

Schedule Design to Guarantee Freedom of Interference in Mixed Criticality Systems

Christoph Ficek, Kai Richter and Nico Feiertag
Symtavision GmbH

ABSTRACT

The integration of mixed-criticality software according to safety standards like ISO 26262 generates new, parasitic mutual effects within the involved software architectures. In this situation, established schedule design patterns like RMS fail to deliver both efficiency and safety, in particular the freedom of interference. In today's practice of building a schedule, certain such measures to fulfill these safety requirements can conflict with efficiency requirements. The target of this paper is to present a sound approach of how to solve such requirement conflicts and to build up schedules that are safe and also efficient. We present a general early-stage procedure to build safe, certifiable, and efficient schedules. The procedure is based on the established design patterns and adds guidelines on how to exploit additional options in both schedule design and software partitioning. This procedure was validated against typical real-world systems and one example is presented.

CITATION: Ficek, C., Richter, K. and Feiertag, N., "Schedule Design to Guarantee Freedom of Interference in Mixed Criticality Systems," *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.* 5(1):2012, doi:10.4271/2012-01-0036.

INTRODUCTION

Selecting a schedule for a hard real-time system is not an easy task. Meeting the timing deadlines and avoiding overload is important for any embedded system, even more so in safety critical ones. Known design patterns like Rate-Monotonic Scheduling (RMS, short period = high priority) yield compact, resource-efficient preemptive schedules and have been used in the industry for a while (in OSEK [1] and AUTOSAR OS [2]). With the appearance of mixed-criticality ECUs, however, safety standards like ISO 26262 require *freedom from interference* on all different safety levels. Obviously, this conflicts with the resource-efficient RMS approach as in RMS, the period (or cycle time) of an application determines fully the scheduling priority. This does not necessarily reflect the application's safety level - for example: a high critical task may be constantly preempted by a low critical task if the latter has the shorter cycle time. From the safety point of view, the requirement *freedom from interference* is violated. In the certification process this violation must be handled. The later in the development such violations are recognized and counter measures must be applied, the more expensive a system gets. What can be done in such situations? Simply switch priorities according to the criticality levels, e.g. high critical tasks get highest priorities and low critical tasks get lowest priorities? What is the overall influence on performance and timing constraints like

deadlines? How can this be reliably analyzed? What are the consequences of such a design decision and how big is the effort to implement it? How about other counter measures such as AUTOSAR Timing Protection? Is this still certifiable according to safety standards?

Trial and error is a dangerous option and can lead to huge costs in case failures are detected in late design stages. Rather an assessment of all possible options is needed prior to taking the decision. This includes considering the consequences and the cost of each option. For instance, a pure RMS scheduling requires timing protection that might lead to additional software cost; alternatively, priority selection per criticality is typically less resource efficient and might lead to additional hardware cost, etc.

The better and less expensive way is to build up and check a schedule which does not have violations or only those which are known and were handled. To achieve this first a schedule is buildup which guarantees the freedom of interference by priority assignment and the use of Timing Protection functions. In the second step, we check the schedulability and show which counter measures can be applied to make the system schedulable.

In our paper, we are showing the different requirements for safety and efficiency. We especially consider the requirements of the automotive safety standard ISO 26262 and AUTOSAR as OS in the context of mixed criticality

systems. From this, we derive guidelines on how a schedule can be systematically built and present measures fulfilling the requirements (safety and timing). For the measures the advantages and disadvantages are discussed. The goal is to support a developer as much as possible to build up a certifiable schedule as much as possible and in a systematic way. The schedule should be efficient and safe according to ISO 26262 and we want to show how to deal with problems and to estimate consequences of options during a design process. In addition, the required steps to verify the correctness of a taken decision will be shown, for example scheduling analysis to detect deadline misses. The approach is applied to an industry example from the chassis domain which had a specific scheduling problem.

In the next section we will present the schedule's safety requirements. Then general possible priority assignment strategies are presented. In the subsequent section AUTOSAR Timing Protection for execution time bounds is discussed. Then our procedure is presented, showing how the priority assignment strategies and the Timing Protection are used to build up a schedule and fulfill the safety requirements. The possible counter measures for non-schedulable systems are shown, too. The next section applies the approach to an example. The conclusion and outlook are the last section.

In our approach we are using only existing and available mechanisms for industrial systems in the series. Using standardized schedulers and mechanisms can be beneficial for designers and developers. This is the main difference between our approach and others. Many of the approaches in the research community require changing the OS scheduler to support explicit criticality levels. Algorithms are applied during the run time of the system and influence the scheduler. For example in [5] de Niz et al. have introduced a *slack-aware* approach and extended in [6]. This approach calculates offline a time point for a specific task. At the latest the task must start the execution at this point and cannot be disturbed by lower priority tasks to guarantee its deadline. To monitor these points and guarantee the execution, extensions for the implemented scheduler are necessary. The approach sounds reasonable and promises efficiency considering certification issues.

However, the schedule designer in the industry today (we mean series production, not research) cannot use it, because there is no industrial available operating system with such a scheduler, which is also certified or certifiable for today automotive system. In automotive, the operating system and scheduler are mostly just purchased and not developed in-house. Therefore changing the scheduler is not possible.

Work for influencing the RMS pattern for mixed criticality reasons already exists. In [7] and [8] *period transformation* is presented. By splitting the task execution in smaller parts and a more often execution, the RMS pattern will increase the priority of such a task. We are using this approach in a reduced way. Section 6.3.2 shows how.

SAFETY REQUIREMENTS

Mixed criticality means there is a mix of applications, tasks or processes with different criticality levels on one CPU. Safety standards like ISO 26262 (automotive)[9], IEC 61508 (industrial) or DO-178B (avionics) require spatial and temporal separation in such mixed criticality systems. Spatial separation means the isolation of the different memory spaces between the applications. In this paper we will focus on the temporal isolation. This means that the CPU time, which has to be shared between all applications should be allocated correctly from a safety point of view. All standards have nearly identical requirements: It has to be guaranteed that applications with lower criticality cannot interfere with applications with higher criticality. This property is referred to as *freedom of interference*.

In this paper we will focus on the automotive domain where ISO 26262 is the certification standard. ISO 26262 defines four criticality levels called *Automotive Safety Integrity Level* (ASIL), derived from IEC61508 SIL levels. The levels are ASIL-D (highest criticality), ASIL-C, ASIL-B and ASIL-A (lowest criticality level). Furthermore "no criticality level" (often named as QM for Quality Management) is defined in the standard.

ISO 26262[9]-Part 6 Annex D (informative annex) proposes several mechanisms for the scheduler to guarantee the freedom of interference like time triggered scheduling (used e.g. in ARINC 653 IMA approach) or fixed priority based scheduling. Also monitoring of processor execution time is suggested.

In today's practice of building a schedule, certain such measures to fulfill these safety requirements can conflict with efficiency requirements. The target of this paper is to present a sound approach on how to solve such requirement collisions.

TASK PRIORITY ASSIGNMENT

We are focusing on the automotive domain in this paper. Here, the scheduling is typically priority-driven, as in OSEK and AUTOSAR as operating system standards. Static priority preemptive (SPP) systems are the normal case with possibility of non-preemptable (critical) sections. In the schedule design phase, the task priorities are assigned during the configuration of the system. Higher priority tasks are allowed to preempt lower priority ones.

ISO 26262 requires the freedom of interference in mixed criticality systems between software partitions with different criticality levels. In OSEK and AUTOSAR the software partitions are represented by runnables (kind of sub processes or function). One software partition contains several runnables. These runnables are grouped to tasks (often according to their activation period). The tasks are the entities scheduled by the OS. In general, runnables of different software partitions can be grouped to one task. Criticality levels constrain this grouping, so that only runnables of the same criticality level can be grouped to one task. The task

itself inherits the criticality level of the runnables. (Mixing criticality levels into one task is not prohibited, but it will be hard to guarantee the time and space separation. Tasks in OSEK and AUTOSAR use usually one task context and one memory space for all their runnables.)

The challenge is to assign the priorities in the best way to fulfill system requirements like deadlines, get good resource efficiency, and still make the design safe and certifiable. There are two general approaches to ensure this: RMS (Rate Monotonic Scheduling) or CAPA (Criticality as priority assignment)[5]. Both approaches have advantages and disadvantages, which will be shown in the following parts.

CRITICALITY ASSIGNMENT AS PRIORITY - CAPA

One approach is to assign task priorities by criticality levels. High critical tasks get a high priority, low critical get low priority. This guarantees the freedom of interference (lower critical tasks cannot preempt high critical ones). The advantage for CAPA is that the freedom of interference is guaranteed by design and follows suggested approaches like in ISO26262 - Part 6 Annex D [9]. On the downside, CAPA is not considering resource efficiency sufficiently. The resulting schedule could be not even schedulable.

RATE MONOTONIC SCHEDULING - RMS

RMS is a well-known pattern to assign priorities for periodic tasks. A small task period leads to a high priority and longer periods to lower priorities. Research in the past has shown that RMS results in very resource efficient scheduling [3][4]. Introducing mixed criticality, RMS has the problem that a task priority does not reflect the criticality level of the task. This can lead to situations in which a low critical task with a short period would have a high priority but the high critical task with long period a low priority. The low critical task can delay or preempt the high critical one. In [5] this is called *criticality inversion*. Such a schedule raises problems during the certification phase. In later sections, we show how the freedom of interference can still be secured.

At this point we introduce an extension for the classical RMS. We allow several tasks with the same cycle time (period) to exist. This is necessary as several tasks of different criticality levels but with the same cycle time can occur. The priority assignment works as described in the following: the highest priority is assigned to the task with the highest criticality level. The task with the same cycle time (period) but lower criticality level is getting the next lower priority, etc. This way just shows how to assign priorities. The freedom of interference must be guaranteed in other ways (described later).

TIMING PROTECTION

Timing protection is a concept implemented by an AUTOSAR service that monitors (during system run time)

parameters of tasks (and interrupts), compares them to a previously configured budget, interprets any budget overrun as an error, and then executes certain error recovery measures. A detailed analysis of the AUTOSAR timing protection mechanism has been carried out in [12].

The monitoring of different parameters during the runtime of a system is a widely used technique to recognize any undesired behavior. This is also suggested by ISO 26262-Part 6. The following three issues exist for timing and scheduling aspects in AUTOSAR:

- The execution time of tasks or interrupts
- The inter-arrival time of events
- The locking time for shared resources or interrupts

Assuming that an overrun of an estimated task execution time is an error (for example an infinite loop, which is never left), the execution time monitoring can recognize this and react on it. In AUTOSAR, the options are kill the task, restart an OS-Application or shut-down and restart a whole ECU. This feature is also defined in other RTOS specifications and available in implementations. The Timing Protection implementation itself is targeting a specific criticality level. That way an ASIL-B watch dog cannot be used to monitor and secure an ASIL-C function. Tasks monitored till ASIL-B are still only ASIL-B. The preemption of ASIL-C tasks is then not allowed (an ASIL-C monitor would be required). In other words, the timing protection mechanisms allow preemptions between tasks of different criticality levels up to the protected level, i.e. the level of the timing protection services.

The inter-arrival time monitoring guarantees that e.g. interrupts by a broken sensor cannot lead to a system locking as a result of the Interrupt Service Routines turning into a “babbling idiot” and consuming the complete CPU performance. This kind of watch dog is mainly designed to bind the influence of event driven task executions.

The locking time monitoring is used to avoid a too long interrupt disabling or shared resource occupation. Critical sections in task executions are often disabling interrupts during this time to have a full, undisturbed access to occupied shared resource. The allowed time can be bounded by the locking time monitor.

In this paper we only investigate the execution time protection as this is the most important mechanism from the schedule building point of view. All other mechanisms are out of focus for this paper. We also assume a “correct” Timing Protection configuration and furthermore that the error and recovery cases are also analyzed and validated against safety goals. The intention is to use the Timing Protection to bind the maximum execution time of tasks. This way, the interferences between tasks (in preemption cases) are bound. So a low critical task with high priority cannot preempt a high critical task with low priority in an unbounded