15th Triennial World Congress, Barcelona, Spain

GNAT/ORK: AN OPEN CROSS-DEVELOPMENT ENVIRONMENT FOR EMBEDDED RAVENSCAR–ADA SOFTWARE

Juan Zamorano [*] **José F. Ruiz** [**]

[*] *Departamento de Arquitectura y Tecnología de Sistemas Informáticos*
Universidad Politécnica de Madrid, E-28660 Madrid, Spain
[**] *Departamento de Ingeniería de Sistemas Telemáticos*
Universidad Politécnica de Madrid, E-28040 Madrid, Spain
jzamora@fi.upm.es, jfruiz@dit.upm.es

Abstract: Ada tasking is a powerful abstraction mechanism for developing concurrent embedded systems. However, many implementations of concurrent tasking have been seen as potentially unsafe for critical systems because of their high degree of indeterminism. The *Ravenscar profile* is a subset of Ada 95 tasking with purpose of providing a basis for the implementation of certifiable critical systems. ORK is an open-source real-time kernel which provides full conformance with the Ravenscar profile on embedded computers. The kernel has a reduced size and complexity, and has been carefully designed to allow the building of reliable software for embedded applications. This kernel is integrated in a cross-compilation system based on GNAT 3.13, supporting the subset of Ada 95 tasking which is allowed by the Ravenscar profile in an efficient and compact way. It is closely integrated with other tools, including a tasking-aware version of GDB.

Keywords: Safety–critical, Real–time systems, Real–time languages, Ada tasking programs, Software tools

1. INTRODUCTION

Mission-critical embedded software has usually been developed on top of a cyclic executive that invokes the execution of application tasks according to a predefined static schedule. There are thus no concurrent threads of execution, and the application code is made of a set of purely sequential procedures.

This approach leads to simple, robust implementations, and provides a deterministic time behavior which has often been considered a requirement for critical real-time systems. However, as the functionality and complexity of embedded software increases, and there is more and more pressure for shortening development times and reducing costs, while keeping up with critical reliability requirements, its low-level nature and lack of flexibility make it less appropriate. As a consequence, more attention is being devoted to higher level, abstract development methods that include concurrency as a means of decoupling application tasks and making software easier to design and test (Vardanega, 1998).

Indeed, many implementations of concurrent tasking have been seen as potentially unsafe for critical systems because of their high degree of indeterminism, which may make programs difficult to validate. This has led to either completely banning tasking for critical software applications, which is the traditional approach, or to the more flexible approach of building specialized kernels with reduced functionality. By limiting the way tasks are executed and synchronized, it can be expected that concurrent systems can be analyzed and tested, so that safe concurrent systems can be built in a way that has significant advantages over cyclic executives from the point of view of flexibility and structuring.

Ada (Ada, 1995) is the language of choice for many critical systems due to its careful design and the existence of clear guidelines for building safe systems (Ada, 2000). While the first approaches to developing safe Ada software did not make use of Ada tasking (Holzapfel and Winterstein, 1988; Barnes, 1997), recent advances in real-time systems timing analysis methods (Audsley *et al.*, 1996) have paved the way to safe tasking in Ada. The *Ravenscar profile* (Burns, 1999) is a subset of Ada 95 tasking that was initially defined at the 8th International Real-Time Ada Workshop (IRTAW8) with purpose of providing a basis for the implementation of certifiable critical systems.

This paper describes the design and implementation of the *Open Ravenscar Real-time kernel (ORK)*, which is an open-source kernel compliant with the Ravenscar profile. The kernel is fully integrated with the GNAT compilation system. Debugging support, including tasking, is based on an enhanced version of the GDB debugger and the DDD graphic front-end. The distribution of the cross-compilation system includes an adapted version of GNAT hosted on i386 Linux or SPARC Solaris workstations and targeted to ERC32 [1] [2] and PC-compatible embedded computers, the kernel itself, adapted version of GDB and DDD, and some additional libraries and tools. It is freely available as an open source product, with a GPL license [3].

2. THE RAVENSCAR PROFILE

The Ravenscar profile defines a subset of the tasking features of Ada 95 for high-integrity applications. The profile is based on a computation model with the following features:

- A single processor.
- A fixed number of tasks.
- A single invocation event for each task. The invocation event may be generated by the passing of time (for time-triggered tasks) or by a signal from either another task or the environment (for sporadic tasks).
- Task interaction only by means of shared data (protected objects) with mutually exclusive access.

The profile forbids many of the most complex tasking features, including dynamic and nested tasks and protected objects, requeue, asynchronous transfer of control, task termination, task abortion, task entries and rendez-vous, dynamic priorities, relative delays, select statements, and multiple protected entries.

The tasking model defined by the profile includes tasks and protected types and objects at the library level, a maximum of one protected entry with a simple boolean barrier for synchronization, a real-time clock, absolute delays, preemptive priority scheduling with ceiling locking access to protected objects, and protected procedure interrupt handlers, as well as some other features, which allow the development of embedded real-time systems. For a full description, see the full profile definition (Burns, 1999).

The Ravenscar profile defines a computation model similar to the one proposed by Vardanega (Vardanega, 1998), which is based on the HRT-HOOD method (Burns and Wellings, 1995). The profile allows implementing embedded systems with the tasking facilities provided by Ada, restricted so as to ensure that the system can be analyzed for accurate timing and safety requirements. Preliminary experience confirms the value of this approach for space embedded software development (García *et al.*, 2001).

3. THE OPEN RAVENSCAR REAL-TIME KERNEL

3.1 *Runtime Architecture*

Ada tasking is implemented in GNAT by means of the run-time library, called GNARL (GNU Ada Runtime Library) (Baker *et al.*, 1996). The parts of GNARL which are dependent on a particular machine and operating system are known as GNULL (GNU Low-level Library), and its interface to the platform-independent part of the GNARL is called GNULLI (GNULL Interface). Most implementations of GNULL are built as a layer of glue code added on top of an existing set of POSIX thread functions (IEEE, 1990), which in turn may be implemented on top of an operating system.

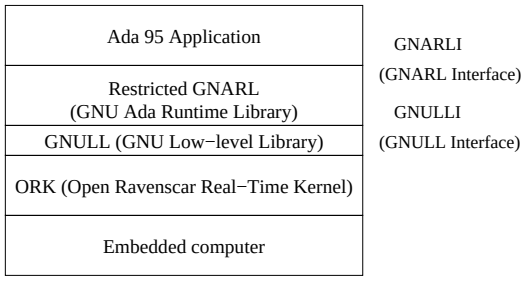

Fig. 1. Architecture of the GNAT run-time system based on ORK

The ORK version of GNAT uses this approach, but instead of providing a full POSIX conformant kernel interface, which would impose too much overhead on the system, as GNARL already provides most of the functionality which is needed for tasking, takes advantage of a simpler interface at the kernel level, which provides an almost direct implementation of GNULLI, with the GNULL packages acting as a thin glue layer for GNAT (figure 1).

[1] ERC32 is a radiation-hardened implementation of the SPARC v7 architecture.

[2] This work has been funded by ESA/ESTEC contract no. No.13863/99/NL/MV.

[3] ORK and its associated tools for developing embedded software can be downloaded from `http://www.openravenscar.org`

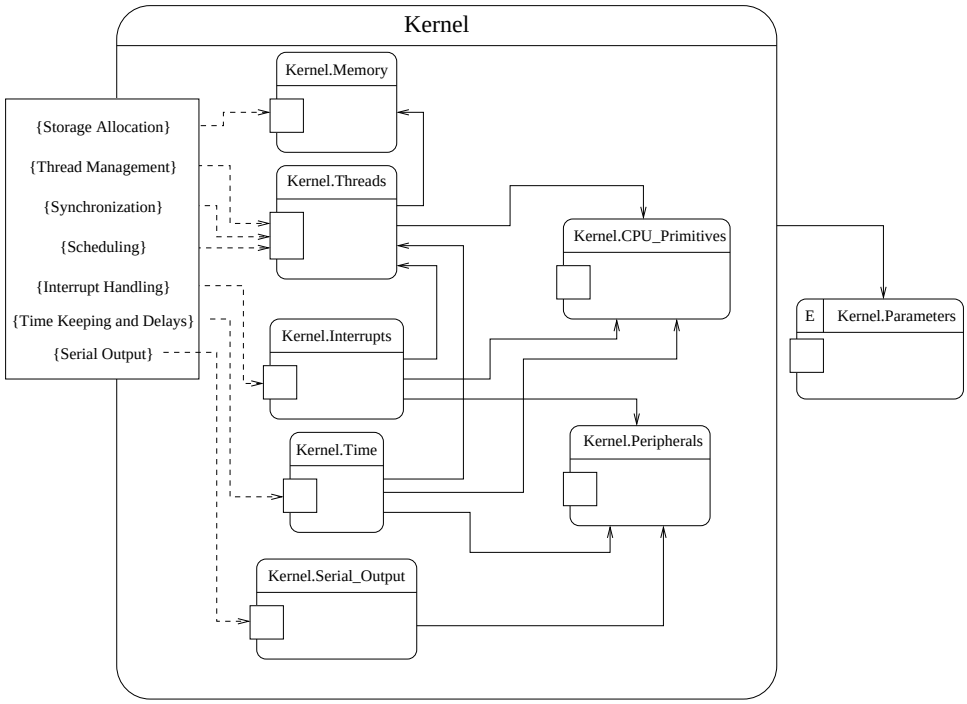

Fig. 2. ORK packages.

3.2 *ORK Architecture*

The kernel itself consists of a set of Ada packages, all of them children of an empty root package called Kernel (figure 2). This structure is similar to that of the JTK (Ruiz and González-Barahona, 1999) and Top-Layer (Kamrad and Spinney, 1999) kernels. Some of the packages have additional children that extend their interfaces so that some of their internal functionality is made visible to other kernel packages.

Kernel primitives in ORK are always non-threaded (interrupts are disabled while accessing the kernel), so that kernel operations are only executed on behalf of a specific user-level thread (to which the relevant overhead can thus be charged). There are no hidden threads within the kernel.

4. IMPLEMENTATION ISSUES

The kernel is written in Ada 95, except for a small part which is written in assembly language. A sequential subset of Ada has been defined based on the recommendations of the *Ada High Integrity Systems* standard (Ada, 2000). The Ada features which are not used in the subset are detailed elsewhere (UPM, 2000).

4.1 *Thread Scheduling and Synchronization*

The ready thread queue is implemented as a priority-ordered double-linked list. Space for the maximum number of threads that can exist in the system is reserved at initialization, thus avoiding the need for dynamic storage management.

Delayed threads are put on a single queue, ordered by delay expiration time. The queue is implemented as a linear linked list. An “alarm clock” approach is used to signal delay expiration and the subsequent thread activation (Zamorano *et al.*, 2001).

Ada 95 protected objects provide the synchronization mechanisms to operate on shared data in mutual exclusion. In addition, the Ravenscar profile requires protected objects to implement the ceiling locking protocol. The implementation of protected objects, as well as the runtime internal data protection, requires the kernel to implement two synchronization objects: mutexes and condition variables.

Mutexes provide access with mutual exclusion to shared data. ORK takes great advantage of being targeted primarily to a monoprocessor system, and its implementation of mutexes is very simple and efficient. It suffices with raising the priority of the locking thread to the ceiling priority of the mutex.

Condition variables provide the functionality required by a thread to voluntarily suspend itself to wait for some condition to be satisfied. The semantics of condition variables have also been dramatically simplified with respect to POSIX (Baker *et al.*, 1996) because the Ravenscar profile restrictions avoid the queueing of tasks.

4.2 *Fast Context Switch*

One issue to take into account is that not all the tasks will use the floating point unit. Thus, the floating point context should not be stored until necessary. It should remain in the floating point registers and not

disturbed until another floating point task is switched to. The current implementation saves the floating point context only when necessary.

Another issue must be take into account for the ERC32 version of the ORK kernel. The SPARC v7 has a total of 167 user-allocable registers and 128 of these are used for the overlapping register windows. The 128 window registers are grouped into eight sets of 24 registers called *windows*. During a context switch, the register windows of the current thread must be flushed onto the thread stack before a window will be loaded with the top frame of the new thread.

There are two different approaches to follow for the flushing policy. The kernel can flush all register windows, or just the windows currently in use. The latter approach gives better average context switch time (Snyder *et al.*, 1995), and is the one used in ORK. However, the worst case value is approximately the same in both approaches.

4.3 *Time Management*

Annex D of the Ada Language Reference Manual (Ada, 1995) requires a maximum resolution of 1 ms and a minimum range of 50 years for Ada.Real_Time.Time. ORK uses 64 bits to store time values with a granularity equal to a nanoseconds, therefore the range is (-292, 292) years. In order to provide high resolution software timers and the monotonic, precise clock maintaining a low overhead, ORK uses two hardware timers.

ERC32 (Temic, 1997) provides better arrangement than PC architecture because ERC32 has two timers which have 32 bits wide down-count registers. Moreover, these timers are able to request interrupts with different priorities. The ERC32 version of ORK configures one of them as a timestamp counter so that it generates periodic interrupts (Zamorano *et al.*, 2001). The least significant part of the clock value is stored in the timer down-count register, while the most significant part of the same value is stored in main memory, and incremented by the timer interrupt handler. Since the timer down-count is automatically restarted by the hardware, the clock precision is given by the hardware and the clock resolution is given by the scaled oscillator frequency.The interrupt period can be much longer (actually up to the down-count register capacity) without loss of precision.

The other one is used as single-shot timer. Therefore interrupts are generated on demand, and not periodically. The single-shot timer is reprogrammed on demand every time an alarm is set so that it interrupts when the alarm expires. This arrangement provides high resolution software timers with a low overhead. For a 10MHz ERC32, the clock and delays resolution are 100 ns, and the interrupt period could be up to 429 seconds.

The timing services implementation for the PC version of ORK uses the i8254 Programmable Interval Timer (PIT) (Intel, 1990). The PIT is a standard device in the PC architecture that has three 16 bits wide timers with a scaled oscillator period of 839 ns, which is the resolution for the PC version. The main problem with the PIT is that only timer 0 is able to request interrupts.

Therefore, The PC version of ORK uses the same strategy than RT-Linux (Barabanov, 1997). Timer 0 is used as single-shot timer and interrupts are generated to support the monotonic clock and alarms. In such a way that, timer 0 is programmed with its maximum interval (55 ms), if there are no closer alarms. Otherwise, it is programmed with the required interval. The most significant part of the clock is incremented accordingly.

However, this method introduces a problem related to the precision of the clock, as the time spent by the processor to recognize an interrupt and reprogram the timer varies depending on the behavior of the software. As a result, the clock can suffer an additional drift, which is different from the usual manufacturing drift which deviates the clock in a linear way. ORK uses timer 2 to measure the time required to reprogram timer 0 and this time is also added to the most significant part of the clock. In this way, the software drift of the clock is avoided.

4.4 *Hardware Exception Handling*

Ada exceptions are entities that represent a kind of exceptional situation that arise during program execution. It is a very powerful mechanism to handle unexpected situations.

The Ravenscar profile does not limit any of the features of sequential Ada (Burns, 1999), and hence it does not place any restriction on exceptions (either user defined or predefined exceptions), so they can be fully used with GNAT/ORK (de la Puente *et al.*, 2000).

The default mechanism for handling exceptions in GNAT is to translate hardware traps to some reserved POSIX signals (SIGFPE, SIGSEGV, ...), and these signals are mapped to standard Ada exceptions (Oh *et al.*, 1996) (Storage_Error and Constraint_Error).

The ORK kernel which is underneath the GNAT runtime is only in charge of redirecting trap occurrence to the appropriate exception handler. It is achieved by filling the trap table with a pointer to a wrapper which in turn calls the appropriate Ada exception handler. This way hardware exceptions are translated into Ada exceptions in ORK.

This mechanism is very simple and efficient, even more if compared to the default mechanism in GNAT which uses the complex POSIX signal mechanism. Of course the ORK scheme has a penalty, as the portability of the POSIX signal approach is higher.

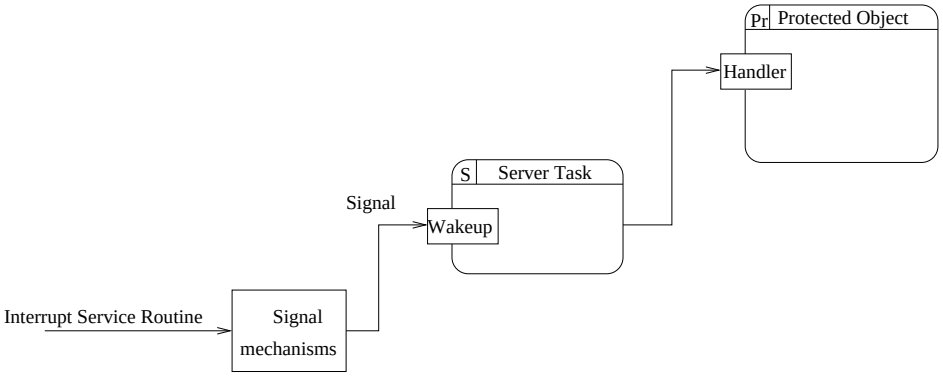

Fig. 3. Interrupt handling in GNARL

4.5 *Interrupt Handling*

The default mechanism for handling interrupt in GNAT is based on a two level approach. At the lowest one, interrupts are translated to POSIX signals. The highest level maps these signals to the standard form of Ada interrupts by means of asynchronous signal handler procedures, attached using sigwait (Oh *et al.*, 1996). This mechanism is very similar to the one used for handling exception, Therefore it also suffer from the performance penalty for preserving portability.

In addition there is another source of overhead in the current GNARL implementation. Interrupt handlers are executed within the context of specially dedicated server tasks, each one associated to an interrupt source (figure 3). This approach simplifies the scheduling of interrupt handlers, and provides a simple way to achieve mutual exclusion between handlers. However, this implementation is very inefficient and uses tasks with entries, which is forbidden in the Ravenscar profile and cannot thus be used with ORK.

The Ada interrupt handling model (Ada, 1995; Intermetrics, 1995; Baker *et al.*, 1996) implies that a protected handler can only be preempted by a higher priority interrupt. ORK masks all interrupts with a lower priority than the currently active priority, by making the hardware priority equal to the active priority. Moreover, the Ravenscar profile requires the ceiling locking protocol, which means that protected interrupt handlers cannot preempt other operations on the same protected objects. As a result, an interrupt handler can never be blocked waiting for a protected object to be free, and protected handlers can be directly invoked from the Interrupt Service Routine (figure 4). The package Kernel.Interrupts provides operations to install and detach interrupt handlers.

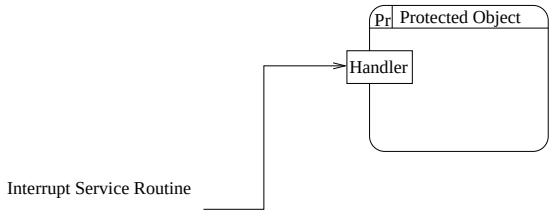

Fig. 4. Interrupt handling in ORK

5. CONCLUSIONS AND FUTURE WORK

The Open Ravenscar Kernel supports the full Ravenscar profile with the GNAT compilation system. It has been thoroughfully tested with a Ravenscar profile compatible subset of the ACVC suite, plus a set of additional tests that have been specifically designed to check compliance with the profile. Some problems have been found, partly due to the GNAT implementation, but all of them can be solved with compiler modifications. The validation process and its results are described in the *Software Validation and Verification Report*, available at the Open Ravenscar web site (CASA, 2000).

The kernel has a reduced size (8 KB plus 4 KB of vector table for the ERC32 version), the measured context switching time for the ERC32 version is around 523 CPU cycles (52.3μs for a 10 MHz ERC32), and the interrupt latency is 1708 CPU cycles (170.8μs for a 10 MHz ERC32).

However, the minimum size program is about 95 KB, mainly due to the high amount of code linked into the executable file as a result of references made by GNARL and the GNAT compiler itself. Many of these references are not used by Ravenscar-compliant programs, and thus the code size should be reduced accordingly. We are currently working in a better adaptation of GNAT and the upper layers of the run time library to the Ravenscar profile. There are still a number of issues that have to be solved so that GNAT supports the profile in an efficient way. Interrupt handling is one example of such issues, which in this case has been solved by the ORK team.

The kernel has been tested on real hardware (PC-Compatible computers and eVAB-695E Temic Evaluation Board) as well as ERC32 (SIS [4] and TSIM [5])

[4] SIS is not free software, and it is not part of ORK. See `http://www.estec.esa.nl/wsmwww/erc32/freesoft.html` for further details.

[5] TSIM is not free software, and it is not part of ORK. See `http://www.gaisler.com/` for further details.

and PC (VMware[6]) simulators. ORK has been carefully designed to isolate hardware dependencies, allowing easy retargeting. We plan to port the kernel to other hardware monoprocessor platforms, such as PowerPC or the CPU32 family of microcontrollers.

6. ACKNOWLEDGMENTS

We gratefully thank the others members of the development team: J.A. de la Puente, R. García, R. Fernández, and P. Palomo. J. Amador and T. Vardanega from the European Space Agency have also contributed with their helpful suggestions.

7. REFERENCES

Ada (1995). *Ada 95 Reference Manual: Language and Standard Libraries. International Standard ANSI/ISO/IEC-8652:1995*. Available from Springer-Verlag, LNCS no. 1246.

Ada (2000). *Guidance for the use of the Ada Programming Language in High Integrity Systems*. ISO/IEC TR 15942:2000.

Audsley, Neil, Ian Bate and Alan Burns (1996). Putting fixed priority scheduling theory into engineering practice for safety critical applications. In: *IEEE Real-Time Technology and Applications Symposium*. IEEE Computer Society Press.

Baker, T.P., Dong-Ik Oh and Seung-Jin Moon (1996). Low-Level Ada tasking support for GNAT - performance and portability problems. In: *Proceedings of the Washington Ada Symposium*.

Barabanov, Michael (1997). A Linux-based real-time operating system. Master's thesis. New Mexico Institute of Mining and Technology. Available at `http://www.rtlinux.org/\~baraban/thesis`.

Barnes, John (1997). *High Integrity Ada. The SPARK Approach*. Addison Wesley.

Burns, Alan (1999). The Ravenscar profile. *Ada Letters* **XIX**(4), 49–52.

Burns, Alan and Andy Wellings (1995). *HRT-HOOD(TM): A Structured Design Method for Hard Real-Time Ada Systems*. North-Holland. Amsterdam.

CASA (2000). *Open Ranvenscar Real-Time Kernel - Software Validation and Verification Report*. CASA, Space Division. Available at `http://www.openravenscar.org`.

de la Puente, Juan A., José F. Ruiz and Juan Zamorano (2000). An open Ravenscar real-time kernel for GNAT. In: *Reliable Software Technologies — Ada-Europe 2000* (Hubert B. Keller and Erhard Ploedereder, Eds.). number 1845 In: *LNCS*. Springer-Verlag. pp. 5–15.

García, Rodrigo, Tullio Vardanega and Juan A. de la Puente (2001). An application case for the Ravenscar profile: Porting OBOSS to GNAT/ORK. In: *Reliable Software Technologies — Ada-Europe 2001* (Alfred Strohmeier and Dirk Craeynest, Eds.). number 2043 In: *LNCS*. Springer-Verlag. pp. 392–404.

Holzapfel, R. and G. Winterstein (1988). Ada in safety critical applications. In: *Ada in Industry* (S. Heilbrunner, Ed.). Cambridge University Press.

IEEE (1990). *Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (Incorporating IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995)*. IEEE. ISO/IEC 9945-1:1996.

Intel (1990). *Peripherals. ISA chip sets*. Intel Corporation.

Intermetrics (1995). *Ada 95 Rationale: Language and Standard Libraries*. Intermetrics. Available from Springer-Verlag, LNCS no. 1247.

Kamrad, M. and B. Spinney (1999). An Ada runtime system implementation of the Ravenscar profile for a high speed application layer data switch. In: *Reliable Software Technologies — Ada-Europe'99* (Michael González-Harbour and Juan A. de la Puente, Eds.). number 1622 In: *LNCS*. Springer-Verlag. pp. 26–38.

Oh, Dong-Ik, T.P. Baker and Seung-Jin Moon (1996). The GNARL implementation of POSIX/Ada signal services. In: *Proceedings of the Ada-Europe'96*.

Ruiz, José F. and Jesús M. González-Barahona (1999). Implementing a new low-level tasking support for the GNAT runtime system. In: *Reliable Software Technologies — Ada-Europe'99* (Michael González-Harbour and Juan A. de la Puente, Eds.). number 1622 In: *LNCS*. Springer-Verlag. pp. 298–307.

Snyder, J.S., D.B. Whalley and T.P. Baker (1995). Fast context switches: Compiler and architectural support for preemptive scheduling. *Microprocessors and Microsystems* **19**(1), 35–42.

Temic (1997). *SPARC RT Memory Controller (MEC) User's Manual*. Temic/Matra Marconi Space.

UPM (2000). *Open Ravenscar Kernel — Software Design Document*.

Vardanega, Tullio (1998). Development of On-Board Embedded Real-Time Systems: An Engineering Approach. PhD thesis. TU Delft. Also available as ESA STR-260.

Zamorano, Juan, José F. Ruiz and Juan A. de la Puente (2001). Implementing Ada.Real_Time.Clock and absolute delays in real-time kernels. In: *Reliable Software Technologies — Ada-Europe 2001* (Alfred Strohmeier and Dirk Craeynest, Eds.). number 2043 In: *LNCS*. Springer-Verlag. pp. 317–327.

[6] VMware is a trade mark of VMware, Inc.