Disclaimer

This copy is a preprint of the article self-produced by the authors for personal archiviation. Use of this material is subject to the following copyright notice.

IEEE Copyright notice

Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ 08855-1331, USA. Telephone: + Intl. 908-562-3966.
Measurement and analysis of schedulability of spacecraft on-board software

Nunzio Cecere‡, Massimo Tipaldi†, Ralf Wenker‡, and Umberto Villano‡

‡Intelligentia s.r.l., Via del Pomerio 7, 82100 Benevento (Italy),
Email: nunzio.cecere@intelligentia.it

*Compagnia Generale per lo Spazio, Via Gallarate 150, 20151 Milano (Italy),
Email: mtipaldi@cgspae.it

¶OHB System AG, Universittsallee 27-29, 28359 Bremen (Germany),
Email: ralf.wenker@ohb.de

†University of Sannio, Department of Engineering, P.zza Roma, 82100 Benevento (Italy),
Email: villano, mtipaldi@unisannio.it

Abstract—Software schedulability analysis is an important aspect for SW real-time verification. This paper proposes a concrete approach for analyzing the schedulability for the SW real-time application running on a spacecraft on-board computer. It is based on an extension of the response time analysis and takes into account many factors that could have a significant impact on the corresponding assessment. Measurements and analysis of schedulability have been carried out in a real space SW project and show the importance of focusing on this aspect in the first phases of the SW development.

Keywords—SW schedulability analysis; spacecraft on-board software; response time analysis; worst-case execution time.

I. INTRODUCTION

Software schedulability analysis has been widely studied in the SW real-time literature (see [1] and [2]). In real-time systems, the correctness of an application does not depend only on the value of the result, but also on when the result is released [3]. The resulting value of an operation can be correct, however the real-time task has not finished its execution phase within its deadline. Hard and soft timing requirements have to be distinguished to capture different temporal needs. While missing a hard timing requirement results in a system failure, missing a soft timing requirement may decrease system performance but does not harm the system integrity [4]. In the first place, the objective of the schedulability analysis is to show that the software is schedulable, hence all defined deadlines are obeyed under all conditions. Therefore, an applicable computational model that enables the analysis of the system schedulability has to be established. Based on this computational model suitable formal verification methods are applied showing the feasibility of the software task set.

The paper has been organized as follows. In section II spacecraft On-board SW (OBSW) architecture of the case study Meteosat Third Generation satellite (MTG), a cooperative undertaking between the European Space Agency (ESA) and the European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT), is presented. Particular emphasis has been given to aspects concerning the SW task definition and interaction. In section III the proposed model based on the extension of the response time analysis [5] is discussed. Section IV reports the schedulability analysis of our case study. Finally, section V concludes the paper with some ideas for future works.

II. SPACECRAFT OBSW ARCHITECTURE

The functional architecture of the On-Board Software running on the MTG Satellite main computer is shown in Fig. 1. In the paper, this SW is referred to as Satellite Control Software (SCSW). It is composed of the following three layers:

- Application Layer: the high level software taking care of the spacecraft system and subsystem management.
- Data Handling SW Layer: middle level software, divided into a set of services and platform device controllers.
- Low Level SW Layer: containing the Real-time Operating System (RTOS) and the hardware drivers.

Since SCSW anomalous behavior can cause or contribute to a system failure impacting the satellite safety and dependability, its criticality level has been assigned to the category B. SCSW is developed according the ECSS standards [6]. All these aspects determine the SW verification and validation needs and effort, including the SW schedulability analysis. The main hardware resources available for the SCSW are the following:

- CPU: LEON2 [7], SPARC V8 architecture [8], 64 MHz, about 45 MIPS.
- Main memory: 8 MB SRAM.

Each task in the SCSW is either a cyclic or a sporadic task. Cyclic tasks are triggered on a periodical basis, whereas the sporadic tasks are only triggered sporadically by internal or external events (for instance, data ready to be processed). Tasks are classified as follows:

- Driver tasks: hardware controllers are supervised by dedicated SW tasks, which need to wait for interrupts, e.g. when a certain data transfer to a satellite equipment is terminated. They are sporadic, however subject to time responsiveness.
Fig. 1: SCSW functional architecture

- Synchronous tasks: this category mainly consists of cyclic tasks performing the processing of data acquired by the driver tasks and providing commands for the next cycle to apply to the hardware. For the SCSW, there are cyclic tasks with a frequency of either 10 Hz or 1 Hz.

- Background tasks: this category hosts tasks not subject to any hard deadline and of rather a-periodic but potentially long-term nature. For instance, the dumping of large memory areas belongs to this category.

- RTOS tasks, e.g. the scheduler itself.

Data integrity has to be guaranteed when the SCSW accesses shared data in order to avoid task conflict and data corruption. For this purpose, different mechanisms of accessing data are adopted, that is to say:

- Space separation: a double buffer (banks A/B) is implemented, the software accesses bank A (respectively B) while bank B (respectively A) is accessed by another entity, for example a hardware controller. After the hardware controller completes its activities, a notification is sent to the software by interrupt. The software processes this notification, swaps the buffer bank usage for the next cycle and re-triggers the hardware process.

- Time separation: this mechanism is used to perform an a-periodic (on request) data update (e.g. spacecraft configuration parameters). The update is executed only by the task using this data, before or after any usage for computation.

- Mutual exclusion: this mechanism is used to access shared data as critical section, such as memory area containing spacecraft house-keeping data.

Data integrity is guaranteed by SW design, by RTOS constructs or both. The operating system used for the SCSW is RTEMS [9] (Real-Time Executive for Multiprocessor Systems). It provides semaphores to protect shared resources from concurrent access and allows the configuration of binary and counting semaphores. Furthermore, it implements two access protocols that mitigate the risk of an unbounded priority inversion: priority inheritance and priority ceiling. For the SCSW, the Immediate Priority Ceiling Protocol (IPCP) has been chosen to modify the task priorities temporarily and, hence allow a successful task scheduling.

III. THE PROPOSED SW SCHEDULABILITY ANALYSIS MODEL

As for the SCSW, priorities of tasks featured by hard real-time deadlines are defined according to the Deadline Monotonic Scheduling (DMS) algorithm, which assigns priorities according to the task relative deadline. The DMS is a generalization of the Rate Monotonic Scheduling (RMS) algorithm [10]. These priorities can be only modified temporarily during their execution in order to deal with synchronization and resource sharing issues. Tasks featured by soft real-time deadlines are given the lowest priorities. The SCSW schedulability analysis is based on the following steps:

- Firstly, a response-time analysis is performed, showing that all the hard deadlines defined for the critical task set are met under all circumstances. The worst-case execution time (WCET) [11] for each of these tasks is measured by static code analysis, and the response times of the timing critical tasks are calculated including blocking and preemption effects.

- Secondly, the schedulability of the activities that are not executed in the context of the critical task set is
analyzed based on a CPU load argumentation. CPU load consumed by the remaining tasks is measured.

- Scenario based measurements are carried out to estimate the total CPU utilization, to check that the CPU load requirements are fulfilled and that critical deadlines are met on the whole.

This approach is modular in the sense that the critical task set and non critical task set are analyzed separately, and then combined in order to obtain the final results.

A. Response time calculation and verification

The response time calculation for each SW task \( \tau_i \in T \), \( i = 1, ..., n \), is hereafter described. Apart from the WCET, various aspects have to be taken into account such as the task context switch and the jitter. They are discussed in this section.

To begin with, RTEMS needs a certain context switching latency to switch between SW tasks. It is denoted by \( CS \). This latency directly affects the response time calculation. The context switching time depends linearly on the size of the respective contexts, on the number of SW tasks and their specific characteristics, e.g. the usage of the floating-point registers. In order to simplify the analysis, it is assumed that all the SW tasks use the Floating Point Unit, thus the values of all its registers are saved/restored on each context switch. Moreover, it is also necessary to consider the installed RTEMS extensions. The SCSW, in fact, adds an extension for the context switch: this means that, for every context switch, a specific function is called.

An Interrupt Service Routine (ISR) interrupts the current SW task, executes some code, and returns to the previous or some other SW task context. Within a time interval \( \Delta_t \), the CPU capacity (in terms of the amount of time) used by the interrupts \( CI(\Delta_t) \) can be calculated as follows:

\[
CI(\Delta_t) = \sum_{r \in I} \Delta_t \cdot [C_r + CIS] ,
\]

where the set of all interrupts is denoted by \( I \), \( C_r \) is the WCET of an ISR, \( T_r \) is its period or minimum inter-arrival time, and \( CIS \) is the ISR context switch. We assume that the interrupts are not correlated, thus within an infinite \( \Delta_t \), all the interrupts can be raised.

As for the cache effects, the LEON 2 processor [12] implements the Harvard architecture with separate instruction and data buses connected to two independent cache controllers. The instruction cache is 32 Kbytes, four-way set associative, and the data cache is 16 Kbytes, two-way set associative. The Least Recently Used (LRU) replacement algorithm is used. For the schedulability calculation respecting Cache Related Preemption Delay (CRPD) effects, the Block Reload Time (BRT) is needed [13]. The BRT describes the actual penalty in case of a cache miss, that is to say the number of cycles needed to fetch a 4-byte cache block. In our case, an SRAM read without error takes certain number of cycles and additional wait states by access to the SRAM module. In order to include CRPD in the schedulability calculation, the maximum number of Useful Cache Blocks (UCBs) is measured for each SW task. A UCB is defined as a memory block that is reused after being cached (leading to a cache hit) at a later point in the execution path. The effect on the response time of task \( \tau_i \) interrupted by task \( \tau_j \), has also to take into account nested preemptions, since \( \tau_j \) may preempt tasks of intermediate priority. The worst-case Cache Related Preemption Delay (CRPD) for \( \tau_i \) is given by the following formula:

\[
CRPD_{i,j} = BRT \cdot \max_{k \in hep(i) \cap lp(j)} \{UCB_k\} ,
\]

where \( hep(i) \) is the set of all tasks with a priority that is higher or equal to the priority of task \( \tau_i \), and \( lp(j) \) is the set of tasks with a lower priority than task \( \tau_j \). Note that the maximum UCB of any task that may be preempted by task \( \tau_j \) and also may preempt task \( \tau_i \) as well as the task \( \tau_i \) itself has been considered. While the cache improves the performance, it adds uncertainty to the WCET as the cache content may be (partly) destroyed due to task context switches. In the SCSW, caches are not frozen for ISR routines and as such they may evict the cache content. As cache freezing is not supported for ISRs, the execution of ISRs may also invalidate the cache content. To model an upper bound for the cache related delay caused by ISRs, the response time of task \( \tau_i \) has to consider the worst-case Cache Related ISR Delay (CRID) given by the following formula:

\[
CRID_{i}(\Delta_t) = \sum_{r \in I} \Delta_t \cdot BRT \cdot UCB_i .
\]

A lower-priority task can block a higher-priority task for the duration it holds a semaphore used to protect a critical section of code or a critical data structure. Every object or class in the model may have a semaphore. Operations of that object or class are protected if they use the respective semaphore object. Since different tasks might use different guarded operations, each task executing such guarded operations has a maximum WCET during which it holds the semaphore object. The safe upper bound for the WCET of the task \( \tau_i \) that holds semaphore \( k \in K \) is denoted by \( C_i(k) \). Every semaphore has a ceiling priority associated with, which is defined as the highest priority of any task executing the critical section:

\[
CP_k = \max_{i \in \{1, ..., n\}} \{ P_i | \tau_i \in T \text{ executing critical section } k \} .
\]

According to the Immediate Ceiling Priority Protocol, the blocking time \( B_i \) of a task \( \tau_i \) is defined as the longest time the task can be blocked by a task \( \tau_j \) with \( P_j \leq P_i \). Therefore, \( B_i \) is the largest WCET of any critical section executed by any task of lower priority that has a ceiling priority equal or larger than \( P_i \):

\[
B_i = \max_{k \in K} \{ usage(k, \tau_i) \cdot C_i(k) \} ,
\]

where:

\[
usage(k, \tau_i) = \begin{cases} 1 & : k \text{ used by at least one task } \tau_j \\ \text{ with } P_j < P_i \text{ and } P_i \leq CP_k & \\ 0 & : \text{ in all other cases} \end{cases}
\]

The release jitter \( J_i \) of any SW task \( \tau_i \) is determined by three factors, that is to say the granularity of the system timer,
the system timer deviation, and the interrupt disabling. The granularity of the system timer does not have any influence in case of the SCSW, as periodic tasks are activated by events issued from ISRs activated by the On-Board Time (OBT). The system timer deviation during the smooth clock adjustment of the three involved OBT units does not exceed a fixed value per definition of the smooth synchronization algorithm implemented in the On-board Computer. This jitter affects only the release of the periodic tasks activated by the OBT. Some RTEMS and SCSW routines disable the interrupts in order to process inherently critical source code. Additionally, a few driver routines use the same mechanism in order to guarantee timing properties when accessing the hardware. The maximum time an interrupt is disabled is determined under normal conditions. Only when writing data into the EEPROM, interrupts are disabled longer.

By taking into account all the previous considerations, the response time $R_i$ for each SW task $\tau_i$ can be expressed as follows:

$$R_i = CS + B_i + J_i + WCET_i + C1(R_1 + J_1) + CRID_i(R_i + J_i) + \sum_{j \in hp(i)} \left( \frac{R_i + J_i}{T_j} \cdot [WCET_j + 2 \cdot CS + CRPD_{i,j}] \right),$$

(7)

where:

$$hp(i) = \{ \tau_j \in T \mid P_j > P_i \}.$$  

(8)

Since $R_i$ is on both sides of the previous equation, an iterative algorithm is required to determine its fixed point. If no $R_i$ can be determined, the tasks are not processable. In this case, there is too much higher-priority computation for the tasks to be finished. The schedulability criterion for a real-time task $\tau_i$ is the comparison of the calculated response time against the required deadline $D_i$, that is to say each task $\tau_i$ is schedulable within its deadline if and only if $R_i \leq D_i$.

IV. Schedulability Analysis Results in Our Case Study

The approach described in section III is hereafter applied to the SCSW project. In particular, the response time for each SCSW task is calculated by using the expression (7) in order to demonstrate each task can perform its own activities within the required deadline.

One of the most critical SCSW components is the AOCS Core library which implements the Satellite AOCS (Attitude & Control Orbit Subsystem) algorithms. The starting point is to determine its WCET. The AiT WCET Analyser by AbsInt has been used to measure an upper bound of the WCET. Source code instrumentation combined with tool configuration scripts as well as some assumption on the WCET values of external SW components are needed. Fig. 2 shows the AOCS Core WCET along with its most contributing functions. The calculated WCET for the entire AOCS Core library is about 42 ms, too high in order to fulfill the required SCSW CPU load margins.

In order to reduce the WCET, some analysis has been performed both at SW design and implementation levels. Some mathematical functions have been identified as responsible for this misbehavior, and therefore corrected. Fig. 2 shows the AOCS Core library WCET calculation after implementing the identified improvements. Thanks to this optimization, the WCET has been reduced to about 24 ms, see also Table I.

A similar approach has been used to measure the WCET of the other SCSW libraries or tasks. If necessary, similarly to what done for the AOCS Core library, some adjustments have been performed in order to reduce their WCET. Figures 4, 5, and 6 show the results of respectively the 10 Hz tasks, the 1 Hz tasks, and the sporadic tasks.

In order to apply the expression (7), it is necessary to measure the WCET of each ISR routine and critical section. They are also calculated by means of AbsInt. As far as RTEMS is concerned, the WCET of its system calls is provided by the documentation of the RTEMS Improvement project [14], an RTEMS version qualified for the specific needs of the European space industry. For brevity’s sake, we report only the RTEMS WCETs, see Table II.

After having measured or estimated all the necessary inputs, it is possible calculate the Worst Case Response Time...
TABLE I: AOCS Core Library WCET Results

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET (ms)</th>
<th>Improved WCET (ms)</th>
<th>Improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AocCore</td>
<td>42</td>
<td>24</td>
<td>43</td>
</tr>
</tbody>
</table>

(WCRT) for the SCSW tasks. Fig. 7 shows that the SCSW is schedulable since all deadlines are met. However, the Telemetry (TM) Task, although below its deadline, raises some concern: at the time of the writing of this paper, the SCSW is not fully completely implemented and adding new features could lead to scheduling problems; in fact, there are still missing important AOCS algorithms, that could significantly affect the performance. For next versions of the software the results need to be updated and consolidated and it will be crucial to work closely to the SW developers for the timely identification and resolution of any SW schedulability issue. As lesson learnt from previous space SW projects, the authors of this paper encourage starting the SW schedulability analysis from the early phases of the SW design.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented a concrete approach for analyzing the SW schedulability of spacecraft real-time On-board SW. An extension of the response time approach has been proposed. Results on a concrete case study have been included. The approach is valid for fixed priority tasks, with jitter problems and resources constraints. Most of the real-time systems belong to this category. However, for some systems, precedence constraints between tasks have to be modeled by means of more adequate means. A possible direction could be the timed automata, where it is possible to express every kind of constraint of real-time systems. A future work could be the integration of the terms identified by our model like cache, context switching, jitter, and blocking time within a timed automata model.

REFERENCES

Fig. 7: SCSW Worst Case Response Time (WCRT)


