respect to its best case (TTP = 1000).
- *PL2*: performance loss for ORK+/XtratuM with respect to its best case (TTP = 1000).
- *PL3*: performance loss for ORK+/XtratuM with respect to ORK+.

Figure 5 shows the performance loss values for different values of the timer task period.

The above results show that for ORK+ on native LEON2, the performance loss due to task switching (PL1) is negligible when the timer period is longer than 10 ms, and only reaches a significant value for a timer period of 1 ms, which is seldom found in the kind of space applications we have in mind. The performance loss for ORK+ on XtratuM (PL2) follows a similar pattern. The PL3 figures give an indication of the cost of virtualization, i.e. the difference in performance between the ORK+/XtratuM and the native ORK+ configurations. Again, it can be seen that it is only significant for very short periods, far below the values

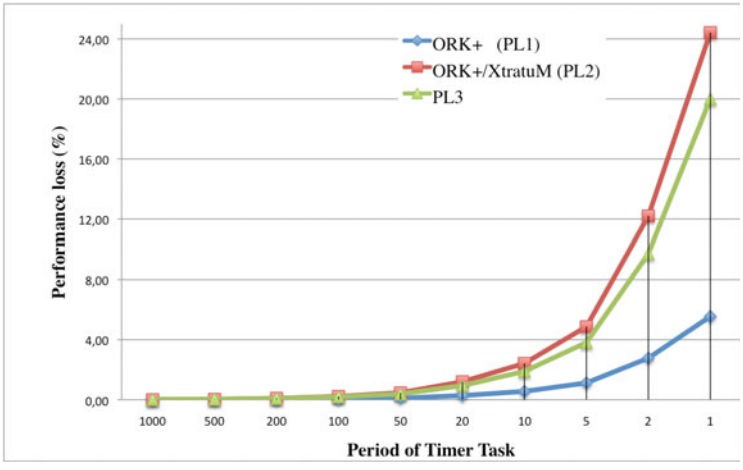

Fig. 5. Evaluation scenario

that are commonly found in on-board software applications. These results can be considered very satisfactory, taking into account the advantages of having several applications running in an isolated (temporal and spatial) partitioning framework.

Finally, it must be noted that PL2 values are about 5 times the corresponding PL1 values. This result roughly means that the periodic task switching overhead in ORK+/XtratuM is about 5 times the native ORK+ overhead. This increment is mainly due to clock management, as XtratuM general hypercalls have to be made not only to set the elapsed timer for the absolute delay, but also to keep the values of the execution-time clocks using the partition time clock. It can thus be concluded that the implementation of Ada timing services on top of a virtualization layer deserves further research.

6 Related Work

Although hypervisors were first developed by IBM in the 1960s, there has been a recent revival of interest in this technology, due to projects such as Xen [7], VMWare [20] and L4 [11]. These projects were aimed at building virtualizers for general purpose systems, including desktop PCs and servers.

More recently, hypervisors have been used in embedded and real-time systems. For example, PikeOS [10] is a microkernel based on L4 which is targeted to embedded systems and uses virtualization techniques. Although microkernels were first developed as an architectural approach for building large and complex operating systems, they can also be used as bare-metal supervisors. The PikeOS architecture has two main components, namely the *separation kernel* and the *system software*. The former runs in supervisor mode and provides a set of basic services: scheduling, memory management and interrupt handling. The

latter runs in user mode and is shared by all partitions. It provides services for inter-partition communication and device drivers. The services provided by the microkernel can be used to build several different operating systems, resulting in a virtualized system.

Other related projects are NOVA [17], which is aimed at constructing a secure virtualization environment, and OKL4 Microvisor [9], which is designed as a highly-efficient hypervisor for use in embedded systems.

Generally speaking, it can be said that para-virtualization is the virtualization method that better fits the requirements of embedded systems with real-time constraints. Other methods which can provide full virtualization introduce a siginficant overhead in the system execution, with a direct impact on the predictability of the the applications running in the different partitions. For example, binary translation works by catching the execution of conflicting instructions and replacing them on the fly, which has a clear cost in terms of execution time.

7 Conclusions and Future Work

Combining ORK+ and Xtratum builds up an efficient partitioning platform that enables real-time Ada applications with different criticality levels to run on a LEON2 platform. Time and space isolation between partitions is implemented by the XtratruM hypervisor, and ORK+ provides timing predictability within each partition. The temporal behaviour of applications running on a hierarchical scheduling environment like that provided by ORK+/XtratuM can be statically analysed using an extension of response-time analysis methods [6,5]. It should be noted that ORK+ provides an additional level of enforcement of the required temporal behaviour by means of execution time clocks and timers [13].

XtratuM and ORK+ are currently targeted to LEON2-based computers. The LEON2 support for spatial isolation is rather primitive, consisting only of a set of fence registers, which do not provide any protection against incorrect read operations. This kind of limited memory protection mechanism also imposes a rigid memory sharing scheme between different partitions. This limitation is expected to be overcome with the next-generation of LEON3 processors, which have a full-featured MMU.[2] Future plans include porting the platform to LEON3 and other common embedded processor architectures.

References

1. Arberet, P., Metge, J.J., Gras, O., Crespo, A.: TSP-based generic payload on-board software. In: DASIA 2009, Data Systems in Aerospace, Istanbul (May 2009)
2. Arberet, P., Miro, J.: IMA for space: status and considerations. In: ERTS 2008, Embedded Real-Time Software, Toulouse France (Jannuary 2008)
3. ARINC: Avionics Application Software Standard Interface — ARINC Specification 653-1 (October 2003)

[2] Memory Management Unit.

4. Tucker Taft, S., Duff, R.A., Brukardt, R.L., Plödereder, E., Leroy, P.: Ada 2005 Reference Manual. LNCS, vol. 4348. Springer, Heidelberg (2006) ISBN 978-3-540-69335-2
5. Balbastre, P., Ripoll, I., Crespo, A.: Exact response time analysis of hierarchical fixed-priority scheduling. In: Proceedings of 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (August 2009)
6. Davis, R., Burns, A.: Hierarchical fixed priority pre-emptive scheduling. In: Proceedingsof the 26th IEEE International Real-Time Systems Symposium — RTSS 2005 (2005)
7. Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., Neugebauer, R.: Xen and the art of virtualization. In: Proceedings of the ACM Symposium on Operating Systems Principles (October 2003), http://www.citeseer.ist.psu.edu/dragovic03xen.html
8. Gaisler Research: LEON2 Processor User's Manual (2005)
9. Heiser, G., Leslie, B.: The OKL4 Microvisor: Convergence point of microkernels and hypervisors. In: Proceedings of the 1st Asia-PacificWorkshop on Systems, New Delhi, India, pp. 19–24 (August 2010)
10. Kaiser, R., Wagner, S.: Evolution of the PikeOS microkernel. In: MIKES 2007: First International Workshop on MicroKernels for Embedded Systems, Sydney, Australia (2007)
11. Liedtke, J.: On microkernel construction. In: Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP-15). Copper Mountain Resort, CO (December 1995), http://www.l4ka.org/publications/
12. Masmano, M., Ripoll, I., Crespo, A., Metge, J., Arberet, P.: Xtratum: An open source hypervisor for TSP embedded systems in aerospace. In: DASIA 2009, Data System in Aerospace, Istanbul (May 2009)
13. Mezzetti, E., Panunzio, M., Vardanega, T.: Preservation of timing properties with the ada ravenscar profile. In: Real, J., Vardanega, T. (eds.) Ada-Europe 2010. LNCS, vol. 6106, pp. 153–166. Springer, Heidelberg (2010)
14. Pulido, J.A., Urueña, S., Zamorano, J., Vardanega, T., de la Puente, J.A.: Hierarchical scheduling with ada 2005. In: Pinho, L.M., González Harbour, M. (eds.) Ada-Europe 2006. LNCS, vol. 4006, pp. 1–12. Springer, Heidelberg (2006)
15. Rushby, J.: Partitioning for safety and security: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center (June 1999), also to be issued by the FAA
16. SPARC International, Upper Saddle River, NJ, USA: The SPARC architecture manual: Version 8 (1992), http://www.sparc.com/standards/V8.pdf
17. Steinberg, U., Kauer, B.: Nova: a microhypervisor-based secure virtualization architecture. In: EuroSys, pp. 209–222 (2010)
18. Urueña, S., Pulido, J.A., López, J., Zamorano, J., de la Puente, J.A.: A new approach to memory partitioning in on-board spacecraft software. In: Kordon, F., Vardanega, T. (eds.) Ada-Europe 2008. LNCS, vol. 5026, pp. 1–14. Springer, Heidelberg (2008)
19. Urueña, S., Pulido, J.A., Redondo, J., Zamorano, J.: Implementing the new Ada 2005 real-time features on a bare board kernel. Ada Letters XXVII(2), 61–66 (2007); Proceedings of the 13th International Real-Time Ada Workshop (IRTAW 2007)
20. White paper: Virtualization overview (2006), http://www.vmware.com/pdf/virtualization.pdf