USING A WORST-CASE EXECUTION TIME TOOL FOR REAL-TIME VERIFICATION OF THE DEBIE SOFTWARE

Niklas Holsti, Thomas Långbacka, Sami Saarinen

Space Systems Finland Ltd.
Kappelitie 6, FIN-02200 ESPOO, FINLAND
Phone +358 9 61328600, Fax +358 9 61328699

{Niklas.Holsti, Thomas.Langbacka, Sami.Saarinen}@ssf.fi

ABSTRACT

Real-time software performance can be verified by testing or by static analysis. We report a static analysis of the on-board software of the Debie instrument, using a new tool called Bound-T that gives bounds on the worst-case execution time of an executable (binary) program. The values are used in an HRT schedulability analysis. The Bound-T tool is being developed by Space Systems Finland and has a modular design for porting to various target processors and platforms; initially the Intel-8051 and ADSP-21020 are supported. We describe the methods and results and compare them to the original test-based verification of the Debie software.

1. INTRODUCTION

This paper describes how we verified the real-time performance of the on-board software for the Debie instrument, using the ESA HRT methodology and a new tool for computing the worst-case execution times (WCET) of computer programs. We begin by sketching the Hard-Real-Time model and the data it needs and provides. As an example, we discuss the on-board software for Debie, an ESA instrument that monitors space debris and micrometeoroids *in situ* by detecting impacts using mechanical and electrical sensors. The program contains periodic and sporadic tasks with response times from 1 ms to 1 s.

Next, we present *Bound-T*, the new WCET tool being developed by Space Systems Finland (SSF). The tool analyses executable machine code (i.e. a compiled and linked program). The tool is largely independent of target programming language and is adaptable to various target processors. The tool is being implemented under ESA contract for the TSC-21020 space DSP. SSF adapted the tool for the Intel 8051 architecture used in Debie.

The main part of this paper reports on the application of the Bound-T/8051 tool to the Debie on-board software, and how this provided the necessary data for an HRT verification. The Debie software performance was originally verified by testing, and so we can compare the two methods.

The Bound-T tool is being extended and applied to new target processors with the support of an ESA technology-transfer contract. We aim for a commercial tool of interest to developers of real-time software in all domains.

2. THE HRT METHODOLOGY

The Hard Real-Time (HRT) methodology (British Aerospace Space Systems 1991; Vardanega 1996) defines a way to model the real-time structure of a program and an algorithm for verifying the schedulability of the program, i.e. that all tasks can meet their deadlines in the worst case, given the WCETs of the program's components. The HRT model is similar to the "Ravenscar" subset of Ada.

Tasks (also called *threads*) are the concurrent processes that animate the program. In the HRT model a task is either a *cyclic task*, an *interrupt sporadic task* or a *software sporadic task*. A cyclic task is activated periodically with a constant period. A software sporadic task is activated by some other task(s) when needed, but with some minimum interval between activations. An interrupt sporadic task is activated by a hardware interrupt (IT).

Once a task is activated, it performs some computation (perhaps including input and output) and then suspends itself to wait for the next activation. The *deadline* of a task is the maximum duration allowed for any activation of the task to complete. As part of the computation, the task may call services of *protected objects*.

A protected object is either a *resource* object or a *synchronisation* object. A resource object provides a set of services (entries, subprograms) that let two or more tasks access some shared data in mutual exclusion. A synchronisation object provides the means to activate a sporadic task. It provides two services: a "wait for activation" service for use by the sporadic task, and an "activate" service for use by the activating tasks. Each synchronisation object is statically associated with a particular software sporadic task, and vice versa.

3. HRT SCHEDULABILITY ANALYSIS

The tasks in an HRT model of a real-time program can be scheduled in various ways. The *deadline-monotonic* scheduling method gives each task a priority proportional to the tightness of its deadline and runs the highest-priority ready task (with pre-emption). It is then possible to compute whether the tasks can meet their deadlines, given the WCET of each task and each protected object service.

For each task and protected object, an *execution skeleton* is also required. The skeleton abstracts the worst-case code execution path into a list of computations with WCET times separated by calls to protected objects. The speed of the real-time kernel (*e.g.* Ada run-time-system) also influences the analysis. The execution times of the task scheduler, lock manager *etc*. must be known, giving an HRT characterisation of the kernel.

4. FINDING WORST CASE TIMES

The worst-case execution time of a program or subprogram can be found in two ways: by measuring the execution of *test cases*, or by a *static analysis* of the program code.

The test approach has several problems. It requires runnable code: an integrated program and a real or simulated responsive environment. The measured time may be skewed by asynchronous interrupts and the like. However, the main problem is that in practice the test path coverage is far from 100 % and thus the measurements only give a lower bound on the real WCET, with unknown accuracy, even if great effort is spent on finding the "worst case". Basing a schedulability analysis on such WCET approximations is risky.

Static analysis generally overestimates the possible paths and gives an upper bound on WCET. Still, schedulability analysis using overestimated WCETs is safe. Furthermore, there are prospects for improving estimates by using

1. programmer guidance and assertions, usually given as source-level annotations, and
2. improved data-sensitive analysis of the code such as elimination of infeasible paths, or bounds on loop counts.

Thus static analysis appears to be the more promising approach (to be supported by timing measurements in validation tests, of course).

Our tool, Bound-T, uses static analysis. It reads the executable program and decodes machine instructions and so constructs the control flow graph of the subprogram being analysed and of all subprograms it calls. For each node in the flow graph, the tool computes the total (worst-case) execution time of the machine instructions.

The tool identifies loops in the flow graph and analyses their data flow to find the loop-counters and their bounds. For complex loops, when the tool cannot automatically determine the number of repetitions, the user asserts repetition bounds in a separate assertion file. Once the bounds on loop repetition are known, the tool uses Integer Linear Programming (ILP) to find an upper bound on the worst-case execution time of each subprogram.

5. THE DEBIE SOFTWARE

The Debie instrument (Drolshagen 1999) consists of up to four Sensor Units mounted on the exterior of a satellite and one Data Processing Unit mounted inside the satellite. Each Sensor Unit is a flat square box, with most of the outward-facing surface formed by aluminum foil, about 10 cm by 10 cm in size and 5 um thick. Particle hits on the foil are sensed mechanically by two piezoelectric sensors in contact with the foil, and electrically by three charged grids that enclose but do not touch the foil. The grids detect plasma ejected by the impact. On the outer side are two grids, for ions and electrons respectively. The single grid on the inner side of the foil detects electrons when the hit is strong enough to penetrate the foil. The sensor signals are amplified and fed into peak-hold circuits for measurement.

The Debie Data Processing Unit uses an 80C32 processor, a member of the Intel 8051 8-bit architecture. It has 32 kB of program memory and 64 kB of data memory and runs at about 1 million 8-bit instructions per second. Debie was developed for ESA with Patria Finavitec Systems as prime contractor. SSF implemented the DPU software, which has the following functions:

- Detection, measurement, classification and storage of debris hits.
- Telecommand reception and execution, including instrument configuration.
- Telemetry generation and transmission.
- Monitoring and housekeeping tasks, including periodic monitoring of supply voltages, sensor temperatures and program-memory checksums.

A hit of sufficient size to pass the programmed thresholds generates an interrupt in the DPU. The interrupt handler must read out the impact data within 2.5 ms of the impact, to avoid droop in the peak-hold signals. The complete handling of the event takes quite a bit longer, and the less urgent actions are placed in another task, activated by the interrupt handler but of lower priority.

Telecommand reception and execution is also divided into an interrupt handler and a lower-priority task, because the first acknowledgement of each 16-bit telecommand must be sent within 1 ms, but commands take up to 10 ms to

execute. Telemetry is transmitted by an interrupt handler, which transmits the next TM word from a prepared buffer within 1 ms of the interrupt.

Monitoring and housekeeping functions are implemented by a single periodic task running at 1 Hz. There are several functions with different periods, from 1 s to 180 s, which should logically be separate tasks, but the single-task solution was chosen because of capacity limitations in the real-time kernel (Keil RTX-51) and to avoid sharing the A/D converter between tasks. The functions with periods larger than one second are programmatically "sliced" so that a part of the function is done in each 1 Hz activation.

The tasks communicate through mailboxes for synchronisation and small messages, with bulk data in shared variables.

6. WCET ANALYSIS OF DEBIE

This section describes how we used a prototype of the Bound-T tool to compute WCETs for Debie as input to an HRT analysis. The Debie project finished before we started the work here reported. The design of the Debie code considered neither WCET nor HRT analysis, and its performance was verified and validated by testing, with some support from WCET analysis with Bound-T. Thus, our analysis results are independent of the testing but can be compared to the test results.

6.1 Sources and Methods

We started from the binary of version 1.4a of the Debie on-board software, consisting of Debie application code written by SSF, run-time libraries from the Keil C51 compiler, and the Keil RTX-51 kernel (from Keil Software, Keil Electronic GmbH). The application code has 11 574 lines of C and 292 lines of assembly language, including comments and headers. The executable is 23 944 bytes, including the kernel and libraries. Source code was at hand for the application code but not for the libraries or kernel.

Static analysis can be prevented by difficult code such as indexed jumps. We did not try to identify such problems in the code beforehand, but just asked Bound-T to compute the WCET of each Debie subprogram that is a task body. Problems detected by Bound-T were iteratively corrected until the analysis was successful, either by changing the Debie source code slightly and rebuilding the binary, or by assertions to support and guide Bound-T. We also extended Bound-T to handle indexed branches (switches) as generated by Keil C51, but this is of general use and is not a Debie-specific change.

At the time of this work, Bound-T did not implement the automatic data-flow analysis. Therefore, the maximum number of repetitions of each loop was manually asserted. As an example, here are the assertions for the *HitTriggerTask* subprogram which handles a particle hit interrupt. Loops are identified by characteristics such as nesting and subprogram calls.

```
subprogram "HitTriggerTask"
   loop_that calls "_SHORTDELAY" and is_in (loop) repeats 4 times;
   end_loop;
   loop_that calls "_WaitInterrupt" repeats 1 times; -- The "forever" loop.
   end_loop;
```

The HRT analysis also requires an upper bound on the duration of interrupt blocking. We used Bound-T to display the worst-case paths of all Debie subprograms with regions of interrupt blocking, and computed the duration of each region as the difference in cumulated time between entering and leaving the region.

6.2 Source Code Changes

To match the needs of Bound-T, the Debie source-code was changed in several places but only in minor ways. Eternal loops (*while* (1) {...}) were made to use a variable (*while* (*forever*) {...}) and asserted to run once. Assignment of *struct* values was replaced by component assignments, or by a byte-wise copy function, because the C51 library routine for *struct* assignments contains indexed branches that Bound-T could not follow. For the same reason, local arrays and structures initialised to constant values were moved to file scope.

Some subprograms contained loops that could not be separately identified with the available subset of Bound-T assertions. Two such loops in Debie were in fact duplicated code (to wait for the end of an A/D conversion) and were merged into a new subprogram. The remaining case was the C51 floating-point addition routine, which was analysed by running Bound-T on this routine interactively. In interactive mode, Bound-T asks the user for bounds on each loop in turn. The resulting WCET on this routine was asserted for analysing Debie.

The C51 routine for floating-point division was found to have an "irreducible" loop structure, which means that the loops intersect rather than nest. This routine was avoided by changing a formula in Debie to use multiplication instead of division. C51 routines for 32-bit operations with one literal operand were found to use non-standard calling

sequences, so literals in Debie were replaced by variables initialised to the literal value. The analysis of RTX-51 kernel routines is discussed in section 7.2.

Finally, the Debie telecommand *Soft-Reset* caused a recursive call cycle, which Bound-T does not handle. This TC calls the *Reboot* function which again invokes the Debie software. An assertion on the WCET of *Reboot* solved this problem.

6.3 Effort and Results

The numerical WCET results are discussed in the next section. Here we evaluate the analysis effort, the usability of the Bound-T tool, and the precision of the analysis results.

Six of the twenty-nine Debie source code files were modified, for a total of 68 modified lines (deleted, changed, or added). The executable size decreased to 23 525 bytes. Naturally no functional changes were introduced. A total of 47 assertions were required, 38 for loops, five for kernel routines, two for C51 library routines, and two for Debie subprograms (*Reboot*, and *PatchCode*, of which more below). Of the 38 loops with assertions, at least 17 and perhaps as many as 30 will be bounded automatically in the completed Bound-T tool.

The analysis took much less time and effort than the timing tests performed in the Debie project. Simulation of the I/O and hardware environment was not required, and no test cases had to be designed. Most of the assertions were quite easy to develop, but some disassembled code had to be read and understood (an alternative would be to time these routines with tests, and assert the times for Bound-T).

As expected, the WCET values were generally slightly larger than the measured times. In some cases we deliberately decreased loop bounds to correspond to the test cases, where the absolute WCET would be much larger, but this was necessary only for time-out loops (e.g. polling for end of A/D conversion) or to match assumed processing-load scenarios (e.g. number of hits per second).

7. HRT ANALYSIS OF DEBIE

7.1 Debie HRT structure

An HRT analysis starts by listing the threads and protected objects and their attributes. If the system can operate in different modes, with different real-time behaviour, each mode has its own version of this list and its own HRT analysis.

For Debie, we distinguish three modes: *TM mode*, *TC-buffering mode*, and *TC-execution mode*. In TM mode, a TM block is being transmitted, TM IT occurs at 1 ms intervals, and no TC ITs occur. When no TM block is being transmitted, Debie can receive and execute TCs. Most TCs consist of one 16-bit word and cause one TC IT; their reception and execution at 1 s intervals is described by TC-execution mode. TCs for patching memory consist of several words and cause as many TC ITs; the reception and buffering of such TC words at 10 ms intervals is described by TC-buffering mode. The difference between the two TC modes lies in the duration and frequency of TC executions. No TM ITs occur in the TC modes. The hit-handling and monitoring and housekeeping tasks are the same in all modes.

The following tables define the HRT structure of the Debie software (IT = interrupt, DL = deadline, MI = minimum interval, SU = Sensor Unit, HK = housekeeping). The WCET values come from Bound-T, with some modifications as explained in section 7.3. All times are in milliseconds. The table of task finishing times, which is the result of the HRT schedulability analysis, is included here for formatting reasons but is discussed in section 7.3.

Table 1. Cyclic Tasks

Mode	*Task*	*Period*	*WCET*
all	Monitoring & HK	1000	115.17

Table 2. Sporadic Tasks

Mode	*Task*	*DL*	*MI*	*WCET*
TM	TM IT	1.0	1	0.17
TC buffer	TC IT	1.0	10	0.56
TC buffer	Exec./buff. TC	10.0	10	3.15
TC exec	TC IT	1.0	1000	0.56
TC exec	Execute TC	10.0	1000	5.23
all	Hit IT	2.5	1000	1.57
all	Store hit	997.0	1000	353.26

Table 3. Protected Objects

Object	*Type*	*User task*	*WCET*
TC mailbox	Sync	TC IT (send)	0.057
		Execute TC (receive)	0.177
Hit mailbox	Sync	Hit IT (send)	0.102
		Store hit (receive)	0.177
SU State	Resource	Execute TC	0.200
		Monitoring & HK	0.200

Table 4. Task Finishing Times

Task	*DL*	*TM*	*TC buff*	*TC exec*
TM IT	1.0	0.386		
TC IT	1.0		0.777	0.777
Hit IT	2.5	2.473	2.492	2.492
Execute TC	10.0		6.670	8.754
Store hit	997.0	437.668	679.540	363.039
Monit. & HK	1000.0	580.514	905.153	479.287

7.2 HRT attributes of the RTX-51 kernel

The HRT schedulability test uses timing attributes of the platform (kernel), such as the worst-case time to switch tasks and the resolution of a timed delay. For the Keil RTX-51 kernel as used in Debie, there are two problems in finding these attributes. Firstly, the services provided by RTX-51 are not exactly matched to the HRT model, so a mapping or selection is needed. Secondly, the RTX-51 data sheet (Keil Software 1996) gives "average" times, not worst-case times.

We could not use Bound-T to find WCETs for most RTX-51 services because they invoke task switching, which is complex code (the control-flow graph has hundreds of edges and many loops) that does not follow normal calling sequences. When Bound-T succeeded for an RTX-51 routine, the WCET was comparable to the "average" time. For example, for the service *os_detach_interrupt* the "average" time is given as 94 cycles and Bound-T gave a WCET of 102 cycles. For the Debie analysis, we used a rough and no doubt inaccurate set of HRT attributes for RTX-51, based mainly on the "average" times given.

RTX-51 provides three types of task. "Standard" tasks, periodically activated with the wait-for-timeout function, are modeled as cyclic tasks in HRT. "Standard" tasks activated by mailbox messages are modeled as (software) sporadic tasks, with the mailbox as a protected object. Interrupt-sporadic tasks model the RTX-51 Interrupt Service Routines (ISR, directly vectored) and also the RTX-51 "fast" tasks that are activated through *os_attach_interrupt*.

A notable feature of RTX-51 is that the time to activate a task (i.e. to release it and switch to it) is very different for the three task types, from 14 cycles to activate an ISR with a hardware interrupt, up to 459 cycles to activate a standard task with a mail message (these are "average" times). Debie uses all three task types: ISRs for the TC IT and TM IT, a "fast" task for the Hit IT, and "standard" tasks for the TC Execution, Hit Storing and Housekeeping/Monitoring.

The Debie code uses RTX-51 mailboxes for task activation. Mailboxes are modeled as HRT synchronisation objects. Updates of shared variables are protected by interrupt disabling (RTX-51 provides semaphores, but these were not used in Debie). The interrupt-blocking regions are modeled as HRT resource-object services, and so the HRT analysis is based on interrupt disabling rather than priority ceiling inheritance, which is not supported by RTX-51.

7.3 Analysis results

HRT analyses were done with an internal SSF tool that implements the basic HRT schedulability test. The HRT Execution Skeleton of each task was defined manually, but the full Bound-T tool will generate them automatically.

The initial analysis results showed deadline misses. We compared the analysis results and WCETs to the Debie timing-test results and the Debie timing model and traced the problems to several causes. Firstly, a poor implementation of a change requested (during the Debie project) to a Debie routine had caused an inadvertent increase in interrupt blocking to 413 microseconds, not reflected in the timing tests. A software correction may be needed to reduce the blocking time. An estimate of the corrected blocking time was used in our HRT analysis.

Secondly, a special function of the TC execution task (clearing the Science Data area after it has been sent in TM) exceeded the 10 ms deadline in the worst-case scenario. This is not a serious problem, because the time can be absorbed into the TM time (over 30 seconds for a full data set). We modified a loop assertion to exclude this function from the analysis of normal TC execution.

Thirdly, after the above changes, the WCET of TC execution was dominated by the *PatchCode* subprogram. This implements the program-patching TC, a non-nominal TC with unusual timing constraints. We excluded this TC from the analysis by asserting a minimal WCET for *PatchCode*.

Finally, the WCET for the TM interrupt service included a heavy action (freezing the 4-byte "DPU time" field in the TM buffer) that occurs rarely. In the TM mode, where TM IT occurs at 1000 Hz, Hit IT was unschedulable because of interference from the TM IT. The time for this rare action (44 cycles as shown by Bound-T) was subtracted from the TM IT WCET for the TM mode analysis.

After these changes, the HRT analysis was successful. Table 4 in section 7.1 shows the main result: the finishing time of each task for each mode. The times are milliseconds from task activation; the deadlines (*DL*) are shown for comparison.

The deadline margins from this analysis are smaller than indicated by the tests. Apart from the RTX-51 modeling approximations, we think that there are two reasons for this difference. Firstly, the real worst-case scenarios were not tested, because of the difficulty of controlling the testing environment (we used the Keil dScope debugger and 8051 simulator). Secondly, some changes in the Debie software were not retested at SSF for timing, because their effect was judged to be minor and because the hardware-based validation environment and EGSE were by then available and system-level timing-tests were easier. Thus, the measured times refer in part to earlier versions of the Debie software, while the static analysis applies to the latest version.

7.4 Conclusions

Our main conclusion is that the Bound-T tool would have been very useful for developing the Debie software, had it been available during the Debie project. Perhaps its most important role would have been to measure WCETs early in the coding phase, before any code was runnable and before the testing system was available. This would have detected the code parts that were poorly suited to the 8051 architecture or to this compiler. Early changes in this code would have had little impact on schedule or cost, but a large impact on performance.

"Regression measuring", i.e. recomputing WCET values after software changes, would also have been far easier than rerunning the timing tests, because Bound-T can be run from a shell-script and is insensitive to small changes in the code of the target program. Assertions and run-commands need to be updated only if the subprograms or loops they refer to are essentially changed, removed or renamed.

8. FUTURE OF THE WCET TOOL

We are developing Bound-T, our WCET tool, into a commercial product. This work is supported by ESA under a technology-transfer contract. The chief areas of development are, firstly, support for additional target processors, and secondly, improvements in the interfaces between the tool, its users and other tools such as compilers. Since Bound-T analyses binary code rather than source code, it is largely independent of the target programming language. It may need adaptations for compiler-specific code patterns; an example from the Keil C51 compiler was noted in section 6.1.

The tool is implemented in Ada, using some of the new features of the 1995 Ada standard: modular types, child packages, string handling, stream input-output. Components used include the *Ada Structured Library* by Corey Minyard, the *OpenToken* lexical-analysis library of Ted Dennison, the *Omega Calculator* by William Pugh et al., and the *lp_solve* ILP program by Michel Berkelaar.

Only two tool modules, *Processor* and *Decoder*, depend on the target processor. For the Intel 8051, these modules are approximately 1k lines of Ada (excluding the data-flow analysis), while the generic part of the tool is approximately 10k lines at present. A third module, *Format*, handles the executable file format, and was about 1.5k lines for the AOMF format of the Keil C51 toolchain. This shows that the tool is indeed adaptable to quite different target architectures.

Currently, the tool runs on the Sun SPARC/Solaris and Intel/Linux platforms with the GNU Ada compiler Gnat. The only platform dependency in the Ada code is in the record types that define executable instruction formats (endianness and bit numbering issues), and automatic conversions can be implemented for these. We expect that the tool can be deployed on any platform with an Ada compiler, including MS Windows NT systems.

We are looking for people who would be interested in using Bound-T in pilot projects. We welcome suggestions for improvements and in particular suggestions for additional target processors.

References

British Aerospace Space Systems Limited 1991:
Hard Real-Time Operating System Kernel Study.
TP887.

Drolshagen G 1999:
Standard In-Situ Impact Detector (Debie).
Preparing for the Future Vol. 9 No. 3, p. 24, ESA, *http://esapub.esa.int*.

Keil Software 1996:
RTX-51: Real-Time Multitasking Executive for the 8051 Microcontroller Family.
Keil Elektronik GmbH, Germany.

Vardanega T 1996:
Tool Support for the Construction of Statically Analysable Hard Real-Time Ada Systems.
Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS'96), pp. 129-135,
IEEE Computer Society Press.