Transactions on Engineering and Computer Science

Research | Vol 1 Iss 1

Generalized Models of Mixed-Criticality Systems for Real-Time Scheduling

Junghwan Lee[1,2*] and Myungjun Kim[2]

[1]Department of Battery Management System, Baoding, Hebei 2266, China
[2]Department of Computer Science, Chungbuk National University, Cheongju, Chungbuk, South Korea

*Corresponding author: Lee J, Department of Battery Management System, Baoding, Hebei 2266, China, Tel: +82-10-5459-2016; E-mail: roryjhlee[at]gmail.com

Received: May 05, 2020; Accepted: May 21, 2020; Published: May 29, 2020

Abstract

In this article, we address the divergence theory and practice related to mixed-criticality systems (MCSs) for real-time scheduling from an industry perspective. We identify the practical problems in consideration of design aspects, such as time partitions, the decomposition of tasks, degradation scenarios, fault handling for safety, and criticality mode of MCS. The identified problems in design will induce theoretical limitations and difficulties for adopting prior works to practice. In prior works, criticality modes were viewed as equivalent to the assurance levels; however, these assumptions seem to be too simplified for studies far from practice. They may lead to solutions for real-time scheduling in MCS going in different directions that are highly divergent from the real-world situation. Hence, we propose new models for MCS that are closer to practice. Here, criticality modes are separate from the assurance levels of tasks since the criticality modes exist for degradation or safety scenarios, whereas the assurance levels of tasks, hardware, and software components exist to represent reliability for requirements, designs, verifications, and validations. Based on the newly proposed system models, we will show a way of extending prior works with the proposed mode-based priority protocol for real-time MCSs. The proposed mode-based protocol is easily implanted with existing real-time schedulers or the Automotive Open System Architecture platform.

Keywords: ***Mixed-Criticality systems; Real-Time systems; AUTOSAR; Safety.***

1. Introduction

In the automotive industry, the requirements of autonomous driving, connected cars, and electric vehicles (EVs) are increasing, with the resulting growth of software complexity and architecture [1-3]. For EV, the following main design principles for future electrical/electronic (E/E) architectures and software design are proposed in [4]: *1) flexible E/E architecture: future E/E architectures should be based on a centralized or hierarchical topology with scalable electronic control units (ECUs) to execute all hardware-independent software functions; 2) highly integrated mechatronic components:*

sensors and actuators should become the smart components that communicate over standardized data interfaces with the E/E architecture; 3) a standardized communication backbone: a time-triggered homogeneous in-vehicle core network must be developed that can guarantee the required quality of service; and 4) operating system/middleware: new technology must be developed and deployed to take extra-functional properties into account. With this design, mixed-criticality systems on multiprocessors will be widely developed for EVs, while traditional safety-critical systems, such as powertrains, battery management systems (BMSs), and inverters, will be developed for safety and their specific functionalities. Hence, complex scheduling can be used for future automotive systems, while static or dynamic fixed priorities on uniprocessors are used in the traditional automotive industry. There is an opportunity for previously studied scheduling to be used in the automotive industry since real-time scheduling is more important to this industry as compared with the consumer industry.

Many studies have been published on real-time mixed-criticality systems (RTMCSs) since Vestal proposed scheduling algorithms for a mixed-criticality system (MCS) in 2007 [5-6]. However, the divergence between theory and practice in RTMCSs is still a barrier to adopting academic studies in industry. Misinterpretations of RTMCSs in theory have been addressed from a safety perspective in [6-8]. Again, academic scope and limitations are addressed in [9].

The term "system criticality" in Vestal's model described the operation modes instead of assurance levels in [6,7]. However, we can observe that the numbers of operation modes and assurance levels are equivalent in the vestal's models, and consecutive studies have used the term "criticality mode" for the operation mode. A simplified model, where the number of assurance levels is equivalent to the number of criticality modes, and tasks are dropped when their assurance levels correspond to criticality modes, seems to restrict the extension of studies, since many studies have been conducted based on this model. The term "importance" is used, since the automotive safety integrity level (ASIL) is determined by considering severity, controllability, and probability of occurrence according to ISO26262; one task will have a higher ASIL than another task if they have the same severity and controllability but different probabilities of occurrence [6,7]. However, we claim that the term "importance" in prior works signifies the assurance level. The reason that probability of occurrence is one factor used to determine the assurance level is that failures of functions cause hazards only in specific conditions or in operation modes [10-13].

The probability of occurrence is equivalent to the probability of specific conditions or operation modes occurring. For instance, there are tasks τ_1 and τ_2 assigned to ASIL-A and ASIL-B, respectively. They have the same severity and controllability but different probabilities of occurrence. Here, τ_2 is assigned to ASIL-B and represents a relatively high probability of occurrence, while τ_1 is assigned ASIL-A and represents a low probability of occurrence. The question is, if the system is in overload condition and needs to drop either τ_1 or τ_2, which task needs to be dropped? Dropping τ_1 is comparatively safe, because in specific conditions or operation modes, the system has a low probability of causing a hazard with τ_1. Hence, "importance" in prior studies relates to the assurance level in a simplified MCS model that does not consider strategies to drop tasks as specific conditions or operation modes. Most MCSs in practice still do not have such strategies [11]. The timing isolation by scheduling decision is also issued in [6,7]. However, a scheduler is the highest ASIL level, which is the lowest failure rate and able to access other lower ASIL components and the timing isolation by scheduling decisions as ASIL is a property of schedulers and scheduling algorithms to separate from traditional schedulers and scheduling algorithms.

In this article, we address the design considerations of real-time scheduling for MCS and difficulties and limitations of prior works to adopt practice from an industry perspective in consideration of requirements, designs, and implementations for MCS. In addition, we propose the MCS models close to industry practice for a real-time scheduling and develop a mode-based priority protocol (MPP) for real-time scheduling on the proposed MCS models. We report a case study on designing MCSs in practice to elucidate the proposed MCS models.

2. RTMCS Design Considerations

There are two main elements distinguishing an RTMCS from traditional real-time systems. One is that it handles a timing fault caused by an actual execution time (AET) exceeding the estimated worst-case execution time (WCET), while the other is handling systematic faults. From a safety perspective, a timing fault is a subcategory of systematic faults, and such faults may cause system failures. An AET exceeding the estimated WCET can be improved by the development process. However, there is no specific guideline for WCET measurement, while there are guidelines for design and test methods like assurance levels in ISO26262 and IEC61508. For real-time scheduling, we need to distinguish timing faults from other faults.
The list of faults causing timing failures comprises the following:

- F1) Systematic faults in a scheduling algorithm.
- F2) Systematic faults in a schedulability analysis.
- F3) Systematic faults in an implemented scheduler.
- F4) Systematic faults in estimated WCETs.
- F5) Systematic faults in estimated interarrival rates (EIRs).
- F6) Systematic faults in high-criticality (HC) tasks.
- F7) Random hardware (HW) faults in HW handled by HC tasks.
- F8) Systematic faults in low-criticality (LC) tasks.
- F9) Random HW faults in HW handled by LC tasks.

F1 to F3 are managed by the development process requiring the highest effort. Hence, we can assume that F1, F2, and F3 will not occur. F4 and F5 are managed in verification and validation, but there is no safety guidance. Static analysis and task-based runtime measurements are performed in the verification phase. Scenario-based runtime measurement is performed in the validation phase. However, it is hard to ensure the correctness of estimated WCETs with scenario-based runtime measurement due to HW uncertainty [14,15]. The accuracy and F5 affects F2, but we separate F4 and F5 from F2 for future studies, since methods to measure the WCET can be separately researched for increasing accuracy and decreasing efforts and costs. The accuracy of estimated WCETs depends highly on test efforts in practice. F6 and F7 are managed by the development process, requiring efforts and costs depending on their ASIL according to safety the guidance. F8 or F9 affects F6 or F7, since the wrong software code or abnormal HW operation can cause AETs exceeding the estimated WCETs or actual interarrival rates (AIRs) compared with EIRs. F8 and F9 can be solved by scheduling the algorithm in consideration of criticality inversion. Hence, solving problems F4, F5, F8, and F9 represents novel areas with F1 and F2 for RTMCSs, and they are mostly issued in practice.

Lemma 1. Systematic faults in tasks or random HW faults can cause AETs exceeding WCETs or AIRs exceeding EIRs.

Proof: An unexpected infinite loop is a systematic fault that can cause an AET of the task exceeding an estimated WCET of the task. An unexpected event invoking tasks is a systematic fault that can cause an AET of the tasks exceeding the EIRs of the tasks. An HW fault causing a delay in response time can cause an AET of a task waiting for an HW response exceeding the WCET of the task. An HW fault causing an event can invoke a task unexpectedly.

2.1 Criticality inversion

In the criticality inversion, LC tasks preempt HC tasks. The criticality inversion will not cause timing failure if a task set is passed schedulability test based on traditional RTS, such as deadline monotonic (DM), rate monotonic (RM), or earliest deadline first (EDF) approaches. However, HC tasks may not meet the deadlines if F4, F5, F8, or F9 occurs. For instance, if AETs exceed the estimated WCETs and the priorities of LC tasks are higher than the priorities of HC tasks, then the HC tasks may miss their deadlines. If the AIRs of sporadic tasks are higher than the EIRs of the tasks, and the priorities of LC tasks are higher than the priorities of HC tasks, then the HC tasks may miss their deadlines since schedulability analysis is no longer valid.

Definition 1. Criticality inversion failure is when HC tasks miss the deadline due to lower criticality tasks running.

2.2 Hard, soft, and hybrid RTMCS types

The criticality modes are for safe states or graceful degradation. The fail-operational, fail-safe, and fail-silent modes are well-known degradation strategies. These are defined in [16-18] as follows:

- Fail operational: One failure is tolerated (i.e., the component remains operational after one failure). This is required if no safe state exists immediately after the component fails.
- Fail safe: After one or several failures, the component directly reaches a safe state (passive fail safe, without external power) or is brought to a safe state by a special action (active fail safe, with external power).
- Fail silent: After one or several failures, the component exhibits a quiet behavior externally (i.e., remains passive by switching off), and therefore, does not negatively affect other components.

In practice, system designers can give constraints for deadlines and periods corresponding to criticality modes according to the degradation scenario or safety scenario. Generally, the implicit deadline is more prevalent in automotive practice since an explicit deadline increases the complexity of requirements, SW design, and HW design for a whole system, as well as its subsystems. Criticality modes are generally switched as safety scenarios in a fail-safe system and degradation scenario in a fail-operational system. The fail-safe system focuses on how to ensure the system is safely halted according to a safety scenario when faults are detected. Each criticality mode as a safety scenario does not require all SW and HW components to run; thus, only SW and HW components corresponding to a criticality mode will run. A safety scenario defines faults corresponding to criticality modes; that is, faults can trigger the system to go to one of the criticality modes according to the safety scenarios. However, the fail-operational system focuses on how to make a system keep running as long as possible because a halted system also causes critical damages. Flights or autonomous driving cars will give critical damages when systems are halted. An MCS design highly depends on the degradation scenarios or the safety scenarios, which also affect scheduling policy. A set of tasks corresponding to a criticality mode must generally meet the deadlines in a fail-safe system, which is a hard real-time (HRT) characteristic. In a fail-operational system, some tasks in a set of tasks corresponding to a criticality mode can tolerate missed deadlines. Hence, the tasks have soft real-time (SRT) characteristics. Traditional automotive systems are generally require a fail-safe system, but autonomous driving requires a fail-operational system.

Definition 2. In a hard RTMCS, the sets of tasks corresponding to the criticality modes meet deadlines in the criticality modes if F4, F5, F6, F7, F8, and F9 do not occur. These faults cause tasks to miss deadlines, but criticality inversion failure is not allowed.

Definition 3. In a soft RTMCS, the sets of tasks corresponding to criticality modes missing a deadline is tolerable in the criticality modes without criticality inversion failure.

Definition 4. In a hybrid RTMCS, there are SRT tasks and HRT tasks. The sets of SRT tasks and HRT tasks correspond to the criticality modes. The SRT tasks missing deadlines are tolerable in criticality modes without criticality inversion failure. The HRT tasks meet the deadlines in the criticality modes if F4, F5, F6, F7, F8, and F9 do not occur. Those faults cause HRT tasks to miss their deadlines, but the criticality inversion failure is not allowed.

2.3 Timing Partitioning

A challenge in RTMCSs compared with traditional RTSs is time partitioning. Traditional hard real-time scheduling (HRTS) has also been developed for safety-criticality systems, but this is not considered in terms of the time partition. A time-partitioned system means that low-assurance tasks do not interfere with high-assurance tasks. (That is, low-assurance tasks do not affect high-assurance tasks becoming not schedulable). This is similar to a memory-partitioned system in that low-assurance software components (SWCs) cannot corrupt the memory of high-assurance SWCs. Criticality inversion induces problems only if criticality inversion failure occurs.

Definition 5. A time partition is a mechanism to prevent criticality inversion failure.

Theorem 2. DM for fixed priority and EDF for dynamic priority with execution and arrival time monitoring are optimal for the hard RTMCS.

Proof: The sets of tasks corresponding to criticality modes must be feasible in each criticality mode, regardless of the criticality of tasks for the hard RTMCS. Hence, DM for fixed priority and EDF for dynamic priority are optimal. AETs and actual interarrival times (AITs) are bounded to estimated WCETs and estimated minimum interarrival times (EMITs) under execution time and arrival time monitoring, respectively.

2.4 Decomposition to Degrade the assurance level

The SWCs are typically decomposed to degrade the ASIL as the decomposition helps to reduce development efforts while a system still fulfills the safety requirements [24-28]. The development cost and efforts increase exponentially as the required ASIL is changed from ASIL-A to ASIL-D [29,30]. Supposing that we set the WCET of tasks as Quality Management (QM), ASIL-A, ASIL-B, ASIL-C, and ASIL-D in ISO26262, we decompose the ASIL-D task to four ASIL-A(D) tasks. The four ASIL-A(D) tasks are deemed to have ASIL-A criticality. In previous studies, four ASIL-A(D) tasks have probably been simultaneously dropped in admission control, which causes critical system failure. Hence, we cannot drop the tasks as only ASIL.

According to ISO26262, ASIL-D can be decomposed to ASIL-C(D) and ASIL-A(D), ASIL-B(D) and ASIL-B(D), or ASIL-D(D) and QM(D). ASIL-C can be decomposed to ASIL-B(C) and ASIL-A(C) or ASIL-C(C) and QM(C). ASIL-B can be decomposed to ASIL-A(B) and ASIL-A(B) or ASIL-B(B) and QM(B). ASIL-A can be decomposed to ASIL-A and QM(A).

2.5 Criticality Modes

The system designer defines system behaviors corresponding to sets of tasks necessary to run in different criticality modes. A set of tasks runs in a mode. The tasks necessary to meet deadlines can have different assurance levels in practice, but they were assigned the same assurance levels in prior studies [31-49].

Definition 6. An implicit criticality mode (ICM) is when the criticality modes are equivalent to the assurance levels. The number of assurance levels is the same as the number of criticality modes, and a set of tasks with the same assurance level is guaranteed to run in a criticality mode:
A set of criticality modes: $M = \{M_1, \dots, M_n\}$,
A set of assurance levels: $L = \{L_1, \dots, L_n\}$,
A set of system behaviors: $s = \{s_1, \dots, s_n\}$,
A set of tasks: $\tau = \{\tau_1, \dots, \tau_m\}$,
$s_i = \{\tau_k | \tau_k \in \tau \wedge L_n = \mathrm{L}(\tau_k) \wedge (\tau_k \ run\ in\ M_i)\}$.

Definition 7. An explicit criticality mode (ECM) is when the criticality modes are not equivalent to the assurance levels. The number of assurance levels is not the same as the number of criticality modes, and a set of tasks with different assurance levels can be guaranteed to run in a criticality mode:
A set of criticality modes: $M = \{M_1, \dots, M_n\}$,
A set of system behaviors: $s = \{s_1, \dots, s_n\}$,
A set of tasks: $\tau = \{\tau_1, \dots, \tau_m\}$,
$s_i = \{\tau_k | \tau_k \in \tau \wedge (\tau_k \ run\ in\ M_i))$.

Theorem. 3 The ICM model cannot meet safety-critical systems if there is a set of tasks with decomposed tasks.

Proof: A high-assurance task can be decomposed to two tasks where the assurance level will be lower than the original assurance level. If the criticality mode is switched from low to high, the system guarantees that high-assurance tasks will run and drops lower-assurance tasks than when the criticality mode is switched. However, the original assurance level of lower assurance tasks can be high. Hence, the system cannot guarantee high assurance level tasks to run in a high-level criticality mode. For instance, there could be four assurance levels, A, B, C, and D. In ICM, the system guarantees A, B, C, and D assurance tasks will run in A, B, C, and D criticality modes, respectively. A D assurance task can be decomposed to two B assurance tasks, and a B assurance task can again be decomposed to two A assurance tasks. If the criticality mode is switched from A to B, then the system guarantees that B assurance tasks will run and can drop A assurance tasks.

2.6 Estimated WCET

Schedulability analysis requires an exact estimated WCET. Generally, the complexity of MCSs is much higher than that of single-criticality systems due to its number of functions to fulfill requirements and hardware architecture. Hence, it is hard to achieve an exact WCET. However, if the estimated WCETs are incorrect, then the result of schedulability analysis will also be incorrect, which may cause severe damage. There are two main concerns for setting estimated WCETs for tasks in MCSs, which are as follows: 1) If we measure a WCET for a task in the worst condition, such as disabling the cache and the longest execution path, then the estimated WCET becomes too pessimistic since there is almost no probability of task executing in such a condition; and 2) if we measure WCETs as use cases, then the correctness of the WCET depends on test efforts and system complexity. (That is, the correctness of the estimated WCET will increase as test efforts—which again depend on system complexity—increase). However, we still cannot prove how correct the estimated WCET is in measuring WCET as a use case. In practice, the second method is prevailed upon to set WCETs for tasks with specific margin values; this normally comes from the experience of engineers or field data, since standards like ISO26262 and IEC61508 do not provide guidance on timing tests as assurance levels. Therefore, we typically generate internal guidance to have the estimated WCET as an assurance level. For instance, we can increase the number of test counts, code coverage, and number of use cases as assurance levels.

Definition 8. In an implicit WCET (IWCET), the correctness of WCET for a task depends on the assurance level of the task.

Definition 9. In an explicit WCET (EWCET), the correctness of the WCET for a task does not depend on the assurance level of the task.

In an IWCET, it is necessary to make efforts to use the estimated WCETs as assurance levels of tasks. However, in the EWCET, we must expend a lot of effort into having exact estimated WCETs of all tasks, regardless of the assurance level. The EWCET is prevailed in practice since it is hard to find rationales for using the IWCET. In fact, if we consider only safety certification, then we do not need to make a significant effort to measure the WCET since the safety accessor usually focuses on the development processor rather than specific real-time models or schedulability analysis and they do not know whether methods to measure the WCET are correct since there is no guidance in ISO26262 or IEC61508. Nevertheless, we manage the WCET in a significant effort for safety and availability, which directly affects system quality in practice. Hence, discovering rational methods of setting the estimated WCET as assurance levels of tasks, such as probability WCETs, can dramatically reduce development efforts and costs [19-23].

3. Limitations of Prior Works to the Industry

3.1 Static mixed-criticality with no monitoring

Vestal assumed that in $C_{iA} \geq C_{iB} \geq C_{iC} \geq C_{iD}$ for static mixed criticality with no runtime monitoring (SMC-NO), A is the highest assurance level and D is the lowest assurance level [5]. He also mentioned, "*For each task τ_i we want to assure to level L_i that τ_i never misses a deadline. We assume this level of assurance is achieved when the analysis is carried out using compute times having the same level of assurance*" [5].

Proposition 4. The criticality mode of SMC-NO is an ICM.

Proposition 5. The SMC-NO is a hybrid RTMCS.

Proposition 6. The WCET of SMC-NO is an IWCET.

Theorem 7. SMC-NO is not a timing-partitioned system in F4, F5, F8, and F9.

Proof: There can be LC, high-priority (HP) tasks. If AETs exceed the estimated WCET, or AITs exceed the EMITs in LCHP tasks, then HC, low-priority (LP) tasks miss the deadline. Hence, the system will not support time partitions in F4 and F5. F8 and F9 can cause F4 and F5, as shown in Lemma 1.

Intuitively, we can know that the measured WCETs for a low assurance level task are less exact than those for a high assurance level task due to different test efforts for execution time measurement. Measured WCETs for high assurance level tasks can be directly used for WCETs for both a high assurance level and low assurance level because it is the exact value that can be used in HC mode. However, measured WCETs for low-assurance tasks cannot be used directly for high-assurance WCETs. A margin value needs to be added to the original measured WCET for high-assurance WCETs in low-assurance tasks. Hence, the gap between WCETs for a low assurance level and high assurance level in a low-assurance task will be larger than that between WCETs for a low assurance level and high assurance level in a high-assurance task. For example, $C_{iA}, C_{iB}, C_{iC}, and\ C_{iD}$ of a level A assurance task τ_i can be equivalent, since a WCET is measured in assurance level A. However, $C_{jA}\ and\ C_{jB}$ of level B assurance task τ_j can be different, since a WCET is measured in assurance level B, and the WCET of assurance level A needs to be set to a more conservative value than that of assurance level B. $C_{jB}, C_{jC}, and\ C_{jD}$ of level B assurance task τ_j will again be equivalent. Hence, high-assurance tasks are likely to be deemed HP, even in LC mode, and this characteristic brings them close to becoming criticality as priority assignments (CAPAs). In other words, if a set of level A and level B assurance tasks are not schedulable, then the B assurance tasks will be assigned to a lower priority despite having a smaller relative deadline. Another characteristic is that, if a fault from an LCHP task occurs in LC mode without execution time monitoring, inducing the execution time to exceed WCET, then the HC task may also miss the deadline. Hence, criticality inversion may result in HC tasks missing deadlines.

3.2 Static mixed Criticality

WCETs are set differently based on the criticality of a task because executed jobs have different levels of criticality (as opposed to conservativity) of WCETs compared with static mixed criticality (SMC) [31,32]. The same priority assignment as in Audsley's approach and SMC response time analysis (RTA) are used for both SMC-NO and SMC. The only difference between SMC-NO and SMC is that jobs will be aborted in SMC if the AET exceeds WCET with execution monitoring, whereas jobs cannot be aborted in SMC-NO even if AET exceeds WCET since there is no execution monitoring.

Proposition 8. The criticality mode of SMC is an ICM.
Proposition 9. The SMC is a hybrid RTMCS.
Proposition 10. The WCET of SMC is an IWCET.

Theorem 11. The SMC is not a timing-partitioned system in F5, F8, and F9.

Proof: There can be LCHP tasks. If AITs exceed EMITs in LCHP tasks, then HCLP tasks miss their deadline. Hence, the system will not support time partitioning in F5. As shown in Lemma 1, F8 and F9 can cause F5.

3.3 Adaptive mixed criticality

If AET exceeds $C_l(LO)$ for tasks in a low criticality level, then the system criticality level is changed from low to high, and all LC tasks are dropped in adaptive mixed criticality (AMC) [31,32]. The main difference between AMC and SMC is that AMC drops all LC tasks if AET is over $C_l(LO)$ for any task, while SMC only drops a task if AET is over $C_l(LO)$ for an LC task. In the discussion regarding SMC-NO in Vestal's work, the main purpose of different WCETs is said to be the conservativity of criticality. A conservative WCET can be used for a high-assurance task, which may reduce the test effort for determining the exact WCET. Vestal focused on preventing the high-assurance task from missing the deadline of criticality inversion with conservative WCET (without execution monitoring and with reasonable WCET) for schedulability, since there is a low probability of the worst case occurring in all tasks at the same time. Both high-assurance and low-assurance tasks run on HC mode. The main purpose of different WCETs is executing different jobs according to criticality [31]. Hence, the WCET of every job with a high assurance level is tested, and low-assurance tasks will not be executed in the HC mode in AMC. However, executed jobs of high-assurance tasks may not differ between being in HC mode and LC mode in practice. Rather, the jobs required to run on HC mode will be allocated to high-assurance tasks, and other jobs required to run on LC mode will be allocated to low-assurance tasks.

RTA for the stable mode and the AMC-rtb and AMC-max for the criticality change are proposed in [31]. However, generally, the criticality mode is switched to HC mode for degradation or the safe state of the system that is given to recalculate the deadline within the fault-tolerant time interval (FTTI) at the beginning of a timing fault occurring in LC mode in practice. AMC has been extended for minimization of stack size and multiple frequency [33,34].

Proposition 12. The criticality mode of AMC is an ICM.
Proposition 13. The AMC is a hybrid RTMCS.
Proposition 14. The WCET of AMC is an IWCET.

Theorem 15. The AMC is not a timing-partitioned system in F5, F8, and F9.

Proof: There can be LCHP tasks. If AITs exceed EMITs in LCHP tasks, then HCLP tasks miss the deadline. Hence, the system will not support time partitions in F5. As shown in Lemma 1, F8 and F9 can cause F5.

3.4 Zero-Slack scheduling

Zero-slack scheduling (ZSS) has been developed to solve criticality inversion in an overload condition, since criticality inversion is the only problem in this condition [35,36]. Task t_i in ZSS has two WCETs for normal mode C_i^0 and critical mode C_i, and $C_i \leq C_i^0$ because normal mode allows an overload condition, while critical mode does not. Hence, C_i^0 is deemed a conservative WCET. In normal mode, the scheduler is intended to maximize schedulability with C_i^0 as an RM or DM parameter; RM and DM are used for optimal fixed-priority scheduling. During runtime, admission control is required for mode switching. This calculates the slack time of higher criticality and lower priority tasks compared with a running task. If the slack time for C_i^0 is not enough in a task with higher criticality and lower priority, then the mode is switched from normal to critical mode, and tasks are scheduled as CAPAs. In critical mode, priority blocking of lower criticality tasks is increased, with C_i^0 of higher criticality tasks with lower priority. Hence, the schedulability of ZSS in criticality mode will be poor, since the schedulability of CAPA is much worse than that of DM or RM. A disadvantage is that the scheduling overhead will not be minor, since the slack time of higher criticality tasks is continuously calculated.

Proposition 16. The criticality mode of ZSS is an ICM.
Proposition 17. ZSS is a hybrid RTMCS.
Proposition 18. The WCET of ZSS is an IWCET.

Theorem 19. ZSS is a timing-partitioned system.

Proof: The slack times of tasks are monitored during runtime. The slack time will not be enough if F4 and F5 occur. Thereafter, the criticality mode will be switched from LC to HC mode. In HC mode, there is no criticality inversion in CAPA.

3.5 EDF with virtual deadline

EDF with virtual deadline (EDF-VD) reduces the deadline for HC tasks to solve the criticality inversion problem in an overload condition [39,40]. To guarantee HC tasks meeting the deadline in the offline phase, HC tasks use a virtual deadline $\hat{d}_i$, which is calculated by $\hat{d}_i = xd_i$. Scaling factor x is defined as follows: In the runtime phase, tasks with lower criticality than the current critical mode are discarded, and the virtual deadline of HC tasks is not applied in favor of the original deadline. Hence, the algorithm can guarantee HC tasks meeting their deadlines. However, total utilization will be decreased in the overload condition, as total utilization is calculated in the LC level because a smaller deadline increases utilization. It follows that the criticality mode is easily switched from the LC mode to the HC mode. Similarly, EDF scheduling based on the demand-bound function was developed for MCSs in [41,42]. These studies assumed that the relative deadlines of tasks can be freely tuned if the deadlines are not beyond the true relative deadline specified by the system designer. Hence, they set the relative deadline to be smaller than the original relative deadline for lower criticality mode. This has an effect like that of the larger WCET in HC mode. However, both the smaller deadline and larger WCET decrease schedulability. In practice, we do not have enough CPU resources since computing power is related to cost. Generally, it is hard to make a feasible system even with the maximum relative deadline being specified by the system designer. The EDF-VD has recently been extended in many studies. Under this assumption, multiple-criticality modes exist, and a fault from HC tasks does not trigger switching the criticality mode to HC mode [43,44]. Under the assumption of the imprecise WCET of LC tasks, EDF-VD has been extended, and the criticality mode can switch back from HC mode to LC mode according to the completion rate of the tasks [45,46].

Proposition 20. The criticality mode of EDF-VD is an ICM.
Proposition 21. EDF-VD is a hybrid RTMCS.
Proposition 22. The WCET of EDF-VD is an IWCET.

Theorem 23. EDF-VD is not a timing-partitioned system in F4, F5, F8, and F9.

Proof: There can be LCHP tasks. If AETs exceed the estimated WCET or AITs exceed EMITs in LCHP tasks, then HCLP tasks miss the deadline. Hence, the system will not support time partitioning in F4 and F5. As shown in Lemma 1, F8 and F9 can cause F4 and F5.

3.6 Latest works

The proposed system model is a context-aware, mixed criticality model that extends the execution time budget of tasks for degraded systems in [47]. However, the model is still not matched with the practical model, and an execution time budget must be defined in consideration of FTTI. A runnable mapping algorithm based on DM and preemption threshold was proposed in [48]. This study assumed that runnable in the same SWCs had different criticality levels. However, this is not a practical design, since it violates freedom from interference in memory partitioning. In addition, if different criticality runnables are assigned in the same SWC, modifiability and extensibility will be decreased from a software design perspective, and there will be no advantage using the Automotive Open System Architecture (AUTOSAR), a component-based platform. From a safety perspective, software failure mode and effect analysis (FMEA) will be much more difficult in such a design.

EDF-VD was extended in [37,38]. The studies proposed EDF with a universal virtual deadline (EDF-UVD) in which both LC tasks and HC tasks have a virtual deadline, while HC tasks have a virtual deadline in EDF-VD. Instead, LC tasks have smaller execution budgets than the original execution budgets in LC mode. The proposed fault-tolerant mixed-criticality (FTMC) scheduling assumes a system model where a task has WCETs to consider executing backups in fault cases. However, functions for backups or diagnostics are deployed to different SWCs and tasks in the design and safety perspective, since the criticality levels of tasks for detecting faults and backups are not equal to the criticality levels of tasks for normal functions. Generally, normal functions are not safety-related functions, while detecting faults and backups are safety-related functions [49]. Deadline monotonic mixed-criticality (DM^{2}C) on ECM is recently published. RM schedulability analysis is used for normal operation during design phase and online response time analysis (ORTA) with conservative WCETs is used to let system go to safe sate if analysis is failed during run-time [50].

4. RTMCS Models

There are three MCS models described in prior research, which are as follows: 1) tasks with a WCET per criticality mode; 2) tasks with a WCET and deadline per criticality mode; and 3) tasks with a WCET, deadline, and period per criticality mode. The WCETs, deadlines, and periods are set by the criticality mode, and the assurance levels are same as the criticality modes. For instance, if there are four assurance levels A, B, C, and D, then A assurance tasks run in A criticality mode, B assurance tasks run in B criticality mode, C assurance tasks run in C criticality mode, and D assurance tasks run in D criticality mode. However, generally, the number of criticality modes is greater than that of assurance tasks, and mixed assurance tasks can be required to run in a criticality mode in practice. In addition, a single WCET for all criticality modes is prevalent in practice. Rather, a single WCET for all criticality modes and a period and/or deadline per criticality mode is more prevalent than models of prior research in practice because system engineers can set the different period or deadline as a criticality mode but they do not care about WCET generally.

4.1 IWCET and EWCET with ICM

System model 1. A WCET per ICM (WCI)

$$\tau_i \in \tau, \tau_i = (\vec{C}_i, T_i, D_i, L_i)$$

System model 2. A WCET and a deadline per ICM (WDCI)

$$\tau_i \in \tau, \tau_i = (\vec{C}_i, T_i, \vec{D}_i, L_i)$$

System model 3. A WCET, deadline, and period per ICM (WDPCI)

$$\tau_i \in \tau, \tau_i = (\vec{C}_i, \vec{T}_i, \vec{D}_i, L_i)$$

The WCM, WDCM, and WDPCM are IWCETs. IWCETs are attractive models for reducing efforts and costs. The three models were proposed in prior works. However, EWCETs are still predominant in practice, since it is not easy to provide a rationale and evidence for IWCET.

System model 4. Single WCET (s-I)

$$\tau_i \in \tau, \tau_i = (C_i, T_i, D_i, L_i)$$

In practice, s-I is comparable to traditional RTS models. However, it is still predominant with time partitioning in practice. Generally, the requirement or safety engineer gives time constraints with the period according to the criticality modes, and the implicit deadline is prevalent in reducing the complexity of requirements and design.

4.2 IWCET and EWCET with ECM

As mentioned in section II, the system cannot reach safety with an ICM wherein tasks are decomposed. However, the decomposition is the most important design method in MCS.

System model 5. A WCET per ECM (WCE)

$$\tau_i \in \tau, M_k \in M, M_k(\tau_i) = \left(\vec{C}_i, T_i, D_i, L_i\right)$$

System model 6. A WCET and deadline per ECM (WDCE)

$$\tau_i \in \tau, M_k \in M, M_k(\tau_i) = (\vec{C}_i, T_i, \vec{D}_i, L_i)$$

System model 7. AWCET, deadline, and period per ECM (WDPCE)

$$\tau_i \in \tau, M_k \in M, M_k(\tau_i) = (\vec{C}_i, \vec{T}_i, \vec{D}_i, L_i)$$

The task sets correspond to criticality modes, regardless of their assurance levels in ECM. Different EMITs, estimated WCETs, and deadlines for a task can be set as criticality modes and set again as assurance levels. A task has three properties for each mode, namely, HRT, SRT, and none. Let us assume that there are three criticality modes and two assurance levels with WDPCE as in the following:

$$assurance\ level\ L_i = \{L_1, L_2\}$$
$$criticality\ mode\ M_i = \{M_1, M_2, M_3\}$$

We can present a task as follows:

$$M_1 = HRT, M_2 = HRT, M_3 = SRT$$

$$M_1(\tau_i) = \begin{matrix} C_{i,1}(L_1) & T_{i,1}(L_1) & D_{i,1}(L_1) \\ C_{i,1}(L_2) & T_{i,1}(L_2) & D_{i,1}(L_2) \end{matrix}$$

$$M_2(\tau_i) = \begin{matrix} C_{i,2}(L_1) & T_{i,2}(L_1) & D_{i,2}(L_1) \\ C_{i,2}(L_2) & T_{i,2}(L_2) & D_{i,2}(L_2) \end{matrix}$$

$$M_3(\tau_i) = \begin{matrix} C_{i,3}(L_1) & T_{i,3}(L_1) & D_{i,3}(L_1) \\ C_{i,3}(L_2) & T_{i,3}(L_2) & D_{i,3}(L_2) \end{matrix}$$

In hard RTMCS, M_3 can be $NONE$, and hard RTMCS with WCE is the predominant model in practice. In WCE, $T(L_1)$ is equal to $T(L_2)$, and $D(L_1)$ is equal to $D(L_2)$. However, $M_i(T)$ is not equal to $M_j(T)$, nor is $M_i(D)$ equal to $M_j(D)$.

System model 8. A single WCET and ECM (s-E)

$$\tau_i \in \tau, M_k \in M, M_k(\tau_i) = (C_i, T_i, D_i, L_i)$$

In s-E, estimated WCETs, EMITs, and deadlines are set only as criticality modes, with the following assurance levels:

$$M_1(\tau_i) = C_{i,1} \quad T_{i,1} \quad D_{i,1}$$
$$M_2(\tau_i) = C_{i,2} \quad T_{i,2} \quad D_{i,2}$$
$$M_3(\tau_i) = C_{i,3} \quad T_{i,3} \quad D_{i,3}$$

Generally, EMITs are more exactly estimated than estimated WCETs are by static analysis. Hence, hard RTMCS with WCE or s-E is a predominant model in practice. However, WCE is still a novel area, and probabilistic WCET (pWCET) can be an alternative method to provide a rationale for the estimated WCET as the assurance level [19-23]. There are two purposes for using the estimated WCET for assurance levels. The first is to reduce the efforts in the runtime measurement of WCET. Runtime measurement cannot fully guarantee an exact estimated WCET, and we cannot spend unlimited efforts on the test. Hence, we can add additional execution time for high-assurance tasks instead of reducing efforts. The second purpose is avoiding a pessimistic estimated WCET since this will lead to poor schedulability. In fact, we may set single estimated WCETs as an assurance level of a task in schedulability analysis. If the assurance level of the task is low, then the reliability of the task is low. We can assume that the assurance level of the estimated WCET also depends on the assurance level of the task. Hence, schedulability will increase if we use a single estimated WCET as the assurance level of a task. However, this can affect system availability due to the probability that the AET of low-assurance tasks will exceed the estimated WCET. Hence, this is still hypothetical.

5. Scheduling Algorithm in ECM

As mentioned above, system models based on ECM are widespread in practice, and fixed priority scheduling is still prevalent for safety-critical systems. Hence, fixed priority scheduling on ECM can be easily adopted in practice.

5.1 SMC based on ECM

In WCE, WDCE, WDPCE, and s-E, a set of tasks exists for a criticality mode. If there are three criticality modes, then it is likely that three sets of tasks exist in one system.

Let us assume there are three criticality modes, two assurance levels, and an implicit deadline based on WDPCE, as follows in example 1:

$$M_1 = HRT, M_2 = HRT, M_3 = HRT$$
$$L = \{B, A\}, B > A, C(B) \geq C(A), T(B) \leq T(A),$$
$$L_1 = A, L_2 = B, L_3 = B$$

$$M_1(\tau_1) = \begin{matrix} 2 & 5 \\ 1 & 5 \end{matrix}, M_2(\tau_1) = \begin{matrix} 2 & 10 \\ 1 & 10 \end{matrix}, M_3(\tau_1) = \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$$

$$M_1(\tau_2) = \begin{matrix} 4 & 10 \\ 4 & 10 \end{matrix}, M_2(\tau_2) = \begin{matrix} 4 & 10 \\ 4 & 10 \end{matrix}, M_3(\tau_2) = \begin{matrix} 4 & 10 \\ 4 & 10 \end{matrix}$$

$$M_1(\tau_3) = \begin{matrix} 3 & 20 \\ 3 & 20 \end{matrix}, M_2(\tau_3) = \begin{matrix} 2 & 5 \\ 2 & 5 \end{matrix}, M_3(\tau_3) = \begin{matrix} 2 & 5 \\ 2 & 5 \end{matrix}$$

For M_1, we, can assign the lowest priority to t_3 based on the SMC RTA as

$$R_i = C_i + \sum_{t_j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j(L)$$

This can be verified for M_1 as follows:

$$R_3 = 3 + \left\lceil \frac{R_3}{5} \right\rceil \cdot 2 + \left\lceil \frac{R_3}{10} \right\rceil \cdot 4 \, for \, M_1$$

$$R_3 = 3 + \left\lceil \frac{R_3}{5} \right\rceil \cdot 2 + \left\lceil \frac{R_3}{10} \right\rceil \cdot 4 \, for \, M_2$$

The solution is 19 for M_1 but there is no solution that involves assigning the lowest priority τ_3 to M_2. Hence, fixed priority scheduling for MCS, such as SMC and AMC, is no longer optimal. Example 1 is close to practice, since the high assurance task τ_3 works slowly with a longer execution time in normal mode M_1, and the period of the high assurance task τ_3 is shorter to meet FTTI instead of reduced execution time to uphold important safety functions. For M_2, if we assign the lowest priority to τ_2, then the set of tasks is feasible at time 10; this is shown in the following:

$$R_2 = 4 + \left\lceil \frac{R_2}{5} \right\rceil \cdot 2 + \left\lceil \frac{R_2}{10} \right\rceil \cdot 2 \, for \, M_2$$

5.2 MPP

With a practical example, we showed that prior fixed scheduling for MCSs cannot schedule for different criticality modes. The MPP involves the following steps:

- R1) Priority assignment with fixed priority scheduling, such as SMC or AMC, is performed for a set of tasks on a criticality mode.
- R2) Schedulability analysis is performed for a set of tasks on the next criticality mode with the previously assigned priorities. If the task set is not feasible, then go to R1.
- R3) A task has priorities corresponding to criticality modes based on R1 and R2.
- R4) If the mode is switched, the priorities of the tasks are also changed.
- R5) For time partition, execution and arrival monitoring are performed. If faults occur, then the mode is switched as a safety scenario.

The MPP with SMC in WDPCE is as follows:

$$M_l(R_i) = M_l(C_i) + \sum_{t_j \in hp(i)} \left\lceil \frac{M_l(R_i)}{M_l(T_j)} \right\rceil M_l\left(C_j(L)\right). \qquad (1)$$

The MPP with SMC in WCE is expressed as

$$M_l(R_i) = M_l(C_i) + \sum_{t_j \in hp(i)} \left\lceil \frac{M_l(R_i)}{T_j} \right\rceil M_l\left(C_j(L)\right). \qquad (2)$$

Finally, the MPP in s-E is given by

$$M_l(R_i) = M_l(C_i) + \sum_{t_j \in hp(i)} \left\lceil \frac{M_l(R_i)}{T_j} \right\rceil M_l\left(C_j\right). \qquad (3)$$

RTA should be performed for each criticality mode in (1)–(3). Equations (1) and (2) are based on SMC in WDPCE and WCE, respectively, and (3) is based on traditional RTA in s-E. We show the transformed RTA with SMC and traditional approaches. However, a priori works for fixed scheduling in MCS can be transformed with the MPP.

5.3 Implementation Considerations

A scheduler for fixed priority scheduling generally requires a data structure with a single priority for a task. To extend existing fixed priority schedulers to support MPP, the scheduler also requires only mode information and corresponding priority. Typically, the system does not have many criticality modes; the number will usually not be over 10. Therefore, increased memory size is also a minor issue for implementation in consideration of execution and arrival time monitoring. From a performance perspective, when criticality mode is switched, the scheduler directly uses priorities corresponding to the criticality mode. There is no additional scheduling overhead. Simply, the data structure can be modified as follows:

$$\tau_i = (P_i, C_i, T_i, L_i) \rightarrow \tau_i = \left(\vec{M}_i, \vec{P}_i, \vec{C}_i, \vec{T}_i L_i\right)$$

where:

P_i: Priority

C_i: Execution time threshold

T_i: Interarrival time threshold

L_i: Assurance level

$\vec{M}_i$: Vector of criticality mode

$\vec{P}_i$: Vector of priority

$\vec{C}_i$: Vector of execution time threshold

$\vec{T}_i$: Vector of interarrival time threshold

6. A Case Study of MCS Design In the Automotive Industry

At the beginning the development phase for MCS design, hazard, and risk analysis are performed based on the architecture and design. The safety goal comes from the hazard and risk analysis, as follows:

- SG1: Avoid high-voltage battery out gassing (ASIL-D); and
- SG2: Avoid unintended high voltage (ASIL-A).

Each safety goal is assigned as an ASIL. After defining the safety goals, we generate functional safety concepts (FSCs) that derive safety mechanisms and safety requirements with ASIL. Based on the FSCs, we generate technical safety concepts (TSCs) that map logical elements to technical ones. During generation of the FSCs and TSCs, we generally decompose to degrade ASIL as much as possible in consideration of design aspects like modifiability, extensibility, complexity, and performance. Thereafter, SW and HW architectures are designed iteratively.

MCSs require memory partitioning for freedom from interference. Lower ASIL SWCs are not allowed to write data to the memory area of higher ASIL SWCs. In AUTOSAR, we can make partitions with application containers that include tasks. In supervisor mode, we give full permission to the highest SIL SWCs, which can access all memory areas and registers. Other SWCs are required to set the accessible memory range in user mode. In Fig. 1, each core has three application containers, where Application_ABC0 is ASIL-A on core 0, Application BBC0 is ASIL-B on core 0, and System_Application_C0 is ASIL-C on core 0.

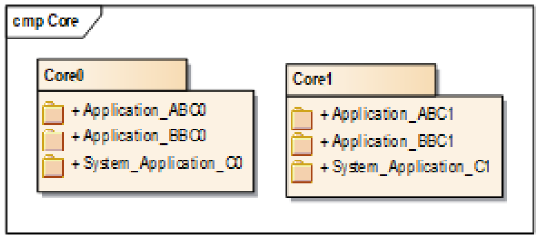

Fig. 1. Application containers.

Figs. 2 and 3 show tasks assigned to cores 0 and 1, respectively. AUTOSAR has the limitation that a BSW stack must be placed on only one core. We allocate the BSW stack to core 0, as MPC5746 supports the lock-step function for this core. The number of tasks in the system application on core 0 is more than that on core 1, since all device drivers are allocated to core 0. Instead, we allocate all application SWs to core 1 except for the sensor/actuator components.

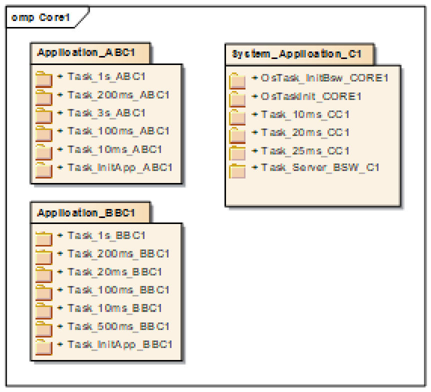

Fig. 2. Tasks on core 0.

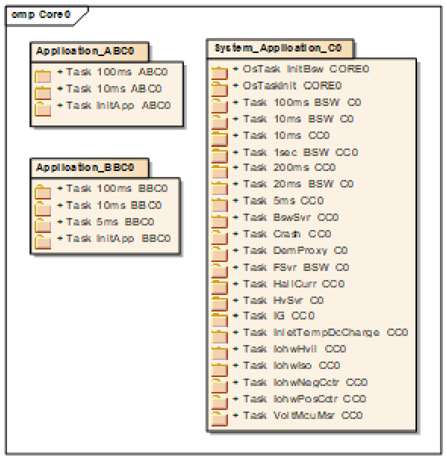

Fig. 3. Tasks on core 1.

For time partition, we use execution time and arrival time monitoring. However, generally, the threshold values are much larger than the estimated WCETs and EMITs since the purpose is to detect systematic and random HW faults rather than time faults from scheduling. Hence, we can also consider time partition from scheduling policy or algorithms with the estimated WCETs and EMITs.

7. Conclusion

Prior studies for RTMCS have been researched based on ICM. Although EMC is more complex than ICM is, ECM must be considered for developing practical real-time scheduling algorithms. The proposed system models based on ECM can be used for the assumption of MCS in future studies. In addition, the MPP can be easily extended to overcome limitations of prior works and adapted to existing real-time schedulers, as well as the AUTOSAR platform, with implementation considerations. This article only considered fixed priority in ECM. Hence, dynamic priority scheduling algorithms and locking protocols based on ECM needs to be developed in future studies.

8. Acknowledgement

This work was supported by Great Wall Motors and Volt Energies under the EV/PHEV platform Program (EC01, ME01).

REFERENCES

1. Reinhardt D and Morgan G. An embedded hypervisor for safety-relevant automotive E/E-systems. In Proc. 9th IEEE Int. Symp. Ind. Embedded Syst. (SIES 2014), Pisa, Italy, 2014, 189–198p.
2. Wu EC, and Sun C. Research advances and challenges of autonomous and connected ground vehicles. IEEE Trans Intell Transp Syst. 2019:1–29.
3. Siegel J, Erb D, and Sarma S. Algorithms and architectures: A case study in when, where and how to connect vehicles," IEEE Intell Transport Syst Mag. 2018;10(1):74–87.
4. Chakraborty S, Lukasiewycz M, Buckl C, et al. Embedded systems and software challenges in electric vehicles. in Proc. Design, Automat. & Test in Europe Conf. & Exhib. (DATE), Dresden, Germany, 2012, 424–429p.
5. Vestal S. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In proc. 28th IEEE Int Real-Time Systems Symposium (RTSS 2007), Tucson, AZ, USA, 2007, 239–243p.
6. Nelissen EG, Nélis V, and Tovar E. An industrial view on the common academic understanding of mixed-criticality systems," Real-Time Syst. 2018;54(3)745–795.
7. Nelissen EG, Nélis V, and Tovar E. How realistic is the mixed-criticality real-time system model? In Proc. of the 23rd Int. Conf. on Real Time and Networks Systems (RTNS '15), Lille, France, 2015, 139–148p.
8. Ernst Rand M. Di Natale M. Mixed criticality systems—a history of misconceptions? IEEE Des Test. 2016;33(5):65–74.
9. Baruah S. Mixed- criticality scheduling theory: scope, promise, and limitations. IEEE Des. Test. 2018;35(2): 31–37.
10. Ward DD, Rivett RS, Jesty PH. A generic approach to hazard analysis for programmable automotive systems. Presented at the SAE World Congress & Exhibition, 2007, 2007-01–1620p.
11. Czerny J, Suchala R, and Runyon M. A scenario-based approach to assess exposure for ASIL determination. Presented at the SAE 2014 World Congress & Exhibition, 2014, 2014-01–0211p.
12. Christiaens S, Ogrzewalla J, and Pischinger S. Functional safety for hybrid and electric vehicles. Presented at the SAE 2012 World Congress & Exhibition, 2012, pp. 2012-01–0032p.

13. Eriksson DH, Bartha FA, Xu F, et al. Using rigorous simulation to support ISO 26262 hazard analysis and risk assessment. In Proc. 2015 IEEE 17th Int. Conf. on High Performance Computing and Communications, New York, NY, 2015, 1093–1096p.
14. Heckmann R, Langebach M, Thesing S, et al. The influence of processor architecture on the design and the results of WCET tools. Proc. IEEE. 2003;91(7):1038–1054.
15. Lundqvist T and Stenstrom P. Timing anomalies in dynamically scheduled microprocessors. In Proc. 20th IEEE Real-Time Systems Symposium (Cat. No. 99CB37054), Phoenix, AZ, USA, 1999, 12–21p.
16. Isermann R, Schwarz R, and Stolzl S. Fault-tolerant drive-by-wire systems. IEEE Control Syst Mag. 2002;22(5) 64-81.
17. Nicolaidis M, Nom S, and Courtois B. A generalized theory of fail-safe systems. In *Proc.* FTCS-19, IEEE CS Press, Chicago, IL, USA, 1989, 398–406p.
18. Lubaszewski M and Courtois B. A reliable fail-safe system. IEEE Trans Comput. 1998;47(2):236–241.
19. del Castillo XCJ and Abella J. A reliable statistical analysis of the best-fit distribution for high execution times. In Proc. 2018 21st Euromicro Conf. on Digital System Design (DSD), Prague, 2018, 727–734.
20. Altmeyer S, Cucu-Grosjean L, Davis RI. Static probabilistic timing analysis for real-time systems using random replacement caches. Real-Time Syst. 2015;51(1):77–123.
21. Cucu-Grosjean L, Santinelli L, Houston M, et al. Measurement-based probabilistic timing analysis for multi-path programs. In Proc.2012 24th Euromicro Conference on Real-Time Systems, Pisa, Italy, 2012, 91–101p.
22. Davis RI, Santinelli L, Altmeyer S, et al. Analysis of probabilistic cache related pre-emption delays. In Proc. 2013 25th Euromicro Conf. on Real-Time Systems, Los Alamitos, CA, USA, 2013,168–179p.
23. Edgar S, and Burns A. Statistical analysis of WCET for scheduling. In Proc. 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No. 01PR1420), London, UK, 2001, 215–224p.
24. Avizienis A. The N-version approach to fault-tolerant software. IEEE Trans Softw Eng. 1985;SE-11(12):1491–1501.
25. Eckhardt E and Lee LD. A theoretical basis for the analysis of multiversion software subject to coincident errors. IEEE Trans Softw Eng. 1985;SE-11(12):1511–1517.
26. Ammann PE and Knight JC. Data diversity: An approach to software fault tolerance. IEEE Trans Comput. 1988;37(4):418–425.
27. Brilliant SS, Knight JC, and Leveson NG. Analysis of faults in an N-version software experiment. IEEE Trans Softw Eng. 1990;16(2):238–247.
28. Latif-Shabgahi G, Bass JM, and Bennett S. A taxonomy for software voting algorithms used in safety-critical systems. IEEE Trans Rel. 2004;53(3):319–328.
29. Vermeulen FB and Goossens K. A generic method for a bottom-up ASIL decomposition. In Proc. Develop. Lang. Theory, vol. 11088, Hoshi M and Seki S, Eds., New York, NY, USA, Springer International Publishing, 2018, 12–26p.
30. Vermeulen FB and Goossens K. Component-level ASIL decomposition for automotive architectures. In Proc. 2019 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. and Netw. Workshops (DSN-W), Portland, OR, USA, 2019, 62–69p.

31. Baruah SK, Burns A, and Davis RI. Response-time analysis for mixed criticality systems. In Proc. 2011 IEEE 32nd Real-Time Systems Symposium, Vienna, Austria, 2011, 34–43p.
32. Baruah S, and Chattopadhyay B. Response-time analysis of mixed criticality systems with pessimistic frequency specification. In Proc. 2013 IEEE 19th Int. Conf. on Embedded and Real-Time Computing Systems and Applications, Taipei, Taiwan, 2013, 237–246p.
33. Zhao Q, Gu Z, Zeng H, et al. Schedulability analysis and stack size minimization with preemption thresholds and mixed-criticality scheduling. J Systs Archit. 2018;83:57–74.
34. Baruah S. Schedulability analysis of mixed-criticality systems with multiple frequency specifications. In Proc. of the 13th Inter. Conf. on Embedded Software (EMSOFT '16), Pittsburgh, PA, 2016, 1–10p.
35. de Niz D and Phan LTX. Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms. In proc. 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), Berlin, Germany, 2014, 111–122p.
36. De Niz, L. Rowe WA, and Rajkumar R. Utility- based resource overbooking for cyber-physical systems. ACM Trans Embed Comput Syst. 2014;13(5s):1–25.
37. Gu X and Easwaran A. Dynamic budget management with service guarantees for mixed-criticality systems. In Proc. 2016 IEEE Real-Time Systems Symposium (RTSS), Porto, Portugal, 2016, 47–56p.
38. Gu X and Easwaran A. Dynamic budget management and budget reclamation for mixed-criticality systems. Real-Time Syst. 2019;55(3):552–597.
39. Baruah SK, Bonifaci V, D'Angelo G, et al. Mixed-criticality scheduling of sporadic task systems. In Proc. Algorithms—ESA 2011, vol. 6942, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, 555–566p.
40. Baruah S, Bonifaci V, D'angelo G, et al. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. J ACM. 2015;62(2):1–33.
41. Ekberg P and Yi W. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. Real-Time Syst. 2014;50(1):48–86.
42. Ekberg P and Yi W. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. Real-Time Syst. 2014;50(1):48–86.
43. Guan CN, Hu B, and Yi W. EDF-VD scheduling of flexible mixed-criticality system with multiple-shot transitions. IEEE Trans Comput Aided Des Integr Circuits Syst. 2018;37(11):2393–2403.
44. Chen G, Guan N, Liu Di, et al. Utilization-based scheduling of flexible mixed-criticality real-time tasks. IEEE Trans Comput. 2018;67(4):543–558.
45. Liu Di, Guan N, Spasic J, et al. Scheduling analysis of imprecise mixed-criticality real-time tasks. IEEE Trans Comput. 2018;67(7):975–991.
46. Guo Z, Yang K, Vaidhun S, et al. Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate. In Proc. 2018 IEEE Real-Time Systems Symposium (RTSS), Nashville, TN, 2018,373–383p.
47. Sundar VK, and Easwaran A. A practical degradation model for mixed-criticality systems. In Proc. 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), Valencia, Spain, 2019, 171–180p.
48. Zhao Q, Gu Z, and Zeng H. Design optimization for AUTOSAR models with preemption thresholds and mixed-criticality scheduling. J. Syst Arch. 2017;72:61–68.

49. Pathan RM. Fault-tolerant and real-time scheduling for mixed-criticality systems. Real-Time Syst. 2014;50(4):509–547.
50. Lee J and Kim M. Real-Time scheduling for mixed-criticality systems in the automotive industry. J Comput Sci Eng. 2020;14(1):9-18.

Authors Biography

Junghwan Lee **(ORCID)** received the M.S. degree in computer science from Chungbuk National University, Chungbuk, South Korea in 2001. He is currently working with the Battery Management System, Department of Great Wall Motors, Hebei, China as an Architect and pursuing the Ph.D. degree in computer science at Chungbuk National University. He worked at Texas Instruments, Seoul, South Korea from 2012 to 2017 and LG Electronics, Seoul, South Korea from 2005 to 2012. His research interests include real-time systems, architecture, and safety.

Myungjun Kim **(ORCID)** received the M.S. degree in computer science from the Florida Institute of Technology, FL, USA in 1984 and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA in 1992. He is currently a professor with the Department of Computer Science at Chungbuk National University, Chungbuk, South Korea. His research interests include real-time systems and distributed computing systems.

Citation: Lee J and Kim M. Generalized models of mixed-criticality systems for real-time scheduling. Trans Eng Comput Sci. 2020;1(1):106.