A Performance Analysis of a Hard Real-time System

C.M. Bailey
British Aerospace, Space Systems Limited, Stevenage, UK

A. Burns and A.J. Wellings
Real-time Systems Research Group, Department of Computer Science
University of York, UK.

C.H. Forsyth
York Software Engineering, Heslington, York

Abstract

This paper analyses the performance of an Attitude and Orbital Control System for the Olympus Satellite. The system was designed using HRT-HOOD and implemented in Ada 83 augmented with some of the facilities of Ada 9X. The program consisted of multiple tasks communicating via protected shared data areas. A worst case execution time analysis tool was used to estimate the execution time of each task, and Deadline Monotonic schedulability analysis was used to predict the system performance, taking into account all kernel overheads. An instrumented version was also produced to gather statistics on the actual run-time performance.

1. Introduction

Many current industrial real-time systems use simple scheduling techniques based on cyclic schedulers. These, unfortunately, can require a considerable amount of effort to ensure that the scheduling load is balanced — an apparently simple programming change can result in an inordinate amount of readjustment of minor and major schedules. Currently many academic research groups claim that pre-emptive fixed priority process scheduling is now a mature enough technology to use in the implementation of new real-time systems. Although the arguments in favour of process-based scheduling are convincing [Locke1992], industry still remains sceptical about both the predictability of the resulting systems and the degree of inaccuracy associated with the current analysis techniques and tools.

In January 1991, the European Space Agency commissioned a study into the practicality of using process-based scheduling techniques in an on-board application environment, with Ada as the implementation language. We have reported our experiences of using these techniques in a realistic industrial application, the Attitude and Orbital Control System for the Olympus Satellite, elsewhere [Bailey1993]. In this paper we present a performance evaluation of the actual system and compare the results to that predicted by the analysis tools.

The structure of the paper is as follows. In section 2 we define the computational model that is appropriate for our application area, and the representation of that model in our design method, HRT-HOOD [Burns1994] and the implementation language, Ada [Defense1983]. In section 3, we briefly describe the tool support that we have for analysing systems developed according to the computational model. In section 4 we describes the requirements and the design of Olympus Satellite case study, then in section 5 and 6 we present a performance evaluation of the actual system. Finally, in section 7 we present our overall conclusions.

2. The Computational Model

Before deciding on a scheduling approach, a broad survey of current and future space applications was carried out to identify their typical characteristics [Bailey1991]. This indicated that:

- the process mix is predominately periodic (with well defined periods) but with a significant aperiodic presence
- aperiodic/sporadic sources generally represent external sources such as telemetry, telecommands and sensors, and all have known minimum arrival intervals
- the environment is typically deterministic in the sense that the number and types of objects to be monitored, controlled or analysed is fixed during early design, and the software concerned all has theoretically predictable worst case execution times
- both hard and soft timing constraints exist.

Given the above characteristics it was decided that pre-emptive priority based scheduling was the most appropriate technology to use, and that Deadline Monotonic Scheduling Analysis [Audsley1993, Leung1982] was more suitable than the usual Rate Monotonic Analysis [Liu1973] because:

- it allows the deadline of a process to be less than its period
- it allows sporadic processes to be more easily modelled
- it allows for better control over input and output jitter
- it allows kernel overheads to be easily incorporated.

The Attitude and Orbital Control System (AOCS) was designed using HRT-HOOD, an object oriented design method that has explicit abstractions to support the common hard real-time design paradigms. HRT-HOOD recognises the following object types:

- passive — objects which have no concurrency and no synchronisation constraints
- active — general objects modelling concurrent entities
- cyclic — objects which represent periodic activities
- sporadic — objects which represent aperiodic, or sporadic, activities
- protected — objects which control access to resources
- class — generic objects

Objects are described by their operations, their threads of control, their synchronisation with other objects, and their real-time attributes (e.g. deadline, period, offset, worst case execution time, criticality). For a full description of HRT-HOOD, the reader is referred to the literature. [Burns1994, Burns1992a, Burns1992b].

Although Ada 83 has been widely criticised for its lack of support for real-time, the limitations are well understood [Sha1990, Burns1990a, Burns1990b]and extensive changes have been made in Ada 9X to make the language more responsive to the needs of the real-time community [Ada 9X Mapping/Revision Team1993]. The approach adopted by the project has been to use a restricted subset of Ada (e.g., no dynamic tasking, no dynamic memory allocation, no recursion) and to extend an Ada Compiler and its run-time system to support: [Burns1993a]

- a large priority range (64 levels of software priority)
- passive tasks (with the semantics of Ada 9X protected types); a passive task is one which is optimised so that it has no thread of execution; it is used to implemented a protected object.
- a "delay until" function (to allow a more accurate representation of a periodic task)
- monotonic time (to avoid the problems with using a time-of-day clock)
- time fences (for the detection of worst case execution time overruns of tasks)

Note that all of the above functionality, except time fences, are likely to be available in Ada 9X [Ada 9X Mapping/Revision Team1993].

3. Tool Support

In order to support the project it was necessary to write several tools. The following were constructed.

3.1. Worst Case Execution Time (WCET) Analyser

A worst case execution time (WCET) tool is an essential component of any tool set which supports hard real-time systems development. Schedulability analysis requires that the worst case times for all tasks be known, along with the worse case execution times of any sections of code that must be executed in mutual exclusion (i.e. that can cause blocking — represented as protected objects in the design, and passive tasks in the implementation).

The computational model defined in section 2 imposes constraints on the use of Ada in hard real-time systems. Even with these constraints it is possible to construct programs which generate code which cannot be analysed by the WCET tool. It is therefore necessary to introduce further restrictions. In particular any construction the programmer uses in the time critical regions of the program must be analysable for its maximum execution time. This constraint does not apply to sections of a program that are not time-critical (for instance, initialisation sections of a program, such as the elaboration of library package specifications). The following are required by our compiler and analyser [Forsyth1992].

Pragma LOOPCOUNT

Most loop statements must have an associated implementation dependent pragma LOOPCOUNT to indicate the maximum number of times the loop will run. The actual compiler used issues a warning when it encounters a loop without a preceding pragma LOOPCOUNT, except in the following two special cases.

1 When it finds a "for loop" with constant bounds for which the compiler can calculate the loop count.

2. When it finds an infinite loop of the form:

```
loop
    -- do something
end loop;
```

which implements a periodic task.

It is worth noting that in practice application of these rules mean that very few LOOPCOUNT pragmas are needed: mainly for "while" loops and "for" statements with non-static bounds.

No GOTO statements

A goto statement makes it difficult for the analyser to detect loops, and therefore is banned.

No recursion, including mutual recursion

As with the goto statement, recursion and mutual recursion makes it difficult for the analyser to detect loops. Our compiler does not provide enough information to diagnose recursive calls when a program is compiled or linked. Instead, the WCET analyser rejects a program containing a nested sequence of calls more than 50 calls deep. This also detects and rejects recursive calls.

Bodies and specifications in separate source files

This condition is imposed to allow the time for a library unit's initialisation (the code generated for a library unit's specification) to be kept separate from the time for the execution of the unit's body.

No hidden loops with non-static bounds

A "hidden loop" is one the compiler generates to implement Ada operations on arrays, such as assignment, comparison, or concatenation. Provided the bounds of the array(s) are static, the compiler will automatically generate the internal equivalent of a pragma LOOPCOUNT for an array operation. A hidden loop can also be generated to initialise a subprogram's stack frame, and to create aggregates containing arrays. Thus, the size of all objects allocated on the run-time stack must be known statically, as must the size of any aggregate created. The compiler issues a diagnostic if it finds operations on objects of non-static size. Some of these restrictions could be removed with minor changes to the compiler.

Analysis Approach

Our approach has been to analyse the object code produced by the compiler and the assembler. The York Ada compiler adds the Ada structural data from the source code to the symbolic debugging section of each file of object code [Forsyth1992].

The York WCET tool consists of two basic components:

1. A disassembler reads text segments from the object code file and converts each machine instruction and its operands into a convenient internal form.

2. The disassembled output is then analysed to form basic blocks at the assembler level, and a flow graph is produced. The information placed by the compiler in the symbol table is used to help reduce this graph until a figure for the worst case execution time of Ada subprograms and tasks can be found. The times for the assembler instructions are obtained from table lookup.

The York approach uses individual object files rather than the fully linked load file. This is because much of the York run-time is written in C and does not have the required annotated symbol table information. When the WCET is unable to resolve a call to a subprogram (all run-time routines have a subprogram interface) it looks in a standard file to see whether there is a known time for the execution of that routine. If there is, then graph reduction continues. If there is not, then the tool continues and tries to reduce the basic blocks before and after the call.

Run-time interface routines which potentially block or cause context switches have their execution times set to zero in the file of known execution times. The schedulability analysis will take into account these operations.

3.3. Schedulability Analysis Tool

This tool takes as its input a description of the real-time characteristics of an application and the real-time characteristics of the execution environment (including the overheads incurred by the Ada run-time environment). It performs deadline monotonic scheduling analyses using the test given by

Audsley et al. [Audsley1993]. The tool outputs details of the results and allows the user to perform refinement techniques such as period transformations etc. [Audsley1993].

The following symbols are used in this paper to represent the various real-time characteristics of the task set and the execution environment

C	A task's computation time requirement
D	A task's deadline
T	A periodic task's period
B	A task's blocking time
O	A task's offset
G	A sporadic task's minimum time gap between successive invocations
t	Time
I	The interference a task gets from a high priority task
WCET	The Worst Case Execution Time
BLOCK	The Worst Case Blocking Time from Application Resources
c_s	The time cost of a context switch
c_r	The time cost of releasing a task from the delay queue and moving it to the run queue (dispatch queue);
c_d	The time cost of putting a task in the delay queue (at the end of its period)
c_n	The time cost of entering a protected object
c_l	The time cost of leaving a protected object
B_e	The maximum period (of time) of non-preemption exhibited by the execution environment

The computational requirement of a task is calculated as follows. For each periodic and sporadic thread the WCET must be extended by the overheads of its own execution:

$$TEMP := WCET + c_r + 2*c_s + c_d$$

if the thread is a sporadic with non-zero G then at run-time the gap is enforced by using an Ada delay statement; this results in extra context switches

$$TEMP := TEMP + 2*c_s$$

the overheads of making NP calls on Protected Objects must then be taken into account:

$$C := TEMP + NP*(c_p + c_l)$$

The tool assumes that each resource is encapsulated in a passive task and Immediate Ceiling Priority Inheritance is used. For all tasks the blocking time they can experience must be compared with the blocking time that could be caused by the execution environment:

$$B := MAXIMUM(B_e, BLOCK)$$

The analysis tool also assumes that there is a real-time clock which is periodic and has a known worst case execution.

3.4. Schedulability Simulation Tool

This tool allows the run-time scheduling decisions to be simulated in the host environment. It takes as its input a description of the real-time characteristics of the application and the real-time characteristics of the execution environment. Its output includes a trace of the scheduling that would be performed at

run-time. The tool can be used in conjunction with the Schedulability Analysis Tool to help the system designer construct a schedulable system.

4. The Olympus Attitude and Orbital Control System

The Olympus satellite was launched in July 1989 as the world's largest and most powerful civil three-axis-stabilised communications satellite. Situated at longitude 19 degrees West, Olympus provided direct broadcast TV and 'distance learning' experiments to Italy and Northern Europe.

An AOCS subsystem exists to acquire and maintain the desired spacecraft position and orientation. The Olympus AOCS software may operate in six modes of operation, of these the Normal Mode is the most complex and is used for the greatest percentage of the satellites lifetime. This paper considers normal mode and an "Earth Acquisition and Lock" mode.

4.1. Hardware Architecture

The Normal Mode software is embedded in the Spacecraft Microcomputer Module (SMM) and communicates with the following devices over a serial data bus

- A Telemetry & Telecommand Subsystem (TMTC),
- An InfraRed Earth Sensor (IRES),
- A Digital Sun Sensor (DSS),
- A Rate Gyro Sensor (RGS),
- Four reaction wheels (RWs),
- Thrusters.

There is a cold-standby SMM which is powered up should a failure be detected. In this case study we are interested in real-time aspects of the system, and therefore we shall assume the system's hardware and software is reliable.

Serial bus messages are placed on the bus according to the priority of the transmitting device. Gyros have the highest priority, followed by the TMTC and the SMMs. The bus has time slots reserved for replies, ensuring that SMM requests for sensor data receive responses within a 960 μs time slot. There is a source of regular 10ms clock interrupts.

The system described in this paper runs on 2 VME boards containing a 68020 processor, a 68881 floating point coprocessor, 1 M Byte RAM, timers, and dedicated chips for communication over the Olympus serial bus. These new cards replace the current SMM in the Olympus Engineering Model test facility to demonstrate successful operation of the unit.

4.2. Software Requirements

The object of `Normal Mode' attitude control (NRM Mode) is to maintain the satellite's orientation to Earth. This is to be achieved by a 200 ms cyclic task that forces IRES roll and pitch angles and a rate-gyro derived yaw angle to zero by controlling the speed of four reaction wheels. For two spells per day of 15 minutes, a one second period task calibrates the gyro drift rate by comparing the yaw estimate, derived from an integration of gyro rate, against the sun and earth positions. The gyro angle and gyro drift rate are corrected at the end of each spell. The gyro data is received approximately every 100ms (without being requested).

In addition to the regular activities identified above, further attitude control functions occur at less frequent or irregular intervals.

- Momentum dumping is triggered when the speed of any reaction wheel exceeds a preset threshold. This consists of a reduction in the reaction wheel speed in a series of steps, while compensating

bursts of thruster firings prevent the loss of Earth-pointing. Dumping on the three axes operates independently.

- The Telemetry and Telecommand Subsystem (TMTC) routinely requests status information from the AOCS software. This task, which has a minimum period of 62.5 ms, keeps the ground informed of the spacecraft state. The SMM is unable to respond to a telemetry request in the same bus time slot, so it transmits a response in a later time slot.

- A telecommand function allows ground to enable or disable control, to enable or disable dumping, to trigger gyro calibration, or to set a reaction wheel failed or operational. Telecommands can occur at a minimum interval of 190μs

The other mode which was implemented was Earth Acquisition and Lock Mode (EAL). This, simpler, mode has the goal of placing the satellite in a standard attitude. Thrusters rather than reaction wheels are used to control the spacecraft orientation.

A full description of the Requirements of the AOCS is given by Bailey[Bailey1992].

4.3. Software Design

We have presented the design of the AOCS elsewhere [Burns1993b, Burns1993c]. The software for the two modes consists of 10 cyclic objects, 3 sporadic objects, 14 protected objects, and 16 passive objects. Overall over 3,000 lines of Ada code was produced.

5. Performance Evaluation

Although the schedulability analysis techniques allow the response time of each task to be predicted, in any realistic system it is very difficult to measure the worst case response times of all the tasks, as it is usually not possible to exercise the system to produce its worst case behaviour. Furthermore, even measuring the average response time is fraught with problems unless the underlying run-time support systems keeps track of when tasks are released and when they finish each cycle. In our case, the run-time systems provided only a mechanism by which system idle time could be measured, and therefore we shall concentrate on the difference between the predicted utilisation and the measured utilisation.

Utilisation figures may be obtained from three different sources, calculation, simulation and measurement. These sources are discussed below:

Calculated Utilisation

Values of processor utilisation may be calculated by taking the sum of the utilisations of each task. The [kernel] Clock thread must be included [Burns1993d].

The utilisation of task i is calculated as C_i/T_i, where C_i is the worst case execution time of Task i (reported by the worst case execution analyser) and T_i is the period or minimum arrival time of Task i. C_i must include the context switch times to and from task i.

Simulator Utilisation Figures

The Schedulability Simulator reports processor utilisation. This figure excludes context switching overheads and the cost of entering and leaving protected objects.

A figure that includes these runtime system overheads may be obtained by adding an idle task to the simulation to absorb all unused CPU time. The processor utilisation may be calculated as $100 - U_{idle}$. Although this figure will include the costs of extra context switching to and from the idle task, these extra costs are expected to be small.

Execution time figures are available for the worst case, but not for the best or average execution time. However the simulator is able to simulate the arrival of the sporadic threads at their average rate as well as their worst case rate. The Simulator is therefore able to yield two estimates for processor utilisation: a worst case figure and one that is more realistic in the arrival of the sporadic tasks.

Measured Utilisation

The software was instrumented to monitor the amount of idle time spent at run-time. Telecommands are used to initiate and stop monitoring. The duration of recording is returned in Telemetry as is the time spent by the Run time System in its busy wait loop. The utilisation is calculated using these two values.

The measured utilisation is a 'typical' or 'average' value rather than the worst case value that is produced by the above two methods. The measured utilisation values presented in this report were measured over periods of 500 seconds.

5.1. Results

Calculated Worst Case Utilisation

The table below calculates worst case utilisation figures for the two modes of AOCS. The sporadic tasks are assumed to arrive at their minimum arrival interval.

				Normal Mode		EAL Mode	
Task	WCET note a	Context Switch Overhead	Total WCET	Period /MAT	Utilisation	Period /MAT	Utilisation
	ms	ms	ms	ms	%	ms	%
Clock	0.408	0	0.408	10.0	4.08	10.0	4.08
onemsg_here	0.098	0.108	0.206	100.0	0.21	50.0	0.41
twomsg_here	0.304	0.108	0.412	50.0	0.82	100.0	0.41
fourmsg_here	0.717	0.108	0.825	200.0	0.41	#######	0.00
tm_here	0.098	0.108	0.206	62.5	0.33	62.5	0.33
tc_here	0.098	0.108	0.206	187.0	0.11	187.0	0.11
z1_here	0.098	0.108	0.206	100.0	0.21	100.0	0.21
Calibrate_gyro	9.46	0.472	9.932	1000.0	0.99	#######	0.00
Command_Actuators	4.11	0.472	4.582	200.0	2.29	100.0	4.58
EAL_Mode	8.16	0.472	8.632	#######	0.00	100.0	8.63
Normal_Mode	42.11	0.472	42.582	200.0	21.29	#######	0.00
Process_DSS_Data	5.38	0.472	5.852	1000.0	0.59	#######	0.00
Process_IRES_Data	7.83	0.472	8.302	100.0	8.30	100.0	8.30
Read_Bus_Ip	1.99	0.472	2.462	10.0	24.62	10.0	24.62
Read_Yaw_Gyro	5.57	0.8	6.37	100.0	6.37	100.0	6.37
Real_Time_Clock	0.06	0.472	0.532	50.0	1.06	50.0	1.06
Request_DSS_Data	2.73	0.472	3.202	200.0	1.60	#######	0.00
Request_IRES_Data	1.78	0.472	2.252	100.0	2.25	100.0	2.25
Request_Wheel_Speeds	2.73	0.472	3.202	200.0	1.60	#######	0.00
Telecommands	4.35	0.8	5.15	187.0	2.75	187.0	2.75
Telemetry_Response	4.18	0.8	4.98	62.5	7.97	62.5	7.97
Time0_Update	0.47	0.472	0.942	3600.0	0.03	3600.0	0.03
Total					87.89		72.13

Note:

a) The WCET figure identified in this column include the cost of entering and leaving protected objects , the cost of exception handling and error messages.

b) The periods of tasks not active in the specified mode are shown as '########'.

c) Tasks running in both modes execute the same code, and so have the same WCET, in both modes.

d) onemsg_here, twomsg_here etc. are tasks representing interrupts, they are used for analysis purposes only.

5.2. Comparisons with Simulator and Measured Utilisation

The utilisation figures yielded by the three methods are tabled below.

	Utilisation (%)	
	Normal Mode	**EAL**
Calculated Utilisation	87.9	72.1
Simulator Worst Case	83.2	67.7
Simulator, average arrivals	74.2	58.8
measured	37.7	29.2

As can be seen, there are large differences between the predicted utilisation and those actually measured. This difference is due to the inaccuracy in the analysis and the tools supporting the analysis, and the difficult of measuring actual worst case. This is discussed further below.

5.3. Causes of Inaccuracy

Two main causes of inaccuracy may be identified in the analysis techniques:

Inaccuracy in the scheduling analysis techniques

The analysis embodied in the scheduling analysis tool makes some assumptions including

- Execution times are always assumed to be at the maximum. In practise a predictable pattern of executions may be observed. (e.g. the Telemetry task executes its worst case pathway only once every 33 calls).

- Sporadic inter-arrival intervals are always assumed to be the minimum. In practise a regular pattern of arrivals may be observed over a time period that is greater than the minimum inter-arrival interval. (This may also be observed in the Telemetry task).

- A simple model was used for accounting for the overheads associated with the handling of clock interrupts and manipulating the Ada delay queue [Burns1993d].

Recent advances in scheduling theory may allow some of these assumptions to be relaxed.

Inaccuracies in the WCET analysis tool

The worst case execution analyser returns worst case execution time estimates for each task; these will be pessimistic as simplifying assumptions have been made in order to simplify construction of the tool .

- The improved processor efficiency provided by pipelining is not predicted.

- Cache is to be disabled to allow the calculation of execution times and must remain disabled during actual runs. This reduces the performance of the microprocessor.

- Compiler optimisation has been prohibited to allow the calculation of execution times. This also reduces the effective power of the microprocessor.

- Instructions may have execution times that are data dependent. In practice these instructions may never receive the data that leads to the worst case instruction time.

- Calculated execution times assume that the worst case path through the code is always taken. This worst case path is determined simply and may not be legal, e.g. in the code fragment below the calculated worst case execution time includes the execution times of both branches a and b; such a path is illegal and could never be followed.

```
procedure p1( val : in out integer ) is
begin
      if  val < 1000 then
            -- branch a : code not shown
      end if;
      -- code not shown.
         val := val + 3;
      -- code not shown.
      if  val > 2000 then
            -- branch b : code not shown
      end if;
      -- analyser assumes that the exception with the most
      -- expensive handler is raised here
exception
      when ...
end;
```

(The above code fragment is assumed to show all changes to the variable 'val'.)

Elements of this inaccuracy could be eliminated by the use of semantic information, nevertheless, it is not obvious that all such cases could be eliminated.

5.4. Analysis of Inaccuracy

Worst Case Utilisation Figures

The following table compares the worst case execution times yielded by calculation and by the simulator respectively.

	Utilisation (%)		
	Calculated worst case	Simulator worst case	Difference
EAL Mode	72.1	67.7	4.4
NRM Mode	87.9	83.2	4.7

The 'calculated worst case' utilisation assumes that each task that executes interrupts the execution of a lower priority task and therefore adds twice the context switch time to each task WCET. This assumption mirrors that of the Schedulability Analyser. In contrast, the 'simulator worst case' calculates the context switch cost on the basis of the priorities of the current and newly released tasks; if the newly released task has priority that is lower than that of the currently executing task, or if there is no executing task, it does not interrupt that task. As shown in Appendix A1, this difference in assumptions is able to explain a utilisation difference of 2.4 % .

Simulator Utilisation Figures

The figures below identify the utilisation predicted by the simulator for the two cases of average and worst case arrival frequencies. In both cases the execution time of the threads is assumed to be worst case.

	Utilisation (%)		
	Simulator worst case	**Simulator 'average arrivals'**	**Difference**
EAL Mode	67.7	58.8	8.9
NRM Mode	83.2	74.2	9.0

The 'expected' differences in utilisation are calculated below at 9.83%. The expected and simulator differences therefore correspond to the working accuracy.

		Utilisation Difference (%)	
		NRM	EAL
1.	The Telemetry sporadic has a minimum arrival interval of 62.5 ms, however, over the 24 second format only 43 of a possible 384 telemetry requests are targeted at the software. This reduces the average utilisation of Telemetry_Response from 7.97% to 0.89%.	7.08	7.08
2.	The minimum arrival interval of the Telecommands sporadic is 187 ms, however the average approaches zero.	2.75	2.75
	Total expected difference	9.83	9.83

The first of the two expected differences reflects the assumption of a critical instant by the schedulability analyser; if the schedulability analysis took offset information into account a more sophisticated timing data file could be produced that would model the arrival pattern of the telemetry sporadic. Scheduling theory is now available that takes account of offset information [Tindell1992].

Average Utilisation Figures

The figures below compare the 'average arrival' simulator utilisation and the measured average. The measured average was observed with both cache and optimisation disabled. For Normal Mode, Dumping and Control was enabled and all three axis reaction wheels were operational.

	Utilisation (%)		
	Simulator 'average arrivals'	**Measured average**	**Difference**
EAL Mode	58.8	29.2	29.6
NRM Mode	74.2	37.7	36.5

The measured average utilisation may properly be expected to be less than the simulated (average arrivals) where the execution time is not worst case. Six such reasons are identified below.

		Difference in utilisation: simulator 'average arrivals'- measured average	
		NRM (%)	EAL (%)
1	The read_bus_ip task expects in the worst case to read six messages from the FIFO. Over an interval of 200 ms an average of approximately one message per 10 ms execution period is read; see appendix A2.	4.00	4.24
2	Calibration occurs as two 15 minute spells per 24 hours, the average utilisation is therefore not 0.95% but 0.02%.	0.93	0
3	Dumping of momentum occurs as reaction wheel speeds reach a threshold. The tests were of insufficient duration for this to occur; see appendix A3.	0.92	0
4	Except for the quick and simple interrupt tasks, threads have an exception handler to give warning to ground of their occurrence and to prevent the permanent loss of the task. Since the code is designed such that exceptions should never happen, the worst case assumption that an exception is raised in each task at each execution is clearly pessimistic. The average occurrence of exceptions should approach zero; see appendix A5.	6.16	5.99
5	A similar observation may be applied to error messages. These are raised by many threads to flag unexpected events, the occurrence of which should approach zero; see appendix A4.	6.45	4.13
6	Less utilisation present in both 4 & 5 above[1]	-4.4	-4.28
	Total explained difference in Utilisation simulator average : measured average.	14.06	10.08

	Utilisation Difference %	
	NRM	EAL
Simulator 'average arrivals'	74.2	58.8
Measured average	37.7	29.2
Total difference in Utilisation	36.5	29.6
Total explained difference in Utilisation	14.06	10.08
Unexplained difference in Utilisation	22.44	19.52

The above causes of difference between the worst case & measured average utilisation are not exhaustive. The as yet unexplained difference in utilisation will include further components:

- so far neglected differences between the 'typical' and worst case paths through the code, this will include both 'large scale' effects, for example, the use of default Earth angles when the Process_IRES_Data task determines that the Earth is outside the IRES field of view, and 'small scale' effects, such as the presence of a 'carry' bit in a multiple precision arithmetic operation.

- the effect of the instruction pipeline which the worst case execution time analyser cannot model; in other experiments the WCET tool gives approximately 15-25% inaccuracy for straight-line code.

[1] To avoid counting this component twice.

Effect of Cache & Optimisation

The table below compares the measured utilisation for three conditions of cache and compiler optimisation. The figures relating to Normal Mode have Dumping and Control enabled and all three axis reaction wheels operational.

		Measured Utilisation (%)	
Cache	**Optimisation**	**EAL**	**NRM**
Disabled	Disabled	29.2	37.7
Enabled	Disabled	28.0	36.11
Disabled	Enabled	27.75	35.24

The cache for the 68020 is 64 long (32 bit) words; this holds instructions only.

Cache and optimisation are therefore observed to raise the processor performance by 4% and 5.2% respectively (EAL mode) or by 4.4% & 7% (NRM mode). The low increase in performance observed with the use of cache may be due to the case study containing only a few small loops.

6. Variation in Thread Execution Times

The analysis above has identified the major constituents of the worst case execution time estimates. It does not give any indication as to whether the distribution of the execution times may be modelled as a Gaussian or uniform distribution, nor does it identify whether the series may be considered a random series or whether successive execution times are correlated.

A special version of the software was generated to poll the budget timer at the end of a task execution to determine the remaining budget for that task. These values were stored within the onboard software and output with a resolution of 4 ms via Telemetry. The execution times were calculated from these values and the initial budget values.

While the direct observation of thread execution times cannot be guaranteed to yield the extreme values, it does give an indication of the 'typical' variation. The table below records, for four tasks, the maximum, minimum and average values of execution time in a series of 372 observations sampled at intervals of 24 seconds. All times recorded in the table below are in ms.

Task	WCEA tool WCET	Number of samples	Mode	Max observed execution time	Average observed execution time	Min observed execution time
Normal Mode	42.11	372	NRM	20.136	19.931	19.824
EAL Mode	8.16	372	EAL	4.564	4.532	4.464
Request IRES	1.78	372	NRM	0.472	0.397	0.364
Request IRES	1.78	372	EAL	0.364	0.364	0.364
Process IRES	7.83	372	NRM	4.964	4.900	4.800
Process IRES	7.83	372	EAL	5.12	5.020	4.564

Notes:

- These times were recorded with normally executing software, i.e. no exceptions and no errors were raised. The WCEA tool predicts that the occurrence of exceptions would raise the execution time of each task by 0.84 ms.

- The normal mode task readings were recorded with control enabled and both calibration and momentum dumping inhibited; zero wheel speed avoidance was enabled and did occur.

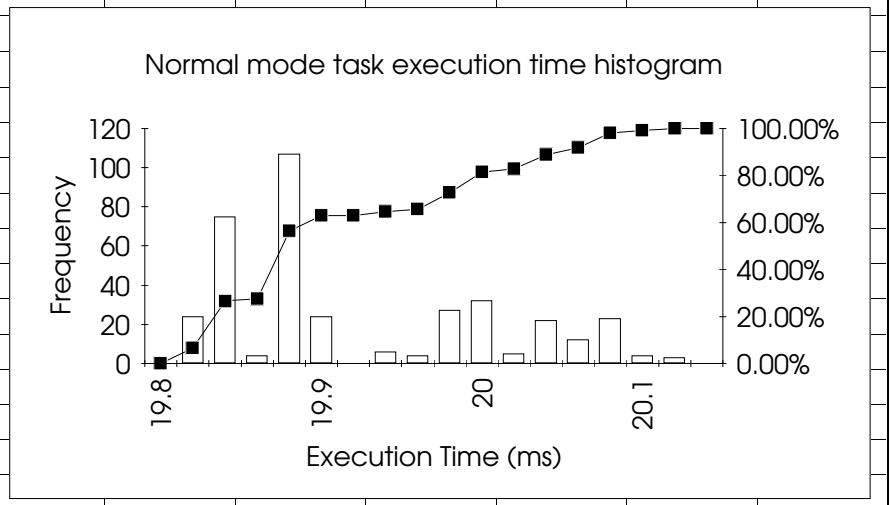
- EAL mode has no submodes that will influence the execution time of this task. The observed variation in execution times is assumed to be related to the firing or non-firing of the thrusters. In principle all three axes could fire simultaneously, although the chance of observing this is low. The variation in execution times should therefore be trebled to cover this case.

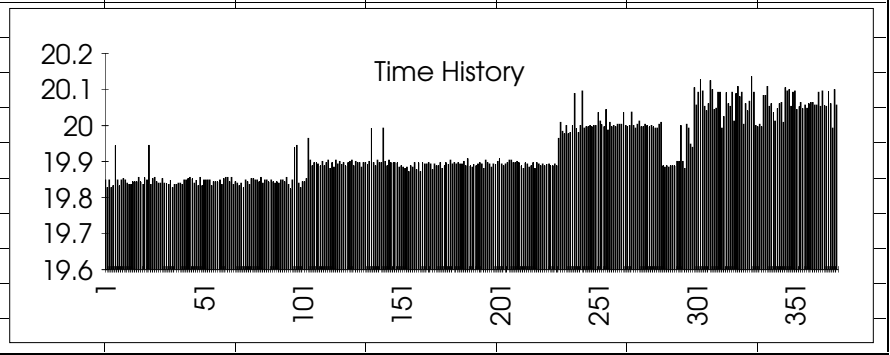
- Although the process_IRES_data thread code shows a spread of execution times, it does not show all possible paths through the code. In particular it does not include the path taken when the sign but not the magnitude of the angles is known; such a path will only be followed immediately prior to Earth capture shortly after launch. The existence of paths that execute only under such rare conditions implies that great caution is required in the drawing of conclusions from 'typical' data series such as these.

The following four pages illustrate the execution times as time series and as a histogram. It may be observed that execution times are not normally distributed. Neither do they form a random series, they are strongly correlated over time.

Normal_Mode task execution time in normal mode.

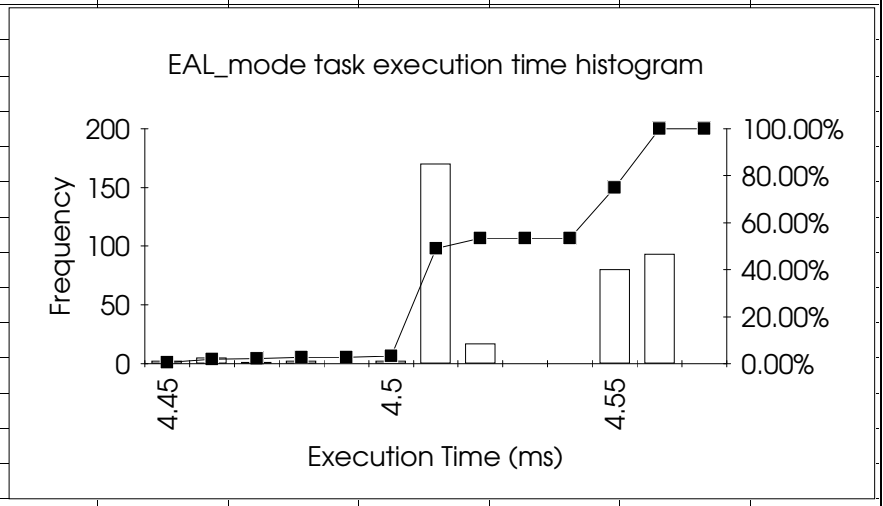
Bin	Frequency	Cumulative %
19.8	0	0.00%
19.82	24	6.45%
19.84	75	26.61%
19.86	4	27.69%
19.88	107	56.45%
19.9	24	62.90%
19.92	0	62.90%
19.94	6	64.52%
19.96	4	65.59%
19.98	27	72.85%
20	32	81.45%
20.02	5	82.80%
20.04	22	88.71%
20.06	12	91.94%
20.08	23	98.12%
20.1	4	99.19%
20.12	3	100.00%
210.14	0	100.00%
	372	

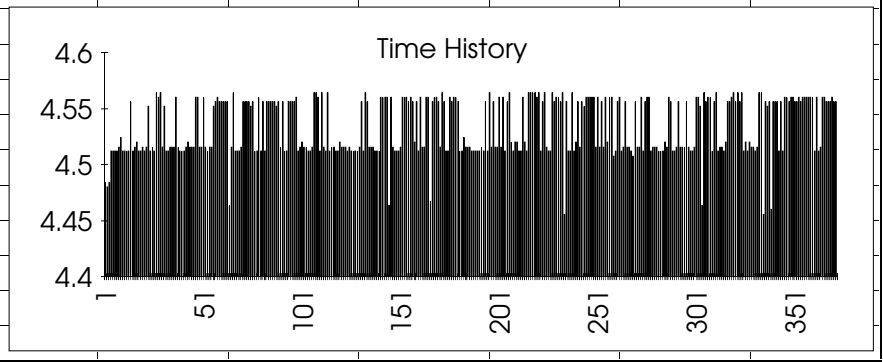




EAL_mode task execution time distribution in EAL mode

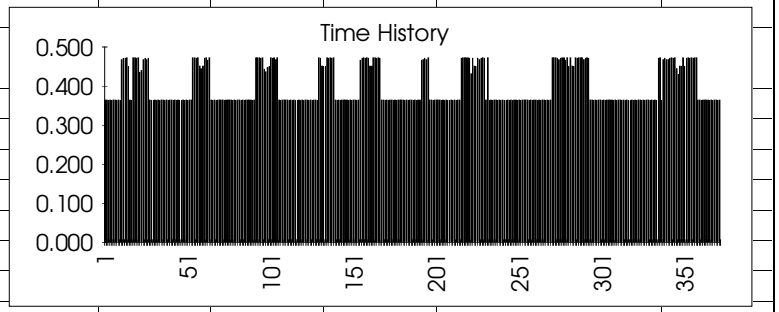
Bin	Frequency	Cumulative %
4.45	2	0.54%
4.46	5	1.88%
4.47	1	2.15%
4.48	2	2.69%
4.49	0	2.69%
4.5	2	3.23%
4.51	170	48.92%
4.52	17	53.49%
4.53	0	53.49%
4.54	0	53.49%
4.55	80	75.00%
4.56	93	100.00%
4.57	0	100.00%
	372	





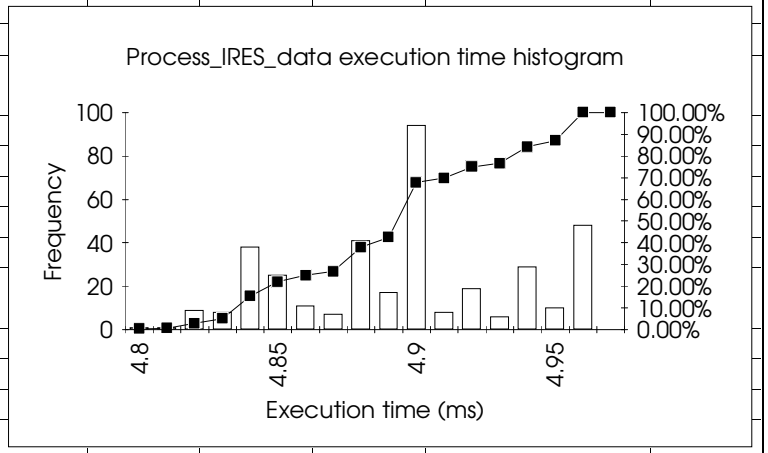
Request_IRES_data task execution time distribution in Normal mode.

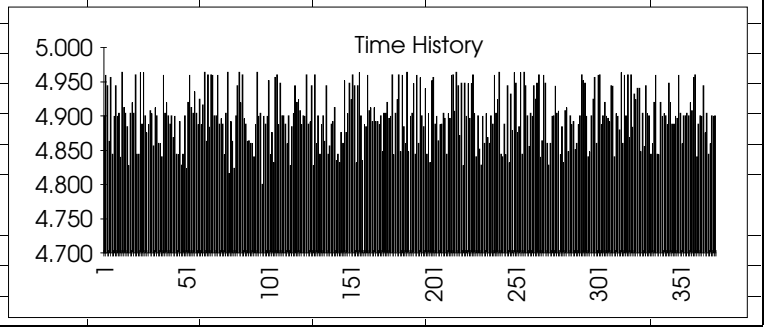
Bin	Frequency	Cumulative %
0.355	0	0.00%
0.36	242	65.05%
0.365	0	65.05%
0.37	0	65.05%
0.375	0	65.05%
0.38	0	65.05%
0.385	0	65.05%
0.39	0	65.05%
0.395	0	65.05%
0.4	0	65.05%
0.405	0	65.05%
0.41	0	65.05%
0.415	0	65.05%
0.42	0	65.05%
0.425	0	65.05%
0.43	2	65.59%
0.435	3	66.40%
0.44	3	67.20%
0.445	1	67.47%
0.45	17	72.04%
0.455	0	72.04%
0.46	0	72.04%
0.465	16	76.34%
0.47	88	100.00%
	372	



Process_IRES_data task execution time distribution in Normal mode.

Bin	Frequency	Cumulative %
4.8	1	0.27%
4.81	1	0.54%
4.82	9	2.96%
4.83	8	5.11%
4.84	38	15.32%
4.85	25	22.04%
4.86	11	25.00%
4.87	7	26.88%
4.88	41	37.90%
4.89	17	42.47%
4.9	94	67.74%
4.91	8	69.89%
4.92	19	75.00%
4.93	6	76.61%
4.94	29	84.41%
4.95	10	87.10%
4.96	48	100.00%
4.97	0	100.00%
	372	





7. Conclusions

If schedulability analysis for hard real-time (potentially safety critical) systems is to migrate from the research environment to industrial use, it must be possible to identify the degree of inaccuracy introduced. From the results identified in sections 5 and 6 we can conclude that the worst case execution time estimates generated by the Worst Case Execution Time Analyser and processed by the Schedulability Analyser and Simulator have the following components:

1. A component that corresponds to 'typical' executions.
2. A component that corresponds to executions that are 'atypical' but in some sense expected.
3. A component that corresponds to executions that in principle can occur but which are not expected.
4. A component that corresponds to undue inaccuracy that could be removed with further enhancement to the tools (technical or removeable pessimism).
5. A component that corresponds to undue inaccuracy, but which cannot be removed without development of the theory (structural or immoveable pessimism).

Taking these in reverse order,

Item 5 relates to the benefits that would be available to us if the techniques of compiler optimisation, cache and pipelining could be analysed. If it were possible to guarantee that the use of these techniques always yielded lower execution time code than without them then the use of these techniques could provide a 'safety margin'; if not then the techniques could not be used in hard real-time applications. The pattern of context switches also partly falls in this category.

Item 4 relates to benefits that could be achieved with the analysis of:

- Patterns of sporadic task arrival that are predictable but which do not correspond to a single minimum inter-arrival interval.
- Patterns of execution time that are not constant but which vary predictably.

Item 3 relates to the reporting of exceptions and (arguably) errors.

Where processor requirements are heavy there may be a conflict between the requirement of verified schedulability and other requirements. This will be illustrated with an example:

> The code development and test procedures are such that exceptions should never occur in flight. Never-the-less, an exception handler is present that catches these extremely unlikely events and reports them via telemetry. The philosophy applied in this study is that the cost of these handlers must be charged for every execution of every task; this costs 6% of processor utilisation.

It is the authors' conjecture that this philosophy might be reappraised if a strict insistence on worst case schedulability threatens the loss of full diagnostic information from the spacecraft. Probabilistic arguments might be employed to argue, for instance, that the probability of **n** tasks simultaneously raising exceptions is effectively zero.

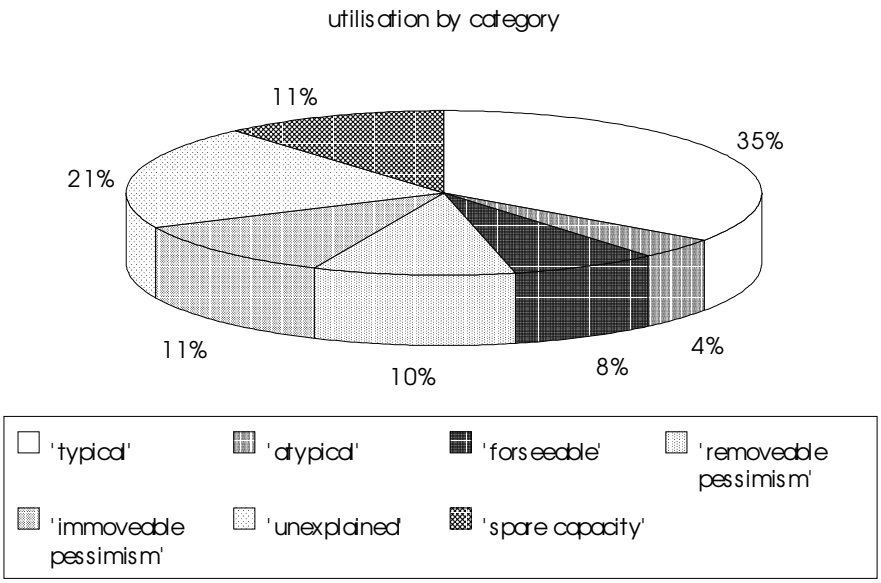
Item 2 is the sum of those components that occur in the expected worst case scenario, but not in 'typical' execution. This includes:

- Telecommand reception
- Calibration
- Momentum dumping

Item 1 reflects the measured, 'typical', utilisation. This is the maximum that might be observed during testing; items 2, 3 and 4 could add almost 50% to this value. Thereby pointing to the value of predicting over measurement.

Numerical estimates

A breakdown of the AOCS utilisation into the above categories is shown below, executing in Normal Mode on an optimised system using cache. In this diagram the figures presented in section 5 have been rebased to reflect the improvements that cache and optimisation bring.[1,2]



Our performance evaluation was not able to identify the "unexplained" component of the utilisation. We can only assume that it is related to variable instruction times, memory wait states etc.

Execution time distributions

Inspection of the recorded task execution time histories identifies that the execution times do not constitute a random series, rather successive samples are strongly correlated.

Such an observation is powerful evidence that schedulability analysis should be undertaken with worst case execution times, except where knowledge of the application environment can be applied. It is not possible, in general, to develop statistical arguments of the form that the probability of n consecutive executions having execution times greater than l and less than p.

Deadlines

This report has concentrated on the analysis of utilisation, which provides a broad brush view of the system as a whole. This should not be interpreted in any way as denigrating the importance of deadlines, which is the undisputed goal of the schedulability analysis tools.

---

[1] The order that the pie slices appear in the key is clockwise from the largest slice.

[2] The figures were measured on a processor that had cache disabled and with code that had not been optimised. The measured utilisation figures were reduced by a factor of 0.92 to reflect the improvements that optimisation and cache could (theoretically) bring. The extra utilisation provided by cache and optimisation was added to the 'immoveable pessimism' component as the theory does not support their use for hard real-time systems.

Acknowledgement

This work has been supported by the European Space Agency (ESTEC Contract 9198/90/NL/SF). The authors would like to thank Eric Fyfe, Paco Gomez Molinero and Tullio Vardanega for their help during the course of the project.

Appendix A

Calculation of Utilisation Impacts

A1. Context switch Utilisation

The component of the calculated utilisation due to context switches is calculated below:

		Normal Mode		EAL Mode	
Task	Context Switch Overhead ms	Period /MAT ms	Utilisation %	Period /MAT ms	Utilisation %
Clock	0	10.0	0.00	10.0	0.00
onemsg_here	0.108	100.0	0.11	50.0	0.22
twomsg_here	0.108	50.0	0.22	100.0	0.11
fourmsg_here	0.108	200.0	0.05	########	0.00
tm_here	0.108	62.5	0.17	62.5	0.17
tc_here	0.108	187.0	0.06	187.0	0.06
z1_here	0.108	100.0	0.11	100.0	0.11
Calibrate_gyro	0.472	1000.0	0.05	########	0.00
Command_Actuators	0.472	200.0	0.24	100.0	0.47
EAL_Mode	0.472	########	0.00	100.0	0.47
Normal_Mode	0.472	200.0	0.24	########	0.00
Process_DSS_Data	0.472	1000.0	0.05	########	0.00
Process_IRES_Data	0.472	100.0	0.47	100.0	0.47
Read_Bus_Ip	0.472	10.0	4.72	10.0	4.72
Read_Yaw_Gyro	0.8	100.0	0.80	100.0	0.80
Real_Time_Clock	0.472	50.0	0.94	50.0	0.94
Request_DSS_Data	0.472	200.0	0.24	########	0.00
Request_IRES_Data	0.472	100.0	0.47	100.0	0.47
Request_Wheel_Speeds	0.472	200.0	0.24	########	0.00
Telecommands	0.8	187.0	0.43	187.0	0.43
Telemetry_Response	0.8	62.5	1.28	62.5	1.28
Time0_Update	0.472	3600.0	0.01	3600.0	0.01
Total			10.88		10.74

The calculated utilisation assumes that each newly released thread interrupts a currently executing thread of lower priority. On this basis each thread execution requires two context switches: one from the interrupted thread and one returning to it. Where there is no thread executing or where the newly released thread is of lower priority than the currently executing thread then the current thread will run to completion and the newly released thread requires only one context switch.

For cyclic threads these two cases correspond to overheads of:

$C_r + 2*C_s + C_d = 0.472$ ms pre-emption (worst case)

and

$C_r + C_s + C_d = 0.308$ ms no pre-emption (typical)

respectively.

Similarly for the (gap-enforced) sporadics, which may suffer two pre-emptions, these two cases correspond to overheads of:

$C_r + 4*C_s + C_d = 0.800$ ms pre-emption (worst case)

and

$C_r + 2*C_s + C_d = 0.472$ ms no pre-emption (typical)

respectively.

The context switch overhead on the assumption that no threads were preempted would be as illustrated in the table below:

		Normal Mode		EAL Mode	
	Context				
Task	Switch	Period		Period	
	Overhead	/MAT	Utilisation	/MAT	Utilisation
	ms	ms	%	ms	%
Clock	0	10.0	0.00	10.0	0.00
onemsg_here	0.108	100.0	0.11	50.0	0.22
twomsg_here	0.108	50.0	0.22	100.0	0.11
fourmsg_here	0.108	200.0	0.05	########	0.00
tm_here	0.108	62.5	0.17	62.5	0.17
tc_here	0.108	187.0	0.06	187.0	0.06
z1_here	0.108	100.0	0.11	100.0	0.11
Calibrate_gyro	0.308	1000.0	0.03	########	0.00
Command_Actuators	0.308	200.0	0.15	100.0	0.31
EAL_Mode	0.308	########	0.00	100.0	0.31
Normal_Mode	0.308	200.0	0.15	########	0.00
Process_DSS_Data	0.308	1000.0	0.03	########	0.00
Process_IRES_Data	0.308	100.0	0.31	100.0	0.31
Read_Bus_Ip	0.308	10.0	3.08	10.0	3.08
Read_Yaw_Gyro	0.472	100.0	0.47	100.0	0.47
Real_Time_Clock	0.308	50.0	0.62	50.0	0.62
Request_DSS_Data	0.308	200.0	0.15	########	0.00
Request_IRES_Data	0.308	100.0	0.31	100.0	0.31
Request_Wheel_Speeds	0.308	200.0	0.15	########	0.00
Telecommands	0.472	187.0	0.25	187.0	0.25
Telemetry_Response	0.472	62.5	0.76	62.5	0.76
Time0_Update	0.308	3600.0	0.01	3600.0	0.01
Total			7.19		7.08

Analysis of the simulator reports indicate that, for Normal mode, 1066 thread executions occurred within a period of 3 seconds. During this time only 361 interruptions of currently executing threads were

observed. We may therefore weight the two figures to determine the context switch utilisation that is required for a 'typical' Normal mode measurement:

(361*10.88 + (1066-361)*7.19) / 1066 = 8.44 %

Analysis of the EAL mode reveals that within a duration of 2.5 seconds 870 thread executions and 295 interruptions of currently executing threads were observed. The context switch overhead for this mode was therefore:

(295*10.74 + (870-295)*7.08) / 870 = 8.32 %

A2. Receive_from_Bus Object

The receive_from_bus task will read a maximum of six bus messages from its input FIFO every execution. The execution time required to read one bus message is calculated from the read_bus_message task profile as: 60.6 + 19.6 = 80.2 ms.

The average number of bus messages expected over a period of 200 ms may be calculated as:

	Normal Mode	**EAL Mode**
onemsg_here	2	4
twomsg_here	8	4
fourmsg_here	4	0
tm_here	3.2	3.2
tc_here	1.1	1.1
z1_here	2	2
total	20.3	14.3

Over this period of 200 ms the read_from_bus task will execute 20 times. The difference in execution time between the worst case and average cases is calculated as:

		Normal Mode	EAL Mode
Msgs per 10 ms:	Worst case	6	6
	Average	1.015	0.715
	Difference	4.985	5.285
Cost (ms per 10 ms execution):		4.985 * 0.0802 ms	5.285 * 0.0802 ms
Utilisation Difference:		3.998 %	4.239 %

A.3. Dumping Utilisation

Momentum dumping and zero wheel speed avoidance are both sporadic functions that are undertaken by the cyclic Normal mode thread. The difference in the Normal mode thread execution time that is explained by these two functions is investigated below.

Dumping is an activity that is triggered when reaction wheel speeds reach a threshold. The activity of momentum dumping returns the wheel speeds to within the acceptable speed band. At the completion of dumping a flag is set for input to the zero wheel speed avoidance routine.

Zero wheel speed avoidance is triggered when wheel speeds are within a predefined speed band centred on zero. When triggered zero wheel speed avoidance causes the wheel speed to pass through zero with a single thruster pulse.

A monitoring function executes regardless of whether dumping or wheel speed avoidance is triggered. For this reason the monitoring function is excluded from our calculation.

The three axes roll, pitch and yaw execute independently. Each axis may be operating in a dumping submode, a zero wheel speed avoidance submode, or a submode in which neither dumping nor zero wheel speed avoidance is executing. Axes cannot be both dumping and avoiding zero.

Either zero_dumping or (momentum_dumping + zero_crossing_update) takes place for each axis. The execution times are:

zero_dump.dump = momentum_dump.dump	=	0.217 ms.
zero_dump.zero_crossing_update	=	0.395 ms.
Total		0.612 ms

The variation in utilisation explainable by the above functions is therefore:

$$\frac{0.612 \text{ ms}}{200 \text{ ms}} * 3 \text{ axes} * 100\ \% = 0.918\ \%$$

A4. Effect of error message utilisation:

The processor utilisation excluding the component attributable to error message generation is calculated below. The worst case execution times were generated by the worst case execution time analyser tool with the procedure write_error_message set to a null procedure.

				Normal Mode		EAL Mode	
		Context					
Task	WCET	Switch	Total	Period		Period	
	note a	Overhead	WCET	/MAT	Utilisation	/MAT	Utilisation
	ms	ms	ms	ms	%	ms	%
Clock	0.408	0	0.408	10.0	4.08	10.0	4.08
onemsg_here	0.098	0.108	0.206	100.0	0.21	50.0	0.41
twomsg_here	0.304	0.108	0.412	50.0	0.82	100.0	0.41
fourmsg_here	0.717	0.108	0.825	200.0	0.41	########	0.00
tm_here	0.098	0.108	0.206	62.5	0.33	62.5	0.33
tc_here	0.098	0.108	0.206	187.0	0.11	187.0	0.11
z1_here	0.098	0.108	0.206	100.0	0.21	100.0	0.21
Calibrate_gyro	8.28	0.472	8.752	1000.0	0.88	########	0.00
Command_Actuators	3.52	0.472	3.992	200.0	2.00	100.0	3.99
EAL_Mode	9.79	0.472	10.262	########	0.00	100.0	10.26
Normal_Mode	41.68	0.472	42.152	200.0	21.08	########	0.00
Process_DSS_Data	4.79	0.472	5.262	1000.0	0.53	########	0.00
Process_IRES_Data	7.24	0.472	7.712	100.0	7.71	100.0	7.71
Read_Bus_Ip	2	0.472	2.472	10.0	24.72	10.0	24.72
Read_Yaw_Gyro	4.38	0.8	5.18	100.0	5.18	100.0	5.18
Real_Time_Clock	0.06	0.472	0.532	50.0	1.06	50.0	1.06
Request_DSS_Data	2.14	0.472	2.612	200.0	1.31	########	0.00
Request_IRES_Data	1.18	0.472	1.652	100.0	1.65	100.0	1.65
Request_Wheel_Speeds	2.14	0.472	2.612	200.0	1.31	########	0.00
Telecommands	2.48	0.8	3.28	187.0	1.75	187.0	1.75
Telemetry_Response	3	0.8	3.8	62.5	6.08	62.5	6.08
Time0_Update	0.47	0.472	0.942	3600.0	0.03	3600.0	0.03
Total					81.44		68.00

By comparison with the calculated utilisation of section 4, the utilisation required for error messages is :

NRM : 87.89 - 81.44 := 6.45 %
EAL : 72.13 - 68.00 := 4.13 %

A5. Cost of exceptions

The cost of exception handling was determined to be 0.84 ms by applying the Worst Case Execution Analyser to code from which the exception handler code had been removed.

The component of processor utilisation applicable to exception handling is calculated below for both Normal Mode and EAL.

A second set of calculations identifies the exception handling component when the cost of errors is set to zero. The cost of exception handling was determined to be 0.24 ms.

Both sets of calculations are necessary because of a strong cross coupling between the cost of exceptions and the cost of outputting error messages.

			errors at normal execution time			errors not charged time.		
	Normal Mode	EAL Mode	Exception cost (Nrm)			Exception cost (Nrm)		
Task	Period /MAT ms	Period /MAT ms	deltaT ms	NRM deltaU %	EAL deltaU %	deltaT ms	NRM deltaU %	EAL deltaU %
Clock	10.0	10.0	0	0.00	0.00	0	0.00	0.00
onemsg_here	100.0	50.0	0	0.00	0.00	0	0.00	0.00
twomsg_here	50.0	100.0	0	0.00	0.00	0	0.00	0.00
fourmsg_here	200.0	########	0	0.00	0.00	0	0.00	0.00
tm_here	62.5	62.5	0	0.00	0.00	0	0.00	0.00
tc_here	187.0	187.0	0	0.00	0.00	0	0.00	0.00
z1_here	100.0	100.0	0	0.00	0.00	0	0.00	0.00
Calibrate_gyro	1000.0	########	0.84	0.08	0.00	0.24	0.02	0.00
Command Actuators	200.0	100.0	0.84	0.42	0.84	0.24	0.12	0.24
EAL Mode	1000000.0	100.0	0.84	0.00	0.84	0.24	0.00	0.24
Normal_Mode	200.0	########	0.84	0.42	0.00	0.24	0.12	0.00
Process_DSS_Data	1000.0	########	0.84	0.08	0.00	0.24	0.02	0.00
Process_IRES_Data	100.0	100.0	0.84	0.84	0.84	0.24	0.24	0.24
Read Bus Ip	10.0	10.0	0	0.00	0.00	0	0.00	0.00
Read_Yaw_Gyro	100.0	100.0	0.84	0.84	0.84	0.24	0.24	0.24
Real_Time_Clock	50.0	50.0	0	0.00	0.00	0	0.00	0.00
Request_DSS_Data	200.0	########	0.84	0.42	0.00	0.24	0.12	0.00
Request_IRES_Data	100.0	100.0	0.84	0.84	0.84	0.24	0.24	0.24
Request_Wheel_Speeds	200.0	########	0.84	0.42	0.00	0.24	0.12	0.00
Telecommands	187.0	187.0	0.84	0.45	0.45	0.24	0.13	0.13
Telemetry_Response	62.5	62.5	0.84	1.34	1.34	0.24	0.38	0.38
Time0_Update	3600.0	3600.0	0	0.00	0.00	0	0.00	0.00
Total				6.16	5.99		1.76	1.71

The cross coupling between the cost of exceptions and the cost of error messages is :

- 0.84 - 0.24 := 0.6 ms per task
- 6.16 - 1.76 = 4.4 % utilisation for Normal Mode
- 5.99 - 1.71 = 4.28 % utilisation for EAL.

References

Ada 9X Mapping/Revision Team1993.

Ada 9X Mapping/Revision Team, Intermetrics, "Ada 9X Reference Manual, Draft Version 3.0", Ada 9X Project Report (June 1993).

Audsley1993.

N.C. Audsley, A. Burns, K.W. Tindell, M.F. Richardson, and A.J. Wellings, "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling", Software Engineering Journal, 8(5), p284-292 (September 1993).

Bailey1991.

C. Bailey, "Survey of Typical Space Applications", Task 6 Deliverable on ESTEC Contract 9198/90/NL/SF, British Aerospace Space Systems Ltd. (September 1991).

Bailey1992.

C. Bailey, "Software Requirements Document for the Olympus AOCS", Task 10 Deliverable on ESTEC Contract 9198/90/NL/SF, British Aerospace Space Systems Ltd. (March 1992).

Bailey1993.

C.M. Bailey, E. Fyfe, T Vardanega and A.J. Wellings, "The Use of Preemptive Priority-Based Scheduling in Space Applications", Proceedings Real Time Systems Symposium, IEEE Computer Society, North Carolina, pp. 253-257 (December 1993).

Burns1990a.

A. Burns and A.J. Wellings, "Real-time Ada: Outstanding Problem Areas", Proceedings of the 3nd International Workshop on Real Time Ada Issues, ACM Ada Letters, Ada Letters, Vol. X(4), pp. 5-14 (1990).

Burns1990b.

A. Burns and A.J. Wellings, "Usability of the Ada Tasking Model", Proceedings of the 3nd International Workshop on Real Time Ada Issues, ACM Ada Letters, Ada Letters, Vol. X(4), pp. 49-56 (1990).

Burns1992a.

A. Burns and A.J. Wellings, Hard Real-time HOOD: A Design Method for Hard Real-time Ada 9X Systems, Towards Ada 9X, Proceedings of 1991 Ada UK International Conference, IOS Press (1992).

Burns1992b.

A. Burns and A.J. Wellings, "Designing Hard Real-time Systems", pp. 116-127 in Ada: Moving Towards 2000, Proceedings of the 11th Ada-Europe Conference, Lecture Notes in Computer Science Vol 603, Springer-Verlag (1992).

Burns1993a.

A. Burns and A.J. Wellings, "Bridging the Real-time Gap between Ada 83 and Ada 9X", pp. 71-86 in Ada Year Book, ed. C. Loftus, IOS Press (1993).

Burns1993b.

A. Burns, A.J. Wellings, C.M. Bailey and E. Fyfe, "The Olympus Attitude and Orbital Control System: A Case Study in Hard Real-time System Design and Implementation", YCS 190, Department of Computer Science, University of York (1993).

Burns1993c.
A. Burns, A.J. Wellings, C.M. Bailey and E. Fyfe, "The Olympus Attitude and Orbital Control System: A Case Study in Hard Real-time System Design and Implementation", pp. 19-35 in Ada sans frontieres Proceedings of the 12th Ada-Europe Conference, Lecture Notes in Computer Science 688, Springer-Verlag (1993).

Burns1993d.
A. Burns, A.J. Wellings and A.D. Hutcheon, "The Impact of an Ada Run-time System's Performance Characteristics on Scheduling Models", pp. 240-248 in Ada sans frontieres Proceedings of the 12th Ada Europe Conference, Lecture Notes in Computer Science 688, Springer-Verlag (1993).

Burns1994.
A. Burns and A.J. Wellings, "HRT-HOOD: A Design Method for Hard Real-time Ada", Real-Time Systems, Vol. 6, pp. 73-114, Also appears as YCS 199, Department of Computer Science, University of York (1994).

Defense1983.
U.S. Department of Defense, "Reference Manual for the Ada Programming Language", ANSI/MIL-STD 1815 A (January 1983).

Forsyth1992.
C.H. Forsyth, "Implementation of the Worst-Case Execution Time Analyser", Task 8 Volume E, Deliverable on ESTEC Contract 9198/90/NL/SF, York Software Engineering Limited, University of York (June 1992).

Hutcheon1992.
A.D. Hutcheon, "Timings of Run-time Operations in Modified York Ada", Task 8 Volume C, Deliverable on ESTEC Contract 9198/90/NL/SF, York Software Engineering Limited, University of York (July 1992).

ILeung1982.
J.Y.T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks", Performance Evaluation (Netherlands), Vol. 2(4), pp. 237-250 (December 1982).

Liu1973.
C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", JACM, Vol. 20(1), pp. 46-61 (1973).

Locke1992.
C.D. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives", Real-Time Systems, Vol. 4(1), pp. 37-53, Real-Time Syst. (Netherlands) (March 1992).

Sha1990.
L. Sha and J. B. Goodenough, "Real-Time Scheduling Theory and Ada", IEEE Computer (April 1990).

Tindell1992.
K. Tindell, "Using Offset Information to Analyse Static Pre-emptive Scheduled Task Sets", YCS 182, Department of Computer Science, University of York (September 1992).