Analysis of WCET in an experimental satellite software development*

Jorge Garrido, Daniel Brosnan, Juan A. de la Puente, Alejandro Alonso, and Juan Zamorano

Real-Time Systems group (STRAST)
Universidad Politécnica de Madrid (UPM), Spain
`str@dit.upm.es`

Abstract

This paper describes a case study in WCET analysis of an on-board spacecraft software system. The attitude control system of UPMSat-2, an experimental micro-satellite which is scheduled to be launched in 2013, is used for an experiment on analysing the worst-case execution time of code automatically generated from a Simulink model. In order to properly test the code, a hardware-in-the-loop configuration with a simulation model of the spacecraft environment has been used as a test bench. The code has been analysed with RapiTime, with some modifications to the original instrumentation routines, in order to take into account the particularities of the test configuration. Results from the experiment are described and commented in the paper.

1998 ACM Subject Classification C.3 Real-time and embedded systems

Keywords and phrases Real-time systems, embedded systems, timing analysis, WCET calculation

Digital Object Identifier 10.4230/OASIcs.WCET.2012.81

1 Introduction

UPMSat-2 is a project aimed at developing an experimental micro-satellite that can be used as a technology demonstrator for several research groups at UPM, the Technical University of Madrid. The Real-Time Systems Group at UPM (STRAST)[1] is responsible for designing and building all the on-board and ground-segment software for the satellite. The software is being coded in Ada with the Ravenscar profile tasking restrictions [4], and runs on a LEON3 [9] computer board. The GNATforLEON compilation chain [14], including the Open Ravenscar Real-time kernel (ORK) [5], is being used for software development.

Software standards for on-board spacecraft software [8, 7] require schedulability analysis to be used in the verification process. This kind of analysis, in turn, requires the worst case execution time (WCET) of each task to be known. Therefore, WCET measuring methods and tools [16] must be used as a first step in performing timing analysis on embedded on-board systems.

This paper describes the approach that the authors have taken to calculate the WCET of the UPMSat-2 Attitude Determination and Control System (ADCS) subsystem. The ADCS functional code has been automatically generated from a Simulink[2] engineering model, and

* This work has been partly funded by the Spanish Ministry of Economy and Competitiveness (MINECO), project TIN2011-28567-C03-01 (HI-PARTES).
[1] `www.dit.upm.es/str`
[2] *Simulink* is a registered trade mark of The MathWorks Inc.

then integrated with concurrent, real-time container code following a model-driven approach. The WCET of the resulting code has been analysed using RapiTime[3], a well-known tool for hard real-time systems analysis.

The rest of the paper is organised as follows: Section 2 describes the ADCS subsystem and its relationship to other components of the UPMSat-2 software. Section 3 presents the methodological approach to WCET analysis and the details of the analysis process. Section 4 summarizes the results obtained so far. Finally, section 5 presents the conclusions of the analysis and some ideas for future work.

2 System description

2.1 Hardware platform

All the computer-related functions on board of the UPMSat-2 satellite will be executed on a single computer platform, called the On-Board Computer (OBC) [6]. The OBC hardware will be based on a LEON computer with SRAM main memory and a solid state disk (SDD), as well as a number of peripherals for interaction with the satellite sensors and actuators.

The LEON family of processors[4] is a 32-bit synthesizable VHDL processor core that implements the SPARC V8 architecture [15]. The flight version of the OBC, which will be based on LEON3, is still under development. For this reason, an engineering model has been used for this experiment. The engineering model is based on a GR-XC3S1500 Spartan3 development board[5] with a LEON2 processor at 40 MHz clock frequency and 64 MB of SDRAM. Cache memory is not used in this implementation. The main difference between the LEON2 processor used in the engineering model and the envisaged production LEON3 are that the latter has a 7-stage pipeline instead of the 5-stage pipeline of LEON2. Other differences, such as the presence of an MMU and SMP support in LEON3, are not significant as these features are not used in this project. Since the purpose of this work is to validate the WCET calculation methodology, it can be assumed that the engineering model is a representative instance of the flight computer.

2.2 Software architecture

The on-board software system is composed of the following subsystems (figure 1):

- *Platform monitoring*: this subsystem is in charge of assessing the state of the satellite by periodically reading housekeeping sensor data (battery and solar panel voltages and currents, temperatures at various points of the spacecraft, etc.)
- *Telemetry and telecommand (TMTC)*: this subsystem interacts with the telecommunications hardware (TMC) in order to send messages (telemetry) to a ground-based station and receive telecommands from it.
- *Attitude determination and control system (ADCS)*: this subsystem computes the orientation (attitude) of the satellite with respect to the Earth, and takes corrective actions in order to keep it within the specified values. The attitude is derived from 3-axis measurements of the Earth magnetic field made with special sensors called *magnetometers* (MM). Control actions are performed by means of *magnetorquers* (MT), which are electromagnets

[3] `http://www.rapitasystems.com/rapitime`
[4] `http://www.gaisler.com/leonmain.html`
[5] `http://www.pender.ch/products_xc3s.shtml`

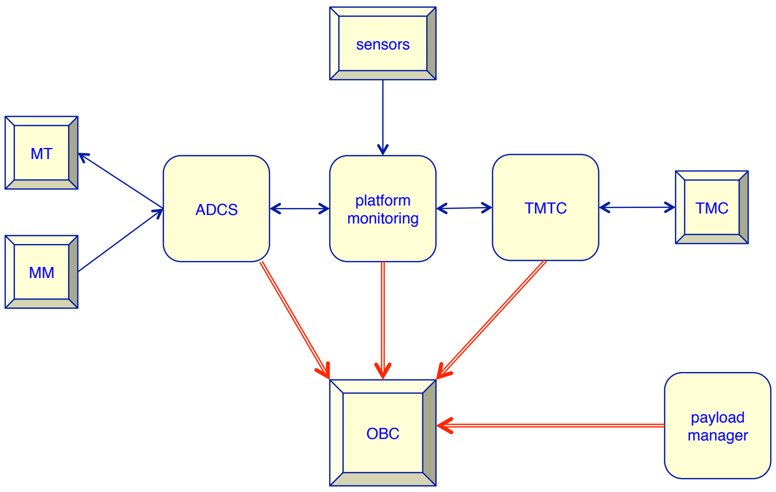


Figure 1 UPMSat-2 software architecture.

that can develop a magnetic field that is used to change the attitude of the satellite as needed.

In addition to the above main subsystems, there is another software component which manages the operation of the satellite payload. In this case, the payload consists of a series of experiments to be performed when ordered by specific telecommands.

The code for the UPMSat-2 software is written in Ada [1], and is being produced using a model-driven approach which was first developed in a previous project [11]. Under this approach, a software system is made of a number of components with well-defined interfaces. The internals of a component have three main parts, which implement separation of concerns between the functional, concurrent, and synchronisation aspects of the component. In particular, functional code generated with high-level modelling tools such as Simulink can be incorporated in the functional part of a component, leaving the complexity of concurrency, timing and synchronization to the other parts (figure 2). Concurrency and synchronization are implemented by Ada tasks as restricted by the Ravenscar profile, i.e. the number of tasks if fixed, the scheduling method is fixed-priority pre-emptive scheduling (FPPS), and communication among tasks is limited to shared data objects accessed with an immediate ceiling priority protocol (ICPP) [4]. The concurrent constituents of components are automatically generated from a reduced set of archetypes [3, 12]. Since the code of the concurrent elements is derived from a few simple patterns, the key issue for analysing the timing behaviour of the system is to get accurate estimates of the WCET of the functional code. This code can be rather complex, depending on the particular functions that are executed by each component. In the following, we will centre on the functional code of the ADCS subsystem as a representative example of the issues involved.

2.3 The Attitude Determination and Control System (ADCS)

The attitude of the satellite is represented by three angles measuring the orientation of the spacecraft with respect to an Earth-based reference frame. The attitude angles are computed from the readings of three magnetometers, which measure the intensity of the Earth magnetic field on the platform reference axes. There are also three magnetorquers that are used as actuators in order to adjust the attitude to the required angle values.

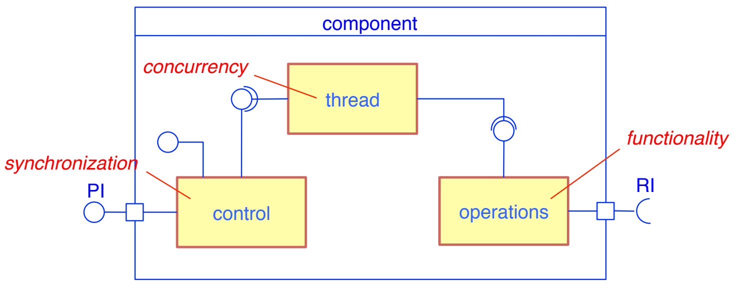


Figure 2 Structure of a software component.

The angular orientation of the satellite may change due to perturbations originated at the environment, mainly by the fluctuation of the Earth magnetic field, but also by the drag of residual atmosphere and the solar radiation pressure. There are three magnetorquers that are used as actuators in order to adjust the attitude to the required angle values in the presence of perturbations. A control algorithm is used to compute the appropriate values for the magnetorquers intensity signals.

The attitude control algorithm is designed by aerospace engineers based on a mathematical model of the spacecraft dynamics and the torque perturbations. Due to the complexity of the model, a simulation model has been created using Simulink in order to design, test and validate the structure of the control algorithm and to tune its parameters to the most appropriate values.

Figure 3 shows the general structure of the simulation model. The blocks represent the spacecraft dynamics and the environment elements that are relevant for the design, including the Earth magnetic field, environmental perturbations, and the changes in the gravitational acceleration. The variables in the model represent different torques acting on the spacecraft, the quaternions representing the attitude of the satellite with respect to a vertical reference frame, the rotation matrix from the vertical reference frame to the satellite reference frame ($\mathbf{C}_{\mathrm{BV}}$), and the angular velocity of the body with respect to the vertical reference frame ($\mathbf{pqr}$).

The block labelled "controller" represents the control algorithm, which has as inputs the measurements of the magnetic field provided by the magnetometers ($\mathbf{B}_{\mathrm{b\_T}}$) and its derivatives ($\mathbf{B}_{\mathrm{dot}}$). The outputs of the control block are the intensity supplied to the magnetorquers ($\mathbf{I}_{\mathrm{A}}$) and the corresponding actuator torques ($\mathbf{T}_{\mathrm{Nm}}$). It must be noted that the sensor and actuator models are included in the controller model.

The functional code of the attitude controller is automatically generated using the Simulink code generation tools. In order to do this, and to properly model the interface elements as well, a discrete-time model of the attitude controller is required. While a discrete-time model could be obtained from a continuous-time controller model, a direct approach using discrete-time blocks in the whole model has been followed. A sampling frequency of 1Hz has been used for this purpose, based on the control engineers' knowledge of the spacecraft dynamics.

The current version of the controller functional code is comparatively simple, with no loops or conditional control structures. It has about 50 lines of C code, which are integrated with the concurrent and control constituents of the ADCS subsystem by means of the Ada interface facilities. The code implements the well-known PID (proportional-integral-derivative) control algorithm [2]. Further developments in the control algorithm are expected to produce longer and more complex code.

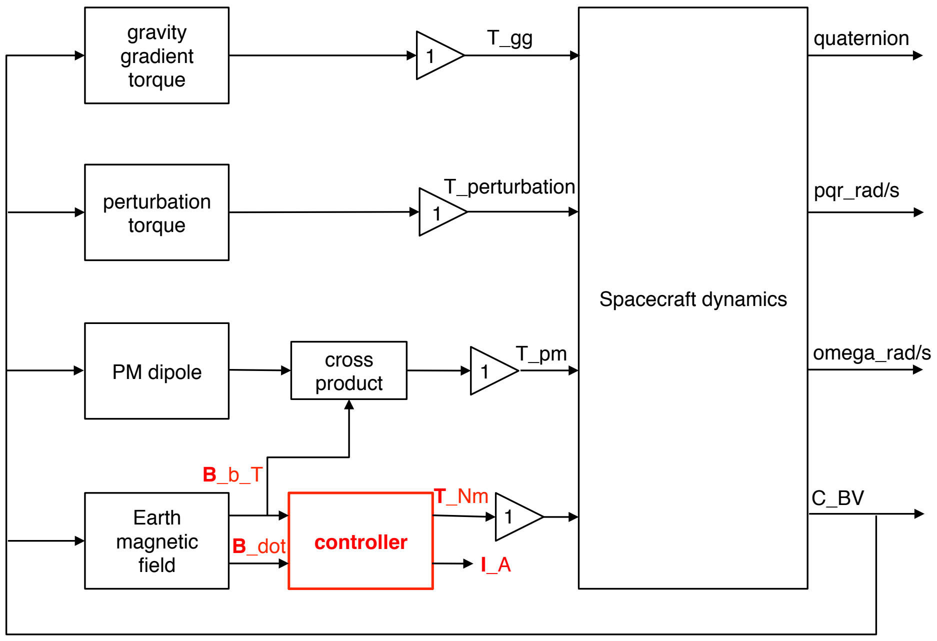


Figure 3 ADCS simulation model.

3 Analysis of the ADCS execution time

3.1 Object of the analysis

The code that has been analysed is the ADCS functional code automatically generated by the above described simulation model. While using a purely static code analysis could be feasible, a measurement-based approach has been used in order to get accurate estimates on the WCET on real hardware [16]. The hardware platform is the engineering version of the OBC that was introduced in section 2.1. As it has already been discussed, this platform is representative of the final flight computer hardware which is still to be built.

An important aspect of the analysis is the software context in which the code runs. In this work we have chosen to execute the ADCS functional code on its own, with only the underlying ORK kernel in place. The only active interrupt source is the system clock interrupt, which has a fairly low frequency, about 0.59Hz. The execution of the interrupt handler would only add a few tens of microseconds to the measured value of execution time in the rare case that the interrupt occurs while the ADCS task is executing.

It could be argued that measuring the execution time in isolation does not account for cache interference and other effects that may result from preemption in a concurrent execution environment. Since the current configuration of the OBC does not include a cache, we can still consider that executing the code in isolation provides realistic measurements of the execution time, at least for the engineering model. Further development of the flight version of the OBC are open to using the LEON3 cache, though, which should be taken into account in future measurements. Another question is the possible effects of control flow breaks on the processor pipeline. The SPARC V8 architecture does not perform branch prediction, using delayed branching instead. Therefore, in this case such effects can be safely ignored.

3.2 Test environment

As it frequently happens in real-time embedded system projects, it is not possible to test the satellite software in its real environment. Hardware-in-the-loop (HIL) techniques provide a useful testing framework in such cases. The basic idea is to test the embedded system against a simulation model instead of the real environment. The test environment is built from the simulation model depicted in figure 3 above, replacing the controller block by the real computer and controller software. Some additional components have been included as well, in order to model the sensors and actuators that carry out the interaction between the computer and the modelled environment.

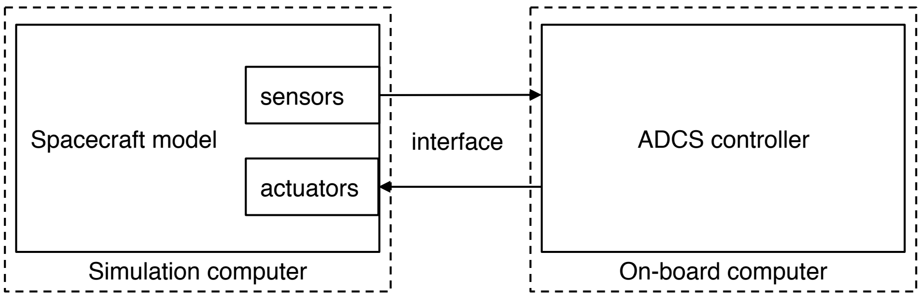


Figure 4 ADCS test configuration.

Figure 4 shows the architecture of the ADCS test configuration. The simulation model is implemented in Simulink on a PC-based platform, whereas the system under test is the ADCS code running on an engineering version of the satellite computer board. The interface between them is built on a serial line during the first stages of the development cycle, to be replaced with actual analog lines as the OBC implementation is advanced enough. This interface is modelled by replacing the original controller model (controller block in figure 3) with two new blocks: one than sends magnetic field data from the simulation model to the computer by means of the serial line, and another one that receives actuator data from the computer and inputs them to the attitude simulation block.

3.3 WCET measurement

We have chosen RapiTime[6] for WCET measurements as we are interested in on-target execution time measurements, using the above described test configuration. The version used is RVS 3.0.

RapiTime collects execution traces to generate time measurement statistics, among which worst case execution time measurements as required for schedulability analysis. In order to do this, RapiTime analyses the structure of the source code and adds instrumentation points at relevant places. The instrumentation points are calls to a procedure that records the execution time at which the point is reached. The instrumented code is repeatedly executed long enough to acquire relevant statistical information about the executions. Once this is done, the generated trace is checked for the correct format and compliance with previously extracted structural information, and a statistics report is generated.

In our case, the ADCS functional code has been instrumented with the RapiTime tools. In a first step, only the auto-generated controller code has been analysed, leaving out the

[6] `http://www.rapitasystems.com`

Listing 1 Instrumentation procedure

```
procedure Ipoint ( Id : Natural ) is
begin
   trace_buffer(trace_idx).ident := Id ;
   trace_buffer(trace_idx).timestamp := Ada.Real_Time.Clock ;
   trace_idx := trace_idx + 1;
end Ipoint;
```

interface code that is required for reading the magnetometers, which is simulated in the test platform.

The default setup for the RapiTime instrumentation, in which the trace is output to a serial line during the execution, is not suitable for our test platform, due to the bias that the serial communication would add to the measurement. Therefore, we changed the instrumentation code so that it writes the traces to an array instead. The trace elements are pairs of system time values and instrumentation point identifiers. The `Ada.Real_Time.Clock` function provided by ORK was used for system time measurements. The clock granularity is 100ns, and the time type representation is 64 bits long. RapiTime uses natural number values as instrumentation point identifiers, which were mapped to 32 bit Ada integers. Consequently, trace values use 12 bytes for each instrumentation point. The trace array is stored in the target computer memory, and later extracted to the developer workstation with the GRMON[7] debugging tool. Listing 1 shows the code of the instrumentation procedure.

In order to have enough data for the results to be relevant, the simulation should cover at least one complete orbit. The reason for it is the need to cover different values of the Earth magnetic field at different orbital positions. Since the devised orbital period for the satellite is about 1.64 hours, 5930 iterations of the control method have to be executed to cover a full orbit at the specified 1Hz rate. The size of the trace array, considering that there are two instrumentation points for each iteration, is thus 140 KB, which can be easily stored in the 64 MB RAM of the target computer.

The final step is to process the data on the development workstation. The results are described in the next section.

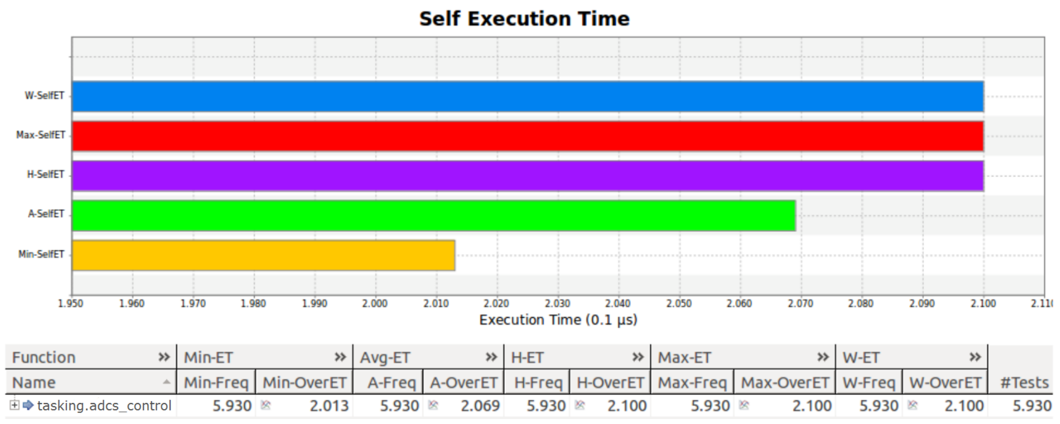
4 Results

In addition to WCET measurements, RapiTime generates a large amount of data about the ADCS code. The most relevant data refer to code coverage. RapiTime reports the number of times that each instrumentation point has been reached, which can be easily used to derive the execution count for every possible execution path.

For our purposes, the main results refer to statistics of time measurements. Figure 5 shows the minimum (Min-ET), average (A-ET), high water mark (H-ET), and maximum (Max-ET) execution time registered for the ADCS controller function, together with an estimate of the worst case execution time (W-ET) [13]. The term “Self” in the figure means that subprogram calls are not included, which is all right in this case as the code user test does not make any such calls. WCET estimation, which does not necessarily match the worst execution time registered in the experiments, is one of the features that make this tool

[7] `http://www.gaisler.com/`

really powerful for systems validation. “Over” refers to measurements including function calls. “Freq” refers to the number of times the code has been executed during the test.



Function	Min-ET		Avg-ET		H-ET		Max-ET		W-ET		
Name	Min-Freq	Min-OverET	A-Freq	A-OverET	H-Freq	H-OverET	Max-Freq	Max-OverET	W-Freq	W-OverET	#Tests
tasking.adcs_control	5.930	2.013	5.930	2.069	5.930	2.100	5.930	2.100	5.930	2.100	5.930

Figure 5 Execution time results for the ADCS control function.

In our case, the WCET estimate has a value of 2100 (210μs). Since the controller task period is 1s, the processor utilization is 0.21%, which is a fairly low value leaving time for more complex control algorithms if needed, and for the rest of the subsystems as well.

As can be seen from figures 5 and 6, the difference between the minimum and the worst case execution time is just 8.7μs, and the mode is 207μs with a probability of 0.147. With such a small difference, the distribution spans a very narrow range. However, the most interesting point is that, although the actual WCET might be greater than the estimated value of 210μs, the probability of such a situation is very low, $1.69 \cdot 10^{-4}$. We can take this result as an indication that we can safely use the estimated WCET for response time analysis.

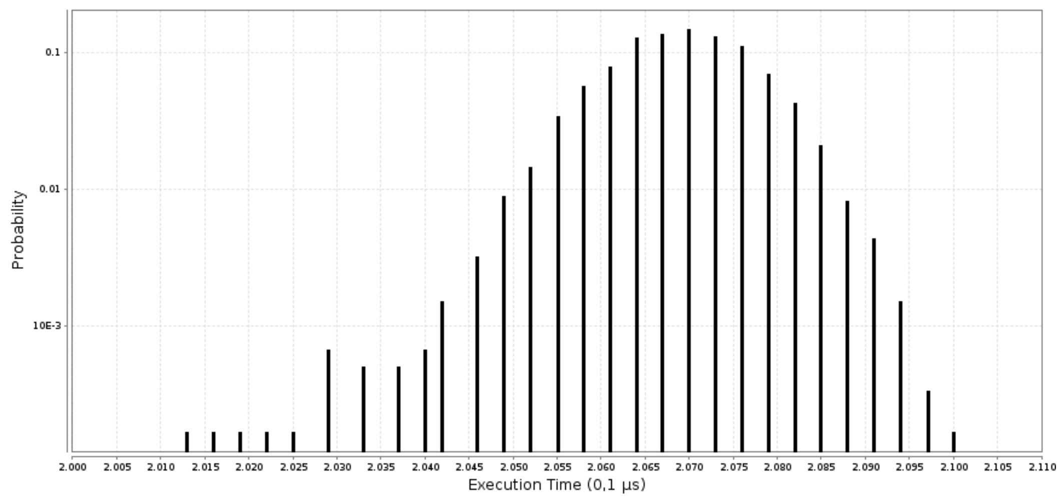


Figure 6 Probability distribution of execution time.

5 Conclusions and future work

The UPMSat-2 project provides an excellent testbed for many of the results obtained by UPM researchers over the recent years. In spite of being an experimental satellite, it is still a relevant project in terms of size and complexity. In particular, using the ORK technology developed by the real-time systems group, together with timing analysis methods such as implemented by RapiTime and other tools, provides an opportunity to validate the technology on a real satellite mission and consolidate our approach to embedded software development.

Hardware in the loop, supported by RapiTime and Simulink tools, has proved to be a successful approach to WCET analysis in terms of time, cost and results. Accordingly, it will be one of the techniques used for the final validation of the attitude determination and control system of the UPMSat-2, as well other subsystems of the spacecraft on-board software. Although the current implementation of the controller function is very simple and consequently does not require some of the advanced WCET techniques implemented by the tool, the experiment has shown that the testing infrastructure is valid and can be used to analyse the execution time of other, more complex functions as the development advances.

Future work includes adding real sensors and actuators to the test environment. This will allow us to get real and complete timing measurements on the whole system and not just the auto-generated code. The next step is to perform measurements with all the OBC subsystems in place, and use response time analysis tools to estimate the time behaviour of the full system, including interference among different tasks and blocking due to access to common resources. This may require optimizing the instrumentation routine in order to get a more efficient implementation of measurements.

The flight version of the OBC will include a LEON3 processor with cache memory. The issue of the effects of multitasking on cache affinity will have to be further investigated. A possible way to improve timing predictability may be freezing the cache during the execution of interrupt handlers, so that the interrupted task preserves its cache affinity, at the possible cost of increasing the execution time of interrupt handlers. It is also possible to disable the cache during the execution of critical application tasks, making the cache control registers part of the task context. This technique has been implemented in an extension to ORK that has been used in a related project [10].

On-going work also includes higher-level modelling of the on-board software using some of the tools developed in the CHESS project.[8]

Acknowledgements

The authors wish to acknowledge the collaboration of the rest of the UPMSat-2 team, especially Assal Farrahi and Javier Cubas from the “Ignacio Da Riva” institute (IDR/UPM).

References

1 *ISO/IEC 8652:1995(E)/TC1(2000)/AMD1(2007): Information Technology — Programming Languages — Ada.*

2 Karl Johan Åström and Tore Hägglund. *Advanced PID Control.* ISA - The Instrumentation, Systems, and Automation Society, Research Triangle Park, NC 27709, 2005.

[8] CHESS (Composition with Guarantees for High-integrity Embedded Software Components Assembly) is an 7FP project funded under the Artemis JTU.

3 Matteo Bordin and Tullio Vardanega. Automated model-based generation of Ravenscar-compliant source code. In *Proc. 17th Euromicro Conference on Real-Time System*, ECRTS '05, pages 59–67, Washington, DC, USA, 2005. IEEE Computer Society.

4 Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the Ada Ravenscar profile in high integrity systems. *Ada Letters*, XXIV:1–74, June 2004.

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

6 Juan A. de la Puente, Juan Zamorano, Alejandro Alonso, and Daniel Brosnan. A real-time computer control platform for an experimental satellite. In *Jornadas de Tiempo Real — JTR-2012*, 2012. Available at `http://www.ctr.unican.es/jtr12/programme.html`.

7 European Cooperation for Space Standardization. *ECSS-E-ST-40C Space engineering — Software*, March 2009. Available from ESA.

8 European Cooperation for Space Standardization. *ECSS-Q-ST-80C Space Product Assurance — Software Product Assurance*, March 2009. Available from ESA.

9 Gaisler Research. *LEON3 Product Sheet*, 2008.

10 Enrico Mezzetti, Adam Betts, José Ruiz, and Tullio Vardanega. Cache-aware development of high-integrity systems. In Jorge Real and Tullio Vardanega, editors, *Reliable Software Technologiey –– Ada-Europe 2010*, volume 6106 of *Lecture Notes in Computer Science*, pages 139–152. Springer Berlin / Heidelberg, 2010.

11 Marco Panunzio and Tullio Vardanega. A component model for on-board software applications. In *36th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2010*, pages 57–64, 2010.

12 José Pulido, Juan A. de la Puente, Matteo Bordin, Tullio Vardanega, and Jérôme Hugues. Ada 2005 code patterns for metamodel-based code generation. *Ada Letters*, XXVII(2):53–58, August 2007. Proceedings of the 13th International Ada Real-Time Workshop (IRTAW13).

13 Rapita Systems Ltd. *RVS Reference Guide*, 2011. Version 3.0.

14 José F. Ruiz. GNAT Pro for on-board mission-critical space applications. In Tullio Vardanega and Andy Wellings, editors, *Reliable Software Technologies — Ada-Europe 2005*, volume 3555 of *LNCS*. Springer-Verlag, 2005.

15 SPARC International, Upper Saddle River, NJ, USA. *The SPARC architecture manual: Version 8*, 1992.

16 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–53, 2008.