Applying the AUTOSAR timing protection to build safe and efficient ISO 26262 mixed-criticality systems

Christoph Ficek, Nico Feiertag, Dr. Kai Richter
Symtavision GmbH
Braunschweig, Germany
{lastname}@symtavision.com

Abstract

Integrating applications with different criticality levels into one electronic control unit requires special care. An unlucky priority assignment (as part of the overall dynamic software architecture) can result in the violation of safety requirements of the ISO 26262. Hence, a certain protection is needed to guarantee the so called *freedom from interference*. In this paper, we evaluate the three priority-assignment strategies: CAPA, RMS and a hybrid approach. The optimal selection requires a trade-off between safety and resource-efficiency. We show how the AUTOSAR Timing Protection mechanism – when configured wisely – facilitates combining the advantages of the three approaches and yields safe and very efficiency schedules. We show a method for determining good configurations for the Timing Protection service.

1. Introduction

One key trend in automotive E/E development is the integration of more and more software on a decreasing number of increasingly powerful electronic control units (ECUs). This consequently leads to new situations where highly critical software, e.g. steering or damper control, is integrated with less critical software on the same ECU. We need new ways of dealing with the safety requirements for such *mixed-criticality systems*. Essentially, we must guarantee that an error in a low-criticality software part must not compromise the correct functioning of any software part with higher criticality (no criticality inversion). Otherwise, the certification according to ISO 26262 will be hardly possible. In the ISO 26262[1], this requirement is captured under the terms *error propagation* and *freedom from interference*.

If we look at other industries such as avionics (with standards such as DO-178B and ARINC 653), we find middle-ware and operating systems that use guaranteed memory- and time-partitioning mechanisms (based on a fixed, time-slotted schedule) to satisfy the requirements. Memory partitioning is applied in automotive systems already. Time-partitioning, however, is not feasible in most cases because the automotive OSEK/VDX[2] and AUTOSAR[3] standards are based on a different resource sharing and scheduling policy. Hence, a different type of protection is needed that is suitable for the used scheduling policy.

In OSEK and AUTOSAR systems, applications are scheduled preemptively under a fixed priority scheme. It is obvious that the priority assignment has a dominant impact on the interference and error propagation among the tasks and applications: higher-priority tasks and application can preempt and delay lower-priority ones. Depending on the priority assignment, a system exhibits criticality inversion or not. In industry practice, two different priority assignment strategies can be found with different pros and cons:

- Rate-Monotonic Scheduling (RMS)[4]. Pro: very resource efficient in error-free case. Con: not aware of criticality, thus criticality inversions possible. This is the most widely used approach. It is intuitive, efficient, easy to deploy, etc.

- Criticality-As-Priority-Assignment (CAPA)[5]. Pro: no criticality inversion in schedule. Con: less resource efficient (in all cases). This is quite new and used only in high-criticality systems, so far.

We explain both approaches later in detail. For this introduction, we conclude that the established strategies let us build systems that are either resource-efficient or safe, but not both.

We have shown in[6] that there exist a third, hybrid priority assignment approach that balances the pros and cons of pure CAPA and pure RMS. The approach uses a) CAPA for the highest criticality levels, and b) RMS for those criticality levels that are either b1) low-criticality or b2) are protected by using the *Timing Protection* mechanism of AUTOSAR. The contribution in [6] introduces the overall methodology to select the right strategy. It was shown that the use of Timing Protection can enlarge the design space, i.e. the flexibility in setting up the dynamic software architecture.

In this paper, we explain in detail, how the *Timing Protection* service works, how it facilitates safe and efficient schedules, and how the service itself must be configured. Especially we show how good (reasonable) *Timing Protection Budget* values (TPB) can be determined and how they interacts with *Recovery Times* (RCT). We will see that this is only possible through an assessment of the expected execution times of tasks, and a reliable scheduling analysis in the error-free *and* the error-case.

The paper is organized as follows: In the next Section, we review the relevant requirements from the ISO 26262 on the software architecture, in particular the task schedule. Then, we illustrate the three strategies mentioned (RMS, CAPA, hybrid) using a comprehensive example. In Section 4 we introduce the details of Timing Protection and explain the parameters and how TPB values can be determined. This will include a certain amount of analyses that must be performed. Finally, we draw our conclusions.

2. Safety requirements

2.1. ISO 26262

In 1998, the International Electrotechnical Commission (IEC) issued the first edition of IEC 61508. This standard regulates the development of electrical, electronic and programmable electronic (E/E/PE) systems with safety functions.

With the increasing complexity of electronic components in the vehicle, it was necessary to capture the specific properties in the development of these components in an own standard. ISO 26262 ("road vehicles - functional safety"). This standard regulates the implementation of the functional safety of a system with electrical/electronic components in a vehicle.

For the development of components it is necessary to know the assigned automotive safety integrity level ASIL, because the measures and methods in the development process, as well as the necessary technical mechanisms depend on this classification. The ASIL classification is the result of the risk analysis and risk assessment. Four levels of security exist A, B, C, D, and the classification "not critical". The security requirements of the system and thus the requirements for the methods, measures and mechanisms increase with increasing ASIL.

2.2. Freedom From Interference

ISO 26262 contains 10 parts. Part 6 - Product development at the software level - describes the necessary measures and methods for the software part of a systems. Chapter 7 and annex D define the requirements to fulfill if a software system is buildup of components with different criticality levels.

The ISO demanded to use the highest ASIL in the system for all components if interference between the components cannot be excluded. Interference includes memory, execution time and communication.

The development according to this approach, using the highest ASIL, is not practical. To realize such mixed criticality systems, which are certifiable, measures, methods and technical mechanism are required to guarantee the freedom from interference between the components. Certification can also be achieved if interference is allowed on purpose, but it is shown or guaranteed that all interference are recognized and processed in a way that safety goals are not violated. In this paper we are concentrating on the scheduling and the interference in execution time.

3. Contradicting design patterns

In this section we illustrate the two established priority-assignment strategies, and their pros and cons for mixed-criticality systems. This is demonstrated in the following example:

Application / Task	ASIL level	Cycle time / max. runtime (Deadline)[ms]	Runtime [ms]	RMS priority	CAPA priority
T1	ASIL-C	5	0,5	4	10
T2	ASIL-B	10	1,2	8	8
T3	ASIL-A	1	0,2	10	6
T4	QM	10	3	6	4

Table 1 ASIL levels, priorities and ASIL levels of the example system

Table 1 includes the resource and safety requirements of four applications (tasks), and their maximum runtime requirements (deadlines) which correspond to the period in this example. One of the most important properties that needs to be determined during the integration is the task priority. According to the Rate-Monotonic Scheduling method (RMS) [4], the priority is derived from the cycle time: the shorter the cycle time, the higher the priority. According to the Criticality Aware Priority Assignment method (CAPA) [5], the task with the highest safety level is given the highest priority. Table 1 shows the resulting priorities for both methods.

Neither of the two methods yields to a satisfying result. The priorities from the RMS method result in a task schedule that meets all deadlines. However, the schedule does not guarantee the necessary freedom from interference. The ASIL-C application can be interrupted by all other applications, this is called *criticality inversion*[5]. For instance, a runtime error in the ASIL-A function could possibly block the ASIL-C-function permanently. illustrates the schedule and the interferences in a Gantt diagram. The RMS approach therefore does not provide sufficient safety. Such criticality inversions are naturally avoided when using the CAPA approach; only higher ASIL levels can preempt and block lower ASIL levels, not the other way round. According to ISO 26262, this is not considered critical. However, the resources – particularly CPU time – are not being used efficiently. Figure 2 illustrates the schedule: due to the modified priorities, the deadline of task T3 is being violated.

The example shows the mutual impact of design patterns and safety requirements on the fulfillment of real-time properties. The existing patterns (RMS and CAPA) do not make it possible to fulfill both requirements at the same time. Therefore, new design paradigms are needed, or the existing ones have to be adjusted to the new requirements.

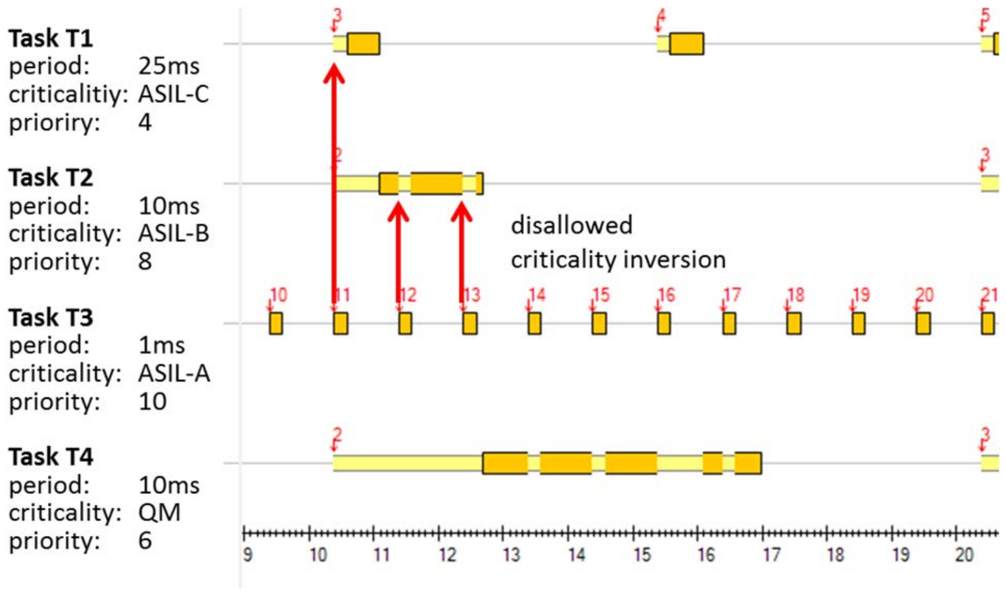

Figure 1 Software schedule according to RMS (Rate-Monotonic-Scheduling)

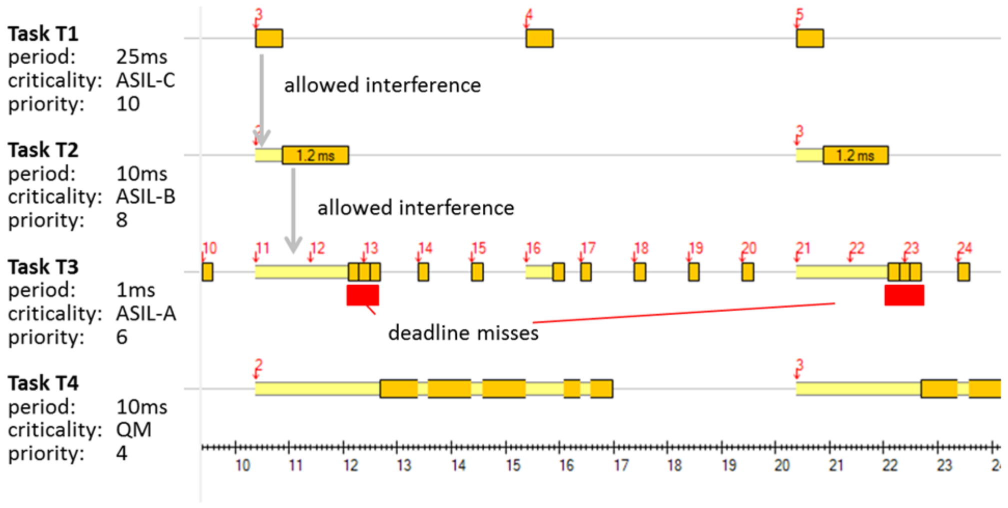

Figure 2 Software schedule according to CAPA (Criticality-as-priority assignment)

4. AUTOSAR Timing Protection

Timing Protection is an AUTOSAR system service that monitors different parameters during the run time of the system. These are:

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

All are important and useful, but for this paper we are concentrating on the execution time monitoring. For this a budget for the execution time of a task is defined. In the event that any task exceeds the preset execution budgets, this is being interpreted as an error and a configurable error handling is executed. The Timing Protection does not fully eliminate criticality inversions, but the amount of possible interference is reliably bounded. The safety requirements of ISO 26262, especially the *absence of error propagation*, are therefore fulfilled.

If Timing Protection is available in a system, the design space for the schedule is naturally enlarged, i.e. we have more design options because some of the criticality inversions can be secured explicitly. Limitations are introduced through the necessity to certify Timing Protection. In general, the Timing Protection should be developed according to the highest ASIL level of all applications (or higher). For the proper configuration of the Timing Protection (especially the runtime budgets), a reliable analysis and forecast of the critical schedule situations is mandatory. Figure 3 shows that all safety and real-time requirements can be met with a combination of RMS priority assignment, CAPA and Timing Protection. There are no disallowed interferences anymore, and all tasks reliably complete within their cycle times (deadlines).

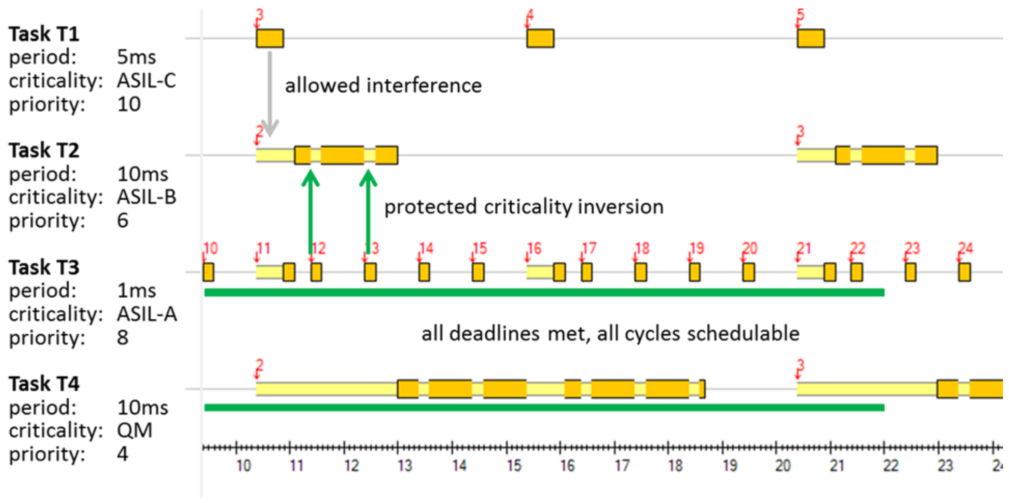

Figure 3 Mixed CAPA and RMS schedule with Timing Protection

4.1. Timing Protection configuration

In the previous section, we have shown that the Timing Protection Service can yield very efficient and safe schedules, if properly configured. In this section, we present a method, to derive a proper configuration. This means, we show methods that let us determine the Timing Protection Budget (TPB) values, as the key configuration parameter of the Timing Protection service.

What are the goals?

- All tasks/applications must be schedulable in the error-free case
- In the event of an error of a single task/application, the rest of the system must be schedulable
- The probability of false positives (error has been detected even if there was no error) shall be as low as possible (soft quality goal)
- The resource efficiency shall be as large as possible (soft efficiency goal)

These are reached by the following procedure:

- Determine the maximum execution times of all tasks

- Perform a scheduling analysis for the error-free case, i.e. under the assumption that all tasks use their core execution times determined, but no timing protection steps in, etc. This must be schedulable.
    - Like shown in Figure 3 for our example.
- Select a TPB for each task; obviously this shall be larger than the maximum core execution time.
    - Section 4.2 shows how
- Perform scheduling analysis with TPB as maximum execution time
    - Because TPB are larger than core execution time, this will be less efficient (more "waste"). We will come to that later.
    - If this is schedulable, the system will work, even when the tasks exploit their full budget.
- Consider the error case
    - The cases to check depends on the chosen error strategy and will be explained in detail later

If all is schedulable, or at least, no criticality inversion exists then a possible configuration has been found.

4.2. Timing budget determination

Until now, we have not shown how the budget value itself can be determined. We look into this now.

The goal of the timing protection budget is to bind the execution time of a task. To do this it is necessary to know the worst case execution time of tasks or the functions/runnables in the task. However, most likely the exact worst case execution time (WCET) is not known. Like in [7] we can assume that the execution time of high critical functions is determined with a bigger effort than for low critical ones. Developers have to spend more time and effort in validating and testing high criticality functions. The knowledge about the time critical paths is well known. With high probability the WCET will be not overrun during the runtime, except in real error cases.

In practice several tools and methods exist to get the WCET. For example static code analysis like it is available by AbsInt[8] can be used to get the WCET depending on the different HW platforms. A further method is simulation for example with VaST tools[9] or the tracing on the real target HW with e.g. Gliwa tools[10].

Low critical functions, like quality management, are less important and the effort for validating and testing is lower, mostly because of cost reasons. The determined WCET for such functions is not as safe as for high criticality ones. With higher probability the determined WCET will be overrun during the run time even in correct situations.

So we can define different times, which are illustrated in Figure 4:

- CET : Core Execution Time
    - The time a function needs to run.
    - Depends on the executed code and changes during run time
- BCET Best Case Execution Time
    - Minimum possible CET
- WCET : Determined Worst Case Execution Time
    - Assumed WCET for a function
- $WCET_R$: Real Worst Case Execution Time
    - The real but may be unknown maximum execution time for a task
- TPB: Timing Protection Budget value
- RCT : Recovery time
    - Time needed by recovery mechanisms in error cases

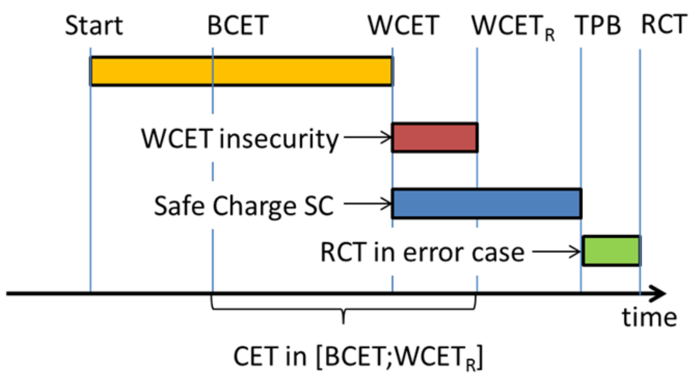

Figure 4 Task execution times

The more we know about a function the closer the WCET gets to the WCET_R and the insecurity is reduced. However, as we do not know how close we are, we are setting the TPB value a little bit higher than the WCET. This should avoid, that a task is killed not for error reasons, but because the WCET was not determined correctly. We call this extra amount of time Safe Charge SC. The safe charge is more or less a risk factor for not knowing the exact WCET_R and the risk to configure the budget wrong.

A practical suggestion could be this:

- ASIL-D: WCET + 5%
- ASIL-C: WCET + 10%
- ASIL-B: WCET + 15%
- ASIL-A: WCET + 20%
- ASIL-0: WCET + 25%

We are granting for example for ASIL-A tasks 20% more run time than determined by a WCET assumption. However, this is only an example. In the end, the software developers, together with the integrator, must make this decision, how much they budget (TPB). The proposed TPBs may be safe, but of course the efficiency is decreasing because of over dimensioning the system. As a design decision (and with reasoning) the false positives for less critical functions can be accepted due to increase the performance and reduced the overestimation.

A kind of tailoring between efficiency on the ECU, the effort for better WCET determination and the acceptance of error for non-critical functions takes place.

4.3. Error case and recovery strategy

In the case of an overrun of the execution time budget an error recovery is executed. What happens exactly is configurable. AUTOSAR describes in general three different scenarios:

- Task kill
- Application kill and restart (optional)
- ECU shutdown and restart (optional)

The pure task kill means, that the task just ends. If it is a periodic task, it will be scheduled again according to his period when the triggering event arrives again. All code, which has not been executed until this point, will not be executed anymore. The functional consequences must be analyzed in detail, but are out of our focus. The kill itself needs some time, the recovery time RCT. To check the schedule for this situation, we are adding the RTC value to the TPB and use this value in the scheduling analysis and investigate the consequences. Table 2 shows the different TPB and RCT values for the task schedule and Figure 5 shows the schedule for this configuration. The green boxes mark the RCT. The concrete overhead for this time must be determined together with the Operating System vendor and also depends on the concrete hardware. We assumed, that all tasks will raise an error and need the RCT time. For this case the system is still schedulable in the example. Of course this is very pessimistic. In reality not all tasks will raise an error every execution. Detailed analysis can be made and assume that only one task will raise an error not so often and never simultaneously with other tasks, or only in few cases. In the end it depends on the detailed system and the reasoning, but it must be done during the certification process.

Application / Task	ASIL level	Cycle time / max. runtime (Deadline)[ms]	Runtime [ms]	Time Protection Budget TPB	Recovery Time RCT	priority
T1	ASIL-C	5	0,5	0,55	50µs	10
T2	ASIL-B	10	1,2	1,38	50µs	6
T3	ASIL-A	1	0,2	0,24	50µs	8
T4	QM	10	3	3,75	50µs	4

Table 2 Task configuration with TPB and RCT values

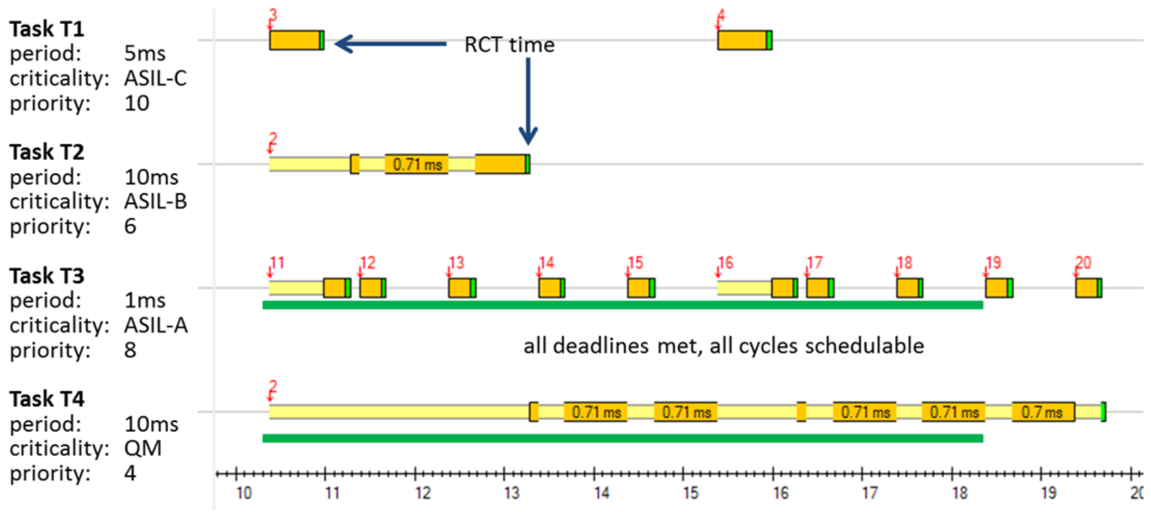

Figure 5 Schedule with Recovery Time RCT

The application kill means the end of the current task, which violates the budget, and all other tasks belonging to a certain application. If the application is stopped, the schedulability can be checked like for the task kill described before. For a restart of the application a specific restart task has to be configured in AUTOSAR. Thereafter the normal application is scheduled. This restart task must be explicit analyzed and its influence on the entire system depends on its priority and its execution time. Using scheduling analysis, it can be determined how long a specific restart task will need, considering its priority, to restart the application and what the influences on the other tasks are. If the timing is also not violated in this case, the configuration is valid and can be used for the AUTOSAR Timing Protection configuration. This has to be repeated for all applications in the system.

The third case is an ECU restart. This does not need any further scheduling analysis for this ECU because all tasks stop and the schedule does not exist anymore. What has to be determined is the restart time of the ECU and of its side effects, for example system wide timing in distributed systems over different ECUs. As an example, the restarted ECU could send some frames on a bus to signal the restart. This will influence the bus schedule and may be the receiver ECUs.

4.4. Scheduling Analysis

To check the schedulability of the system, we are using a scheduling analysis. In general every approach for scheduling analysis, like from Palencia[11] can be used. We are using the tool SymTA/S by Symtavision[12], which follows the theory by Palencia and are based on the SymTA/S approach[13].

All Analysis approaches need the WCET of the scheduled entity as input. As described we are using the different TPBs values as input and maximum for the WCET of a task in the analysis.

The outcome of the analysis is the worst case system behavior, when all tasks will use their maximum budget (but are not exceeding it). If the system is schedulable in this case, it is designed to handle the worst case in the normal case. If the system is not schedulable, it needs a closer look, to identify what is not schedulable. Different counter measures for this case and for optimizing the schedule are for example in [6].

If high critical functions are affected, the schedule design can be treated as not feasible. If low critical functions like Quality Management are affected, the design could be accepted. Simulation approaches, like the SymTA/S Distribution analysis, can give hints how often, for example a deadlines, are violated for an ASIL-A task.

4.5. Tasks and runnables

One issue to consider for the timing protection in AUTOSAR is the relationship between runnables and tasks. The Timing protection mechanisms are used for tasks, not runnables. Also a task is killed or restarted in the error and recovery case. In the same way like the tasks are composed of the runnables, we can compose the protection values from the runnables. The WCET for each runnable is determined. The protection value is determined per runnable and the summarized as one value for the task.

In practice the WCET for some simple runnables can be determined very accurately, even with low effort. For these runnables we can reduce the overestimation for the TPB.

As the tasks consist of runnables, one or several runnables will be responsible for an execution time overrun in the error case. In the worst case, the first runnable in a task will be responsible and overrun the execution time, and other runnables in the task will not even be executed.

We are assuming first, that the runnable order is fully customizable. In this situation, the more important runnables and the ones with the safest WCET should be executed first. This increases the probability, that at least these runnables will be executed. If the order is only partly customizable, the important and reliable parts should be executed first. In some cases it makes sense to add very important runnables into an own task. This increases the scheduling overhead and task count, but reduces the risk for these runnables to be disturbed by others.

5. Conclusion

Integrating several applications with different criticality levels into one electronic control unit requires special care. An unlucky priority assignment (as part of the overall dynamic software architecture) can result in the violation of safety requirements of the ISO 26262. Hence, a certain protection is needed to guarantee the so called *freedom from interference*.

In this paper, we have evaluated the three most commonly used priority-assignment strategies: CAPA, RMS, and hybrid approach. It became clear that each of those has advantages and disadvantages with respect to safety and resource-efficiency; these goals appear quite contradicting. We showed that AUTOSAR Timing Protection helps to solve the conflict between efficiency and safety requirements. Furthermore we showed how budget values for the protections mechanism can be determined and checked for feasibility.

The example has shown that the actual configuration parameters of the Timing Protection service must be selected with care. If set too tight, the system produces too many false positives (detected errors in case no error has happened). If set to loose, the system exhibits poor performance in case of an error. The method of scheduling analysis helps evaluating different error-free and error- -recovery scenarios upfront, i.e. during SW architecture design. This ensures that the best / most sufficient configuration can be found.

We showed how the AUTOSAR Timing Protection mechanism – when configured wisely – facilitates combining the advantages of the three approaches and yields safe and very efficiency schedules. We showed a method for determining good configurations for the Timing Protection service. The method is validated using a set of experiments.

6. Acknowledgment

This work was jointly funded by Advanced Research & Technology for Embedded Intelligence and Systems Joint Undertaking (ARTEMIS JU) and the Federal Ministry of Education and Research (BMBF) Germany within the project 'RECOMP', support code 01IS10001A.

7. References

[1] ISO 26262ISO/DIS 26262 Road vehicles – Functional safety – Part 1-10, URL http://www.iso.org

[2] OSEK/VDX Group. Operating System Specification 2.2.3. OSEK/VDX Group, Feb.2005. http://www.osek-vdx.org/.

[3] www.autosar.org Official website of the AUTOSAR partnership

[4] Lui Sha; Rajkumar, R.; Sathaye, S.S.; , "Generalized rate-monotonic scheduling theory: a framework for developing real-time systems," Proceedings of the IEEE , vol.82, no.1, pp.68-82, Jan 1994RMS

[5] De Niz, D.; Lakshmanan, K.; Rajkumar, R.; , "On the Scheduling of Mixed-Criticality Real-Time Task Sets", Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE , vol., no., pp.291-300, 1-4 Dec. 2009

[6] Ficek, C.; Feiertag, N; Richter, K.; , "Schedule design to guarantee freedom of interference in mixed criticality systems", SAE World Congress, Detroit, 2012, under reviwe

[7] Vestal, Steve, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance", In Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS '07), 2007

[8] AbsInt, http://www.absint.de/

[9] VaST tools by Synopsis, www.synopsis.com

[10] Gliwa GmbH, http://www.gliwa.com/

[11] J. Palencia and M. Gonz´alez Harbour, "Schedulability analysis for tasks with static and dynamic offsets", In Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE, pages 26–37. IEEE, 1998. [Palencia, Tindel, Thiele]

[12] SymTA/S by Symtavision GmbH, www.symtavision.com

[13] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach", in IEEE Proceedings Computers and Digital Techniques, 2005