Implementing the New Ada 2005 Real-Time Features on a Bare Board Kernel*

Santiago Urueña suruena@datsi.fi.upm.es
José Pulido pulido@dit.upm.es
José Redondo jredondoh@dit.upm.es
Juan Zamorano jzamora@datsi.fi.upm.es

Universidad Politécnica de Madrid (UPM)

Abstract

A real-time kernel providing timing services is a key component of any real-time system. The current revision of the Ada standard provides a range of real-time mechanisms that can be used to ensure the required temporal behaviour of real-time tasks. However, kernel timing services must be implemented carefully in order to avoid overheads and inaccuracies. This paper describes the implementation of the Ada timing services in an evolved version of the Open Ravenscar Kernel. The interrelation among the different timing mechanisms is also analysed and evaluated.

1. Introduction

High-integrity real-time systems usually have hard timing requirements, which have to be guaranteed by using an appropriate engineering approach for their design and implementation (see e.g. [19]). Such an approach is usually based on a computation model which enables the temporal behaviour of a system to be analysed and adjusted if necessary.

Ensuring the required real-time behaviour usually relies on an accurate knowledge of the worst-case computation times (WCET) of all the real-time tasks. Although some good techniques for computing WCET are available [16], there is still a large degree of uncertainty, especially when modern processors with cache memories, speculative execution and segmentation are used. Pessimistic WCET estimations lead to an underutilisation of resources, and thus tight estimates are usually sought. The risk with tight WCET estimates is, on the other hand, to be optimistic, and then occasionally get an actual execution time which is larger than the estimated value. This situation is called an *overrun*, and may give raise to a generalised miss of deadlines by tasks by a domino effect.

*This work has been partially funded by the IST Programme of the European Commission under project IST-004033 (ASSERT) and the Spanish Ministry of Science and Technology (MCYT), project TICTIC2005-08665-C03-01 (THREAD).

The new Ada real-time mechanisms can be used to monitor the run-time behaviour of tasks. In this way it is possible to detect overruns and deadline misses and take corrective actions before other tasks are affected [14]. This paper presents the implementation of the Ada real-time features on GNATforLEON, an open-source cross-compilation system that implements Ravenscar tasking for LEON2 [10] targets. GNATforLEON is a port to LEON2 targets of GNAT Pro for ERC32 [17]. GNAT Pro for ERC32 and thus GNATforLEON uses a version of the GNAT run-time library (GNARL) specially developed to support the Ravenscar profile on top of a bare board kernel which is an evolved version of the Open Ravenscar Kernel (ORK) [5, 7].

2. The new Ada 2005 real-time mechanisms

The current revision of the Ada standard provides a range of real-time mechanisms that can be used to ensure the required temporal behaviour of real-time tasks. The Ada.Real_Time package includes a monotonic real-time Clock as well as a definition of Time which are appropriate for real-time systems. The package was already part of Ada 95 [12, Annex D] and can be used to check real-time related properties, such as minimum inter-arrival times or task deadlines. Real-time timers were not provided as such in Ada 95, but delay statements and asynchronous transfer of control (ATC) provided a similar functionality at a higher abstraction level (see e.g. [4]). However, ATC is excluded from the Ravenscar profile due to its complex implementation. Nevertheless, there are new real-time mechanisms which can be used to efficiently detect deadline overruns in critical systems.

Timing events [1] is an Ada 2005 lower-level mechanism that can be used with the Ravenscar profile [18, D.15] for detecting deadline overruns [14]. Timing events are a light-weight mechanism for specifying an action to be executed at a given time without the need to use a task or a delay statement. A timing event can be set to occur at an absolute time or after a real-time interval. A protected procedure handler is executed whenever the event occurs, unless it is cancelled before that time. The functionality

of timing events is provided by the library-level package Ada.Real_Time.Timing_Events, in this way it is no needed to change the compiler to implement this mechanism but just to add the support to the Ada run-time library as well as to the underlying kernel. It is worth noting that only library-level timing events are allowed by the Ravenscar profile.

Ada 2005 also includes mechanisms for measuring and monitoring execution-time, namely *execution-time clocks* and *timers* [2], and *group execution-time budgets* [3]. These mechanisms can be used to estimate the execution time of code segments, to handle some kinds of aperiodic events, and to detect execution-time related temporal faults. These mechanisms are also provided by library-level packages: Ada.Execution_Time, Ada.Execution_Time.Timers, and Ada.Execution_Time.Group_Budgets.

In Ada 2005 each task has an *execution-time clock* that computes the amount of CPU time it has consumed, including the run-time services invoked by the task. It should be noticed that it is implementation-defined which task is charged with the execution time for system services, which include interrupt service routines, or even whether it is charged to no task [18, D.14(13a/2)]. *Execution-time timers* are objects that are associated with a task —and hence with the task execution-time clock— when they are declared. A timer can be armed to expire at an absolute value of that clock or after some execution-time interval. When the timer expires, a protected procedure handler is executed. Setting again the handler replaces the handler and the time of execution and the timer remains set. *Group execution-time budgets* is a similar mechanism, which can be used with a set of tasks instead of a single task. A task can belong to at most one such group. A global budget of execution-time can be allocated to the whole group, and then it is decreased when any task in the group consumes execution time. As with timers, a protected procedure handler can be specified to be executed whenever the budget is exhausted. The budget can also be replenished at any time.

Execution-time clocks are allowed in the Ravenscar profile, but timers and group budgets are not. However, we believe that these mechanisms can be safely and efficiently used in high-integrity systems, provided that they are only declared at library level and there is at most one execution-time timer per task [6].

3. Kernel support for timing services

The described timing services has been implemented on GNATforLEON which is an evolved version of ORK for LEON2 based computers. LEON2 is a radiation-hardened implementation of the SPARC V8 architecture, which has been adopted by the European Space Agency (ESA) as the new standard processor for spacecraft on-board computer systems as an upgrade of the ERC32 [9].

GNATforLEON provides direct support for the Ravenscar profile [18, D.13.1], including the following Ada 2005 timing services:

- Global timing events;
- Execution-time clocks.

Execution-time timers and group budgets are also supported by the kernel in spite of being not allowed by the Ravenscar profile. These mechanisms are needed to enforce temporal separation in logical partitioned systems where subsystems with possibly different levels of criticality can share computer nodes. This is a strong requirement for the kind of on-board aerospace embedded systems envisaged in the ASSERT project[1] [15].

The implementation of Ada.Real_Time.Clock and absolute delays for ORK/ERC32 is thoroughly described in [21]. It is based on the two 32-bit hardware timers of the ERC32 processor. That implementation has been ported to the LEON2 processor which has two 24-bit hardware timers. It is worth noting that Annex D of the Ada Language Reference Manual [18] requirements for Ada.Real_Time.Time lead to at least 41 bits for that type. As a result, the implementation uses the hardware timer register as the least significant part (LSP) of the clock and a 32-bit word in memory as the most significant part (MSP). This arrangement provides an accurate tick with low overhead.

Execution-time clocks and timers were also supported by ORK/ERC32 whose implementation is described in [20]. The implementation only allows one execution-time timer per task, as suggested in previous IRTAW discussions [8, 6] and permitted by the Ada Language Reference Manual [18, D.14.1(28/2)]. Although execution-time timers are not allowed in the Ravenscar profile, the Ravenscar profile restrictions enable a simple and efficient implementation which was ported to GNATforLEON. Group budgets were not implemented in ORK but recently on GNATforLEON. However, the implementation is built on top of the execution-time timers one and thus only little support is needed for group budgets at kernel level. The main part is at Ada run-time level in the body of the Ada.Execution_Time.Group_Budgets whereas at kernel level is only needed a flag to record if the armed execution-time alarm of the task correspond to its execution-time timer or to the group budget timer at which the task belongs. In this way, the proper handler may be invoked if the execution-time alarm expires.

The overall implementation is schematically shown in figure 1. As said, timer 1 is used in periodic mode to sup-

[1] ASSERT (Automated proof based System and Software Engineering for Real-Time) is an FP6 Integrated Project coordinated by the European Space Agency. The main goal of the project is to improve the system-and-software development process for critical embedded real-time systems, in the Aerospace and Transportation domains.

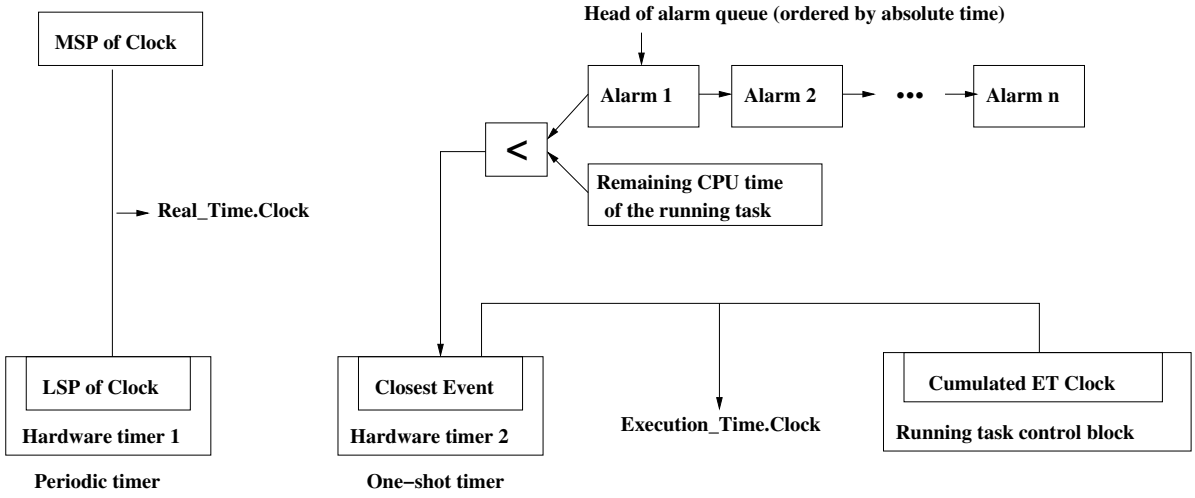

Figure 1. Schema of clocks and timers implementation.

port Ada.Real_Time.Clock and timer 2 is used in one-shot mode and is armed to expire with the closest event. This event can be an absolute delay or the execution-time timer of the running task, which in turn could be its own timer or the timer of its group budget. In a similar way to real-time clock, the execution-time clock of the running tasks is built up by using the hardware timer register and the cumulated execution time. However, if the timer 2 is armed with an absolute delay is more complex to build up the execution-time clock of the running tasks.

4. Implementation of timing-events

The Ravenscar profile restrictions avoids delay cancellation and therefore the alarm queue of figure 1 is simply linked. However timing-events can be cancelled and the alarm queue can not be efficiently used for this purpose. Nevertheless it is possible to manage timing-events in the same way of absolute delays, that is by inserting them in the simply linked alarm queue and if a timing event is cancelled its associate handler is set to null.

The overhead of the above approach could be intolerable in applications with timing event cancellations. It should be noticed that the processing of the timer interrupts implies the execution of the preamble and the epilogue together with the run-time alarm handler. This alarm handler has to clear the interrupt, identify the type of the event, jump to an Ada code which is the handler and finally look for the next closest event in order to arm the hardware timer. In our opinion, this is a pretty amount of instructions for a null handler.

As a result, it was decided to use a new doubly linked queue for timing events. Therefore, timing events can be simply and efficiently located and removed when they are cancelled. The timing event queue is also ordered by absolute time in spite of the timing event was set by using relative time, as it is the best approach [21].

The figure 2 shows the new queues arrangement. In general, it is more simple and efficient to maintain two queues than the combined one and thus the implementation has a lower overhead. It could be argued that it is needed to compare among three events in order to identify the closest event and therefore more comparisons are made in order to arm the hardware timer than without a new queue. This is a fallacy because the total number of comparisons is even lower because it is made much more comparisons for inserting the event in a longer queue.

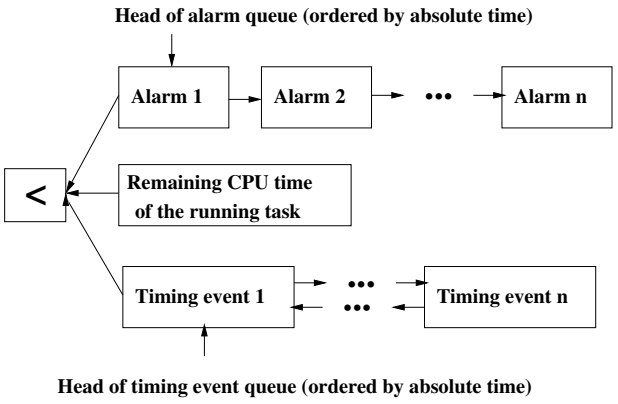

Figure 2. Schema of event queues.

Finally, it must be said that the implementation could be much more simple with a little bit of hardware support. For instance, the implementation of real-time clock and absolute delays of the ORK version for PC computers is much more efficient and simple because it takes advantage of the

Time Stamp Counter which can be found in Pentium processors. The Time Stamp Counter is an up-count 64-bit timer and thus it is able to maintain the monotonic real-time clock itself. This is fairly convenient making more simple to operate the alarm queue because reading the clock is just one instruction.

5. Implementation inaccuracies

The described implementation allocates the time spent in interrupt service routines to the currently running task, which is allowed by Ada. However, it allocates the time spent in timing event handlers too. It should be noticed that to stop and restart the execution-time clock of the running task is not easy because it is not just a matter of stopping the hardware timer. As it is shown in figure 2, the hardware timer 2 does not hold the remaining CPU time of the running task when a timing event expires but the expired timing event. As a result, the real-time clock should be recorded at the beginning and the end of the hardware timer 2 interrupt handler in order to properly updated the remaining CPU time of the running task by subtracting this elapsed time.

In this way, it is not only complex to avoid the allocation of the time spent in timing event handlers to the running task but the time spent for avoiding this could be greater than the time spent in the timing event handler itself. Indeed, it is a time-spending operation to read the clock with the little hardware support of LEON2 processors.

Nevertheless, we believe that the implementation is implicitly allowed by the standard because the Ada Language Reference Manual [18, D.15(25/2)] says that "The protected handler procedure should be executed directly by the real-time clock interrupt mechanism". As it is implementation-defined which task is charged execution time for the time spent in interrupt service routines, it can be concluded that it is also allowed to charge the time spent in timing event handlers. In our honest opinion, it should be clarified in an Ada Issue.

Another source of inaccuracy of execution-time clocks is the so-called *proxy model* for servicing the entries of protected objects which is used by the GNAT compiler [13]. With this approach the task exiting the eggshell executes all the waiting entry calls whose barriers are open on behalf the awaiting tasks and reevaluates the barriers every time. As a result, this time spent for other tasks is charged to the execution-time clock of the exiting task.

The Ravenscar profile only allows one awaiting task per protected object and therefore this inaccuracy can be bounded but as there is no language-imposed restriction on the number of such calls that can be pending, the inaccuracy could be intolerable for general Ada programs. A way to avoid this inaccuracy could be to use the so-called *self-service model*, although the number of context switchings would increase. Moreover, the GNAT compiler and the GNAT Ada run-time library should be modified in order to do that.

6. Metrics

The Ada 2005 timing mechanisms have been implemented by the authors on GNATforLEON, a compilation system for the LEON2 processor, a radiation-hardened derivative of the SPARCv8 RISC architecture for the space domain. The implementation has been based on a previous experimental implementation on top of the Open Ravenscar kernel [20]. The modified compilation system is being used as the execution platform for the ASSERT project.

The overhead of the new timing mechanisms (execution-time timers, group budgets, and timing events) has been measured by comparing footprint size and context switch duration between GNATforLEON 1.0 and GNATforLEON 1.3. GNATforLEON 1.0 is the first version of the compilation system which does not have the new timing mechanisms. Conversely GNATforLEON 1.3 includes all of them. The values shown in tables 1 and 2 have been measured using a pilot application, and therefore should be considered as average values, not as worst-case metrics.

Run-time system	**Context switch** (instructions)
GNATforLEON 1.0	405
GNATforLEON 1.3	606

Table 1. Context switch in GNATforLEON

Section	**Size** (kilobytes)	
	GNATforLEON 1.0	GNATforLEON 1.3
.text	79	87
.data	8	8
.bss	362	365
Total	449	460

Table 2. Memory footprint

The overhead is moderate as about 200 new instructions have to be executed per context switching to support execution time clock and group budget on GNATforLEON. The absolute timing impact depends highly on actual CPI (Cycles Per Instruction) which in turns depends on the status of pipeline, caches and register window. The ideal CPI is 1 and the clock frequency of LEON2 processor is 50 MHz therefore the minimum absolute overhead is 4 μs.

Table 3 shows the instructions required for timing service primitives. It should be noticed that 71 instructions are

just needed to read the clock and the implementation needs to read the clock during context switching and timing service primitives. As a result, the poor hardware support of LEON2 processors highly impact on the duration of timing service primitives.

Operation	Instructions
Real_Time.Clock	71
Timing_Events.Set_Handler	240
Execution_Time.Timers.Set_Handler	271

Table 3. Primitives

Table 4 shows the latencies for executing the corresponding handler when an execution time or group budget timer expires. 400 instructions are needed from the first instruction of the low-level interrupt handler to the first instruction of the Ada handler. It is fairly low providing that LEON2 is a RISC processor.

Operation	Instructions
Timing event handler	396
Execution-time handler	415

Table 4. Handler latencies

The footprint increases in 11 kbytes which are mainly due to the 8 kbytes augment in the code (text section). The other 3 kbytes are due to the need of larger ATCB and structures for individual objects. Table 5 shows the footprint of the required structures.

Type	Size (bytes)
Timing_Event	24
Execution_Time.Timers.Timer	20
Group_Budget	2064

Table 5. Memory size

7. Hardware support

The overhead introduced by real-time mechanisms in the kernel primitives is moderate. However, as said above, 71 instructions are needed to read the clock and the clock must be read to obtain the relative down-count that should be loaded in the hardware register. In this way, the fourth part of the extra instructions in a context switching are used just for reading the clock. There are also another operations which are time consuming such as to compare 64-bit time values and to convert a relative time in the corresponding value that should be load in the down-count timer register.

As a result, a significant part of the introduced overhead can be avoided with just a little bit of hardware support. Hardware timers are fairly simple devices and they can be included in a processor board at a very low cost. It can be envisage a very simple implementation of the described real-time mechanisms just with four 64-bit up-count hardware timers.

In this way, it can be used one timer to support the monotonic real-time clock on hardware, as with the Pentium Time Stamp Counter, without any software support. A second one can be used for the absolute delay queue which is ordered by absolute time and thus the absolute expiration time would be loaded in the so-called *c*omparator value register of the hardware timer. Therefore, an interrupt request should be delivered when the up-count timer reaches the comparator value. A separate timer can be used for timing events which can be managed with the same approach although the timing event queue should be doubly linked. The last timer is dedicated to count for the execution time of the running task, in this way it would be easy read the execution-time clock of the running task. Moreover, to stop and restart this timer would be the needed simple operations to avoid charging the time spent in interrupt service routines and timing events to the running task.

Recently, Intel has specified the so-called High Precision Event Timers [11] for the PC architecture. The specification defines a block of up-count 64-bit timers and each timer can be configured to generate a separate interrupt. The specification allows for a block of 32 timers, with support for up to 8 blocks, which allows a total of 256 timers.

The specification fulfils the requirements to implement the real-time mechanism with a low overhead because timers are implemented as a single up-counter with a set of comparators. Each timer includes a match register and a comparator, and can generate an interrupt when the value in its match register equals the value of the free-running counter. Moreover, the counters increases monotonically and some of the timers can be enabled to generate a periodic interrupt.

It can be easily envisage a very simple implementation of the real-time services with such population of timers. Every timing event could use its own hardware timer and thus queueing is avoided. However, it should be needed to limit the maximum number of timing events with the corresponding pragma Restrictions. In a similar way, every task could own a hardware timer in order to support its execution-time clock and timer. As a result, the overhead in context switching would be reduced to stop and restart the corresponding timers of both tasks. It should be noticed that the maximum number of tasks can be limited by a pragma Restrictions and the implementation may limit the number of timers that can be defined for each task to one, and thus this implementation is allowed by the standard.

Finally, it could be possible to use a periodic timer to activate each periodic task and to eliminate the alarm queue

too. Unfortunately, Ada has not a way of specifying the period of a real-time periodic task and it would be needed to add this feature by a specific implementation pragma.

8. Conclusions

The Ada 2005 real-time services are of paramount importance for detecting temporal faults and thus they enable the development of fault tolerant systems. The implementation described in this paper has a moderate overhead for a Ravenscar kernel and does not introduce much complexity to the underlying kernel. Therefore, they can be used for building high integrity systems.

It should be noticed that the hardware timer devices of LEON2 processor are not adequate to support the real-time features which are needed in a real-time system. Even the monotonic real-time clock needs a significant software support. We believe that the overhead can be highly reduced with a little bit of hardware support which can be found in the Intel PC architecture.

Additionally, some inaccuracies in the implementation of execution-time timers are derived from this poor hardware support, as well as due to the proxy model. These inaccuracies can not be completely avoided in the general case and implementation advices should be provided.

References

[1] Ada Rapporteur Group. Ada Issue 297 — Timing events. *Ada Letters*, XXV(3), September 2005.

[2] Ada Rapporteur Group. Ada Issue 307 — Execution-time clocks. *Ada Letters*, XXVI(1), April 2006.

[3] Ada Rapporteur Group. Ada Issue 354 — Group execution-time budgets. *Ada Letters*, XXVI(2), August 2006.

[4] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, 3 edition, 2001.

[5] J. A. de la Puente, J. F. Ruiz, and J. Zamorano. An open Ravenscar real-time kernel for GNAT. In H. B. Keller and E. Plöedereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

[6] J. A. de la Puente and J. Zamorano. Execution-time clocks and Ravenscar kernels. *Ada Letters*, XXIII(4):82–86, December 2003. Proceedings of the 12th International Ada Real-Time Workshop (IRTAW12).

[7] J. A. de la Puente, J. Zamorano, J. F. Ruiz, R. Fernández, and R. García. The design and implementation of the Open Ravenscar Kernel. *Ada Letters*, XXI(1), 2001.

[8] B. Dobbing and J. A. de la Puente. Session report: Status and future of the Ravenscar profile. *Ada Letters*, XXIII(4):55–57, December 2003. Proceedings of the 12th International Real-Time Ada Workshop (IRTAW 12).

[9] ESA. *32 Bit Microprocessor and Computer System Development*, 1992. Report 9848/92/NL/FM.

[10] GR. *LEON2 Processor User's Manual*, 2005. Gaisler Research.

[11] Intel Corporation. *IA-PC HPET (High Precision Event Timers) Specification*, 2004. Intel Corporation.

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

[13] J. Miranda. *A Detailed Description of the GNU Ada Run Time*. http://www.iuma.ulpgc.es/users/jmiranda/gnat-rts/, 2003.

[14] J. A. Pulido, S. Urueña, J. Zamorano, and J. A. de la Puente. Handling temporal faults in Ada 2005. In N. Abdennadher and F. Kordon, editors, *Reliable Software Technologies — Ada-Europe 2007*, number 4498 in LNCS, pages 15–28. Springer-Verlag, 2007.

[15] J. A. Pulido, S. Urueña, J. Zamorano, T. Vardanega, and J. A. de la Puente. Hierarchical scheduling with Ada 2005. In L. M. Pinho and M. González Harbour, editors, *Reliable Software Technologies — Ada-Europe 2006*, volume 4006 of *LNCS*. Springer Berlin / Heidelberg, 2006.

[16] P. Puschner and A. Burns. A review of worst-case execution time analysis. *Real-Time Systems*, 18(2/3):115–128, May 2000.

[17] J. F. Ruiz. GNAT Pro for on-board mission-critical space applications. In T. Vardanega and A. Wellings, editors, *Reliable Software Technologies — Ada-Europe 2005*, volume 3555 of *LNCS*. Springer-Verlag, 2005.

[18] S. T. Taft, R. A. Duff, R. L. Brukardt, E. Plöedereder, and P. Leroy, editors. *Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652/1995(E) with Technical Corrigendum 1 and Amendment 1*. Number 4348 in Lecture Notes in Computer Science. Springer-Verlag, 2006.

[19] T. Vardanega. Development of on-board embedded real-time systems: An engineering approach. Technical Report ESA STR-260, European Space Agency, 1999.

[20] J. Zamorano, A. Alonso, J. A. Pulido, and J. A. de la Puente. Implementing execution-time clocks for the Ada Ravenscar profile. In A. Llamosí and A. Strohmeier, editors, *Reliable Software Technologies — Ada-Europe 2004*, volume 3063 of *LNCS*. Springer-Verlag, 2004.

[21] J. Zamorano, J. F. Ruiz, and J. A. de la Puente. Implementing Ada.Real_Time.Clock and absolute delays in real-time kernels. In A. Strohmeier and D. Craeynest, editors, *Reliable Software Technologies — Ada-Europe 2001*, number 2043 in LNCS, pages 317–327. Springer-Verlag, 2001.